BACKTRACKING

ETSII - URJC

Sebastián Ramiro Entierros García,
Adrián Martín Martín y
Álvaro Martínez Quiroga

Fundamentos del diseño del software

Índice

Contenido

Introducción y ficheros

En esta memoria se va a desarrollar la cuarta práctica de la asignatura de *Fundamentos del Diseño del Software* del grado de *Ingeniería de Computadores* de la Universidad Rey Juan Carlos. La práctica consiste en el estudio de la técnica de **backtracking** a través del juego del ajedrez y los posibles movimientos de un caballo por el tablero. También responderemos a unas cuestiones comunes para este tipo de algoritmos y finalizaremos esta memoria con las conclusiones personales sobre el proyecto.

El paquete entregado, con nombre "Practica4_FDS", incluye los siguientes ficheros y archivos:

- Ficheros de código Java del programa.
- Memoria PDF.

Algoritmo

Como hemos mencionado, el problema consiste en recorrer todas las casillas de un tablero de ajedrez con la ficha del caballo. Esta ficha se mueve en forma de "L" entre las casillas. Hemos desarrollado un programa que permite hacer pruebas de dos maneras estas son:

- Pruebas de usuario: el usuario introduce su propio tablero con la casilla de salida del caballo.
- Pruebas por defecto: el programa llevará a cabo los algoritmos para una serie de pruebas del problema y los resultados se volcarán en un fichero de texto.

El objetivo, además del propio recorrido, es conseguir 2 tipos de maneras de hacerlo, estas con:

- Recorrido abierto: el caballo iterará tantas veces como pueda hasta haber visitado todas las casillas del tablero.
- Recorrido cerrado: el caballo iterará tantas veces como pueda hasta haber visitado todas las casillas del tablero y además deberá de terminar en la casilla donde comenzó su viaje por el tablero.

Función de pre-inicio

```
	public Datos buscaCamino(int pos1, int pos2, int filas, int columnas, int
tipo) {
		boolean exito = false;
		int[][] tablero = new int[filas + 1][columnas + 1];
		for (int x = 1; x <= filas; x++) {
			for (int y = 1; y <= columnas; y++) {
				tablero[x][y] = -1;
			}
		}

		// Orden de movimientos
		int movimientoFilasV1[] = { -2, -1, 1, 2, -2, -1, 1, 2 };
		int movimientoColumnasV1[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
```

```
            int movimientoFilasV2[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
            int movimientoColumnasV2[] = { 1, 2, 2, 1, -1, -2, -2, -1 };
            if (0 == pos1 || 0 == pos2 || pos1 > tablero.length || pos2 >
tablero[0].length) {
                  System.out.println("Fallo en las posiciones iniciales, se
excede el limite del tablero.");
                  return new Datos(exito, 0);
            }

            tablero[pos1][pos2] = 0; // Casilla ya visitada (la inicial, 0)
            long t_comienzo = System.nanoTime();
            switch (tipo) {
            case 1:
                  exito = buscarCaminoAbierto(pos1, pos2, 1, tablero,
movimientoFilasV2, movimientoColumnasV2);
                  break;
            case 2:
                  exito = buscarCaminoCerrado(pos1, pos2, 0, tablero,
movimientoFilasV1, movimientoColumnasV1);
                  break;
            default:
                  System.out.println("Tipo de recorrido erroneo.");
                  return new Datos(exito, 0);
            }

            long t_fin = System.nanoTime();
            imprimeTablero(tablero, exito, (t_fin - t_comienzo));

            return new Datos(exito, (t_fin - t_comienzo));
      }
```

Antes de comenzar el algoritmo, la ejecución de este dependerá de los parámetros de entrada de la función.

Como podemos observar comenzamos aplicando una transformación de datos a las casillas del tablero pasándolas cada una de ellas a -1. Después, anotamos los posibles movimientos del tablero, hay un par para probar. Antes de comenzar el algoritmo, transformamos la casilla inicial a la casilla inicio igualándola a 0 y finalmente llamamos al método de *bactracking*.

Recorrido abierto

```
      private boolean buscarCaminoAbierto(int pos1, int pos2, int movimiento,
int[][] tablero, int[] movX, int[] movY) {
            if (movimiento == tablero.length * tablero[0].length) {
                  return true;
            }

            int proximaFila, proximaColumna, i;
            int aux = movimiento;
            for (i = 0; i < movX.length; i++) {
                  proximaFila = pos1 + movX[i];
                  proximaColumna = pos2 + movY[i];
                  if (esSegura(proximaFila, proximaColumna, tablero)) {
                        tablero[proximaFila][proximaColumna] = aux + 1;
```

```
                        if (buscarCaminoAbierto(proximaFila, proximaColumna,
movimiento + 1, tablero, movX, movY)) {
                            return true;
                        } else {
                            tablero[proximaFila][proximaColumna] = -1;
                        }
                    }
                }
                return false;
        }
```

La función recibirá 6 parámetros, estos son:

- Casillas de salida para la llamada en un momento x de la ejecución a la función: *pos1* y *pos2*.
- El número del movimiento actual de la llamada: *movimiento*.
- El propio tablero: *tablero*.
- Dos arrays con el orden de movimientos en los ejes x e y del tablero: *movX* y *movY*.

La idea del algoritmo es sencilla, este iterará sobre la casilla actual "CA" tantas veces como movimientos tenga disponibles. Comprobará si cada movimiento es factible para incluirlo en la solución y de ser así, hace una llamada al propio método de nuevo con el valor de la posición actualizado. Finalmente, cuando llega al último movimiento es capaz de identificarlo si este movimiento actual es igual a las dimensiones del tablero sobre el que se está ejecutando, de ser así el algoritmo ha terminado e imprime el resultado.

Tabla

Tamaño	Tiempo (ms)	Tiene solución
3x3	0	No
4x4	0	No
5x5	1	Sí
6x6	11	Sí
3x10	1	Sí
3x12	1	Sí
5x6	493	Sí
3x4	1	Sí
3x5	0	No
3x6	0	No
3x8	2	Sí
4x6	2	Sí
4x3	1	Sí
5x3	0	No
6x3	0	No
8x3	2	Sí
6x4	8	Sí
6x5	11	Sí
10x3	16	Sí

12x3	187	Sí
7x7	87	Sí
8x8	123	Sí
3x14	23	Sí
5x8	13	Sí
6x7	19	Sí
6x8	1990	Sí
4x8	16	Sí
5x7	1	Sí
8x4	168	Sí
7x5	5	Sí
14x3	5021	Sí

Recorrido cerrado

```
	public boolean buscarCaminoCerrado(int pos1, int pos2, int i, int[][]
tablero, int[] movX, int[] movY) {
		if (i == tablero.length * tablero[0].length) {
			return true;
		}

		for (int j = 0; j < movX.length; j++) {
			int proximaFila = pos1 + movX[j];
			int proximaColumna = pos2 + movY[j];
			if (rango(proximaFila, proximaColumna, i + 1, tablero)) {
				tablero[pos1][pos2] = i + 1;
				if (buscarCaminoCerrado(proximaFila, proximaColumna, i
+ 1, tablero, movX, movY)) {
					tablero[pos1][pos2] = i + 1;
					return true;
				}
			}
		}
		tablero[pos1][pos2] = -1;
		return false;
	}
```

La función del backtracking es igual que la del algoritmo anterior para el caso abierto recibiendo los mismos parámetros, a diferencia que, en este algoritmo, asignamos el valor a la casilla correspondiente solo si la llamada recursiva siguiente devuelve “true”, lo que significa que la búsqueda ha sido satisfactoria.

Tabla

Tamaño	Tiempo (ms)	Tiene solución
3x3	0	No
4x4	0	No
5x5	37	No
6x6	3316	Sí

3x10	5	Sí
3x12	53	Sí
5x6	224	Sí
3x4	0	No
3x5	0	No
3x6	0	No
3x8	1	No
4x6	11	No
4x3	0	No
5x3	0	No
6x3	0	No
8x3	1	No
6x4	11	No
6x5	392	Sí
10x3	3	Sí
12x3	56	Sí
7x7	No ha terminado	No
8x8	No ha terminado	No
3x14	1646	Sí
5x8	No ha terminado	No
6x7	No ha terminado	No
6x8	No ha terminado	No
4x8	No ha terminado	No
5x7	No ha terminado	No
8x4	No ha terminado	No
7x5	No ha terminado	No
14x3	1827	Sí

Las tablas anteriores han seguido el siguiente orden de saltos del caballo en las casillas:

	3		**2**	
4				1
		C		
5				8
	6		7	

Preguntas de la práctica

¿Cuántos niveles tendrá el árbol de exploración? ¿y en el caso del camino cerrado?

El árbol de exploración tendrá tantos niveles como casillas hay en el tablero, es decir si es un 5x5 tendrá un total de 25 niveles. En el caso cerrado tendrá la misma cantidad de niveles que su semejante abierto.

¿Cuántos candidatos tendrá el árbol de exploración en cada nivel?

El árbol de candidatos en cada nivel dispondrá de 8 opciones, es decir, 8 candidatos. Esto se debe a que el caballo desde una casilla cualquiera, este puede realizar 8 movimientos diferentes. Sin embargo, los candidatos pueden no ser directamente factibles por que el movimiento a realizar se salga de las dimensiones del tablero. También, el tiempo cambia si se decide comenzar a explorar desde una casilla diferente, en algunos casos, llegando a disminuir el tiempo de exploración si se comienza en una casilla más centrada del tablero.

¿Cómo influyen los tiempos en caso de modificar el orden de salida o de exploración?

Por ejemplo, en el caso abierto del tablero con tamaño igual a 5x6:

- Usando el orden 2 de casillas ha tardado 493 ms
- Usando el orden 1 de casillas ha tardado 24ms.

Con esto, podemos llegar a la conclusión de que el árbol de exploración dependerá del orden de exploración de las casillas del tablero.

Conclusiones de la práctica.

Consideramos fundamental esta práctica para entender la técnica del backtracking. Nos ha sorprendido también que en cada ordenador personal nos ha llegado a dar resultados diferentes para los diferentes casos de búsqueda. Incluso, en ocasiones, el resultado ha mejorado cambiando el orden de exploración de las casillas.

Referencias

- Temario del aula virtual.