Memoria Practica IA Búsqueda A*

Introducción:

El algoritmo de búsqueda A* es uno de los algoritmos de búsqueda de grafos heurístico o informado, el cual encuentra siempre el camino de menor coste entre un nodo origen y un nodo objetivo.

Estructuras de datos:

El programa está formado por varias clases que permiten almacenar un Grafo no dirigido y realizar el algoritmo de Búsqueda A*:
El conjunto de clases que representan al Grafo son la clase ‘Node’, la clase ‘Arc’ y la clase ‘Graph’.

La clase ‘Node’ representando el Nodo del Grafo, contiene como atributos principales su identificador numérico, el valor que le atribuye la heurística y una lista de arcos correspondientes a la clase ‘Arc’.

La clase ‘Arc’ representa la conexión entre 2 nodos en cuyos atributos tenemos el nodo de origen y el nodo destino del arco, aparte del coste asociado al mismo.

La clase ‘Graph’ es la que representa un Grafo y une las 2 clases anteriores conteniendo como atributo principal un array de todos los nodos del Grafo y un valor numérico del total de arcos del mismo. Esta clase se encarga principalmente de leer los ficheros que contiene la información del Grafo y se encarga también de asignar los valores heurísticos de cada nodo.

Como clases principales separadas de la representación del Grafo, tenemos la encargada de realizar el algoritmo de búsqueda A*, la clase ‘AStarSearch’ y otra clase que representa cada camino que se va generando por el algoritmo llamado ‘Path’.

Para la clase ‘Path’ que representa a cada uno de los caminos que se irán generando en el algoritmo de búsqueda A*, se encuentra un atributo de array con una lista ordenada de cada nodo que recorre el camino, un atributo numérico que representa la suma del coste total

del camino (sin contar el valor de la heurística) y un atributo booleano ‘closed’ que indica si no existe ningún nodo sucesor más que permita expandir al camino, en cuyo caso lo consideraremos como cerrado. Tenemos como función principal ‘closeThePath’ al cual le pasamos el nodo inicial y el nodo final para que compruebe si se puede cerrar el camino y lo haga.
Los casos por los que un camino se puede cerrar son que ya se encuentre en el nodo objetivo, que estemos en el nodo de comienzo o que no exista ningún otro nodo sucesor que ya se encuentre en el propio camino.

La clase principal que ejecuta todo el algoritmo de busqueda A* llamado ‘AStarSearch’ contiene un atributo para la clase ‘Graph’, uno que representa tanto al nodo inicial como al nodo final, un array con todos los caminos que se han generado, un array con los caminos que pueden llevar a una posible solución (es decir, que acaben en el nodo final), y otro que contenga todos los caminos con soluciones del coste mínimo. Aparte tenemos atributos con una lista de todos los nodos generados y otra con los nodos inspeccionados. Como funciones principales tenemos la función ‘aStar’ que es la que ejecuta el algoritmo de búsqueda A* principal y otra función ‘expandNode’ encargada de expandir un nodo cuando lo indique el propio algoritmo.

Por último tenemos una clase ‘Main’ principal, encargada de enviar los archivos para leerlos, y recoger por teclado con que nodo queremos como el inicial y como el final. Se encargará de llamar a la clase ‘AStarSearch’ pasandole todos estos datos y llamara a la función ‘writeResults’ para sacar en el fichero results.txt toda la información que ha generado el algoritmo.

Ejecución del programa:

El programa está hecho en java y el comando necesario para su ejecución es la siguiente:

java Busqueda_A.main [ruta del fichero con el grafo] [ruta del fichero con las heurísticas]

Tabla generada con los valores obtenidos de cada fichero:

Instancia	n	m	Vo	Vd	Camino	Distancia	Nodos Generados	Nodos Inspeccionados
ID1	15	19	1	8	1 - 2 - 4 - 12 - 5 - 13 - 8	43	126	78
ID1	15	19	1	8	1 - 2 - 4 - 12 - 5 - 13 - 8	43	126	78
ID2	15	19	2	14	2 - 4 - 12 - 5 - 13 - 8 - 14	41	175	113
ID2	15	19	2	14	2 - 4 - 12 - 5 - 13 - 8 - 14	41	175	113
ID3	20	25	1	15	1 - 2 - 4 - 8 - 16 - 15	40	87	54
ID3	20	25	1	15	1 - 2 - 4 - 8 - 16 - 15	40	87	54
ID4	20	25	10	18	10 - 5 - 3 - 2 - 4 - 18	31	238	144
ID4	20	25	10	18	10 - 5 - 3 - 2 - 4 - 18	31	238	144
ID5	20		1	2				