****

**Practica1**

**Inteligencia Artificial**

José Miguel Avellana López 18068091G

Jassine El Kihal X8592934L

**Implementacion de cola con prioridad en algoritmos UCS, A\* y BestFS**

Con el objetivo de que el fringe ordene sus nodos se dará uso de la clase *priorityQueue* de *util.py.* A esta cola se le ha de introducir un elemento y una prioridad asociada, y sacará el elemento con menor prioridad.

El problema es que si se quiere tener como elemento un nodo y prioridad su costo el método update de la cola no sirve, ya que este comparará el costo de nodos iguales y no con igual estado. Debido a esto, no se dará uso del método update, en vez de usar el método update se hará lo siquiente con la salida del fringe de cada nodo:

* Si el estado del nodo aun no se había alcanzado, se subirá al fringe con su respectivo costo.
* Si el estado del nodo ya se habia alcanzado, entonces comprueba si ha sido expandido.
* En el caso en que el nodo halla sido expandido,se ignorará y se pasará a coger el siguiente nodo del fringe, pues seguro que su estado no es objetivo ni sus hijos tienen mejor costo que los que ya ha expandido el nodo de mismo estado anterior.

if generated[n.state][0] == n:

Si no se cumple la condición anterior aseguramos lo comentado en este parrafo.

* En el caso en que el nodo de menor costo siga en el fringe, se comprobará si el nodo actual tiene mejor costo y, de tenerlo, se subirá al fringe.

En resumen, en vez de actualizar el nodo, cuando se expanda el nodo con menor coste, de los subidos a la cola hasta ese momento de ese estado, se puede asegurar que el resto de nodos de ese mismo estado serán ignorados aunque permanezcan en la cola.

Y que, además, al haber sido expandido, no podrían haber llegado al fringe nodos de ese mismo estado con un menor coste. Esto es porque estos nodos tendrían que haber sido generados por algun nodo de mayor coste que el nodo expandido, y por tanto, si los costos no son negativos, aseguramos la optimalidad.

**Evaluación experimental**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **tiempo ejec.**  **nodos expan.** | **UCS** | **ASTAR- manhattan** | **ASTAR-**  **euclidean** | **BFSH-**  **manhattan** | **BFSH-**  **euclidean** |
| **bigMaze** | 0.422s  620 | 0.333s  466 | 0.335s  471 | 0.337s  466 | 0.338s  471 |
| **mediumMaze** | 0.116s  269 | 0.125s  78 | 0.172s  159 | 0.154s  78 | 0.161s  159 |
| **openMaze** | 0.128s  682 | 0.126s  89 | 0.137s  54 | 0.141s  89 | 0.143s  54 |
| **testMaze** | 0.071s  7 | 0.068s  7 | 0.070s  7 | 0.074s  7 | 0.077s  7 |
| **tinyMaze** | 0.072s  15 | 0.070s  8 | 0.071s  8 | 0.068s  8 | 0.071s  8 |

Observaciones:

* Los algoritmos informados tienen los mismos valores con las mismas heuristicas ya que, al ser costes identicos, los costos no influyen en la prioridad del A\*, solo la heuristica.
* La heuristica Manhattan obtiene mejores resultados en grafos con menor número de aristas. (El openMaze es una grafo casi completo porque no tiene ‘muros’).
* El tiempo de ejecucion se ha obtenido apartir del comando *time*, se trata del costo real, pero nos da una aproximación de la media. No existe gran variacion entre cada algoritmo.