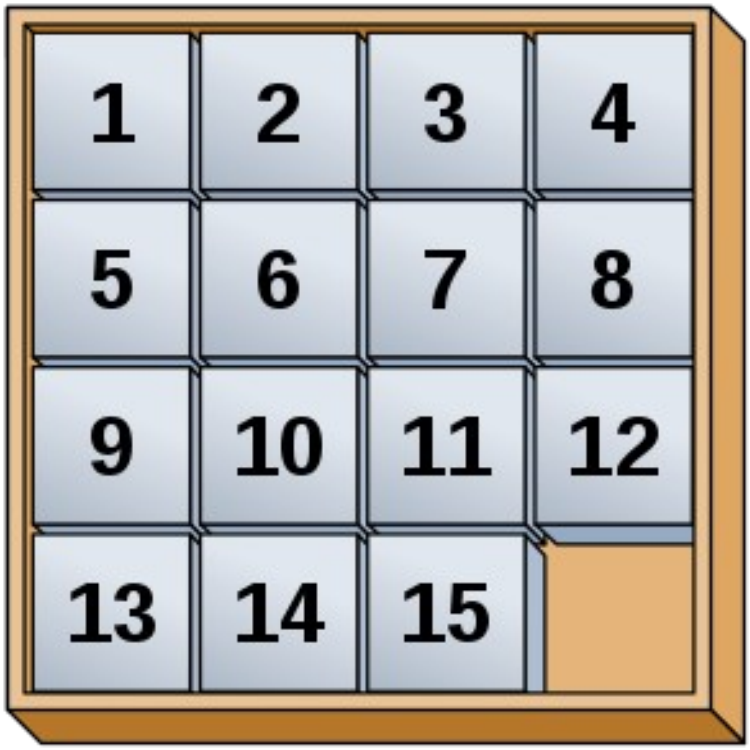
Faculdade de Ciências da Universidade do Porto
Inteligência Artificial CC2006

# Jogo dos 15
# Relatório de entrega




Docente: Inês Dutra
Ano lectivo: 2017/2018
André Pereira up201407074
Pedro Almeida up201403403
Ana Dias up201303164

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

# Índice

# Introdução

No problema iremos estudar estratégias de procura, para cada método teremos de analisar o tempo que demora a resolver o problema e o espaço da memória que utiliza, que é o número de nós que cria. Através destes resultados podemos concluir qual o algoritmo mais eficaz.
Podemos resolver este jogo usando duas estratégias diferentes, procura não guiada ou procura guiada. Dentro da procura não guiada podemos aplicar: busca em profundidade (DFS), busca em largura (BFS) ou busca iterativa limitada em profundidade. Na procura guiada (que usa heurística) temos a busca gulosa (Greedy) e a busca A*.

# Jogo dos 15

O jogo dos 15 é representado por uma grelha 4 por 4 onde existem 15 células que estão numeradas de 1 a 15 e uma célula que se encontra em branco. O objectivo é ordenar as células por ordem crescente e a célula em branco ficar no fim ou início (dependendo da solução final que pretendemos). Inicialmente a grelha está baralhada. Os movimentos possíveis sao deslocar a célula em branco para cima, para baixo, para a esquerda ou para a direita.
Existe a possibilidade de a grelha inicial não ter solução devido ao número de inversões e do formato da grelha.

# Estratégias de Procura

Antes de aplicarmos as estratégias de procura temos de analisar se a grelha dada inicialmente é solúvel. Primeiro colocamos a grelha em forma de vector, de seguida calcula-se o número de inversões que se tem de fazer para o vector ficar ordenado. Depois de ter este número existem duas possibilidades, o caso a largura ser par ou ser ímpar. Como no jogo dos 15 é uma tabela de largura par, para haver solução o espaço em branco, inicialmente tem de estar numa linha ímpar a contar de baixo da grelha e o número de inversões tem de ser par. Caso contrário o programa não consegue encontrar a solução.

## ➔ Procura não guiada

Os algoritmos de procura não guiada, também referida como procura cega, apenas têm acesso à grelha inicial aquando a definição do problema. Isto é, só conseguem gerar sucessores e verificar se o estado actual corresponde ou não à solução final pretendida.

**Busca em profundidade (DFS)**
O algoritmo desta busca é caracterizado por expandir sempre o nó do nível mais profundo da árvore de procura. Começa no nó raíz e explora o mais possível em termos de profundidade, repetindo a tarefa até encontrar a solução pretendida ou não haver mais filhos do último nó que inseriu. No caso de não encontrar a solução final, volta ao nó mais profundo que ainda não foi explorado. Esta procura usa a estrutura de dados LIFO(last-in, first-out).
Em termos de eficácia, este algoritmo não é o mais apropriado pois se a árvora de procura tiver uma profundidade infinita o algoritmo pode não encontrar a solução pretendida. Sendo assim, pode-se tirar mais partido deste algoritmo se houverem múltiplas soluções para o problema. Sendo assim a busca em profundidade tem complexidade temporal igual à procura em largura, O(b^m), mas a nível espacial, este algoritmo tem complexidade O(b*m), sendo b o número de nós sucessores de cada nó e m a profundidade máxima que a árvore pode ter.

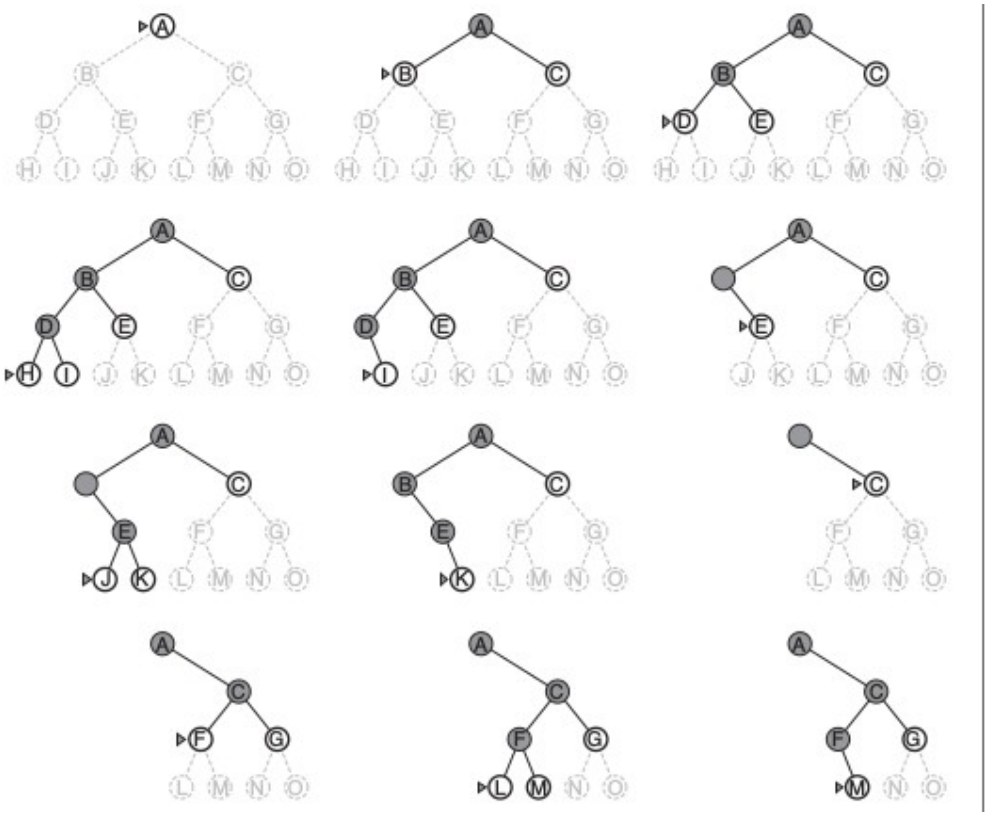


*Imagem 1: Demonstração do funcionamento da árvore de pesquisa em profundidade*

**Busca em largura (BFS)**

A busca em largura inicia no nó raíz e expande todos os nós vizinhos (sucessores). De seguida, repete o processo para os seus sucessores e assim sucessivamente. Logo, todos os nós são expandidos numa dada profundidade da árvore de procura, antes dos do nível seguinte de profundidade serem explorados, até ser encontrada a solução pretendida. Assim, o algoritmo tem de garantir que o mesmo vértice não é visitado mais que uma vez. Para isto, este algoritmo utiliza uma estrutura de dados fila, assegurando a ordem de chegada dos vértices. Resumindo, todos os nós do mesmo nível são expandidos, antes de passar para o próximo nível de profundidade, de modo que encontre a melhor solução de todas, sendo esta a mais próxima do nó raíz.

Nesta busca, o nó raíz da árvore gera b nós sucessores no primeiro nível, que geram cada um b nós adicionais no segundo nível de profundidade, o que resulta em b^2 nós e por assim adiante. Assim, no pior dos casos, se forem gerados d níveis de profundidade a complexidade desta busca será O(b^d).

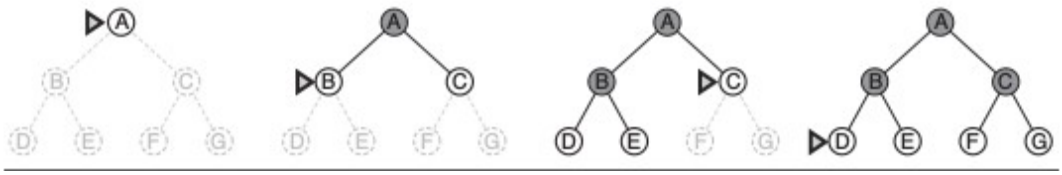


*Imagem 2: Demonstração do funcionamento da árvore de pesquisa em largura*

**Busca iterativa limitada em profundidade**
Esta procura é semelhante ao algoritmo da procura em profundidade, com a distinção de possuir uma característica que impõe um limite na profundidade máxima que a árvore de procura pode atingir. Isto é, os nós são percorridos e expandidos em profundidade até atingirem o limite, passando assim a expandir os restantes nós pelos últimos gerados. Este algoritmo garante que a menor solução possível é primeiro encontrada. Tem a vantagem de não ser infinito, em relação ao algoritmo de procura em profundidade. No entanto, é necessário ter em atenção qual o limite da profundidade imposto, pois se for muito pequeno a busca iterativa limitada em profundidade poderá não encontrar a solução.
Tem complexidade temporal O(b^d) e complexidade espacial O(b*d), sendo b o número de sucessores e d a profundidade até à solução. Sendo assim este algoritmo é preferencialmente utilizado para procuras em que a profundidade até à solução não é conhecida inicialmente.

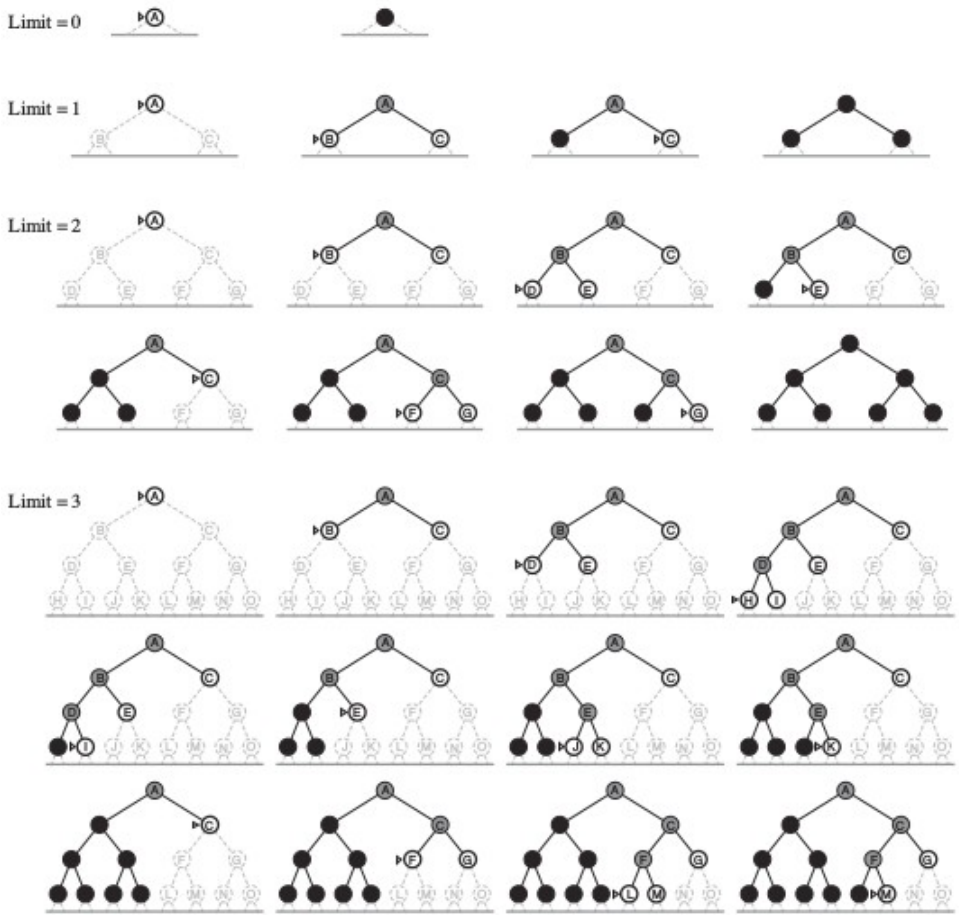


*Imagem 3: Demonstração da pesquisa iterativa limitada em profundidade*

➔ Procura guiada

Os algoritmos de procura guiada distiguem-se pois inicialmente calculam o custo da expansão, pelo que são mais vantajosos do que uma procura não guiada. O seu funcionamento base é a seleção de um nó para expansão, baseado numa função de evolução. Esta função tem como objectivo determinar o custo estimado da expansão, pelo o que permite que o nó escolhido para ser expandido

seja o que tem menor evolução. A maoir parte destes algoritmos incluiem uma componente designada por função heurística.

**Heurística**
Um algortimo heurístico tem como função encontrar a solução ou soluções pretendidas, entre todas as possíveis. Geralmente, encontra a solução mais eficazmente, mais rapidamente e com menor gasto do espaço na memória. Logo, dá um custo estimado do caminho mais eficiente do nó até ao objectivo final.

**Gulosa (Greedy)**
Este algoritmo utiliza a avaliação heurística do nó actual para optar pelo mais eficaz para o problema específico a ser tratado. Sendo assim, é semelhante à busca em profundidade pois tenta expandir o nó mais próximo do final. Deste modo, tem complexidade espacial e temporal O(b^m), tendo b o número de nós expandidos e m a profundidade da árvore de pesquisa.

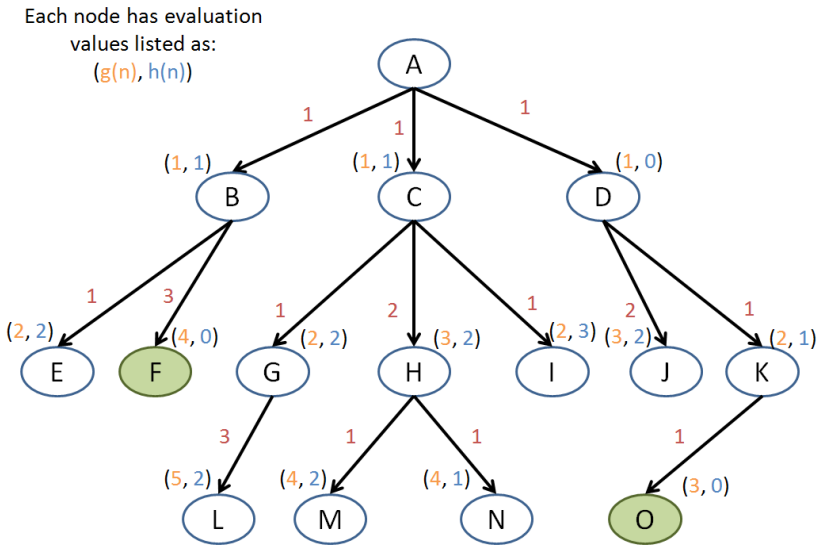



*Imagem 4: Demonstração do funcionamento do algoritmo de pesquisa gulosa (Greedy)*

**Busca A***
Esta busca utiliza a função g(n), que calcula o custo dos nós, isto é, o custo entre o estado inicial e o actual. A* também calcula a função heurística h(n), que determina o custo entre o estado actual e o final. A equação final é dada por f(n)=g(n)+h(n), retornado assim o custo entre o estado inicial e o final, devendo o algoritmo, depois de calculada esta diferença, escolher o menor valor.
Com este método, o algoritmo deduz a solução óptima, combinando a busca gulosa com a busca de custo uniforme.
Tem complexidade temporal e espacial O(log (h*(n))), onde h*(n) é o custo real entre n e o estado inicial.

## Descrição da Implementação

A linguagem utilizada neste trabalho foi o Java, visto que é uma linguagem mais familiar para os membros do grupo e também 'user-friendly'. Em relação a este problema, existe a vantagem que o Java trabalha sobre uma maquina virtual(JVM), dando assim portabilidade ao programa, podendo ser executado em qualquer máquina. É também uma linguagem orientada a objetos, facilitando a implementação de estruturas de dados e algoritmos necessários a este trabalho.
Estruturas de dados utilizadas no código foram:

- Stack
- Matriz(Nó)
- PriorityQueue
- Queue

## Resultados

**1ªTeste**
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 (MB) | Encontrou a solução? | Profundidade/Custo |
|---|---|---|---|---|
| DFS | 0,122 | 15,386 | Sim | 22 |
| BFS | 0,005 | 2,485 | Sim | 4 |
| IDFS | 0,003 | 2,485 | Sim | 4 |
| Gulosa | 0,001 | 1,865 | Sim | 4 |
| A* | 0,001 | 1,865 | Sim | 4 |

**2º Teste**
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 (MB) | Encontrou a solução? | Profundidade/Custo |
|---|---|---|---|---|
| DFS | 6,873 | 71,705 | Sim | 21 |
| BFS | 0,124 | 33,138 | Sim | 13 |
| IDFS | 0,225 | 45,176 | Sim | 13 |
| Gulosa | 0,002 | 1,865 | Sim | 13 |
| A* | 0,003 | 2,486 | Sim | 13 |

**3º Teste**
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 (MB) | Encontrou a solução? | Profundidade/Custo |
|---|---|---|---|---|
| DFS | 1,903 | 251,773 | Sim | 20 |
| BFS | 0,015 | 8,065 | Sim | 8 |
| IDFS | 0,028 | 19,225 | Sim | 8 |
| Gulosa | 0,002 | 1,865 | Sim | 8 |
| A* | 0,002 | 1,865 | Sim | 8 |

## Conclusão

Após analisarmos os resuldados dos testes realizados, concluimos que os algortimos mais eficazes a nível de complexidade, espacial e temporal, os melhores são os de procura guiada, como seria de esperar. De seguida é o de pesquisa em largura, depois o de busca iterativa limitada em profundidade e por fim o de pesquisa em profundidade.
Com a realização deste trabalho conseguimos perceber melhor os algoritmos de procura, também como e quando utilizá-los. Devido à sua aplicação neste jogo, conseguimos entender a usa utilização no quotidiano e num caso mais prático.

## Referências bibliográficas

**Apontamentos fornecidos pela professora:**
http://www.dcc.fc.up.pt/~ines/aulas/1718/IA/t1/index.html
https://www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/TilesSolvability.html
http://www.math.ubc.ca/~cass/courses/m308-02b/projects/grant/fifteen.html
http://kociemba.org/fifteen/fifteensolver.html
**Livros:**
Artificial Intelligence: a Modern Approach, by Stuart Russell and Peter Norvig, 3rd edition, Prentice Hall
Artificial Intelligence: a new synthesis, by Nils Nilsson
**Sites:**
http://web.cs.ucla.edu/~forns/classes/winter-2016/cs-161/midterm-practice.html
http://students.ceid.upatras.gr/~papagel/project/contents.htm
https://en.wikipedia.org/wiki/Heuristic_(computer_science)
https://pt.wikipedia.org/wiki/O_jogo_do_15