IA - Métodos de busca

Aluno: Antônio Carlos Durães da Silva

Dado o limite de 1 a 3 páginas, para acessar o repositório com código atualizado, animações dos algoritmos, mais casos de teste, descrição dos códigos e diretórios, clique aqui.

Como executar o código

A partir do diretório do projeto, instale as dependências:

```
pip install -r requirements.txt
```

Basta executar o arquivo `main.py`. Para customizar algo da execução para alterar o arquivo main e/ou o `config.py`.

Algoritmos

Com exceção do A*, todos são métodos de busca não informada, isto é, não contam com uma heurística para orientar/direcionar a busca de um estado específico até o objetivo.

Sobre otimalidade, dado que o nosso problema tem custos distintos de movimentação, apenas algoritmos (A* e UCS) com heurísticas que consideram os custos podem ser ótimos. O BFS encontra o caminho ótimo para problemas em que as ações tenham mesmo custo (não é o caso).

Os algoritmos compartilham boa parte da implementação, a principal variação se dá nas estruturas de dados utilizadas, como fila (para o BFS), pilha (para o DFS) e fila de prioridade (para UCS e A*).

Breadth-First Search (BFS)

Dado um nó inicial ("X", por exemplo), o BFS explora todos os seus vizinhos até encontrar o estado objetivo. Para cada vizinho verificado, seus vizinhos são adicionados à fila para repetirem o processo. Seu comportamento se resume a uma busca exaustiva.

Depth-First Search (DFS)

Dado um nó inicial ("X", por exemplo), o DFS explora todo o ramo até encontrar o estado objetivo ou até findar o ramo (encontrar um nó folha, sem filhos). Caso finde o ramo sem encontrar o objetivo, o DFS retrocede e explora os ramos dos nós vizinhos do nó expandido anteriormente ("X").

Uniform-Cost Search (UCS)

O UCS pode ser visto como uma extensão do BFS. Ao invés de expandir todos os nós vizinhos de um nó, o UCS prioriza os nós com menor custo g, além de verificar se é possível melhorar o caminho até um nó já visto, desde que esse caminho seja menos custoso.

A* (A Star)

O diferencial deste algoritmo é que ele considera não só o custo g (nó atual até o nó objetivo) mas também o custo h (estado atual até o objetivo), entregue por uma heurística. Por esse diferencial, o A* mescla velocidade de execução com o encontro de um caminho ótimo.

Experimentos

Para acelerar a execução dos métodos de buscas, algumas decisões de implementação foram feitas:

- Utilizar conjuntos ("set") para fronteira e visitados para aproximar a checagem à complexidade O(1)
- Avançar com loop se o nó já estiver na fronteira ou visitado

Foram realizadas **100 execuções** para cada algoritmo, todas com a mesma **semente (42)**.

Os números de pontos flutuantes são arredondados para duas casas decimais somente no momento de gerar a tabela apresentada na seção de resultados.

Métricas

Para mensurar tempo de execução foram utilizados a média aritmética e o desvio padrão populacional. A mensuração do tempo só inicia ao invocar o método de busca, portanto, etapas como criação do mapa e escrita dos resultados em arquivos não são inclusas nessa medição.

O tamanho e custo do caminho final incluem o nó inicial e o nó objetivo, isto é, foi considerado o caminho completo.

O número de nós gerados foi tomado como o número remanescentes na fronteira quando o algoritmo terminou. Já o número de expandidos como foi tomado como o número de nós visitados. Ambas as métricas foram replicadas pelo que pode ser observado no código dado como exemplo (template).

Expectativas

É esperado que algoritmos não ótimos, como o DFS, executem mais rápido ao custo de encontrarem um caminho mais custoso. Espera-se também o oposto, isto é, que métodos ótimos, como UCS, executem mais lentamente ao custo de encontrarem um caminho mais barato.

Além da otimalidade, espera-se que o uso de heurística também seja um fator impactante no tempo de execução, tendo em vista adição de processamento para cálculos aritméticos e verificações adicionais.

Configuração da máquina

Software

Testes executados com Python 3.8 e Windows 10.

Hardware

- Notebook Acer Aspire Nitro 5, modelo AN515-54-718D
- 16GB RAM DDR4

- SSD 256GB

Processador:

- Intel® Core™ i7-9750H 9ª Geração
- 6 núcleos, 12 threads
- Frequência máxima de 4.50 GHz e base de 2.60 GHz

Resultados

Resultados com execução de labirinto 300x300:

método	média (ms)	desvio (ms)	desvio (%)	custo caminho	tamanho caminho	nós gerados	nós expandidos
Uniform Cost Search	7422.34	162.06	2.18	**436.32**	323	0	70147
A* (Euclidian)	549.68	25.93	4.72	**436.32**	323	1253	12722
A* (Octile)	267.16	26.16	9.79	**436.32**	323	923	5684
Breath First Search	1607.40	57.92	3.60	439.64	323	0	70147
Depth First Search	8.28	9.51	114.81	495.54	376	881	383

Por não contar com heurística, o DFS apresenta o caminho mais custoso, contudo tira proveito da posição do nó objetivo e seu comportamento simples de expansão para gerar a resposta mais rápida. Seu desvio padrão é alto, pois sua execução é extremamente rápida. Esse algoritmo é ideal para encontrar rapidamente alguma solução, sem se preocupar com heurística ou otimalidade.

Se ainda não há heurística, mas é desejável priorizar um caminho mais curto ou ótimo (caso não haja distinção de custo entre as ações) sobre o tempo de execução, o BFS é o algoritmo ideal. O algoritmo foi o segundo mais lento e com alto número de nós expandidos. Embora a solução encontrada seja satisfatória, não foi possível encontrar a ótima, pois o problema admite ações de custos distintos.

A busca de custo uniforme é uma boa opção quando há a necessidade da solução ótima, uma forma de calcular o custo g (custo do estado inicial até um estado específico), mas não há uma heurística admissível para o problema. O algoritmo foi o mais lento, o que era esperado, pois, além de expandir muitos nós, também inclui o tempo para calcular o custo dos caminhos gerados até encontrar a solução.

De posse de uma heurística, A* é o método recomendado quando o objetivo for combinar a otimalidade da solução com a velocidade de execução, ambas características vão depender da heurística utilizada. Quando usada a heurística euclidiana, por ser admissível (não superestima o custo de atingir o objetivo), o algoritmo gera uma solução ótima. Em contrapartida, ao utilizar a distância octil como heurística, o algoritmo executa com menos da metade do tempo da versão euclidiana e gera uma solução ótima (para esse labirinto em específico, em outros labirintos, o algoritmo gerou uma solução distinta da ótima por um ou dois nós).