|  |
| --- |
| Inteligencia Artificial |
| Práctica 2. Búsquedas |
| Problema del puzle 8 |

|  |
| --- |
| Juan Alberto Rodríguez López  2015/2016 |

1. Desarrollo:

Para resolver el problema del Puzzle 8 contamos con una serie de estrategias de búsqueda informada y no informada, estas son:

- Búsqueda en anchura.

- Búsqueda en profundidad.

- Búsqueda greedy best first.

- Búsqueda A\*.

El código ha sido estructurado empleando las clases Nodo y Estado, además de la clase main Practica2. La clase Nodo está compuesta por:

-public Nodo() 🡪 constructor para un puzle con números aleatorios.

- public Nodo(int [][] estado) 🡪 constructor para un puzle con números adquiridos por parámetros.

- public int[] Localizacero() 🡪 función para controlar en todo momento la posición del 0, el cual representa el espacio vacío del puzle.

- public int Calcularcoste() 🡪 función para calcular el coste de los nodos, utilizando determinados algoritmos.

-gets y sets 🡪 funciones que dan valor y devuelven datos almacenados como los padres de los nodos, la profundidad, el coste, etc.

El código desarrollado en la clase Estado se describe a continuación:

-public Estado() 🡪 construye un estado inicial aleatorio.

- public Estado(int [][] Inicio, int[][] Solucion) 🡪 construye un estado incial por parámetro.

-public void setSolucion(Nodo Solucion) 🡪 se guarda el estado final.

-public void MostrarEstado(int [][] estado) 🡪muestra por pantalla el estado actual.

-public ArrayList<Nodo> GenerarSucesores(Nodo Actual, int modo) 🡪 función para generar los hijos.

-public Nodo Mover\*\*\*\*\*(int[] PCero, Nodo Actual, int modo) 🡪 indica todos los movimientos posibles (arriba, abajo, derecha, izquierda).

- public void CalculoHeuristica(Nodo Actual) 🡪 calcula lo cerca que esta el nodo actual de la solución.

-public void MostrarSolucion(Nodo Actual, int CamSolucion, int Visitados, int MaxC, int MaxA, long tiempo, int Modo) 🡪 se escoge el modo en que se va a visualizar la solución.

- public void Modo\*(Nodo Actual, int CamSolucion, int Visitados, int MaxC, int MaxA,long tiempo, int Modo) 🡪 muestra la solución con el modo de visualización seleccionado (0, 1 o 2).

- public void A\_Estrella(int modo) 🡪 implementación del algoritmo A\*.

- public ArrayList<Nodo> GestionRepetidosA\_Estrella(ArrayList<Nodo> hijos, ArrayList<Nodo> Cerrados, Queue<Nodo> Abiertos, int modo) 🡪 se desarrolla la forma en la que el algoritmo A\* trata los nodos repetidos

- public void BusquedaAnchura(int limite, int modo) 🡪 implementación del algoritmo de busqueda en anchura.

-public ArrayList<Nodo> GestionRepetidosAnchura(ArrayList<Nodo> hijos, ArrayList<Nodo> Cerrados, Queue<Nodo> Abiertos, int modo) 🡪 se desarrolla la forma en la que el algoritmo de búsqueda en anchura trata los nodos repetidos.

-public void Greedy(int modo) 🡪 implementación del algoritmo greedy best first.

-public ArrayList<Nodo> GestionRepetidosGreedy(ArrayList<Nodo> hijos, ArrayList<Nodo> Cerrados, Queue<Nodo> Abiertos, int modo) 🡪 se desarrolla la forma en la que el algoritmo greedy best first trata los nodos repetidos.

- public void BusquedaProfundidad(int limite, int modo) 🡪 implementación del algoritmo de búsqueda en profundidad.

- public ArrayList<Nodo> GestionRepetidosProfundidad(ArrayList<Nodo> hijos, ArrayList<Nodo> Cerrados, Stack<Nodo> Abiertos,int modo) 🡪 se desarrolla la forma en la que el algoritmo de búsqueda en profundidad trata los nodos repetidos.

En la clase Practica2 se encuentra un pequeño menú con el que manipular la practica en ejecución.

1. Experimentación:

Para comprobar las características de cada estrategia se generan 5 estados iniciales, a continuación se mostrará cada uno de ellos junto a los resultados de cada algoritmo con respecto al espacio utilizado, la longitud del camino y el tiempo. Aparecerá resaltado el mejor algoritmo en cada prueba. Estos son los estados iniciales utilizados:

Estado inicial 1: {{1,2,3},{6,4,0},{7,8,5}}

Estado inicial 2: {{1,2,3},{6,4,5},{7,8,0}}

Estado inicial 3: {{1,4,2},{6,0,3},{7,8,5}}

Estado inicial 4: {{4,1,3},{6,2,5},{7,8,0}}

Estado inicial 5: {{0,1,2},{3,4,5},{6,7,8}}

-Tabla de tiempos (en milisegundos):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Alg\Estado | 1 | 2 | 3 | 4 | 5 |
| A\* | 83 | 42 | 746 | 177 | 9787 |
| Anchura | 164 | 92 | 948 | 237 | 15631 |
| Greedy | 79 | 38 | 4456 | 599 | - |
| Profundidad | 40 | 40 | - | 3867 | 19004 |

\*En caso de dato no registrado, se representa como 21000

-Espacio utilizado (número máximo de estados guardados abiertos + cerrados):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Alg\Estado | 1 | 2 | 3 | 4 | 5 |
| A\* | 67 | 37 | 343 | 113 | 5611 |
| Anchura | 78 | 48 | 371 | 143 | 7206 |
| Greedy | 48 | 105 | 11441 | 841 | - |
| Profundidad | 48 | 8 |  | 3317 | 12362 |

\*En caso de dato no registrado o muy elevado, se representa como 12000.

-Longitud del camino (pasos):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Alg\Estado | 1 | 2 | 3 | 4 | 5 |
| A\* | 5 | 4 | 8 | 6 | 14 |
| Anchura | 5 | 4 | 8 | 6 | 14 |
| Greedy | 25 | 4 | 8 | 6 | - |
| Profundidad | 25 | 4 | - | 24 | 24 |

\*En caso de dato no registrado, se indica como 30.

1. Análisis:

En teoría, es el algoritmo A\* el que obtiene la solución óptima con determinadas condiciones. Junto al de búsqueda en anchura son los más eficaces. Búsqueda en profundidad no siempre encuentra la solución y greedy best first es rápido, pero no óptimo. En las pruebas realizadas se demuestra todo esto: A\* y anchura son fiables y garantizan un mejor resultado que greedy y profundidad, que en algunos casos son más rapidos, pero en otros no llegan a la solución.