Relatório

1) Introdução

Um problema de busca/procura contém sempre um conjunto de estados, entre os quais um estado inicial e um estado final (objectivo), porém entre estes estados serão deduzidos outros novos estados intermédios, sempre que se avança na procura pela solução,

Todos estes estados são representados numa árvore de pesquisa ou grafo, onde cada nó é uma estrutura composta por pelo menos 5 campos:

- Estado;
- Nó pai;
- Função aplicada para gerar um nó;
- Profundidade do nó na árvore;
- Custo do caminho desde a raiz.

Existem vários métodos que podem ser utilizados em problemas de procura/busca:

-> Métodos de pesquisa não guiada:

- Largura;
- Custo uniforme;
- Profundidade;
- Profundidade iterativa;
- Bidireccional;

-> Métodos de pesquisa guiada:

- Greedy;
- A*;

2) Estratégias de Pesquisa

a) <u>Pesquisa não guiada</u>

- **Profundidade (DFS)**

A pesquisa em profundidade, começa por expandir os filhos do primeiro nó, seguindo a expansão dos nós do primeiro filho, e assim sucessivamente, ou seja, expande os nós filhos do nível seguinte, antes do anterior, e só retrocede se não existir mais filhos para serem descobertos. Logo se a expansão for infinita e a solução não estiver no caminho que a pesquisa está a seguir, esta pesquisa não irá encontrar solução.

A complexidade espacial é O(b x m), onde b é o fator de ramificação e m a profundidade máxima.

Tem como complexidade temporal $O(b^m)$.

- **Largura (BFS)**

Um algoritmo de pesquisa em largura expande e examina sistematicamente todos os nós. Esta pesquisa exaustiva, não passa para o nível seguinte, sem expandir todos os nós possíveis do nível actual.

Tem complexidade temporal e espacial igual, $O(b^d)$, onde b é o fator de ramificação e d a profundidade.

- **Iterativa Limitada em Profundidade (DFSi)**

A pesquisa iterativa limitada em profundidade, funciona como a DFS, porém é acrescentado um limite, inicialmente a 0, que vai sendo incrementado, e tem como função obrigar que expanda todos os nós da árvore até ser encontrada a solução, ou seja, quando o estado que estiver na profundidade limite, não é mais expandido, começando a busca de novo e incrementa 1 no limite até a solução ser encontrada.

Tem complexidade espacial igual à pesquisa em profundidade (O(b x m)).

b) Pesquisa guiada

- **Gulosa/greedy**

A pesquisa gulosa ou greedy é um método de pesquisa guiada que tenta minimizar o custo estimado para chegar à solução.

O nó que aparenta estar mais próximo do objectivo é aquele que é expandido em primeiro lugar. Utiliza-se uma função que calcula o custo estimado do caminho mínimo entre o estado actual e o objectivo pretendido.

É uma pesquisa similar à pesquisa em profundidade no sentido em que segue sempre a mesma direcção num caminho ao procurar a solução, mas pode, no entanto mudar de direcção consoante o custo dos nós ainda não visitados na árvore de pesquisa.

Este custo, no nosso trabalho, é o numero de posições fora do lugar, entre o estado atual e o objectivo.

- **A***

A pesquisa A*(A-star), segue o exemplo da pesquisa gulosa e tem como principal objectivo a minimização do custo total de um caminho. Como a Gulosa, o caminho segue a ordem da menor, porém além de o custo ser o numero de posições fora do lugar, é adicionado ao custo o nível de profundidade actual, isto para que haja uma relação entre o nível e o verdadeiro custo.

3) Descrição da implementação

No nosso trabalho utilizamos a linguagem C++, pois apesar de já estarmos familiarizados com a mesma do semestre passado, permite fazer uma coisa essencial para este trabalho, que o java não permite, como por exemplo um array booleano de tamanho 876543210. Além disto, o C++ assim como todas as linguagens contem muitas bibliotecas e APIs que nos ajudam bastante

Neste trabalho, usamos varias estruturas de dados para guardar a informação:

- o principal é uma estrutura, onde contem todos os dados relativos a um nó da árvore: matriz, peso, nível, as posições da casa vazia do jogo, e um apontador para a estrutura antecedente ao nó.

- uma stack da estrutura principal, pois permite é lista, onde se adiciona ao topo o ultimo nó encontrado e se retira do topo o nó a ser processado, permitindo assim desenvolver o DFS e DFSi.

- uma queue da estrutura principal, onde se adiciona ao fim da fila o ultimo nó expandido e se retira do inicio o nó a expandir, desenvolvendo o BFS.

- uma priority queue da estrutura principal para a busca Greedy e A*, pois o C++ permite colocar na priority queue por ordem de custo.

- um array de booleanos com o tamanho do maior caso possível de uma matriz, pois para melhorar o programa convertemos uma matriz num inteiro, para marcar e verificar mais rápido os nós já encontrados, impedindo a existência de ciclos.

Estas estruturas de dados são as ideais para a linguagem que usamos e para a resolução deste trabalho.

4) Resultados

Estratégia	Tempo (segundos)	Encontrou a solução ?	Profundidade /Custo
DFS	0.979s	SIM	21797 passos
BFS	0.116s	SIM	23 passos (ÓTIMA)
IDFS	1.837s	SIM	23 passos (ÓTIMA)
Gulosa	0.356s	SIM	47 passos
A*	0.358s	SIM	23 passos (ÓTIMA)

Tabela baseada nas seguintes matrizes:

Inicial

3	4	2
5	1	7
6		8

Final

1	2	3
8		4
7	6	5

5) Conclusão

Este trabalho permitiu-nos aprofundar os nossos conhecimentos relativamente ao funcionamento e implementação dos diferentes tipos de pesquisa, bem como quando os devemos de utilizar.

Com base no trabalho realizado e nos resultados finais, concluimos que a pesquisa em profundidade é o tipo de pesquisa menos eficiente e que necessita de um maior número de passos para atingir o objetivo.

Por outro lado, concluimos que os melhores algoritmos para a resolução deste problema são a pesquisa em largura (BFS), a pesquisa iterativa limitada em profundidade (IDFS) e a pesquisa A* pois ambas têm solução ótima, ou seja, conseguem atingir o objetivo em 23 passos.

TRABALHO REALIZADO POR:

- **André Viana Carvalho up201404878**
- **Miguel Ribeiro Correia up201405219**