INSTITUTO POLITECNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENIERÍA CAMPUS ZACATECAS

Ing. Sistemas Computacionales

Análisis de Algoritmos

Jesús Abelardo Dávila Mauricio

Presenta:

SOLUCIÓN AL PROBLEMA DEL CABALLO EN UN TABLERO DE AJEDREZ

(Reporte)

A 28 de Octubre del 2019

Introducción

En el transcurso del curso de Análisis de Algoritmos, hemos revisado y analizado algoritmos dinámicos, iterativos, entre otros, los cuales presentaban una solución a problemas particulares y casi aplicables a la vida cotidiana, esta vez se nos impuso como practica la búsqueda de un algoritmo que dé solución al *problema del caballo* en un tablero de ajedrez.

Descripción

El problema del caballo básicamente consiste en, buscar jugadas en las cuales recorramos todo el tablero de ajedrez sin repetir posiciones utilizando sus movimientos: dos casillas en horizontal y una vertical (o inversa).

La solución a este problema se presenta en el siguiente algoritmo implementado en JAVA

```
import java.util.Random;


public class Caballo {
   private int n1 = 2, n2= 8, tam = 10, cont = 1;
   int fallos=0;
   public Caballo(){
   }
   public Caballo(int n1,int n2, int tam){
      this.n1=n1;
      this.n2=n2;
      this.tam=tam;
   }
   public void Solucionar(){
      Random aleatorio = new Random();
      int Tablero[][] = new int[tam + 1][tam +1];
      Tablero[1][1] = 1;
      int pos1, pos2, cas=tam;
      while ( cont < cas ){
```

```
        fallos++;
        cont = 1;
        int ciclo = 0;
        for ( int s = 0; s <= tam; s++ ){
            for ( int t = 0; t <= tam; t++ )
                Tablero[s][t] = 0;
        }
    Tablero[1][1] = 1;
    while ( 1000 != ciclo){
        ciclo++;
        pos1 = 1 + aleatorio.nextInt(tam);
        pos2 = 1 + aleatorio.nextInt(tam);
        if ( Math.abs(Math.abs(n1) - Math.abs(pos1)) == 2){
            if ( Math.abs(Math.abs(n2) - Math.abs(pos2)) == 1  )
                if ( 0 == Tablero[pos1][pos2]){
                    Tablero[pos1][pos2] = ++cont;
                    n1 = pos1;
                    n2 = pos2;
                    ciclo = 0;
                }
        }
        if ( Math.abs(Math.abs(n1) - Math.abs(pos1)) == 1){
            if ( Math.abs(Math.abs(n2) - Math.abs(pos2)) == 2  )
                if ( 0 == Tablero[pos1][pos2] ){
                    Tablero[pos1][pos2] = ++cont;
                    n1 = pos1;
                    n2 = pos2;
                    ciclo = 0;
                }
        }
    }
}
```

```
        System.out.printf("Terminado: Se recorrieron %d casillas.\n", cont);
        System.out.printf("\nSe intentaron %d caminos antes de obtener el requerido.\n", fallos);
        Mostrar(Tablero);
    }
    public void Mostrar(int B[][]){
        for ( int k = 1; k <= tam; k++ ){
            for ( int j = 1; j <= tam; j++){
                System.out.printf("%5d", B[k][j]);
            }
            System.out.println("\n");
        }
    }
}
```

El algoritmo mostrado trabaja con dos variables que posicionan al caballo en el tablero; estas variables son **n1** y **n2**, además de una variable **tam** que indica el tamaño del tablero (este algoritmo puede funcionar par un tablero de n*n) y por ultimo una variable **cont** que enumera las posiciones del caballo en el tablero.

Dentro del algoritmo utilizamos la función **Random**, la cual nos ayuda a que cada vez que probemos el algoritmo, posicione al caballo en diferente lugar, para así poder observar cuan tan diferente es la solución proporcionada por el algoritmo.

Demostración

Para un tablero de ajedrez común

```
Se intentaron 1 caminos antes de obtener el requerido.
 1   0  41   4  15  18   7   0
40   0   0  19   8   5  14  17
 0  42   9   0   3  16   0   6
28  39  20  43  32  11   2  13
21  24  27  10   0  44  33   0
38  29  22  31  26   0  12  47
23  50  25  36  45  48   0  34
 0  37  30  49   0  35  46   0

BUILD SUCCESSFUL (total time: 0 seconds)
```

Para un tablero de 10x10 tenemos que el número de movimientos para una posible solución es 39.

```
Terminado: Se recorrieron 39 casillas.

Se intentaron 1 caminos antes de obtener el requerido.
    1     0    17     0     5    24     0    26    39     0

   16     0     6     0    18    27     4    23     0     0

    0     8     0     0    13     2    25    28     0    38

    0    15     0     7     0    19     0     3    22    29

    9     0     0    14     0    12    21    30    37     0

    0     0    10     0    20     0    36     0     0     0

    0     0     0     0    11     0    31     0     0     0

    0     0     0     0    32     0     0    35     0     0

    0     0     0     0     0    34     0     0     0     0

    0     0     0    33     0     0     0     0     0     0

BUILD SUCCESSFUL (total time: 0 seconds)
```