TP 01 -Trabalho Prático 1
Algoritmo de Busca para Menor Caminho

Jean Lucas Almeida Mota

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

jeanlk@ufmg.br

1. Introdução

O problema proposto se baseia em usar inteligência artificial e seus algoritmos de busca nos espaços de estados para encontrar o menor caminho entre dois pontos. No caso, dado um mapa discretizado, queremos discutir quais são os algoritmos mais ou menos eficientes para encontrar o caminho entre dois pontos do mapa. Os algoritmos discutidos nesse trabalho são:

- Busca em Largura (BFS)
- Aprofundamento Iterativo (IDS)
- Busca de Custo Uniforme (UCS)
- Gulosa
- A*

2. Implementação

O programa foi desenvolvido na linguagem python.

2.1 Modelagem e Estruturas de Dados

O tipo de agente escolhido foi o Problem-Solving-Agent, por causa das especificações dos algoritmos a serem discutidos. Dessa forma:

- O objetivo é representado como um conjunto de estados.
- O agente começa de um estado inicial e ações fazer o agente trocar de estado.
- Uma busca no espaço de estados deve ser feita a fim de encontrar a sequência de ações que levam do estado inicial a um dos estados objetivos.

O ambiente do Problem-Solving-Agent é um mapa discretizado com as seguintes condições: estático, observável, discreto e determinístico.

Formulação do problema:

- Estados: cada estado possível é uma posição cartesiana no mapa, ou seja, um par ordenado do tipo (coluna, linha).
- Estado inicial: qualquer um.
- Sequência de Ações: Cima, Baixo, Esquerda, Direita

- Modelo de Transição:
    - Cima: (coluna, linha - 1)
    - Baixo: (coluna, linha + 1)
    - Esquerda: (coluna – 1, linha)
    - Direita: (coluna + 1, linha)
    - OBS: respeitando os limites máximo do mapa.
- Teste objetivo: verificar se o estado encontrado é o estado final.
- Custo: cada ação possuo custo 1.

A modelagem do problema foi feita a fim de facilitar a implementação dos algoritmos de busca em espaço de estados. Dessa forma, a estrutura principal utilizada foi o grafo, onde:

- O vértice representa: estado, custo de passar por esse estado, pai, profundidade.
- OBS: Estamos considerando que todas as ações são válidas para todos os estados. Dessa forma, testamos se o estado encontrado após a ação é viável ou não. Um estado é considerado inviável se sua posição não estiver dentro dos limites do mapa, ou se seu custo for infinito.
- A aresta representa que existe um caminho viável entre dois estados.

Fica evidente que não foi usada nenhuma estrutura padrão para grafos, como listas de adjacência ou matrizes de adjacência. Isso ocorreu porque não havia necessidade de construir tais estruturas. A relação de paternidade foi mais que suficiente para implementar os algoritmos de pesquisa.

A função sucessora foi implementada exatamente como o modelo de transição foi proposto, uma função que recebe uma posição e retorna o conjunto de estados alcançáveis a partir daquele nó com cada uma das ações definidas. As operações realizadas possuem custo O(1), pois se tratam apenas de operações de soma e subtração.

Para os algoritmos de busca com informação, foi necessária a definição de uma heurística. A heurística escolhida para a função de avaliação é a distância euclidiana entre dois pontos no plano cartesiano, no caso, a distância euclidiana entre um ponto qualquer e o ponto objetivo.

- É uma heurística admissível pois a distância euclidiana $d = \sqrt{(X_a - X_b)^2 + (Y_a - Y_b)^2}$ será sempre menor ou igual a distância p percorrida no mapa discreto. Uma vez que d é a distância em linha reta entre dois pontos e p é um caminho que precisa satisfazer restrições do mapa.

Dessa forma, foi possível implementar de maneira eficaz os algoritmos de busca sem informação e com informação.

2.2 Implementação

A fim de modularizar a implementação, foram implementadas duas classes principais, uma de baixo nível e outra de mais alto nível. São elas:

Classe SistemaNPC:

- É a classe de mais alto nível, onde foram feitas as operações de leitura da entrada, escolha do algoritmo a ser utilizado e output da solução.

Classe Agente:

- É a classe de mais baixo nível, onde foi implementado o agente de fato. E também onde foram implementados os algoritmos de busca, são eles:
    - Busca em profundidade
        - Funções: BFS(no_inicial) e BFS_helper(no_inicial)
        - Descrição: Algoritmo que faz uma busca por nível no espaço de estados viáveis. É **completo**. Para este problema **não é ótimo**.
        - Tempo: $O(b^d)$
    - Aprofundamento Iterativo
        - Funções: IDS(no_inicial) e IDS_helper(no_inicial)
        - Descrição: Algoritmo que combina os benefícios do BFS e DFS. Possui o mesmo tempo do BFS com a economia de memória do DFS. É **completo**. Para este problema não é **ótimo.**
        - Tempo: $O(b^d)$
    - Busca de custo uniforme
        - Funções: UCS(no_inicial) e UCS_helper(no_inicial)
        - Descrição: Algoritmo que busca explorar primeiro os nós com o menor custo, assim como Dijkstra. É **completo** e **ótimo.**
        - Tempo: $O(b^{1+C^*/\varepsilon})$
    - Guloso
        - Funções: Greedy(no_inicial) e Greedy_helper(no_inicial)
        - Descrição: Algoritmo que usa a heurística da distância euclidiana para escolher de maneira gulosa qual estado deve ser explorado. Em sua definição formal não é **completo**. No entanto, com a adição de um vetor de explorados, o algoritmo para este problema se tornou **completo**. Não é **ótimo**.
        - Tempo: $O(b^m)$
    - A*
        - Funções: Astar(no_inicial) e Astar_helper(no_inicial)
        - Descrição: Algoritmo que usa uma função de avaliação que leva em conta a heurística e o custo real até determinado estado. Por fim, possui uma estimativa do custo da melhor solução. É **completo**. É **ótimo** desde que a heurística seja admissível.
        - Tempo: Exponencial no pior caso.
- Funções auxiliares:
    - GetCusto: Recebe uma posição e retorna o custo dela.
    - FuncSucessora: Recebe um estado e retorna um conjunto de estados alcançáveis a partir daquele estado.
    - DistObjetivo: Calcula a distância euclidiana de um ponto qualquer até o ponto objetivo.

Classes auxiliares:

- Classe No: Implementação do vértice do grafo modelado anteriormente.
- Classe Estado: Implementação do estado, ou seja, implementação do ponto no plano cartesiano.

3. Análise quantitativa

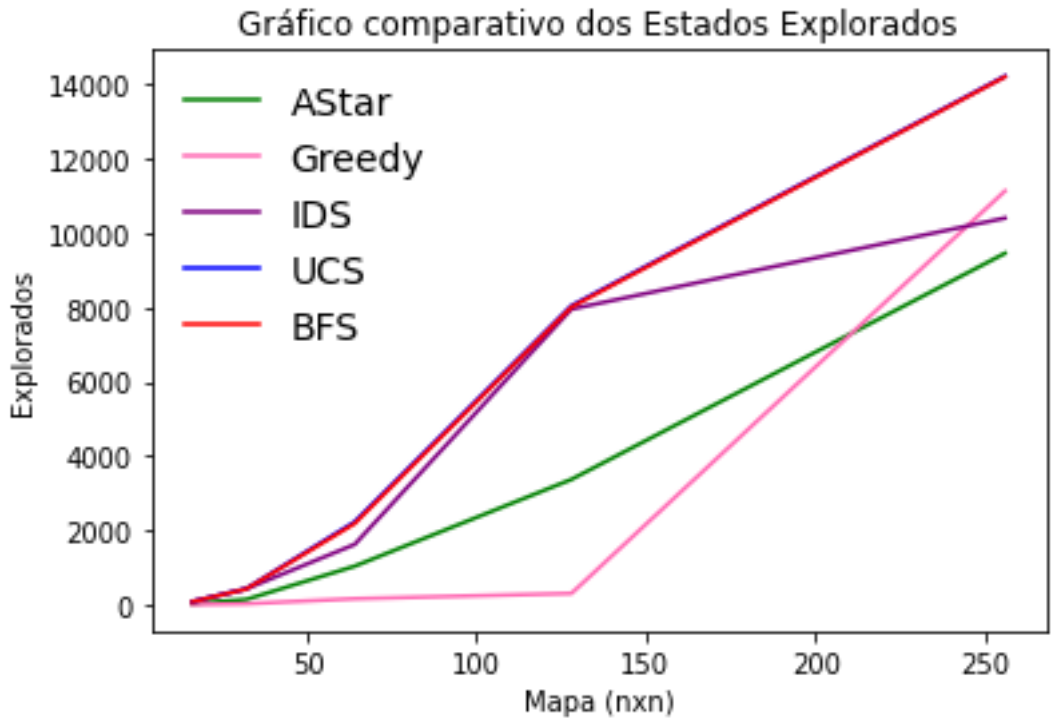
Para comparar os algoritmos em função do número de estados expandidos, foi gerado um gráfico (Estados Expandidos X Tamanho do Mapa), onde existem 5 curvas, cada uma representando um algoritmo. E para comparar em função do tempo de execução, também foi gerado um gráfico (Tempo X Tamanho do Mapa, onde existem 5 curvas, cada uma representando um algoritmo.

Para isso foram efetuadas as seguintes operações:

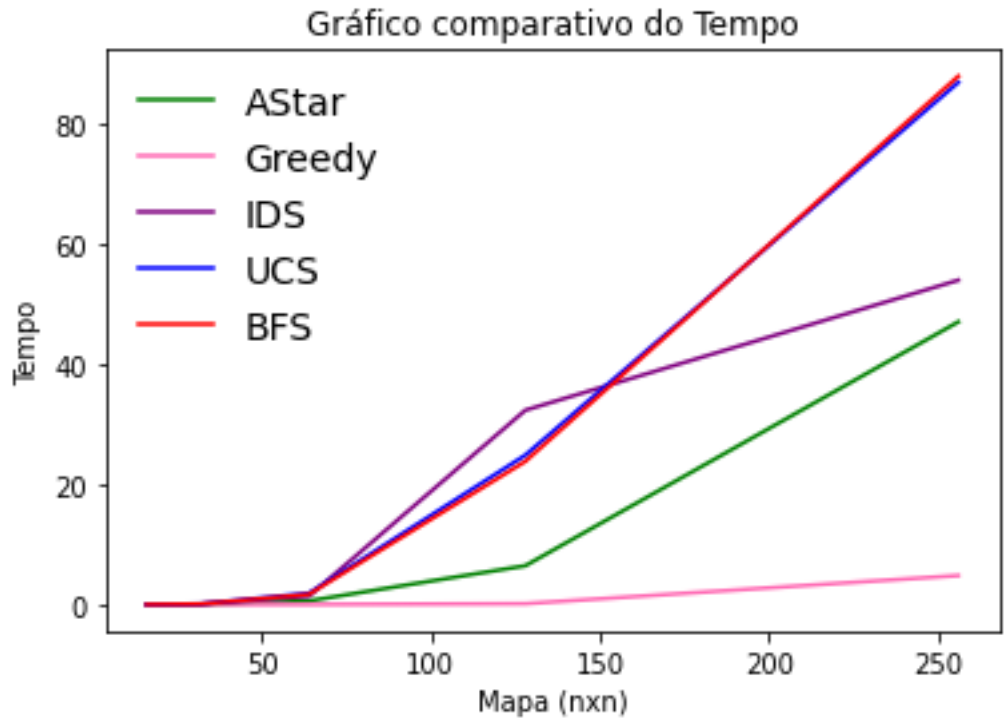
- Criação de 5 mapas, com tamanho: (16x16), (32x32), (64x64), (128x128), (256x256)
- Geração de 5 pontos iniciais e finais em cada um dos mapas, a fim de ter casos testes para obter a média de casos expandidos e a média do tempo em cada um dos mapas.
- Uma vez de posse das médias, basta construir os gráficos.

Segue os gráficos comparativos gerados:

- Estados explorados



- Tempo



Os gráficos individuais bem como os resultados dos casos teste, estão em um arquivo graficos.ipynb e dados.xlsx (excel), respectivamente.

4. Conclusão

Dessa forma, podemos perceber claramente a vantagem dos algoritmos de busca com informação. Uma vez que possuem uma performance melhor tanto em tempo de execução quanto na quantidade de estados explorados.

Percebe-se também que os métodos BFS e UCS são muito semelhantes. E pode dizer que o método UCS é o melhor dos métodos de busca sem informação, pois, além de encontrar a solução ótima, possui um tempo de execução e uma quantidade de estados explorados menor que os seus parentes da busca sem informação.

No entanto, entre os algoritmos de busca com informação, temos algumas ressalvas, pois não há exatamente um algoritmo melhor que o outro. Uma vez que o Greedy possui o melhor tempo de execução, mas perde para o A* na questão dos estados explorados quando o mapa cresce suficientemente. Além de que o A* retorna a solução ótima e o Greedy não. Contudo, o tempo de execução do Greedy é excepcional, até para os casos grandes.

Assim, concluo que a busca com informação, quando possível, será a melhor solução. Caso não seja necessária uma solução ótima, então o algoritmo Greedy é a melhor escolha. Agora quando não for possível a construção de heurísticas, então a melhor opção é o algoritmo UCS, pelo seu desempenho e também por retornar a solução ótima.

5. Execução

O programa foi desenvolvido no Windows 10 e testado no Windows 10.

Para execução no Windows foi executado o seguinte comando:
python pathfinder.py [caminho_para_arquivo_mapa] [metodo] xi yi xf yf
ou
python pathfinder.py mapa.map BFS 1 1 3 1

References

ALMEIDA, M Jussara. Slide Graph Traversal (2021). Disciplina Algoritmos 1 DCC206 UFMG.

Chaimowicz, Luiz e Prates, Raquel. Slide sobre Listas, Pilhas e Filas (2020). Disciplina Estrutura de Dados DCC205 UFMG.

Chaimowicz, Luiz. Slide sobre Solução de Problemas por Busca (2022). Disciplina Introdução a Inteligência Artificial DCC642 UFMG.

Chaimowicz, Luiz. Slide sobre Busca em Espaço de Estados (2022). Disciplina Introdução a Inteligência Artificial DCC642 UFMG.