Inteligencia Artificial

Toral Maldonado Rosa Guadalupe
2153045948

Práctica 3: Algoritmo primero en amplitud

Profesor Orlando Muñoz Texzocotetla

Lunes 5 de junio de 2018

Introducción

Para poder dar solución a un problema con inteligencia artificial se debe pre-definir el espacio de estados. Este puede ser conseguido de varias maneras.

Para esta práctica, el espacio de estados se obtendrá usando el algoritmo de primero en amplitud, el cual consiste en revisar todos los estados que pueden obtenerse a partir de cierto estado inicial y después obtener los estados sucesores a los ya encontrados. Se llama en amplitud, ya que el recorrido del grafo o árbol se hace por niveles.

Se usará para encontrar todas las posibles configuraciones del puzzle que llevan al estado final. Igualmente se determinará si es un árbol o un grafo.

Desarrollo

Para poder implementar el algoritmo de búsqueda ciega primero en amplitud, se usará el juego 8-puzzle.

Se deberá usar 3 colas:

- **cola**: cola principal. Aquí se guardan todas las configuraciones encontradas para analizarlas después.
- **revisados**: cola donde se guardarán todos las configuraciones que hayan sido analizadas.
- **edosNuevos**: cola donde se guardarán todas las configuraciones resultantes dada una configuración actual.
- **hijos**: aquí se guardan las configuraciones resultantes dada una configuración inicial, siempre y cuando no existan en la cola de **revisados**.

Para representar estas colas se hará uso de un objeto de la clase ArrayList; y, dada la naturaleza del ArrayList, los arreglos serán colas con recorrido; es decir, se sacan (o se obtienen, en dado caso de que no se desee eliminarlos de la cola) los elementos de la primer posición y se recorrerá el resto al espacio vacío.

Algoritmo primero en amplitud

A continuación, se mostrarán pasos que se seguirán en la práctica.

1. Se debe guardar en la **cola** el estado “i”.

2. Se revisa. Se generan sus hijos y se guardan en la cola **edosNuevos**.
3. Se revisa si estos hijos no pertenecen a la cola **revisados**. De no existir aquí se agregan a la cola **hijos**.
4. El estado "i" se agrega a la cola de **revisados** y sus hijos de pasan a la **cola**.
5. Se revisa el siguiente estado en la **cola** y se vuelve a repetir el algoritmo.

```
cola.add(estado(i));
siguiente = cola.get(0);
edosNuevos = Tablero.generaEstadosVecinos(siguiente);
insertaEnHijos(edosNuevos, revisados);
revisado = cola.remove(0);
revisados.add(revisado);
while(!hijos.isEmpty())
      cola.add(hijos.remove(0));
siguiente = cola.get(0);
```

Fragmento de código 1. Pasos generales para llegar a la solución de un problema. Esto se repite mientras no se llegue al estado final.

Note que el método **insertaEnHijos** es el encargado de insertar en la cola **hijos** aquellos estados que no hayan sido revisados. Va revisando elemento por elemento de la cola **edosNuevos** y se cerciora de que no hay algún elemento en la cola **revisados** con la misma configuración. De no haberlo, guarda los elementos en la cola **hijos**. El siguiente fragmento de código muestra dicho algoritmo utilizado para comparar las matrices entre estados:

```
if(revisados.isEmpty())
      for(int i = 0; i < edosNuevos.size(); i++)
            hijos.add(edosNuevos.get(i));
else{
      int bandera = 0;
      for(int k = 0; k < edosNuevos.size(); k++){//Recorre el
      arreglo edosNuevos
            for(int m = 0; m < revisados.size(); m++){//Recorre el
            arreglo revisados
                  bandera = 0;
                  if(Arrays.deepEquals(edosNuevos.get(k).getConfigura
                  cion(), revisados.get(m).getConfiguracion())){
                        bandera = 1;
                        break;
                  }
            }
            if(bandera == 0)
                  hijos.add(edosNuevos.get(k));
      }
}
```

Fragmento de código 2. Algoritmo utilizado para agregar los estados a la cola ***hijos.***

A continuación se muestran una ejecución del programa.

Dado el estado inicial:

	1	2
4	5	3
7	8	6

Se quiere llegar al estado final:

1	2	3
4	5	6
7	8	

El árbol resultante tendría que ser el siguiente:

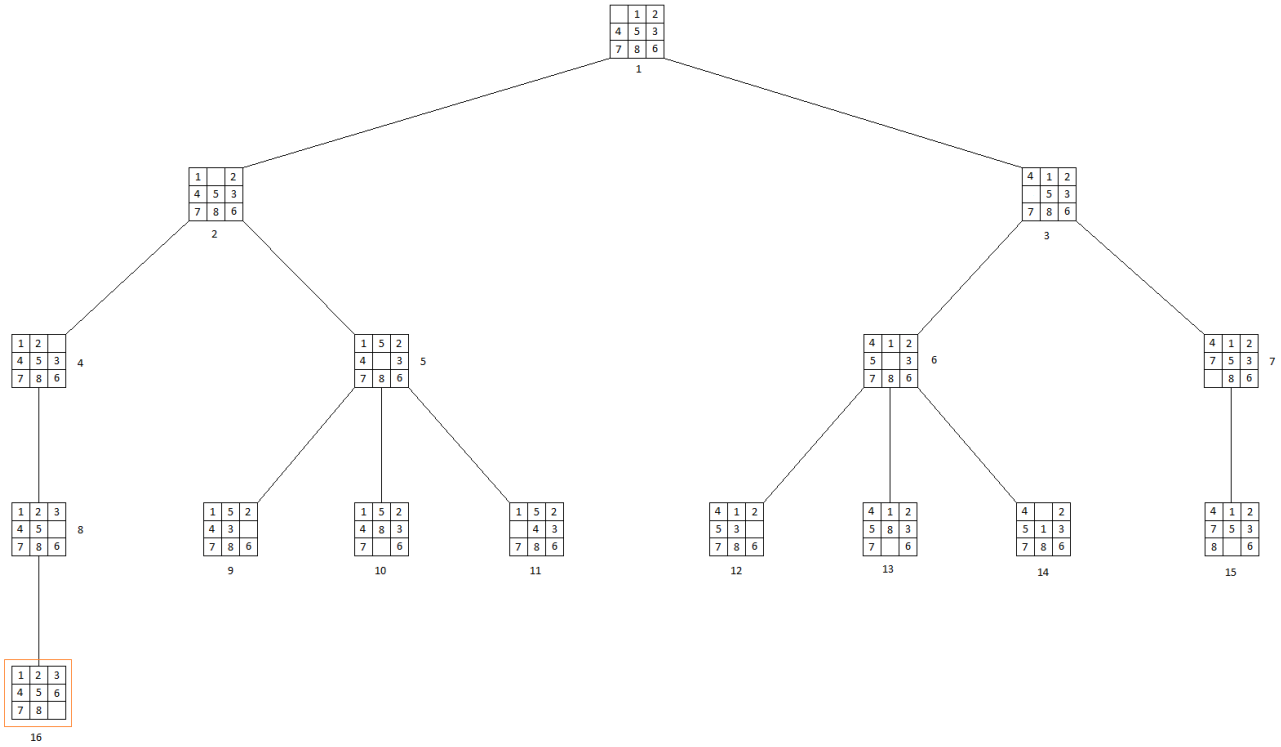

Figura 1. Árbol que resulta de aplicar el algoritmo primero en amplitud a mano.

Se ejecuta el programa para ver que coincida. El orden en cómo se va a recorrer el árbol es:

1, 2, 3, 4, 5, 6, 7, …, 15, 16

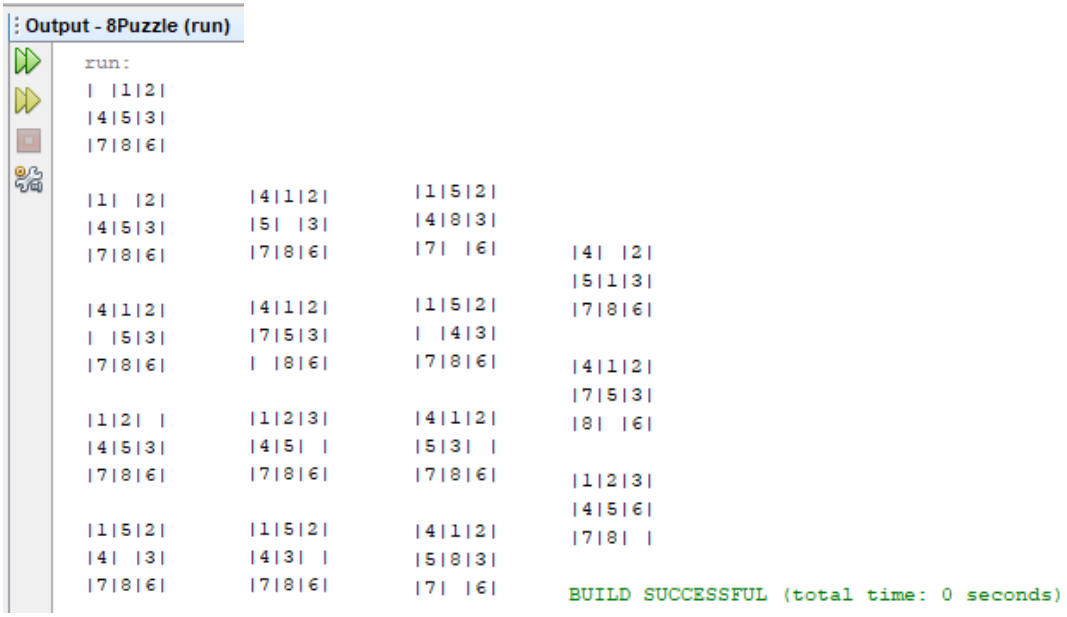

Figura 2. Se muestran los estados posibles que se pueden generar a partir de una configuración inicial del puzzle. Note que la impresión de las configuraciones coincide con las del árbol hecho a mano.

Observaciones finales

Representar el algoritmo primero en amplitud no fue muy difícil, lo difícil fue encontrar la manera de comparar las configuraciones de un objeto del tipo estado con la configuración de otro objeto del mismo tipo.Como se puede notar, el algoritmo para compararlos es un poco extenso debido al uso de banderas y al anidamiento de ciclos for.

En algunas configuraciones para el mismo estado final, el programa seguía generando estados nuevos; sin embargo esto no fue por un error en el código, más bien tiene que ver con la naturaleza del puzzle: cada casilla tiene 8 números más un espacio como opciones para contener, por lo que es más difícil que salgan dos estados parecidos, por lo que sigue generando hijos.