U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Algoritmos de procura
(Jogo dos 15)

Ricardo Araújo Amorim, up202107843
David Rafael Pereira Nogueira, up202108293
Pedro Morim Figueiredo Andrade Leitão, up202107852

Porto
2023

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Índice

Introdução

Um problema de busca/procura pode ser definido como um algoritmo, onde tem como objetivo encontrar uma determinada solução, utilizando métodos ou estratégias específicas.

Existem diversos metódos/estratégias utilizados para resolver problemas de procura, como por exemplo:

Buscas não informadas (que não contem informação além da que é dada na definição do problema)

- Busca em profundidade
- Busca em largura
- Busca em profundidade Iterativa

Busca informada (que contem informação além da que é dada na definição do problema)

- Busca A*
- Busca Greedy

Jogo dos 15

O jogo dos 15 trata-se de uma matriz 4x4 com 15 peças com números de 1-15 e um espaço em branco para onde se pode mover as tais peças

O objetivo do jogo é ordenar as peças a partir de uma configuração inicial embaralhada, numa configuração final dada.

Para mudar a posição das peças só é possível mexer no espaço vazio(para cima, para baixo, para a esquerda e para a direita).

Este jogo, dependendo da sua configuração, poderá não ter solução.

Estratégias de Procura

(a) Procura não guiada (blind - “cega”)

Profundidade (DFS - Depth-First Search)

Esta estratégia expande sempre o nó mais profundo, até chegar ao fim da árvore (se for finita).

Pode ser aplicada por exemplo, se o problema tiver muitas soluções e o objetivo seja encontrar qualquer caminho e não o mais curto.

Complexidade temporal -> b^m; Complexidade espacial -> $b * m$; (onde b = fator de ramificação e m = profundidade máxima)

Largura (BFS - Breadth-First Search)

Esta estratégia expande o nó raiz primeiro, depois todos os sucessores desse nó. Em seguida, esses nós filhos da raiz expandem os seus sucessores, e por aí adiante.

Pode ser aplicada, por exemplo, se o objetivo for encontrar o menor caminho entre dois nós, pois esta pesquisa garante que a solução será a mais ótima possível.

Complexidade temporal -> b^d; Complexidade espacial -> b^d; (onde b = fator de ramificação e d = profundidade)

Busca Iterativa Limitada em Profundidade

Esta estratégia para prevenir a busca em profundidade ser infinita (em alguns casos), usa um limite, e trata os nós na profundidade desse limite como se não tivessem sucessores. Enquanto isso, para escolher o valor desse limite vai experimentando um a um à medida que a profundidade aumenta até encontrar uma solução ou dar uma falha.

Pode ser aplicada, por exemplo, para encontrar a solução mais próxima do nó raiz. Tem vantagens de ambas a busca em largura e busca em profundidade. Consegue ser ótima e completa (ao contrário da busca em profundidade) e gastar menos memória que a busca em largura.

Complexidade temporal -> b^d; Complexidade espacial -> $b * d$; (onde b = fator de ramificação e d = profundidade)

(b) Procura guiada (que usa heurística para orientar a procura)

Uma Heurística é uma técnica que usa informação além da que é dada pelo problema para encontrar uma solução mais rápida e eficiente como também é uma função que estima custos.

Gulosa

Esta estratégia tem como objetivo expandir o nó mais próximo do objeto, pois é mais provável de levar para uma solução, sem pensar nas possíveis consequências futuras. Podemos usar esta estratégia se quisermos encontrar uma solução perto da ótima, dependendo da heurística.

Usamos duas heurísticas, somatório do número de peças fora do lugar e manhattan distance.

Busca A*

Esta estratégia tem em consideração o custo para chegar ao nó e o custo para chegar a solução. Pode ser aplicada para encontrar a melhor solução, pois basta ver o custo menor para chegar o nó e o custo menor para chegar a solução. Pode ser aplicada se quisermos encontrar uma solução ótima e, no geral, dependendo da heurística, eficiente em ambos tempo e em espaço.

Usamos duas heurísticas, somatório do número de peças fora do lugar e manhattan distance.

Descrição da Implementação

Para implementar este problema, usamos a linguagem Java. Escolhemos esta linguagem pois é uma linguagem que todos os membros do grupo estão confortáveis (devido à cadeira de Estruturas de Dados do semestre anterior). O java para resolver este tipo de problemas tem várias vantagens como, o ecossistema de bibliotecas que é grande, ou seja, temos acesso a muitas ferramentas para ajudar nestes problemas, tem também como vantagem o facto de ser eficiente comparado com outras opções famosas como (python) e ser relativamente simples de usar.

Estruturas de dados utilizadas:

Para representar o tabuleiro definimos uma classe, contendo uma matrix inicial, uma matriz final e o *path* que vai ser feito. Escolhemos representar o tabuleiro por uma matriz pois é a solução mais simples para representar as diferentes configurações do jogo.

Usamos uma fila para implementar a BFS, para a DFS usamos uma pilha e para as estratégias Gulosa (Misplaced e Manhattan) e a A* (Misplaced e Manhattan) usamos uma *priority* queue. Escolhemos estas estruturas da biblioteca de java pois achamos que eram as melhores e mais eficientes para representar esses algoritmos.

Como estrutura do código optamos por separar cada algoritmo em ficheiros diferentes, por ser mais conveniente em termos analíticos como também pela sua compatibilidade com a linguagem.

Nomeamos os diferentes ficheiros com os seguintes nomes: BFS, DFS, IDFS, GreedyManhattan, Greedymisplaced, AManhattan, AMisplaced, Board, contendo em todos, o algoritmo referido.

Resultados

Configuração Inicial

1	2	3	4
5	6	8	12
13	9		7
14	11	10	15

Configuração Final

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Estratégia	Tempo (segundos)	Espaço	Encontrou solução?	Profundidade/Custo
DFS	-	-	Não	-
BFS	8,169	37383	Sim	12
IDFS	0,869	424	Sim	12
Gulosa (Misplaced)	0,115	1125	Sim	76
Gulosa (Manhattan)	0,086	42	Sim	12
A* (Misplaced)	0,100	135	Sim	12
A* (Manhattan)	0,088	60	Sim	12

Configuração Inicial

1	2	3	4
13	6	8	12
5	9		7
14	11	10	15

Configuração Final

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Esta configuração não tem solução porque, para uma configuração inicial (Ci) chegar a uma configuração final (Cf), o número de inversões de uma configuração (quando um dado número antecede outro com valor menor) tem de ser par e a linha onde se situa o espaço vazio tem de ser ímpar(a contar de baixo para cima e a começar em 1). Apesar de na configuração final isso se verificar, número de inversões = 0 e linha onde se situa o espaço vazio = 1, na configuração inicial, isso não se verifica, número de inversões = 20 e linha onde se situa o espaço vazio = 2.

Comentários Finais e Conclusão

Depois de analisar as diferentes estratégias para as configurações dadas, podemos observar que as estratégias de procura não guiada tiveram um pior desempenho que as das estratégias de procura guiada. As melhores das não guiada, foi com distinção a IDFS, o que era de esperar porque combina vantagens de tanto a busca em largura como a busca em profundidade. Nas estratégias de procura guiada, a melhor foi a A*, pois em ambas as heurísticas conseguiu ser eficiente em todos os parâmetros, enquanto na gulosa, apesar que na heurística de manhattan foi um pouco mais eficaz, no geral, não teve um desempenho tão bom na heurística misplaced.

Concluindo, a melhor estratégia para este problema é a A*, pois como referido anteriormente, as duas heurísticas foram eficazes tanto em tempo como em espaço.

Referências bibliográficas

. Slides das aulas teóricas da unidade curricular Inteligência Artifical (2022/2023) (12/03/2023).

. Russel,S. & Norvig,P. (2021). Artificial Intelligence: A Modern Approach, Global Edition(4th ed.).Pearson. (10/03/2023)

.Johnson, Wm. Woolsey; Story, William E. (1879), "Notes on the "15" Puzzle", (9/03/2023)

. Archer, Aaron F. (1999), "A modern treatment of the 15 puzzle", (9/03/2023)

. Leyton-Brown, Kevin.(2009), “Graph Search”, (10/03/2023)