Trabalho 1

Algoritmos de Procura (CC2006)

Índice:

Elementos do grupo

Nº mecanográfico	**Nome**
202105587	António Maria Araújo Pinto dos Santos
202203858	Eliandro Ricardo João Luís de Melo
202203859	João Manuel Cardoso Guedes

1. Introdução

O que é um problema de busca/procura?

Um problema de procura é definido por:

- Conjunto de estados
- Estado inicial
- Estado final ou objetivo
- Função Sucessor (Linha de um mapeamento de um estado a um conjunto de novos estados)
- Representação do espaço de estados: árvore de procura

Para resolver os problemas de procura são utilizados algoritmos e estratégias de procura. Estes são divididos entre a Procura Não Guiada e a Procura Guiada / Inteligente, que usa alguma heurística para orientar a sua procura, ao contrário da primeira..

Dentro da Procura Não Guiada usamos pesquisas em: profundidade, largura e iterativa limitada em profundidade. Quanto à Procura Guiada, utilizámos a procura gulosa e a procura A*.

2. Jogo dos 15 - Descrição

O problema de procura estudado foi o jogo dos 15. Este jogo é representado por uma matriz 4x4 onde há 15 células numeradas e uma célula em branco. Variações deste jogo podem conter parte de uma imagem em cada célula. O problema consiste em partir de uma configuração inicial embaralhada das células e chegar a uma configuração final com uma ordenação determinada de algarismos (no caso da matriz de números) ou de imagens (no caso da matriz onde as células representam partes de uma imagem). Os movimentos/operadores possíveis para se chegar de uma configuração a outra são: 1) mover a célula em branco para cima, 2) mover a célula em branco para baixo, 3) mover a célula em branco para a direita e 4) mover a célula em branco para a esquerda.

Para verificar se uma configuração inicial consegue chegar a uma configuração final, será necessário contar o número de trocas das peças contendo números no tabuleiro. Se ambos os estados inicial e final possuírem movimentos pares não será possível “interligar” ambos os estados.

```
if(!(((Invi % 2 == 0) ==
      (BRowi % 2 == 1)) ==
      ((Invf % 2 == 0) ==
      (BRowf % 2 == 1)))){

System.out.println("It is not possible to reach from the initial state to
the final state of the board, and vice versa.");
}
```

Tentamos transformar o tabuleiro inicial numa configuração de jogo igual ao tabuleiro final através de trocas. Se a configuração inicial for igual à final significa que alcançamos a solução, caso contrário nós verificamos o número de inversões das peças do tabuleiro e fazemos trocas entre o zero (espaço em branco) e as peças adjacentes até todas as peças terem zero inversões, ou seja, colocamos todos as peças em ordem até chegarmos à configuração final.

3. Estratégias de Procura

Nota: as definições são iguais às do JavaDocs do código.

a) Procura não guiada(Uninformed Search):

Pesquisa em Profundidade (Depth-first Search) - O algoritmo começa no nó raiz (selecionando algum nó arbitrário como o nó raiz no caso de um gráfico) e explora o máximo possível ao longo de cada ramo antes do backtracking.

Pesquisa em Largura - (Breadth-first Search) - O algoritmo começa na raiz da árvore e explora todos os nós na profundidade atual antes de passar para os nós no próximo nível de profundidade.

Procura iterativa limitada em profundidade (Iterative Deepening Search) - É ótimo como a busca em amplitude, mas usa muito menos memória. A cada iteração, ele visita os nós na árvore de pesquisa na mesma ordem que a pesquisa em profundidade, mas a ordem em que os nós são visitados pela primeira vez é efetivamente a da pesquisa em profundidade.

b) Procura guiada : A procura guiada usa alguma heurística para se orientar.

Heurística é a técnica utilizada para resolver um problema e que ignora se a solução pode ser provada e que está correta, mas que normalmente produz uma boa solução. (Heurísticas são destinadas a obter desempenho computacional ou simplicidade conceitual potencialmente ao custo de precisão.)

Procura Gulosa (Greedy Search) - segue a heurística de resolução de problemas para fazer a escolha do local ideal em cada estágio.

Procura A* (Search A*) - Encontra apenas o caminho mais curto de uma fonte especificada para um objetivo especificado, e não a árvore de caminho mais curto de uma fonte especificada para todos os objetivos possíveis.
Variações:

Somatório do número de peças fora do lugar -

Manhattan Distance - somatório das distâncias de cada peça ao seu lugar na configuração final.

critério	BP	BL	BILP	BG	BA*
tempo	b^m	b^d	b^d	b^m	b x d
espaço	b x m	b^d	b x d	b^m	b^d
otimalidade	não	sim	sim	não	sim
completude	não	sim	sim	não	sim

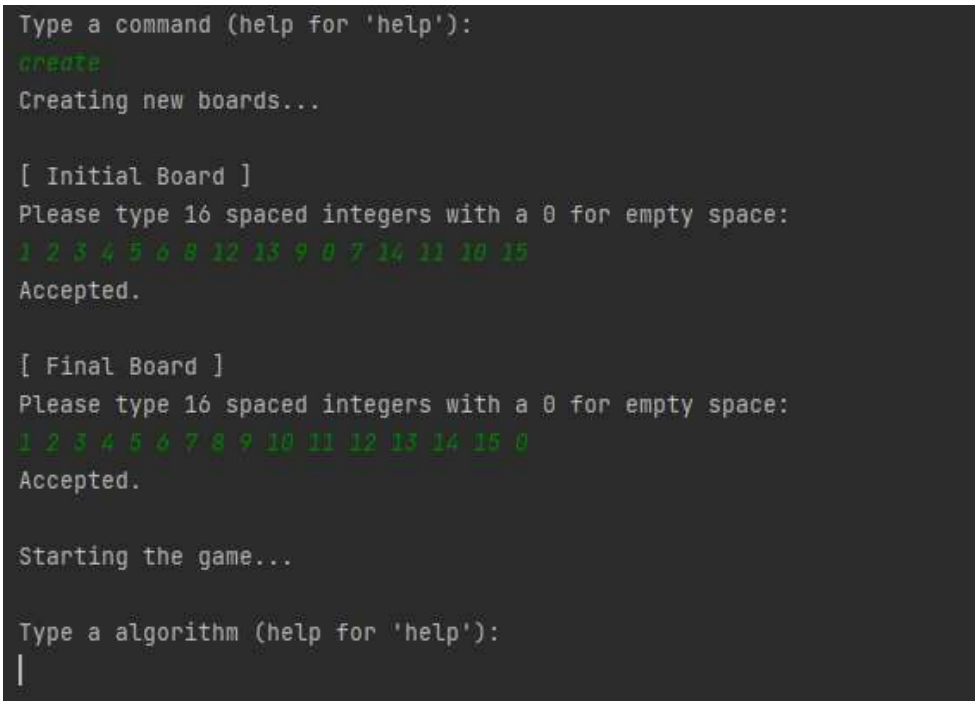
Nota: completude - se houver solução, garante que é encontrada.
otimalidade - se houver solução, garante que encontra a mais rasa primeiro.

Legenda:
b - é o fator de ramificação(branching factor)
d - é a profundidade da solução
m - profundidade máxima da árvore
l - profundidade do limite de pesquisa
BP - Busca em profundidade
BL - Busca em largura
BILP - Busca iterativa limitada em profundidade
BG - Busca gulosa
BA* - Busca A*

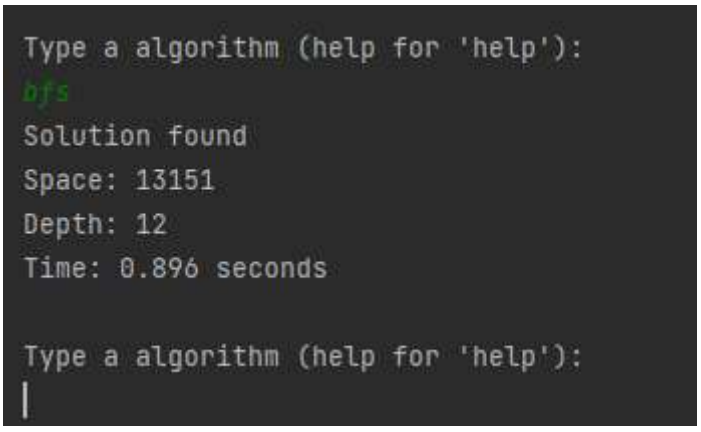
4. Descrição da Implementação

A linguagem utilizada foi o Java, escolhemos o Java acima de tudo porque estamos já bastante habituados ao seu funcionamento, também pela sua flexibilidade e segurança, pode ser utilizado em imensos sistemas operativos maximizando a compatibilidade do nosso trabalho.

Estruturamos o programa com um menu interativo para o utilizador:



Neste momento pode ser escolhida uma estratégia de jogo (algoritmo):



Para manipulação do tabuleiro foi utilizada uma matriz, cada jogo criado possui dois tabuleiros de estado inicial e final. Para consultar uma superior especificação do nosso código, foi gerado um **JavaDocs** presente na pasta **docs.**

5. Resultados obtidos

Resultados obtidos usando as configurações do jogo dadas no enunciado.

```
1 2 3 4 5 6 8 12 13 9 0 7 14 11 10 15
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 terminou)	(não terminou)	(não terminou)	(não terminou)
BFS	0.826	13151	Sim	12
IDFS	0.062	11081	Sim	12
Gulosa	0.007	31	Sim	12
A*	0.001	54	Sim	12

6. Comentários Finais e Conclusões

Os algoritmos de pesquisa são uma parte essencial da área de ciências da computação e desempenham um papel vital em várias aplicações, como localização de rotas, jogos e IA.

Existem muitos algoritmos de busca disponíveis, cada um com seus pontos fortes e fracos, e escolher o algoritmo apropriado para um determinado problema é crucial para alcançar o desempenho ideal, este trabalho foi essencial para obtermos essa conclusão.

Alguns algoritmos, como BFS e DFS, são diretos e fáceis de implementar. No entanto, eles podem nem sempre ser a melhor escolha para problemas de grande escala, pois podem exigir uma quantidade considerável de memória e podem não garantir a localização da solução ideal.

Neste trabalho visualizamos que algoritmos de busca mais complexos, como A*, podem fornecer melhor desempenho e garantir a localização da solução ideal. No entanto, são mais difíceis de implementar e podem exigir mais recursos do computador para processamento.

Em suma, os algoritmos de busca continuarão a evoluir e desempenharão um papel significativo no avanço do campo da ciência da computação.

7. Referências Bibliográficas

https://github.com/FindingBits/POO-Trabalho-2122

https://pt.wikipedia.org/

Google

BOOK: *“AI: a Modern Approach”, fourth edition*

Research: https://chat.openai.com/chat

Slides da Professora Inês Dutra.

https://mathworld.wolfram.com/15Puzzle.html

https://personal.math.ubc.ca/~cass/courses/m308-02b/projects/grant/fifteen.html

https://s3.eu-central-1.amazonaws.com/unitbv.horatiuvlad.com/facultate/inteligenta_artificiala/2015/TilesSolvability.html

http://kociemba.org/themen/fifteen/fifteensolver.html

http://puzzles.nigelcoldwell.co.uk/sixteen.htm