RELATÓRIO

O Jogo Dos 15

Mafalda Ferreira Aires [up202106550]

Gonçalo Luís Garcias Monteiro [up202105821]

PROBLEMA DE BUSCA/PROCURA

Um problema de busca/procura é um problema com o objetivo de chegar a um estado final partindo de um estado inicial, passando por um conjunto de estados. Este conjunto de estados pode ser representado por uma árvore ou grafo, onde cada estado representa uma configuração do problema. A solução para este problema seria a sequência de passos que levam a configuração inicial à configuração final, a qual se alcança através da utilização de estratégias (não informadas ou informadas).

ESTRATÉGIAS DE PROCURA

As estratégias dividem-se em dois tipos: Informadas e Não Informadas.

Qual é a diferença de uma estratégia não informada e de uma estratégia informada?

Ao contrário da pesquisa não informada, a pesquisa informada faz uso de informação externa de modo a tornar a pesquisa mais eficiente em termos de memória, tempo e custo. Esta informação é analisada através de funções que avaliam qual é a melhor próxima jogada, promovendo o avanço para um estado mais próximo do final.

Por sua vez, a pesquisa não informada acaba por ser mais exaustiva e menos eficiente já que se limita a percorrer todos os estados até encontrar a solução.

Estratégias Não Informadas

Todas as estratégias não informadas são utilizadas quando o espaço de busca é pequeno, permitindo uma busca eficiente e com chegada à solução(excepto DFS).

Não são eficientes para árvores com muita profundidade ou profundidade infinita.

- Breadth First Search (BFS)

Este método percorre primeiro os nós com a mesma profundidade, começando na profundidade 0, antes de explorar os nós mais profundos.

Usado quando sabemos que a solução se encontra em pouca profundidade

Complexidade Temporal: O(b^d); b é fator de ramificação; d é profundidade

Complexidade Espacial: O(b^d); b é fator de ramificação; d é profundidade

Completude: Completa

- Depth First Search (DFS)

Este método explora desde do estado inicial até ao nó mais profundo, antes de explorar os seus sucessores.

Complexidade Temporal: O(b^m) ;b fator de ramificação; m profundidade máxima

Complexidade Espacial: b*m ; b fator de ramificação; m profundidade máxima

Completude: Não garante que a solução seja encontrada; Incompleta

- Busca em Profundidade Iterativa

Este método explora aplicando um *Depth Limited Search,* ou seja um DFS com profundidade limitada, iterando a profundidade máxima de 0 até encontrar uma solução (ou infinito, caso não exista solução).

Este algoritmo mistura as vantagens do BFS com DFS já que caso a solução esteja numa profundidade relativamente elevada, este pode evitar percorrer todas as profundidades até chegar a essa profundidade

Complexidade Temporal: O(b^d); b é fator de ramificação; d é profundidade

Complexidade Espacial:b*m ; b fator de ramificação; m profundidade máxima

Completude: completo

O que é uma Heurística?

Uma heurística é uma função que, aplicada a um estado e tendo em conta um estado final faz uma avaliação que permite estimar o custo do caminho entre esse estado inicial e o estado final.

Assim, uma heurística, embora não devolva necessariamente um resultado ótimo, oferece um resultado muitas vezes suficiente para uma pesquisa mais eficiente.

Estratégias Informadas

As estratégias informadas recorrem ao uso de heurística para a sua pesquisa. Ambas as estratégias Gulosa e A* são usadas quando muitos Nodes são explorados(elevada largura e profundidade) sendo importante definir aqueles Nodes que são mais importantes para evitar uso de memória e tempo desnecessário, sendo possível resolver problemas com elevada complexidade em pouco tempo e espaço.

- Estratégia Gulosa

Este método tem apenas em conta o valor da função heurística para escolher a melhor próxima jogada

A melhor heurística para resolver o problema com este algoritmo é Manhattan

- Estratégia A* (A estrela)

Esta estratégia, tem uma vantagem comparativamente à gulosa, já que tem também em conta o custo do caminho até ao estado em questão,(neste contexto o número de jogadas já feitas).

- A melhor heurística para resolver o problema com este algoritmo é Manhattan

IMPLEMENTAÇÃO

Escolha da Linguagem

A linguagem utilizada para a implementação do "Jogo dos 15" foi Python.

Escolhemos usar este linguagem, porque:

- É uma linguagem interpretada, permitindo testar várias funções e classes no modo iterativo de Python;
- Linguagem que estamos mais familiarizados e confiantes;
- Sintaxe mais simples e estrutura mais fácil de compreender;
- Linguagem mais usada em Inteligência Artificial.

Estruturas de Dados Utilizadas

Escolhemos uma class para representar o *Board* e os *Nodes* que contém as configurações/estados.

Vantagens

Esta escolha facilitou a atribuição de um *Board* a um *Node*, a comparação entre Nodes; a expansão e pesquisa em árvore ;e retornar o *path* (subindo na árvore).

Permitiu também guardar a posição do zero (“0”) na class *Board*.

Desvantagens

Contudo, o uso de uma class para representar a *Board* fazia com que os comandos fossem aplicados ao mesmo objeto, resultando em alterações nas configurações anteriores (Resolvemos este problema através de um *deepcopy*).

A class *Node* também teve um aspeto negativo: Dificultou a alternância entre as heurísticas a usar para resolver o problema. Solucionou-se através da criação de duas class *Nodes*, uma para cada heurísticas (Não era a solução pretendida mas, de todas as tentadas, foi a única eficaz).

Estrutura do Código

- Class Board - Atributos

Board.goal (Configuração final)

Board.board (Configuração inicial)

Board.blank (Posição do espaço em branco)

- Class Node - Atributos

Node.brd (Board representa o estado do Node)

Node.parent (Pai do Node atual)

Node.children (Filhos (4 ou menos) do Node atual)

Node.depth (Profundidade do Node atual, também usada para representar o custo)

Node.heur (Heurística)

Cada ficheiro corresponde a cada algoritmo de pesquisa, possuindo todas as funções necessárias para a busca, excepto para os algoritmos BFS e DFS cujo algoritmo base semelhante a ambos se une num só ficheiro: *simpleuninformed*.py .

Para além de ficheiros que contêm os algoritmos, existem também:

- *Board.py* onde se encontra o código com a class *Board* onde está definido e representado o jogo, as suas regras e os seus movimentos.

- *Node* onde se encontra a class *Node* que representa os nós usados para pesquisa não informada e informada apenas com heurísitca "manhattan".

- *NodeMis* onde se encontra a class *NodeMis* que representa os nós usados para pesquisa informada apenas com heurística "misplaced".

RESULTADOS

Para a configuração:

Inicial: 1 2 3 4 5 6 8 12 13 9 0 7 14 11 10 15

Final: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0

Estratégia	Tempo/ segundos	Espaço	Encontrou a solução?	Profundidade/ Custo
DFS	—————————	—————————	Não	—————————
BFS	2.9400	34 714	Sim	12
IDFS	3.6265	8 634	Sim	12
Gulosa - Misplaced	0.0465	747	Sim	62
Gulosa - Manhattan	0.0028	30	Sim	12
A* - Misplaced	0.0100	138	Sim	12
A* - Manhattan	0.0044	53	Sim	12

COMENTÁRIOS

Das estratégias não informadas, a pesquisa em largura (BFS) e a pesquisa iterativa em profundidade (IDFS) têm um bom desempenho na medida que retornam a solução pretendida através do caminho mais eficaz. No entanto, são relativamente demoradas. Ainda nas estratégias não informadas, a pesquisa em profundidade (DFS) não tem um bom desempenho nem é eficaz, pois se visualizarmos a árvore de pesquisa deste algoritmo, verificamos que o nó com a solução pode não se encontrar no ramo pelo qual a pesquisa foi iniciada ou então, estar numa profundidade muito grande.

Das estratégias informadas, as pesquisas com heurística Manhattan sobressaem pelo pouco espaço que ocupam na memória, sendo ainda assim, neste contexto, mais eficiente a Gulosa-Manhattan com um pouco menos de espaço e tempo.

Quanto às pesquisas com heurística Misplaced, o seu espaço na memória e o tempo são um pouco maior em relação às pesquisas com Manhattan, mas ainda assim, suficientes e muito mais eficientes que os algoritmos não informados.

Por fim, o algoritmo com melhor performance é, de acordo com os dados obtidos para este problema, Gulosa-Manhattan.

REFERÊNCIA BIBLIOGRÁFICA

- Artificial Intelligence: a Modern Approach Fourth Edition (Stuart Russell & Peter Norvig) - Chapter 3: "SOLVING PROBLEMS BY SEARCHING"