Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Introdução à Inteligência Artificial
Progessor: Luiz Chaimowicz
Monitor: Tiago Negrisoli

Trabalho Prático I: Algoritmos de Busca para Menor Caminho

Aluno: Ruy Braga Filho

Introdução

O objetivo do trabalho é a implementação dos algoritmos de Busca em Largura (BFS), de Busca em Profundidade com aprofundamento iterativo (IDS), de Busca de Custo Uniforme (UCS), também conhecido como algoritmo de Dijkstra, de Busca Gulosa (Greedy) e o A estrela (Astar).

Esses algoritmos devem ser utilizados para percorrer um mapa bidimensional na forma de grid partindo de um ponto inicial e chegando a um pondo de destino. Cada posição do mapa tem um custo para ser percorrido. Os Algoritmos devem retornar o custo do caminho encontrado e o caminho encontrado.

Modelagem

O mapa foi modelado como uma matriz de caracteres, em que cada célula da matriz representa uma posição no mapa e tem um custo associado para ser percorrido. Cada posição representa um estado. A função de transição determina o próximo estado que o algoritmo deve ir.

O BFS varre o mapa em camadas, visitando inicialmente a posição inicial, depois os vizinhos imediatos do nó inicial, depois os vizinhos imediatos dos vizinhos imediatos da posição inicial e assim por diante.

O IDS faz uma busca em profundidade, cuja profundidade máxima aumenta a cada iteração.

O UCS vai para o vizinho com menor custo acumulado para ir da posição inicial até o vizinho.

O Greedy vai para o vizinho que tem a menor distância aproximada até a posição de destino. A distância aproximada é calculada usando a heurística de Manhattan.

O Astar vai para o vizinho que tem a menor soma do menor custo acumulado para ir da posição inicial até o vizinho com a heurística de Manhattan.

A heurística de Manhattan é admissível, pois seu valor é menor ou igual ao custo real para ir de uma posição à posição de destino.

Implementação

Os algoritmos foram implementados em Python 3. Além da matriz utilizada para representar o mapa, foram utilizadas matrizes auxiliares, uma para armazenar os antecessores de cada posição durante a busca e assim possibilitar a recuperação do caminho encontrado entre a posição inicial e o destino, uma para marcar se as posições que não foram processadas, as que estão na open list e as que foram expandidas. Os algoritmos de busca UCS e Astar utilizaram mais uma matriz, para armazenar o custo acumulado para ir da posição inicial até as outras posições processadas pelos algoritmos.

O programa foi modularizado com as seguintes classes python:

- **mapa.py**: representa o mapa a ser percorrido;
- **pathfinder.py**: uma super classe das classes que implementam os algoritmos de busca. Essa classe contém os atributos e métodos comuns a todos os algoritmos de busca;
- **bfs.py**: subclasse de pathfinder que implementa o algoritmo BFS;
- **ids.py**: subclasse de pathfinder que implementa o algoritmo IDS;
- **ucs.py**: subclasse de pathfinder que implementa o algoritmo UCS;
- **greedy.py**: subclasse de pathfinder que implementa o algoritmo Greedy;
- **a_star.py**: subclasse de pathfinder que implementa o algoritmo A*;
- **main.py**: módulo responsável por processar a linha de comando do usuário do programa, carregar os dados do arquivo, chamar o algoritmo de busca apropriado e por fim, imprimir o resultado.

Análise dos Algoritmos

Sendo **b** o branch factor, **d** a profundidade da busca, **C** o custo da solução ótima e **ε** o custo de cada passo, os algoritmos têm as seguintes complexidades assintóticas e as seguintes propriedades:

BFS

Complexidade de espaço: $O(b^d)$ + os espaços utilizados pelo mapa e pelas 2 matrizes auxiliares = 3 x O(mxn) onde m é o número de linhas e n o número de colunas das matrizes.

Complexidade de tempo: $O(b^d)$

Completude: Sim

Otimalidade: Para esse problema não, já que o custo para atingir uma posição não é uma função não decrescente da profundidade do nó.

IDS

Complexidade de espaço: $O(b^d)$ + os espaços utilizados pelo mapa e pelas 2 matrizes auxiliares = 3 x O(mxn) onde m é o número de linhas e n o número de colunas das matrizes.

Complexidade de tempo: $O(b^d)$

Completude: Sim

Otimalidade: Para esse problema não, já que o custo para atingir uma posição não é uma função não decrescente da profundidade do nó.

UCS

Complexidade de espaço: $O(b^{1+C/\varepsilon})$ + os espaços utilizados pelo mapa e pelas 3 matrizes auxiliares = 4 x O(mxn) onde m é o número de linhas e n o número de colunas das matrizes.

Complexidade de tempo: $O(b^{1+C/\varepsilon})$

Completude: Sim

Otimalidade: Sim, pois nesse problema não há aresta com custo negativo

Greedy

Complexidade de espaço: $O(b^d)$ + os espaços utilizados pelo mapa e pelas 2 matrizes auxiliares = 3 x O(mxn) onde m é o número de linhas e n o número de colunas das matrizes.

Complexidade de tempo: $O(b^d)$

Completude: Para esse problema sim, pois foi utilizada uma matriz auxiliar para para evitar loops e a árvore de busca não tem ramos com profundidade infinita.

Otimalidade: Não

A*

Complexidade de espaço: $O(b^d)$ + os espaços utilizados pelo mapa e pelas 2 matrizes auxiliares = 3 x O(mxn) onde m é o número de linhas e n o número de colunas das matrizes.

Complexidade de tempo: $O(b^d)$

Completude: Sim

Otimalidade: Sim, pois a heurística de Manhattan é admissível.

Como teste para verificar o comportamento do tempo de execução, foram gerados mapas com dimensões 150x150, 300x300, 450x450, 600x600 e 750x750, onde as coordenadas dos pontos iniciais e finais eram $\mathbf{x_i}$=0, $\mathbf{y_i}$=altura//2, $\mathbf{x_f}$=largura-1 e $\mathbf{y_f}$=altura//2.

Segue a tabela com os tempos de usuário determinado pelo comando time do linux

	150x150	300x300	450x450	600x600	750x750
BFS	0.148s	0.576s	1.299s	2.271s	3.759s
IDS	4.603s	36.182s	95.055s	278.664s	366.205s
UCS	0.389s	1.364s	3.588s	6.036s	9.486s
Greddy	0.047s	0.065s	0.116s	0.176s	0.205s
A*	0.131s	0.389s	0.872s	2.366s	2.514s

Conclusão

Em todos os casos testados, os algoritmos UCS e A* encontraram os caminhos de menor custo, sendo que o A* sempre demorou menos tempo para processar. Isso se deve ao fato dele expandir menos nós do que o UCS.

Os algoritmos BFS, IDS e Greedy não encontraram o caminho de menor custo para os mapas utilizados.

O Greedy foi sempre o mais rápido, pois ele sempre vai na direção de menor distância aproximada pela heurística.

Para esse problema, o algoritmo A* se mostrou imbatível, encontrando sempre a solução ótima, a uma custo de tempo de execução bem interessante, sendo sempre o segundo mais rápido.

https://github.com/ruybragafilho/tp01_iia.git