PROYECTO 3

Elaborado por: Ukranio Coronilla

En este proyecto realizaremos una aplicación en Java para resolver el juego de piezas deslizables similar al que se muestra en la imagen:

Para poder resolverlo computacionalmente modelaremos sólo la solución para un juego de 3 x 3 en lugar de 4 x 4.

Lea cuidadosamente los temas del libro **Inteligencia Artificial** de *Nils Nilsson* que se adjuntan a continuación, y elabore una aplicación en Java que recibe en la línea de comandos la configuración inicial del puzzle y la configuración final para que después de hacer los cálculos muestre la serie de movimientos necesarios que son una posible solución del problema.

Especificación del espacio de búsqueda

Muchos de los problemas de interés práctico tienen unos espacios de búsqueda tan grandes que no pueden ser representados mediante un grafo explícito, por tanto, se requieren modificaciones en los procedimientos de búsqueda tratados en el capítulo anterior. Primero, debemos prestar una atención especial a la formulación de dichos problemas para poder aplicar sobre ellos los métodos de búsqueda. Segundo, necesitamos métodos que nos permitan representar grandes espacios de búsqueda mediante grafos implícitos. Tercero, necesitamos métodos eficientes de búsqueda en esos grafos de gran tamaño. En algunos problemas de planificación, como, por ejemplo, el del apilamiento de bloques, no es difícil concebir estructuras de datos para representar los diferentes estados del mundo y las acciones que lo modifican. Normalmente, resulta difícil encontrar representaciones en forma de grafos de estados que sean manejables. Para conseguir esto, se requiere un análisis detallado del problema —buscar simetrías, ignorar detalles irrelevantes y encontrar las abstracciones apropiadas—. Desgraciadamente, plantear un problema para poder aplicar sobre él técnicas de búsqueda es una tarea bastante compleja y se puede considerar como un arte que todavía requiere de intervención humana.

Además del problema del apilamiento de bloques, también se utiliza a menudo el problema del desplazamiento de las piezas de un puzzle para mostrar cómo se pueden utilizar las técnicas de búsqueda en espacios de estados para planificar secuencias de acciones. Para ilustrar las técnicas de búsqueda, que examinaremos en este capítulo y en el siguiente, utilizaremos este tipo de problemas. Un ejemplo típico es el puzzle de 15 piezas, que consiste en un conjunto de 15 piezas dispuestas en un tablero de cuatro por cuatro, en el que se deja una casilla en blanco a la que se pueden desplazar las piezas colocadas en casillas adyacentes. El objetivo es encontrar una secuencia de movimientos que transforme una disposición inicial de piezas en una configuración determinada. El puzzle de ocho piezas es una versión reducida del problema, en el que sólo se tienen ocho piezas en un tablero de tres por tres. Supongamos que el objetivo del puzzle es mover las casillas desde una disposición inicial arbitraria hasta la configuración objetivo, tal y como se muestra en la Figura 8.1.

Figura 8.1.

Configuración inicial y objetivo para el problema del puzzle de ocho piezas.

Dado este problema, una descripción obvia que permite representar los distintos estados es una matriz de tres por tres, en la que cada celda contiene un número del 1 al 8 o un símbolo que represente la celda vacía. El estado objetivo es la matriz representada en la parte derecha de la Figura 8.1. Para pasar de un estado a otro sólo tenemos que desplazar una pieza a la celda vacía. Sin embargo, como ya hemos indicado, podemos elegir entre diferentes formas de representar el espacio de estados. En el problema del puzzle de ocho piezas, podríamos haber considerado que tenemos 8×4 movimientos posibles: desplazarse una casilla hacia arriba, una hacia abajo, una hacia la izquierda, una hacia la derecha, dos casillas hacia arriba... (por supuesto, no todos los movimientos son posibles en todos los estados). Una formulación más compacta sólo tiene en cuenta cuatro posibles movimientos: mover la casilla libre hacia la izquierda, hacia arriba, hacia la derecha y hacia abajo. El grafo con los estados accesibles desde el estado inicial queda definido de forma implícita a partir del estado inicial y del conjunto de movimientos posibles. El conjunto de nodos en el grafo de estados para esta representación del puzzle de ocho piezas es $9! = 362.880$ (en este problema, el grafo de estados puede ser dividido en dos grafos independientes, tales que una configuración de piezas perteneciente a uno de los grafos no puede ser alcanzada desde ninguno de los nodos del otro grafo).

Elementos de un grafo de estados implícito

Como hemos visto en el párrafo anterior, describiendo el estado inicial y los efectos que las acciones producen sobre dicho estado, obtenemos una representación implícita de la parte del grafo de estados que se puede alcanzar a partir del estado inicial. Por tanto, y en principio, es posible transformar la representación implícita del grafo en un grafo explícito. Para hacerlo, sólo hay que generar todos los nodos sucesores del nodo inicial (aplicando todos los posibles operadores al nodo inicial) y, después, volver a realizar la misma operación para cada uno de los nodos sucesores. Para aquellos grafos que sean demasiado grandes para ser representados de forma explícita, el proceso de búsqueda sólo requiere tener de forma explícita la parte del espacio de estados necesaria para calcular el camino hacia el objetivo. El proceso termina cuando se ha encontrado un camino aceptable al nodo objetivo.

Formalmente, podemos decir que existen tres elementos básicos en la representación implícita del grafo de estados:

1. Una descripción para etiquetar el *nodo inicial*. Esta descripción estará formada por alguna estructura de datos que nos permita modelar el estado inicial.
2. Las funciones para transformar las descripciones de los estados (que representan estados del mundo) en las descripciones de los estados resultantes al aplicar las funciones.

   Normalmente, a estas funciones se les denomina *operadores*. En nuestros ejemplos sobre agentes, los operadores son modelos de los efectos de las acciones. Cuando aplicamos un operador a un nodo, se genera uno de sus nodos sucesores.
3. Una *condición de éxito*, que puede ser una función booleana, que se aplica a las descripciones de los estados, o una lista con las descripciones de estados que se corresponden con los estados objetivos.

En este capítulo y en el siguiente examinaremos dos tipos de procesos de búsqueda. El primer tipo corresponde a procesos de búsqueda en los que no se dispone de información específica del problema para establecer preferencias dentro del espacio de estados en lo que concierne a la búsqueda del camino al objetivo. Este tipo de procesos se denominan *búsquedas a ciegas*. En el otro tipo, nos encontraremos con procesos de búsqueda que disponen de información específica sobre el problema, con lo que se puede mejorar la eficiencia del proceso de búsqueda. Este tipo de procesos se denominan *búsquedas heurísticas*[1]. En este capítulo analizaremos el primer tipo, mientras que las búsquedas heurísticas las veremos en el capítulo siguiente.

Búsqueda primero en anchura

Los procedimientos de búsqueda a ciegas operan aplicando los operadores disponibles a los nodos, pero no utilizan ningún conocimiento sobre el dominio del problema (sólo saben cuándo es posible aplicar una determinada acción). El más simple de estos procedimientos es la *búsqueda primero en anchura*. Este método va construyendo un grafo de estados explícito mediante la aplicación de los operadores disponibles al nodo inicial, después aplica los operadores disponibles a los nodos sucesores directos del nodo inicial, y así sucesivamente. Como podemos ver, este procedimiento de búsqueda actúa de manera uniforme a partir del nodo inicial. Teniendo en cuenta que en cada paso se aplican a los nodos todos los operadores disponibles, resulta más eficiente agruparlos en una función denominada *función de estados sucesores*. Cuando apliquemos esta función a un determinado nodo, obtendremos todos los nodos que se producirían al aplicar todos los operadores a dicho nodo. Cada aplicación de la función de estados sucesores se denomina *expansión* del nodo.

En la Figura 8.2 se pueden ver los nodos creados por la búsqueda primero en anchura para el problema del puzzle de ocho piezas. También se pueden apreciar los nodos inicial y objetivo así como el orden en el que se han expandido los nodos, indicado por el número que está próximo a cada nodo. Los nodos que están situados a la misma profundidad se van expandiendo según un orden prefijado de antemano; para cada nodo, el orden de aplicación de los operadores es el siguiente: mover la celda vacía a la izquierda, arriba, a la derecha y abajo. Aunque cada movimiento es reversible, en la Figura 8.2 hemos omitido los arcos que van de los sucesores a sus predecesores. El camino que conforma la solución está marcado por una línea gruesa más oscura. Como veremos más adelante, la búsqueda primero en anchura tiene la propiedad de que una vez que se ha encontrado el nodo objetivo, habremos encontrado el camino de menor longitud. Sin embargo, el principal inconveniente de este método es que requiere la generación y almacenamiento de todo el árbol, cuyo tamaño crece de manera exponencial respecto a la profundidad del nodo objetivo menos profundo.

Una variante de este método es la *búsqueda de coste uniforme* [Dijkstra, 1959], en la que se van expandiendo los nodos que tienen igual coste en vez de expandir los nodos que están a la misma profundidad. Si los costes de todos los arcos del grafo son idénticos, la búsqueda de coste uniforme equivale a una búsqueda primero en anchura. Evidentemente, la búsqueda de coste uniforme se puede considerar como un caso especial de búsqueda heurística, que será descrita en el siguiente capítulo. Este pequeño análisis sobre la búsqueda primero en anchura y la búsqueda de coste uniforme nos pueden dar una idea general sobre los métodos de búsqueda, pero habría que profundizar más en los aspectos técnicos para cubrir los casos en los que el proceso de expansión produzca nodos que ya han sido analizados por el proceso de búsqueda. Dejaremos este análisis para el siguiente capítulo, una vez que tratemos procedimientos de búsqueda más generales.

---

[1] La palabra *heurística* proviene del griego *heuriskein*, que significa descubrir. De la misma palabra se deriva la exclamación ¡*Eureka*!

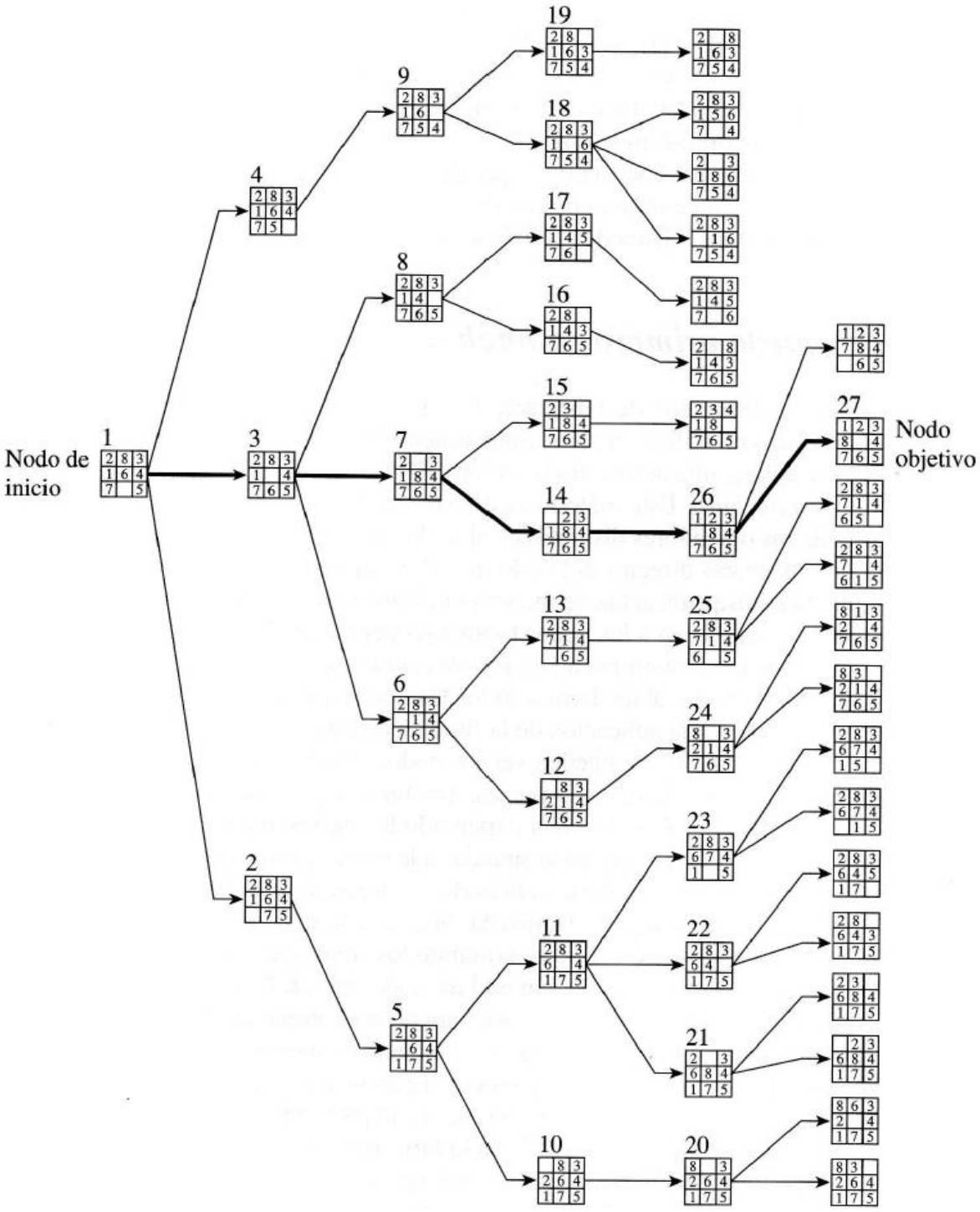

Figura 8.2.

Búsqueda primero en anchura para el puzzle de ocho piezas.

La presentación de la solución debe encontrarse en modo texto y forzosamente debe reutilizar el siguiente código para la creación de los nodos en un árbol n-ario:

Nodo.java

```java
import java.util.ArrayList;
import java.util.List;

class Nodo {
    int valor;
    List<Nodo> hijos;

    public Nodo(int valor) {
        this.valor = valor;
        this.hijos = new ArrayList<>();
    }

    // Método para agregar un hijo al nodo
    public void agregarHijo(Nodo hijo) {
        hijos.add(hijo);
    }
}
```

ArbolNArio.java

```java
class ArbolNArio {
    Nodo raiz;

    // Constructor
    public ArbolNArio(int valor) {
        raiz = new Nodo(valor);
    }

    // Método para realizar un recorrido en profundidad (DFS) del árbol
    public void recorridoDFS(Nodo nodo) {
        if (nodo != null) {
            System.out.print(nodo.valor + " ");
            for (Nodo hijo : nodo.hijos) {
                recorridoDFS(hijo);
            }
        }
    }

    public static void main(String[] args) {
        ArbolNArio arbol = new ArbolNArio(1);

        // Agregamos hijos al nodo raíz
        Nodo nodo2 = new Nodo(2);
        Nodo nodo3 = new Nodo(3);
        Nodo nodo4 = new Nodo(4);
        Nodo nodo5 = new Nodo(5);
        Nodo nodo6 = new Nodo(6);

        arbol.raiz.agregarHijo(nodo2);
        arbol.raiz.agregarHijo(nodo3);
        arbol.raiz.agregarHijo(nodo4);

        nodo2.agregarHijo(nodo5);
        nodo2.agregarHijo(nodo6);

        // Realizamos un recorrido en profundidad del árbol
        System.out.println("Recorrido en profundidad del árbol n-ario:");
        arbol.recorridoDFS(arbol.raiz);
    }
}
```

Importante: Este proyecto es individual. Suba cada clase en un archivo separado, y cada archivo de código que suba debe contener al inicio como comentario el número de proyecto, su nombre completo y el grupo al que pertenece, de no hacerlo así se le descontará un punto de la calificación. No suba archivos comprimidos ni ligas a sitios web externos pues no le será tomado en cuenta el proyecto. Asimismo, deberá subir un video breve mostrando como se ejecuta su proyecto y que efectivamente realiza lo que se pide con dos búsquedas de más de diez movimientos. Se recomienda utilizar OBS Studio con baja resolución y no es necesario que hable en el video.