Inteligência Artificial – Trabalho 1
Relatório Sobre Buscas no Jogo dos 15

Miguel Gomes(201505151), Pedro Moreira(201507254) e Rafael Novais(201508010)

1 – Introdução

Um problema de busca pode ser definido informalmente por procurar respostas que ainda não existem para uma determinada pergunta. Formalmente, um problema de busca é definido por 4 diretivas: conjunto de estados, estado inicial, objetivo final e uma função sucessor (que define os próximos estados a serem visitados).

A resolução de um problema de busca deve ser começada pela parte de modelagem e planeamento do problema. Depois de analisado e com todas as variáveis em mente, procedemos a escolha do algoritmo e estruturas de dados com vista o objetivo final, seja ele eficiência, rapidez ou o uso de pouca memória.

2 – Descrição do Problema

O jogo dos 15 é um jogo "inventado" por Noyes Palmer Chapman que consiste numa matriz 4x4 onde há 15 casas numeradas e uma casa em branco. O problema consiste em partir de uma configuração inicial embaralhada das células e chegar a uma configuração final com uma ordenação determinada de algarismos.

Os movimentos/operadores possíveis para se chegar de uma configuração a outra são:

- mover a casa em branco para cima;
- mover a casa em branco para baixo;
- mover a casa em branco para a direita;
- mover a casa em branco para a esquerda;

3 – Estratégias de Procura

<u>Procura Não Guiada:</u>

- Depth-First Search (DFS): Expande sempre um dos nós com mais profundidade na árvore. Só quando não pode expandir mais nós é que volta atrás e expande um nó com menor profundidade. Para problemas com muitas soluções é mais rápida que BFS pois pode encontrar uma solução expandindo só uma porção do espaço de estados. Deve ser evitada usar quando as árvores de procura têm uma profundidade máxima muito grande ou infinita. Tem complexidade temporal $O(b^m)$ e complexidade espacial O(bm), sendo b o fator de ramificação da árvore de procura e m a profundidade máxima. Não é completa nem ótima.
- Breadth-First Search(BFS): Começa por expandir a raiz primeiro e depois todos os seus sucessores e assim sucessivamente. Todos os nós na profundidade d são expandidos primeiro que os nós na profundidade d+1. Tem complexidade temporal e espacial $O(b^d)$, sendo d a profundidade da solução. É completa e ótima .
- Iterative Depth-First Search (IDFS): Busca iterativa que realiza uma busca limitada em profundidade aumentando o limite após cada iteração. É completa e ótima, tem complexidade temporal $O(b^d)$ e complexidade espacial O(bd).

Procura guiada:

- Greedy Search: Expande o nó cujo estado é julgado ser o mais próximo do estado final. Usa a distância de Manhattan para calcular a sua heurística. Não é completa nem ótima. Tem complexidade temporal e espacial $O(b^m)$.
- A* Search: Semelhante à estratégia Greedy mas para calcular a heurística para além da distância de Manhattan, tem em conta a profundidade do nó. É completa e ótima. Tem complexidade temporal e espacial $O(b^d)$.

4 – Descrição da Implementação

Implementamos todos estes algoritmos na linguagem C++, principalmente porque é uma linguagem simples em que todos nós estamos a vontade para usar e programar.

Relativamente as estruturas de dados optamos por um struct de um Node com os campos para a matriz, heuristica, profundidade e jogada. Utilizamos Listas de Nodes para guardar os nós a serem visitados. No caso do Greedy e do A* foi utilizada uma implementação de uma min-heap para guardar esses mesmos nodes.

Foi também implementada uma lista de nós já visitados com o objetivo de reduzir espaço de memória e eliminar processamento repetido.

```
typedef struct Node{
  int heuristic;
  int matrix[4][4];
  char move;
  int depth;
  Node *father;
}Node;
```

```
int inicial[4][4], final[4][4];
int depth_count  = 1;
minHeap hp = initMinHeap(0);
list <Node*> closed;
```

5 – Resultados:

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

Estratégia	Tempo (segundos)	Espaço (bytes)	Encontrou solução?	Profundidade
DFS	1,22	312767360	Sim	80
BFS	21,84	2777040	Sim	12
IDFS	0,03	5719920	Sim	12
Greedy	0,00	4244	Sim	12
A*	0,00	5544	Sim	12

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

Estratégia	Tempo (segundos)	Espaço (bytes)	Encontrou solução?	Profundidade
DFS	0,03	4837680	Sim	78
BFS	0,00	11680	Sim	4
IDFS	0,00	11840	Sim	4
Greedy	0,00	1144	Sim	4
A*	0,00	1144	Sim	4

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

Estratégia	Tempo (segundos)	Espaço (bytes)	Encontrou solução?	Profundidade
DFS	2,05	528288880	Sim	77
BFS	36,52	3675520	Sim	13
IDFS	0,07	15244960	Sim	13
Greedy	0	3696	Sim	13
A*	0	7656	Sim	13

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

Estratégia	Tempo (segundos)	Espaço (bytes)	Encontrou solução?	Profundidade
DFS	0,51	129989840	Sim	80
BFS	0,04	94800	Sim	8
IDFS	0,00	365440	Sim	8
Greedy	0,00	17512	Sim	18
A*	0,00	3168	Sim	8

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

Estratégia	Tempo (segundos)	Espaço (bytes)	Encontrou solução?	Profundidade
DFS	N/a	N/a	Não	N/a
BFS	0,04	94800	Sim	19
IDFS	5,26	1332895040	Sim	19
Greedy	0,00	16080	Sim	19
A*	0,00	7480	Sim	19

6 – Conclusões

Depois de todos os dados analisados em cada um dos algoritmos, podemos então com estes resultados concluir que:

- o **DFS** nunca retornou uma solução óptima, não resolve todos os casos, e é de todos o que consome mais memória.
- O **BFS** é a opção mais demorada, usa menos memória que o DFS e retorna sempre a solução óptima.
- O **IDFS** retorna sempre em todos os casos testados a solução óptima e com redução de uso de memória face ao DFS.
- O **greedy** retornou sempre solução mesmo que nem sempre a solução óptima.
- O **A*** retorna sempre a solução óptima tão ou mais rápido quanto o greedy.

Concluimos então que o A* é o melhor algoritmo em termos de solução óptima devido ao facto de retornar sempre a solução óptima, no melhor tempo possível e com “pouco” consumo de memória (relativamente aos outros).

7 - Referências Bibliográficas

[1] https://en.wikipedia.org/wiki/Search_problem
[2] http://www.dcc.fc.up.pt/~ines/aulas/1718/IA/t1/index.html
[3] Dutra I. Estratégias informadas de Busca; 2017
[4] Dutra I. Estratégias não informadas de Procura/Busca; 2017.