Inteligencia Artificial

Algoritmos de búsqueda

Búsqueda A*

Diego Cruz Rodríguez
alu0101105802@ull.edu.es
Universidad de La Laguna
Ingeniería Informática
Curso 3º
1º Semestre

Índice

Estructura de datos empleada

Para implementar la clase búsqueda he empleado:
Una estructura de datos en la que guardaré el resultado por ello su nombre “Resultado” está compuesta por: “costeminimo” que guardara el coste del camino mínimo de la búsqueda realizada, “caminominimo” que guardara el camino mínimo encontrado, guardado inversamente de forma que el primer elemento es el destino del camino y el último es el nodo de origen, “generados” guardará los nodos generados, “analizados” que guardara los nodos analizados en la búsqueda y “profundidad” que guardará la máxima profundidad del árbol obtenida.
La clase búsqueda en sí tiene los siguientes atributos, “Grafo_problema_” el cual es un objeto de la clase grafo que guarda el números de nodos del grafo y la tabla de transiciones implementando la en forma de matriz con con un vector de vectores. Aparte de esto la clase búsqueda implementa un vector en el que guarda la heurística “Heuristica_Problema_”, guarda en un atributo los nodos generados “Nodosgenerados_”, en otro atributo los analizados “Nodosanalizados_”, la profundidad del árbol “Profundidad_” el origen de la búsqueda “Origen_” y el árbol generado durante la búsqueda mediante nodos, estos nodos guardan: el nombre del nodo en “Id_”, la heurística del estado “Heuristica_Estado_”, el coste acumulado hasta este nodo ”CosteAcumulado”, un puntero al nodo padre “Padre_”, un vector depunteros de nodos hijos “Hijos_” y la profundidad del nodo “Profundidad_”.

Funcionamiento de la búsqueda

Cuando se genera la búsqueda, se le pasa el fichero donde está contenido el grafo y el fichero de la heurística, y se cargaran en la estructura de datos de la clase anteriormente explicada.
Usando el método “realizarBusquedaAEstrella” le indicaremos el origen y destino de la búsqueda y empezará a desarrollar el camino mínimo. Con este fin este método inicializa la raíz de nuestro árbol, cuenta que se ha generado el primer nodo y llamará el método “analizarGenerarArbol” con la raíz y el destino, este método devolverá con una estructura pareja el camino mínimo y el coste de este, para que búsqueda pueda guardar el resultado en su correspondiente estructura y guardar el árbol en el atributo designado para ello.
El método “analizarGenerarArbol” prepara el coste mínimo poniéndolo a su máximo tamaño, crea un conjunto llamados “nodosAEvaluar” que guardará los nodos pendientes de procesar, y generará el vector para guardar el camino mínimo encontrado hasta el momento, con estos datos, la raíz y destino, llama a “analizarGerenarArbolRecursivo” tras ello, devolverá los datos a el método llamante.
El método “analizarGenerarArbolRecursico” suma uno a nodos analizados y comprueba si es el nodo destino, en caso afirmativo, comprueba que su coste es menor que cualquier otro encontrado anteriormente, en caso afirmativo se establece su coste como el mínimo encontrado hasta el momento, se guarda su camino y se sale del método. En caso de que no fuera el nodo destino se generarán los hijos con el método “generarHijos” y nos devolverá en nodosAEvaluar los nodos que se deberían evaluar, posteriormente se van extrayendo estos nodos con “extraeelmejor”, un método que busca en el conjunto, el nodo

con coste acumulado más heurística menor, con este nodo se analiza si su coste acumulado es mayor que el posible coste mínimo que se ha encontrado, en caso de que sea así, se descarta debido a que esa rama ya solo puede dar un resultado peor que el ya obtenido y se seguirá extrayendo el resto de nodos. En caso de que el nodo extraído tuviera menor coste que el camino actualmente encontrado, se realizará el análisis de ese nodo llamando recursivamente a este método.
El método "generarHijos" valiéndose de la Id del nodo el cual se quieren generar los hijos, se coge la correspondiente fila de la matriz de transiciones y se buscan transiciones, una vez encontrada una transición se verifica que el nodo destino no esté en la rama del nodo actual hasta la raíz, los nodos que corresponden a esta rama nos los da el método "nodosPadre", en caso de haber cumplido estas restricciones, se suma un nodo generado, se generará el nodo y se inicializan sus atributos. Esto se realizará con todas las transiciones de grafo.

El programa contiene 2 main's el primero sirve para de forma manual durante la ejecución, ir introduciendo los datos(archivo del grafo, heurística, origen y destino), el segundo main ejecuta de forma automática todos ejemplos proporcionados, incluido el del enunciado de la práctica y devolviendo el resultado en el archivo out.txt el resultados es podrá encontrar al final de el informe.

La práctica se ha realizado con control de versiones en git y su correspondiente repositorio en github que se hará público tras se cierre el periodo de entrega de la práctica.
Se ha empleado CMake para la creación de los ejecutable, visto que ya en otras asignaturas se me ha comentado que no se conocía la herramienta mantendré la carpeta cmake-build-debug en la cual se encontrarán ambos ejecutables "Pr1" con el main manual y "Pr1analisis" con el segundo main, también se podrá encontrar el Makefile generado con CMake que permitira compilar los programas como cualquier otro Makefile.

Contenido archivo de resultados out.txt

Práctica de Busqueda A* Inteligencia Artificial

Fichero del grafo: Grafo //Ejemplo del
Fichero de heuristica: heuristica_O1F5 //enunciado de la práctica
Origen: 1
Destino: 5
Distancia: 3.65
Camino: 1 --> 2 --> 5
Nodos generados: 5
Nodos analizados: 4
Profundidad Total del arbol: 2

Fichero del grafo: Grafo1
Fichero de heuristica: Grafo1Heuristica
Origen: 1
Destino: 8
Distancia: 43
Camino: 1 --> 2 --> 4 --> 12 --> 5 --> 13 --> 8
Nodos generados: 104
Nodos analizados: 75
Profundidad Total del arbol: 8

Fichero del grafo: Grafo1
Fichero de heuristica: Grafo1Heuristica2
Origen: 1
Destino: 8
Distancia: 43
Camino: 1 --> 2 --> 4 --> 12 --> 5 --> 13 --> 8
Nodos generados: 104
Nodos analizados: 75
Profundidad Total del arbol: 8

Fichero del grafo: Grafo2.txt
Fichero de heuristica: Grafo2Heuristica1.txt
Origen: 2
Destino: 14
Distancia: 41
Camino: 2 --> 4 --> 12 --> 5 --> 13 --> 8 --> 14
Nodos generados: 90
Nodos analizados: 66
Profundidad Total del arbol: 8
Fichero del grafo: Grafo2.txt
Fichero de heuristica: Grafo2Heuristica2.txt
Origen: 2
Destino: 14
Distancia: 41
Camino: 2 --> 4 --> 12 --> 5 --> 13 --> 8 --> 14
Nodos generados: 90
Nodos analizados: 66
Profundidad Total del arbol: 8

Fichero del grafo: Grafo3.txt
Fichero de heuristica: Grafo3Heuristica1.txt
Origen: 1
Destino: 15
Distancia: 40
Camino: 1 --> 2 --> 4 --> 8 --> 16 --> 15
Nodos generados: 65

Nodos analizados: 49
Profundidad Total del arbol: 7
Fichero del grafo: Grafo3.txt
Fichero de heuristica: Grafo3Heuristica2.txt
Origen: 1
Destino: 15
Distancia: 40
Camino: 1 --> 2 --> 4 --> 8 --> 16 --> 15
Nodos generados: 65
Nodos analizados: 49
Profundidad Total del arbol: 7

Fichero del grafo: Grafo4.txt
Fichero de heuristica: Grafo4Heuristica1.txt
Origen: 10
Destino: 18
Distancia: 31
Camino: 10 --> 5 --> 3 --> 2 --> 4 --> 18
Nodos generados: 34
Nodos analizados: 22
Profundidad Total del arbol: 6
Fichero del grafo: Grafo4.txt
Fichero de heuristica: Grafo4Heuristica2.txt
Origen: 10
Destino: 18
Distancia: 31
Camino: 10 --> 5 --> 3 --> 2 --> 4 --> 18
Nodos generados: 34
Nodos analizados: 22
Profundidad Total del arbol: 6

Fichero del grafo: GrafoRumania.txt
Fichero de heuristica: GrafoRumaniaHeuristica.txt
Origen: 1
Destino: 2
Distancia: 540
Camino: 1 --> 20 --> 15 --> 3 --> 14 --> 2
Nodos generados: 22
Nodos analizados: 17
Profundidad Total del arbol: 6