Trabalho Prático
Introdução à Inteligência Artificial

João Antonio Oliveira Pedrosa
Matrícula: 2019006752

[1] Universidade Federal de Minas Gerais
Belo Horizonte - MG - Brasil

joao.pedrosa@dcc.ufmg.br

1. Introdução

O trabalho consiste em implementar diferentes algoritmos de busca vistos em sala de aula, sendo eles: Busca em Largura, Busca de Custo Uniforme, Busca Iterativa, Busca Gulosa e A*. O programa recebe um mapa discretizado, com cada posição representando um tipo de terreno e o custo para andar naquele terreno. Dados um ponto inicial, um ponto final e uma identificação de qual algoritmo usar, o programa deve imprimir o caminho e a distância encontrados.

2. Estruturas e Modelagem

A principal estrutura implementada é uma classe chamada Graph. Todos os algoritmos foram implementados nessa mesma classe.

2.1. Atributos

Para a implementação dos algoritmos, a classe conta com alguns atributos que são utilizados por todos eles.

2.1.1. Vector MV

Um vetor de tuplas, contendo as seguintes tuplas: $((-1, 0), (0, -1), (0, 1), (1, 0))$. Esse vetor representa os possíveis movimentos no mapa, assim, podemos iterar pelas posições deste vetor e calcular o vizinho de algum vértice somando o X do vértice com a primeira posição de cada tupla e o Y com a segunda.

2.1.2. Map Cost

Um *map* onde as chaves são caracteres e os valores são pontos flutuantes. Com esta estrutura, mapeamos o caractere representante de cada terreno ao seu custo.

2.1.3. Vector of Vector - Terrain

Uma matriz, representando o custo de cada nó do mapa passado na entrada.

2.2. Métodos

Cada algoritmo é representado como um método da classe. Todos recebem o mesmo input: Um nó inicial e um nó final e retornam a mesma coisa, uma tupla contendo a Distância e um Vetor representando o caminho. Em todos os algoritmos, uma estrutura contendo o pai de cada nó é mantida. Assim, após o fim da execução do algoritmo, podemos utilizar essa estrutura para recuperar o caminho percorrido.

2.2.1. BFS

Esse método implementa uma busca em largura simples, fazendo a utilização de uma estrutura do tipo *Queue*. Para isso, mantemos um vetor de distância de cada nó, tanto para recuperar a distância percorrida até o nó final, quanto para marcar os nós que ainda não foram visitados com distância igual a -1. Esse algoritmo só é ótimo para custos crescente de acordo com a profundidade mas é completo em geral.

2.2.2. Greedy

Esse método implementa uma busca gulosa. A busca gulosa não utiliza nenhuma estrutura além de um *Set* para manter quais nós foram visitados, nem mesmo a estrutura de pais citada no cabeçalho dessa seção. No lugar disso, o que é feito é manter qual o nó atual em uma simples variável. Feito isso, basta checar qual o melhor vizinho desse nó de acordo com a heurística. Caso o melhor nó já tenha sido visitado, encontramos um loop e o algoritmo levanta uma exceção de loop infinito. Caso contrário, o nó atual é adicionado ao caminho e ao custo e o seu melhor vizinho passa a ser o nó atual processado. Esse algoritmo não é otimo e nem completo.

2.2.3. IDS

Para a Busca Iterativa, foi implementado um outro método, representando uma busca de profundidade limitada. Assim, para a IDS, basta chamarmos o método da busca de profundidade limitada, iterando pela profundidade crescentemente. No método de busca limitada, as estruturas utilizadas são uma pilha, para simular a recursão mantendo o escopo e um vetor de distâncias. O algoritmo é completo, apesar de não ser ótimo.

2.2.4. UCS

A busca de custo uniforme é uma implementação do Algoritmo de Dijkstra. Para isso, são necessárias algumas estruturas, sendo elas uma *Priority Queue* para a fronteira e um vetor para as distâncias. O algoritmo é ótimo e completo.

2.2.5. A*

O mais utilizado dos algoritmos citados até aqui, o A* é uma combinação do Dijkstra com o Greedy e, portanto, possui uma implementação bem similar ao Dijkstra, utilizando

exatamente as mesmas estruturas. A única diferença é que, quando um nó é inserido na fila de prioridade, ao invés de adicionarmos sua distância, adicionamos a sua distância mais o valor de sua heurística. O algoritmo é ótimo e completo, desde que a heurística seja admissível.

2.2.6. Heurística

A heurística utilizada foi a distância de Manhattan. Como o terreno de menor custo possível possui custo igual à um, a distância de Manhattan é uma heurística admissível, afinal, a distância real de um nó sempre será maior ou igual à sua distância de Manhattan.

3. Análise Quantitativa

Para a análise quantitativa, iremos comparar, no mapa *cidade.map*, a distância encontrada de diferentes vértices ao mesmo vértice inicial: (1, 1). Juntamente com isso, será calculado o tempo de inferência de cada algoritmo.

Vértice Final	**Distância Encontrada**				**Tempo de Inferência**			
	BFS	**UCS**	**Greedy**	**A***	**BFS**	**UCS**	**Greedy**	**A***
(4, 4)	6	6	6	6	20ms	210ms	20ms	22ms
(30, 12)	140	50	140	50	40ms	215ms	25ms	231ms
(50, 27)	199	118.5	199	118.5	45ms	217ms	25ms	231ms
(100, 100)	482	254	Loop	254	67ms	212ms	INF	240ms
(150, 100)	567	301	Loop	301	57ms	216ms	INF	249ms
(150, 150)	706.5	369	Loop	369	51ms	236ms	INF	243ms
(250, 250)	1447	571	Loop	571	70ms	216ms	INF	272ms