Enunciado 5

Problema del caballo

INDICE

1 Definición del problema

Se trata de colocar inicialmente el caballo en una casilla de un tablero de ajedrez y moverlo hasta haber pisado todas las casillas sin haber estado más de una vez en la misma casilla. En la figura 1 se muestra un ejemplo del movimiento del caballo. En la figura 2 se muestra un ejemplo de los ocho movimientos posibles de un caballo en un tablero de 8x8.

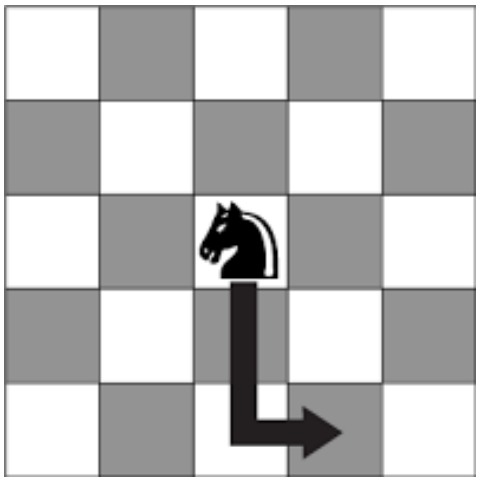

Figura 1: Movimiento del caballo

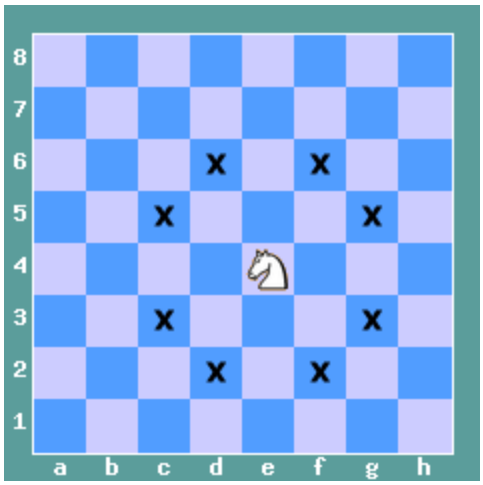

Figura 2: Los 8 movimientos de un caballo

A continuación, podemos observar varias soluciones para tableros de tamaño 5x5 hasta 8x8:

1	12	17	22	3
18	25	2	11	16
13	8	23	4	21
24	19	6	15	10
7	14	9	20	5

1	22	9	34	3	24
10	33	2	23	16	31
21	8	35	32	25	4
36	11	26	17	30	15
7	20	13	28	5	18
12	27	6	19	14	29

1	32	23	42	3	30	27
22	47	2	31	28	43	4
33	24	45	48	41	26	29
46	21	34	25	44	5	38
15	18	49	40	37	8	11
20	35	16	13	10	39	6
17	14	19	36	7	12	9

1	16	39	22	3	18	49	56
38	23	2	17	60	55	4	19
15	40	63	54	21	48	57	50
24	37	42	59	64	61	20	5
41	14	53	62	47	58	51	30
36	25	46	43	52	31	6	9
13	44	27	34	11	8	29	32
26	35	12	45	28	33	10	7

Figura 3: Una de las soluciones para diversos tamaños de tablero

2 Solución secuencial

Utilizaremos un algoritmo recursivo de vuelta atrás (*backtracking*) basado en ir colocando, según un orden, un caballo detrás de otro evitando las casillas ya ocupadas.

El tablero lo guardaremos como una matriz de NxN (siendo N el número de filas y columnas) de tal forma que inicialmente tendrá valores cero (casillas todavía no visitadas). El programa recibe como único parámetro la posición de partida del caballo, donde se ubicará el primer caballo marcándolo con un 1 en el tablero para indicar esta circunstancia.

A partir de la posición inicial, habría ocho movimientos posibles para poner un segundo caballo. El orden en el que se explorarán estos posibles movimientos se refleja a continuación en la Figura 4.

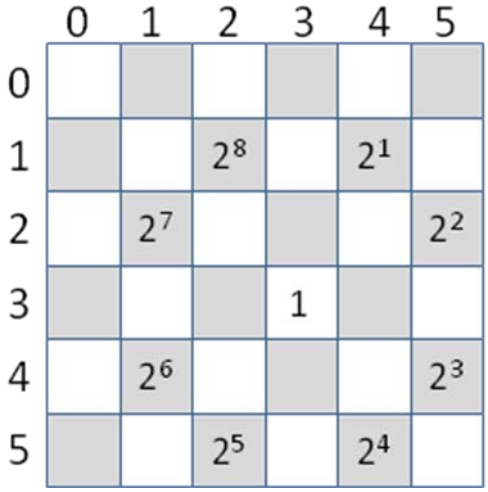

Figura 4: Orden en el que realizaremos los movimientos del caballo

Para cada uno de estos segundos movimientos, imprimiremos la primera solución encontrada, así como el total de soluciones existentes y el tiempo invertido en ello. Un ejemplo de salida para este mismo ejemplo de la figura 4 sería:

```
3:3 ----------------------------------
Ensayando en 1:4
---------- SOLUCION ----------
31  28  33  18   7  26
34  19  30  27   2  17
29  32  21   6  25   8
20  35  12   1  16   3
11  22   5  14   9  24
36  13  10  23   4  15
NumSoluciones = 6411 Tiempo ensayo =>  25:675 seg:miliseg
Ensayando en 2:5
---------- SOLUCION ----------
 7  36  21  24   9  30
20  25   8  29  22  11
35   6  23  10  31   2
26  19  28   1  12  15
 5  34  17  14   3  32
18  27   4  33  16  13
NumSoluciones = 4716 Tiempo ensayo =>  24: 58 seg:miliseg
```

Figura 5: Ejemplo parcial de una ejecución: **caballoSec 3 3**

El algoritmo consistirá en un procedimiento "**saltoCaballo** (int x, int y)" que se invoca con la posición inicial del caballo y que, a su vez, llamará al procedimiento recursivo **saltoCaballoR** con cada uno de los ocho movimientos posibles.

Esta solución secuencial se suministra totalmente escrita en el fichero **caballoSec.c** (inicialmente configurado para un tablero de 5x5). Si generamos el ejecutable mediante el **Makefile** que también se facilita, las soluciones para el caso del caballo en 3 3 se obtendrían ejecutando: ./caballoSec 3 3

La salida que produciría sería como la siguiente:

```
3:3 ------------------------------------
Ensayando en 1:4
---------- SOLUCION ----------
21   6  17  12  19
16  11  20   7   2
 5  22   3  18  13
10  15  24   1   8
23   4   9  14  25
NumSoluciones = 28 Tiempo ensayo =>   0: 34 seg:miliseg
Ensayando en 4:1
---------- SOLUCION ----------
25  16   5  10  19
 6  11  18  15   4
17  24   3  20   9
12   7  22   1  14
23   2  13   8  21
NumSoluciones = 28 Tiempo ensayo =>   0: 25 seg:miliseg
Ensayando en 2:1
NumSoluciones = 0 Tiempo ensayo =>   0: 10 seg:miliseg
Ensayando en 1:2
NumSoluciones = 0 Tiempo ensayo =>   0:  8 seg:miliseg
NumSoluciones = 56 Tiempo total => 0:79 seg:miliseg
```

3 Tabla de tiempos

Los datos que figuran en la tabla siguiente se han obtenido ejecutando el programa "caballoSec" para un tablero de 6x6 con caballo inicial en [3, 3] y en un equipo del lab4401 con Intel i7-10700 a 2,9 GHz con 16MB de caché L3 compartida por los ocho cores [con el HyperThreading deshabilitado]y 8GB de memoria. Se repiten las pruebas en un equipo del lab4406 con Intel i3-8100 a 3,66 GHz con 6MB de caché L3 compartida por los cuatro cores y 8GB de memoria.

Ensayando en	Número de soluciones	Segundos en lab4401	Segundos en lab4406
1:4	6.411	18,162	23,904
2:5	4.716	17,139	22,573
4:5	13.574	20,018	26,335
5:4	13.574	20,049	26,367
5:2	4.716	17,141	22,555
4:1	6.411	18,198	23,933
2:1	1.630	11,638	15,300
1:2	1.630	11,667	15,342
Totales	52.662	134,014	176,313

4 Para saber más

http://es.wikipedia.org/wiki: Aquí puede buscarse la entrada “Problema del caballo” y veremos una descripción del mismo y enlaces a otros sitios de interés.

http://ajedrezpaisa2014.blogspot.com.es/p/ajedrez-y-matem.html: De aquí hemos cogido algunas imágenes de tableros.

5 Listado de programas

El conjunto de ficheros de apoyo que se suministran son los siguientes:

- Makefile: Para obtener los ejecutables de forma cómoda.
- caballoSec.c: Programa secuencial que busca todas las soluciones.

5.1 caballoSec.c

```
//--------------------------------------------------------------------
// PCM Arquitecturas Avanzadas  Curso 16/17                 08/02/16 |
//                                                                   |
//      Problema del caballo en version secuencial                   |
//--------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

#define TRUE 1
#define FALSE 0

#define N 5

// variables globales
static int MovX[8], MovY[8];
static int tablero[N][N];
static int numSoluciones, totSoluciones;

//------------------------------------------------------------------------
void dibujarTablero(void) {
int i, j;
  printf ("---------- SOLUCION ----------\n");
  for (i = 0;i < N; i++) {
    for (j = 0; j < N; j++)
      printf ("%2d  ", tablero[i][j]);
    printf("\n");
  }
}

//------------------------------------------------------------------------
void formarMov(void){
  MovX[0] = -2; MovY[0] =  1;
  MovX[1] = -1; MovY[1] =  2;
  MovX[2] =  1; MovY[2] =  2;
  MovX[3] =  2; MovY[3] =  1;
  MovX[4] =  2; MovY[4] = -1;
  MovX[5] =  1; MovY[5] = -2;
  MovX[6] = -1; MovY[6] = -2;
  MovX[7] = -2; MovY[7] = -1;
}
```

```
//--------------------------------------------------------------------------
int siguienteCasilla(int x, int y, int movimiento) {
int f, c;

    f = x + MovX[movimiento];
    c = y + MovY[movimiento];
    if (   ((f >= 0) && (f < N))
        && ((c >= 0) && (c < N))
        && (tablero[f][c] == 0) )
    return f*N+c; // Se devuelve como indice de array lineal
    return -1;
}

//--------------------------------------------------------------------------
void saltoCaballoR (int x, int y, int salto) {
int i,f,c,fc;

  for (i=0; i<8; i++) {
    fc = siguienteCasilla(x, y, i);
    if (fc >= 0) {
      f = fc / N; c = fc % N;
      tablero[f][c] = salto; // anotar
      if (salto == N*N) {    // si ya ha recorrido todo el tablero
        numSoluciones++;
        if (numSoluciones == 1) dibujarTablero();
      }
      saltoCaballoR (f,c,salto+1);
      tablero[f][c] = 0;     // se desanota el ultimo movimiento
    }
  }
}

//--------------------------------------------------------------------------
void saltoCaballo (int x, int y) {
int i,f,c,fc;
struct timeval t0, tf, t;

  totSoluciones = 0;
  for (i=0; i<8; i++) {
    numSoluciones = 0;
    fc = siguienteCasilla(x, y, i);
    if (fc >= 0) {
      f = fc / N; c = fc % N;
      tablero[f][c] = 2; // anotar
      printf ("Ensayando en %d:%d\n", f, c);
      gettimeofday (&t0, NULL);
      saltoCaballoR (f,c,3);
      gettimeofday (&tf, NULL);
      timersub (&tf, &t0, &t);
     printf ("NumSoluciones = %d Tiempo ensayo => %3ld:%3ld seg:miliseg\n",
                numSoluciones, t.tv_sec, t.tv_usec/1000);
      tablero[f][c] = 0; // se desanota el ultimo movimiento
    }
    totSoluciones += numSoluciones;
  }
}
```

```
//--------------------------------------------------------------------------
int main(int argc, char *argv[]) {
int i,j, filaInicial, colInicial;
struct timeval t0, tf, t;

  if (argc != 3) {
    printf ("Uso: caballoSec filaInicial columnaInicial\n");
    return 0;
  }
  filaInicial = atoi (argv[1]);
  colInicial  = atoi (argv[2]);
  if ((filaInicial < 0) || (filaInicial >= N)) {
    printf ("Fila debe estar en el rango 0..%d\n", N);
    return 0;
  }
  if ((colInicial < 0) || (colInicial >= N)) {
    printf ("Columna debe estar en el rango 0..%d\n", N);
    return 0;
  }

  formarMov();
  for (i = 0;i < N; i++)
    for (j = 0; j < N; j++)
      tablero[i][j] = 0;
  tablero[filaInicial][colInicial] = 1;

  gettimeofday (&t0, NULL);
  printf ("%d:%d ------------------------------------\n",
                                    filaInicial,colInicial);
  saltoCaballo(filaInicial,colInicial);
  gettimeofday (&tf, NULL);
  timersub (&tf, &t0, &t);
  printf ("NumSoluciones = %d Tiempo total => %ld:%ld seg:miliseg\n",
                             totSoluciones, t.tv_sec, t.tv_usec/1000);
  return 0;
}
```