****

**INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO**

**Estructuras de Datos**

Enero - Mayo 2017

**Proyecto 2: Ajedrez.**

**PROFESOR**: Marcelo Mejía

INTEGRANTES

**César García**

**Francisco Altamirano**

**Fernando Merino**

**Índice:**

Descripción del problema . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

Solución diseñada . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

Limitaciones de la solución . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

Posibles mejores y conclusiones . . . . . . . . . . . . . . . . . . . . . . . . . 7

Apéndice . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

**Descripción del problema:**

Objetivo:

Realizar un programa "Caballo" que permitirá al usuario ingresar la posición de un caballo en un tablero de ajedrez de 8x8 con coordenadas x, y; a continuación, utilizará la coordenada que le fue proporcionada para ver cómo mover al caballo sobre el tablero de forma tal que le sea posible tocar las 64 casillas disponibles sin caer 2 veces en una sola. La segunda versión debe de utilizar los conceptos de recursividad vistos en clase para lograr una mayor eficiencia en el código. El resultado se representa imprimiendo la matriz en la que se ve los elementos que incrementan del cero (posición inicial) a la posición final.

Requisitos para la realización del proyecto:

El programa debe:

* Colocar al caballo en la parte deseada del tablero
* Imprimir el resultado de una manera fácilmente comprensible en el cual las casillas estén representadas por números consecutivos según el orden en el que haya pasado el caballo.
* Obtener la trayectoria de un caballo en una matriz de 8 por 8

Algunas de las restricciones del programa son:

* Tiene que ser realizado en la plataforma NetBeans
* Solamente es funcional en los tableros con dimensiones de 8x8
* Solamente es viable la pieza del caballo de ajedrez, ningún otra.
* Se debe utilizar recursividad en el segundo código

**Solución diseñada:**

UML de clases:

|  |
| --- |
| Caballo |
| - x : int  - y : int  - tamaño : int  - contador : int  - ciclos : int  - intentos\_fallidos : int |
| * Recibe() : Void * Imprime(int arreglo[][]) : Void |

|  |
| --- |
| CaballoRecursivo |
| - solucion : int [][]  - camino : int |
| + CaballoRecursivo(int N)  + resolver() : void  + encuentraCamino(int fila, int columna, int cont, int N) : boolean  + puedeMoverse(int fila, int col, int N) : boolean  + imprime() : void  + main(String[]): void |

Algoritmos principales:

Recibe:

1. Se define el arreglo de 9\*9 para evitar el 0
2. Mientras el contador sea menor que las casillas requeridas.
   1. Se define el contador en 1, los ciclos en 0 y se aumenta el número de intentos fallidos en 1.
   2. Se utilizan 2 ciclos for para crear todas las casillas y darles un valor de 0
   3. Se declara la posición inicial del tablero.
   4. Mientras no se alcancen los 1000 ciclos
      1. Se aumenta el número de ciclos en 1 y se definen 2 números aleatorios para dado1 y dado2.
      2. Si el absoluto de x menos el absoluto de dado1 es igual a 2
         1. Si el absoluto de y menos el absoluto de dado2 es igual a 1
            1. Si el arreglo conformado por dado1 y dado2 es igual a cero (Es decir que no ha sido utilizado)

Se aumenta el contador y la posición de partida para el próximo movimiento se vuelve este.

* + 1. Si el absoluto de x menos el absoluto de dado1 es igual a 1
       1. Si el absoluto de y menos el absoluto de dado2 es igual a 2
          1. Si el arreglo conformado por dado1 y dado2 es igual a cero (Es decir que no ha sido utilizado)

Se aumenta el contador y la posición de partida para el próximo movimiento se vuelve este.

3. Se imprime un mensaje de éxito, el número de casillas recorridas y el número de a intentos fallidos.

EncuentraCamino:

➢ Si Aij (el elemento en la fila i y columna j) es diferente de cero, regresar falso

➢ Si no:

○ Aij es igual al valor del camino, y el camino se incrementa

○ Si el contador es igual a el tamaño de la matriz al cuadrado menos uno, regresar verdadero

○ Si el caballo puede moverse abajo y derecha; y al hacer el método recursivo es verdadero, regresa verdadero

○ Si el caballo puede moverse derecha y abajo; y al hacer el método recursivo es verdadero, regresa verdadero

○ Si el caballo puede moverse derecha y arriba; y al hacer el método recursivo es verdadero, regresa verdadero

○ Si el caballo puede moverse arriba y derecha; y al hacer el método recursivo es verdadero, regresa verdadero

○ Si el caballo puede moverse arriba e izquierda; y al hacer el método recursivo es verdadero, regresa verdadero

○ Si el caballo puede moverse izquierda y arriba; y al hacer el método recursivo es verdadero, regresa verdadero

○ Si el caballo puede moverse izquierda y abajo; y al hacer el método recursivo es verdadero, regresa verdadero

○ Si el caballo puede moverse abajo e izquierda; y al hacer el método recursivo es verdadero, regresa verdadero

○ Si no se cumple ninguna de las condiciones anteriores, hacemos el valor de Aij igual a cero, decrementamos el camino una unidad y regresamos falso

Imprime:

1. Utiliza 2 ciclos for para poder imprimir primeramente todas las columnas y después todas las filas dejando espacios entre estas.

**Limitaciones de la solución:**

Casos no contemplados:

* El que los datos correspondientes a la posición inicial del caballo esten fuera de los límites especificados del tablero-
* La implementación de funcionalidad para ingreso de datos con teclado

Restricciones para utilizar el programa común:

* Revisar que no se escriba un dato de posición inicial fuera de los límites del tablero.
* Tener un espacio de tiempo que puede ir desde 30 segundos hasta 7 minutos en el peor de los casos.
* Tener NetBeans

Restricciones para utilizar el programa recursivo:

* El número de fila tiene que estar contenido en el intervalo de [0,7]
* El número de columna tiene que estar contenido en el intervalo de [0,7]
* Las casillas (0,1), (0,4), (1,0), (1,3), (1,7), (2,2), (2,6), (2,7), (3,0), (3,3), (3,7), (5,1), (5,5), (6,0), (6,4), (6,6) llevan a un ciclo de mayor duración al momento de correrse.

**Posibles mejoras y conclusiones:**

El proyecto “Caballo” permite al usuario encontrar una fácil solución al problema del recorrido del caballero sin exigirle más que introducir la posición inicial del caballo; esto es posible gracias al uso de arreglos de 2 dimensiones y estructuras de control for y while.

“Caballo” se podría mejorar añadiendo los métodos correspondientes para ajustar el código a diferentes tamaños de tablero, agregar otras piezas de ajedrez y ver si un recorrido es imposible sin tener que recorrerlo hasta el final, evitando de esta manera una solución ineficiente y que consuma más tiempo del necesario.

**Apéndice:**

**Común**

package caballo;

import java.util.Scanner;

import java.util.Random;

public class Caballo { // Abre clase Caballo

private int x = 4;

private int y = 5;

private int tamano = 8;

private int contador = 1;

int ciclos = 0;

int intentos\_fallidos = 0;

Scanner entrada = new Scanner(System.in);

public void Recibe() { // Abre Recibe

Random aleatorio = new Random();

int Arreglo[][] = new int[tamano + 1][tamano + 1];

// Se define el arreglo de 9\*9 para evitar el 0

Arreglo[4][5] = 1;

int dado1;

int dado2;

int casillas\_requeridas = 64;

while (contador < casillas\_requeridas) {

intentos\_fallidos++;

contador = 1;

int ciclos = 0;

for (int s = 0; s <= tamano; s++) { // Abre for

for (int t = 0; t <= tamano; t++) {

Arreglo[s][t] = 0;

}

} // cierra for

x = 4;

y = 5;

Arreglo[4][5] = 1;

while (1000 != ciclos)

{ // Abre while

ciclos++;

dado1 = 1 + aleatorio.nextInt(8);

dado2 = 1 + aleatorio.nextInt(8);

if (Math.abs(Math.abs(x) - Math.abs(dado1)) == 2) { // Abre if

if (Math.abs(Math.abs(y) - Math.abs(dado2)) == 1) {

if (0 == Arreglo[dado1][dado2]) { // Abre if

Arreglo[dado1][dado2] = ++contador;

x = dado1;

y = dado2;

ciclos = 0;

} // Cierra if

}

} //Cierra if

if (Math.abs(Math.abs(x) - Math.abs(dado1)) == 1) { // abre if

if (Math.abs(Math.abs(y) - Math.abs(dado2)) == 2) {

if (0 == Arreglo[dado1][dado2]) { // Abre if

Arreglo[dado1][dado2] = ++contador;

x = dado1;

y = dado2;

ciclos = 0;

} // Cierra if

}

} // Cierra if

} // Cierra while anidado

} // Cierra while

System.out.println("\nLISTO!");

System.out.printf("\nSe recorrieron %d casillas.\n", contador);

System.out.printf("\nSe intentaron %d circuitos antes de obtener el requerido.\n", intentos\_fallidos);

Imprime(Arreglo);

} // Cierra Recibe

/\*El metodo siguiente despliega el tablero de ajedrez \*/

//////////////////////////////////////////

// Imprime

///////////////////////////////////////////

public void Imprime(int B[][]) { // Abre imprime

for (int k = 1; k <= 8; k++) {

for (int j = 1; j <= 8; j++) {

System.out.printf("%5d", B[k][j]);

}

System.out.println("\n");

}

} // Cierra imprime

public static void main(String[] args) {

Caballo miObjeto = new Caballo();

miObjeto.Recibe();

}

} // Cierra clase Caballo

**Recursivo**

package caballorecursivo;

import java.text.DecimalFormat;

public class CaballoRecursivo {

private int[][] solucion;

private int camino = 0;

/\*\*

\* CaballoRecursivo: Método constructor del arreglo o matriz

\* @param N: es el tamaño que se quiere inicializar el arreglo, NxN

\*/

public CaballoRecursivo(int N) {

solucion = new int[N][N];

}

/\*\*

\* Resolver: método que tiene como condición el cumplimiento de otro

\* método, el cual utiliza la recursividad

\*/

public void resolver() {

if (encuentraCamino(0, 0, 0, solucion.length)) {

imprime();

} else {

System.out.println("NO SE ENCONTRO UN CAMINO");

}

}

/\*\*

\* EncuentraCamino: método que busca todas las posibilidades de movimiento

\* de una pieza de ajedrez que se comporta como el caballo en un arreglo

\* o matriz de NxN

\* @param fila: valor numérico que apunta hacia una cierta fila

\* @param columna: valor numérico que apunta hacia una cierta columna

\* @param cont: el número de veces que se ha realizado el método

\* @param N: la longitud del arreglo en base a sus filas

\* @return booleano que indica si el procedimiento se pudo realizar con

\* éxito o no

\*/

public boolean encuentraCamino(int fila, int columna, int cont, int N) {

// checar si la casilla actual no contiene elemento

if (solucion[fila][columna] != 0) {

return false;

}

// marcar la casilla actual como usada

solucion[fila][columna] = camino++;

if (cont == N \* N - 1) {

// si esto se cumple, se ha resuelto el problema

return true;

}

// ir abajo derecha

if (puedeMoverse(fila + 2, columna + 1, N)

&& encuentraCamino(fila + 2, columna + 1, cont + 1, N)) {

return true;

}

// ir derecha abajo

if (puedeMoverse(fila + 1, columna + 2, N)

&& encuentraCamino(fila + 1, columna + 2, cont + 1, N)) {

return true;

}

// ir derecha arriba

if (puedeMoverse(fila - 1, columna + 2, N)

&& encuentraCamino(fila - 1, columna + 2, cont + 1, N)) {

return true;

}

// ir arriba derecha

if (puedeMoverse(fila - 2, columna + 1, N)

&& encuentraCamino(fila - 2, columna + 1, cont + 1, N)) {

return true;

}

// ir arriba izquierda

if (puedeMoverse(fila - 2, columna - 1, N)

&& encuentraCamino(fila - 2, columna - 1, cont + 1, N)) {

return true;

}

// ir izquierda arriba

if (puedeMoverse(fila - 1, columna - 2, N)

&& encuentraCamino(fila - 1, columna - 2, cont + 1, N)) {

return true;

}

// ir izquierda abajo

if (puedeMoverse(fila + 1, columna - 2, N)

&& encuentraCamino(fila + 1, columna - 2, cont + 1, N)) {

return true;

}

// ir abajo izquierda

if (puedeMoverse(fila + 2, columna - 1, N)

&& encuentraCamino(fila + 2, columna - 1, cont + 1, N)) {

return true;

}

// si nada funciona, regresar un paso en el camino

solucion[fila][columna] = 0;

camino--;

return false;

}

/\*\*

\* PuedeMoverse: Método que indica si una pieza puede moverse en una

\* dirección arbitraria partiendo de una fila y columna dada

\* @param fila: valor numérico que indica la fila en que se encuentra

\* @param col: valor numérico que indica la columna en que se encuentra

\* @param N: la longitud de la matriz o arreglo basado en sus columnas

\* @return booleano que indica si se puede realizar el movimiento o no

\*/

public boolean puedeMoverse(int fila, int col, int N) {

return fila >= 0 && col >= 0 && fila < N && col < N;

}

/\*\*

\* Imprime: método que realiza la impresión de los valores en cada casilla

\* de la matriz o arreglo

\*/

public void imprime() {

DecimalFormat digitos = new DecimalFormat("00");

for (int[] fila : solucion) {

for (int columna : fila) {

System.out.print(" " + digitos.format(columna));

}

System.out.println();

}

}

/\*\*

\* Main: método donde se ejecuta el programa inicializando una variable

\* de tipo CaballoRecursivo

\*/

public static void main(String[] args) {

int N = 8;

CaballoRecursivo i = new CaballoRecursivo(N);

i.resolver();

}

}