**DCC028 - Inteligência Artificial**

**Trabalho Pratico 1: Busca em Mapas**

**Aluno:** Romeu Junio Cunha de Oliveira **Matrícula:** 2012422971

**Introdução**

Neste trabalho prático foi implementado as versões de busca em grafos de quatro algoritmos conhecidos: *Interactive Deepening Seach, Unform Cost Seach*, *Best First Seach* e *A\** utilizando como heurísticas as funções *Manhattan Metric* e *Octile Distance*.

O algoritmo *Interactive Deepening Search* realiza uma iteração sobre um outro algoritmo de pesquisa chamado *Depth-Limited Search*, que mistura os pontos fortes da busca em largura (*Breadth-First Search*) e da busca em profundidade (*Depth-Fist Search*). A cada rodada é incrementado a profundidade na qual o *DFS* irá realizar a busca. Os algoritmos *UCS*, *BFS* e *A\** possuem a mesma implementação de busca em grafos. Neste modelo, é utilizado duas estruturas de dados: a lista **ABERTO(A)** e **FECHADO(F)**. Em *F* é armazenado os nós explorados (que já foram expandidos), e em *A* é armazenado os nós que foram expandidos, porém ainda não foram explorados. Ambos algoritmos expandem o nó de **menor custo**, é aí que eles diferem. Para o *UCS* o menor custo é calculado considerando o custo para chegar do nó início até ele. Para o *BFS* o menor custo é baseado apenas na função heurística, que no caso deste trabalho prático foi considerado apenas a heurística *Octile Distance*. O menor custo do algoritmo *A\** é calculado considerando o custo para chegar ao nó do nó inicio até ele somado com a função heurística.

O algoritmo IDS é completo, porém só é ótimo quando o custo das arestas é constante. O algoritmo UCS é completo e ótimo (*Dijkstra*). O *BFS* não é ótimo pois é um algoritmo guloso, mas esta versão é completa por não entrar em loop. O *A\** é sempre completo, e é ótimo dependendo se a heurística utilizada é consistente ou não. ¹

**Implementação**

Python foi a linguagem utilizada para implementação dos algoritmos descritos (versão 3.6). Para execução do programa foram criados quatro arquivos *shell*: São eles: ***ids.sh****,* ***ucs.sh****,* ***bg.sh*** *e* ***aestrela.sh***. Ambos recebem como parâmetros o mapa para construção do grafo e as coordenadas x e y dos nós de início e fim. Para o algoritmo *A\** um parâmetro adicional: (**1**: para heurística Manhattan e **2**: para a heurística Octile)

Exemplos de execução:

*./ucs.sh map1.map 0 0 255 255*

*./aestrela.sh map2.map 125 125 50 50 1*

Antes de começar a explicar o funcionamento do programa em si, algumas explicações sobre algumas escolhas precisam ser feitas. Primeiramente para a lista aberto decidi utilizar a estrutura ***heap queue*** (um heap com fila de priodidades), que na minha opinião é a estrutura perfeita para lidar com a lista *A.* Pois vemos que todos os algoritmos podem serimplementados como um único módulo de pesquisa (*Search.py*) e que diferem basicamente em como *A* é gerenciada, ou seja, sempre escolho e o expandir o nó de **menor custo.**

¹ uma heurística é consistente se os valores dos custos das arestas são não decrescentes ao longo do caminho.

Então para obter este item de menor custo eu tenho uma complexidade O(1) e para inserir a lista A eu tenho a complexidade de ordenar um heap que no pior caso é O(log n).