**RELATÓRIO**

**Inteligência Artificial**

****

****

**JOGO DOS 15**

**André Cirne – 201505860**

**José Sousa - 2015**

Índice

[Introdução 3](#_Toc475793706)

[Estratégias de Procura 3](#_Toc475793707)

[Procura não guiada 3](#_Toc475793708)

[Profundidade 3](#_Toc475793709)

[Largura 3](#_Toc475793710)

[Busca Iterativa Limitada em Profundidade 3](#_Toc475793711)

[Procura guiada 3](#_Toc475793712)

[Gulosa 3](#_Toc475793713)

[Busca A\* 3](#_Toc475793714)

[Descrição da Implementação 3](#_Toc475793715)

[Linguagem utilizada 3](#_Toc475793716)

[Estruturas de dados utilizadas 4](#_Toc475793717)

[LinkedList 4](#_Toc475793718)

[PriorityQueue 4](#_Toc475793719)

[HashSet e HashMap 4](#_Toc475793720)

[Considerações de implementação 4](#_Toc475793721)

[Resultados 4](#_Toc475793722)

[Conclusão 5](#_Toc475793723)

[Referências 6](#_Toc475793724)

# Introdução

# Estratégias de Procura

## **Procura não guiada**

### Profundidade

### Largura

### Busca Iterativa Limitada em Profundidade

## **Procura guiada**

As procuras guiadas, ao contrário dos métodos anteriormente apresentados, utilizam conhecimentos específicos ao problema a abordar, com o objetivo de ao longo da pesquisa avançar sempre por um caminho que seja mais próximo da solução (1), minimizando assim o custo estimado de chegar à solução.

Ao utilizar este tipo métodos de pesquisa tentamos atingir um numero menor de nós gerados e um tempo menor até chegar à solução.

### Gulosa

A procura gulosa é um tipo de procura guiada, que tenta minimizar o custo da solução, dependendo da função heurística utilizada podemos obter diferentes graus de completude.

Na implementação deste algoritmo usamos como função heurística, a distancia de Manhattan. O calculo desta distancia é efetuado através da soma da distancia horizontal e vertical (2), de cada peça na posição atual em relação à posição final que pretendemos.

A utilização desta heurística, leva a que este algoritmo seja incompleto e não ótimo.

### Busca A\*

A busca A\*, é a tentativa de combinar dois algoritmos a busca gulosa, que tenta estimar o custo para atingir a solução ótima, que na nossa abordagem usou distância de Manhattan, mas que tem como ponto fraco os anteriormente referidos e o algoritmo Dijkstra, algoritmo que descobre o caminho mais curto para um determinado nó, usando a distancia do nó atual à raiz como heurística. O *Dijkstra* é ótimo e completo, no entanto, ineficiente já que todos os nós têm de ser visitados. (3) (4)

Devido à utilização da função de heurística acima descrita, consegue ser ótimo, completo e eficiente.

# Descrição da Implementação

## **Linguagem utilizada**

Desde do inicio do problema devido à forma como se encontra formulado o problema, a existência de tabuleiros, que consideramos importante que a linguagem utilizasse o paradigma de orientação ao objeto. Inicialmente começamos por tentar implementar o a solução ao problema dos jogo dos 15 usando *Python*, já que é uma linguagem multi-paradigma e assim poderíamos tirar também proveito de uma abordagem mais funcional. No entanto chegamos a conclusão de dois problemas, por um lado Python é uma linguagem intrepetada “pouco lenta”, por isso acabamos por optar por Java, continua a ser uma linguagem orientada a objeto, não interpretada, e que apresenta um bom equilíbrio entre rapidez de código e alto nível.

## **Estruturas de dados utilizadas**

### LinkedList

Como não se sabe à partida qual será o número de elementos que vamos ter de guardar, esta é a estrutura de dado que consegue guardar um numero de elementos que só se encontra limitado pela memória física da máquina onde está a correr e que além disso é muito dinâmica, já que permite utilizar qualquer tipo de tipos de dados para formar uma LinkedList. As operações de remoção e de adição no inicio da estrutura com tempo contante, contribuindo assim para um algoritmo mais rápido.

### PriorityQueue

A PriorityQueue em java encontra-se construída com base na Queue, que por sua vez é construída com base na LinkedList, logo todas as características em relação a mais valias desta estrutura de dados, também se aplicam à PriorityQueue. Esta estrutura permite manter uma lista de forma automaticamente ordenada e garantido uma complexidade na ordenação/adição de (5),logo foi utilizada na busca A\* e busca Gulosa onde é essencial esta tarefa.

### HashSet e HashMap

A utilização de HashSet’s e HashMap’s , na nossa implementação deveu-se a dois factos. Por um lado, a complexidade de verificar se um determinado nó já se encontra na estrutura é de complexidade linear, o que é importante nos algoritmos onde é necessário verificar nós repetidos, além disto como não soa utilizados os números em si, mas sim as suas hashs, possibilita um uso menor de memória.

## Considerações de implementação

“”Falar o pq de utilizar as vezes hash map outras hash set””

# Resultados



# Conclusão

# Referências

x

|  |  |
| --- | --- |
| 1. | Wikipedia. [Online].; 2017. Available from: <https://en.wikipedia.org/wiki/Heuristic_(computer_science)#Newell_and_Simon:_heuristic_search_hypothesis>. |
| 2. | Black PE. Manhattan distance. [Online].; 2006 [cited 2017 Fevereiro 25. Available from: <https://xlinux.nist.gov/dads/HTML/manhattanDistance.html>. |
| 3. | Ribeiro P. DAA1617. [Online].; 2016 [cited 2017 Fevereiro 25. Available from: <http://www.dcc.fc.up.pt/~pribeiro/aulas/daa1617/slides/9_distancias_11122016.pdf>. |
| 4. | Eranki R. Pathfinding using A\* (A-Star). [Online].; 2002 [cited 2017 Fevereiro 25. Available from: <http://web.mit.edu/eranki/www/tutorials/search/>. |
| 5. | Oracle. PriorityQueue. [Online].; 2016 [cited 2017 Fevereiro 25. Available from: <http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html>. |

x