



# Projeto Prático

---

## Métodos de Procura

Realizado em:  
ESTGV  
Por  
António Gil Rodrigues, nº 16287  
Jorge Leitão, nº 15934

Viseu, 2019

Instituto Politécnico de Viseu  
Escola Superior de Tecnologia e Gestão de Viseu  
Departamento de Informática

Inteligência Artificial  
Curso de Engenharia Informática

**Projeto Prático**  
Métodos de Procura

Realizado em:  
ESTGV  
Por  
António Gil Rodrigues, nº 16287  
Jorge Leitão, nº 15934

Ano Letivo 2018/2019

Viseu, 2019

# RESUMO

---

O presente relatório serve de suporte escrito ao Projeto Prático desenvolvido na Unidade Curricular de Inteligência Artificial. O objetivo é desenvolver um software em Python capaz de desenhar o caminho entre duas cidades, recorrendo a diferentes métodos de procura. Começamos por procura cega: em profundidade primeiro (*depth-first*) e custo uniforme (*uniform cost*). A seguir procura heurística: procura sôfrega (*greedy search*) e A\*.

**Palavras chave:** A-Star, Depth-First, Greedy, Métodos de procura, Uniform Cost

# ÍNDICE

---

<i>1. Introdução</i>	<i>2</i>
<i>2. Descrição Geral e Interface com o utilizador</i>	<i>3</i>
<i>3. Métodos de procura: Em profundidade primeiro</i>	<i>6</i>
<i>4. Custo Uniforme</i>	<i>9</i>
<i>5. Procura sôfrega</i>	<i>11</i>
<i>6. A* (a-Star)</i>	<i>12</i>
<i>Conclusões</i>	<i>13</i>
<i>7. Referências</i>	<i>14</i>
<i>8. Bibliografia</i>	<i>15</i>

## Índice de Figuras

Figura 1 – Menu inicial do programa	3
Figura 2 – Apresentação dos resultados	3
Figura 3 – Pedido de input ao utilizador	4
Figura 4 – Estrutura dict do grafo	4
Figura 5 – Dicionário com as distâncias de cada cidade até Faro	5
Figura 6 – Menu parcial, evidenciando as variáveis que devem ser reiniciadas para a procura em profundidade	7
Figura 7 – Implementação, de forma recursiva	8
Figura 8 – Aplicação de custo uniforme entre Viseu e Faro, evidenciando o elevado número de iterações.	10
Figura 9 – Pesquisa sôfrega	11
Figura 10 – Algoritmo A*	12

# 1. Introdução

---

O propósito do presente trabalho prático é desenvolver um software em Python capaz de desenhar o caminho entre duas cidades, recorrendo a diferentes métodos de procura.

Desta forma, o objetivo é expor os caminhos selecionados pela aplicação, mas também colocar em evidência as iterações que acontecem em cada algoritmo.

Começamos por expor a interface do programa com o utilizador. De seguida, faremos a revisão do algoritmo utilizado, fazendo também uma avaliação teórica do mesmo e os resultados esperados.

Por fim, vamos tirar ilações sobre os resultados e avaliar se vão ao encontro dos esperados teoricamente.

## 2. Descrição Geral e Interface com o utilizador

---

A aplicação é de linha de comandos, sendo a interação com o utilizador efetuada através de pergunta e resposta.

```
***** TRABALHO PRÁTICO FINAL 2019 *****

----- Disciplina Inteligência Artificial -----

1. Método procura profundidade primeiro
2. Método procura Custo Uniforme
3. Método procura Sôfrega
4. Método procura A*
5. Sair do script

*****

Selecione uma opção [1-4]:
```

Figura 1 – Menu inicial do programa

É questionado o utilizador qual método de procura deseja experimentar. Mediante a resposta, são apresentadas as iterações utilizadas no método, alcançando o resultado esperado. Damos abaixo o exemplo do método 1. Profundidade Primeiro.

```
Iteracao 0:
Aveiro

Iteracao 1:
Aveiro,Porto

Iteracao 2:
Aveiro,Porto,Viana do Castelo

Iteracao 3:
Aveiro,Porto,Viana do Castelo,Braga

Iteracao 4:
Aveiro,Porto,Viana do Castelo,Braga,Vila Real

Iteracao 5:
Aveiro,Porto,Viana do Castelo,Braga,Vila Real,Viseu

Iteracao 6:
Aveiro,Porto,Viana do Castelo,Braga,Vila Real,Viseu,Coimbra

Iteracao 7:
Aveiro,Porto,Viana do Castelo,Braga,Vila Real,Viseu,Coimbra,Leiria
```

Figura 2 – Apresentação dos resultados

É questionado ao utilizador qual a cidade de origem e de destino, nos métodos não informados (Figura 3). Já nos dois métodos heurísticos, só é pedida a cidade de origem, pois o programa apenas suporta a pesquisa com destino Faro.

O programa suporta somente correspondências exatas, e é *case-sensitive* (apenas primeiras letras devem ser maiúsculas).

```
Selecione uma opção [1-4]: 1
Insira a Origem:
Viseu
Insira Destino:
Lisboa
```

Figura 3 – Pedido de input ao utilizador

O grafo que representa as cidades foi traduzido em código na forma de uma estrutura do tipo Dicionário (*dict*), onde as cidades de origem são as chaves (*keys*) e as cidades de destino são os valores (*values*).

Os caminhos são reversíveis, como tal, colocámos as cidades repetidas na origem e no destino (o *dict* é simétrico).

Por sua vez, as cidades encontram-se mapeadas com a distância à cidade de origem, também através de um dict em que as cidades são as *keys* e as distâncias são os *values* (Figura 4).

```
#Dicionário com custos entre cidades
GrafocomCustosdict = {
    'Aveiro': {'Porto':68, 'Viseu':95, 'Coimbra':68, 'Leiria':115},
    'Braga': {'Viana do Castelo':48, 'Vila Real':106, 'Porto':53},
    'Bragança': {'Vila Real':137, 'Guarda':202},
    'Beja': {'Évora':78, 'Faro':152, 'Setúbal':142},
    'Castelo Branco': {'Coimbra':159, 'Guarda':106, 'Portalegre':80, 'Évora':203},
    'Coimbra': {'Viseu':96, 'Leiria':67, 'Aveiro':68, 'Castelo Branco':159},
    'Évora': {'Lisboa':150, 'Santarém':117, 'Portalegre':131, 'Setúbal':103, 'Beja':78, 'Castelo Branco':203},
    'Faro': {'Setúbal':249, 'Lisboa':299, 'Beja':152},
    'Guarda': {'Vila Real':157, 'Viseu':85, 'Bragança':202, 'Castelo Branco':106},
    'Leiria': {'Lisboa':129, 'Santarém':70, 'Aveiro':115, 'Coimbra':67},
    'Lisboa': {'Santarém':78, 'Setúbal':50, 'Faro':299, 'Leiria':129, 'Évora':150},
    'Portalegre': {'Castelo Branco':80, 'Évora':131},
    'Porto': {'Aveiro':68, 'Viana do Castelo':71, 'Vila Real':116, 'Viseu':133, 'Braga':53},
    'Santarém': {'Évora':117, 'Lisboa':78, 'Leiria':70},
    'Setúbal': {'Lisboa':50, 'Évora':103, 'Beja':142, 'Faro':249},
    'Viana do Castelo': {'Braga':48, 'Porto':71},
    'Vila Real': {'Viseu':110, 'Braga':106, 'Bragança':137, 'Guarda':157, 'Porto':116},
    'Viseu': {'Aveiro':95, 'Coimbra':96, 'Porto':133, 'Vila Real':110, 'Guarda':85}
}
```

Figura 4 – Estrutura dict do grafo

---

Para as pesquisas heurísticas, é incluído outro *dict*, que apenas mapeia cada cidade com a sua distância em relação a Faro. Como tal, ambos os grafos devem ser utilizados em simultâneo nas pesquisas heurísticas (Figura 5).

```
26 #Dicionário com custos entre as cidades e Faro
27 distanciasFaro = {
28     'Aveiro':366,
29     'Braga':454,
30     'Bragança':487,
31     'Beja':99,
32     'Castelo Branco':280,
33     'Coimbra':319,
34     'Évora':157,
35     'Faro':0,
36     'Guarda':352,
37     'Leiria':278,
38     'Lisboa':195,
39     'Portalegre':228,
40     'Porto':418,
41     'Santarém':231,
42     'Setúbal':168,
43     'Viana do Castelo':473,
44     'Vila Real':429,
45     'Viseu':363
46 }
47
```

**Figura 5 – Dicionário com as distâncias de cada cidade até Faro**



### 3. Métodos de procura: Em profundidade primeiro

---

O método de procura “em profundidade primeiro” (*depth-first search*) é um método não informado (sem informação do custo do caminho do estado atual para o objetivo) aplicável a estruturas do tipo árvore ou grafo (como é o nosso caso). É um algoritmo recursivo.

O algoritmo inicia num nó origem (no nosso caso, a cidade de origem) e explora o máximo possível os nós seguintes (até chegar a um “beco sem saída”), voltando de seguida para trás para explorar os caminhos alternativos (*backtracking*).

Como este algoritmo tem que explorar a árvore/grafos toda, em árvores de pesquisa muito profundas pode não ser muito eficaz, podendo até ficar preso em ciclos. No entanto, como no problema em estudo não existem muitas cidades, pode ser implementado.[1][2][3]

Também uma desvantagem deste método é não ter em consideração os “pesos” do grafo, ou seja, no caso em estudo, a distância entre as cidades.

O algoritmo foi implementado com uma função recursiva, recorrendo a algumas variáveis que são reiniciadas antes de chamar a função.

```

def get_menu_choice():
    global encontrei, contador
    def print_menu():
        print("\n")
        print("\n")
        print(31 * "*", " TRABALHO PRÁTICO FINAL 2019", 31 * "*")
        print("\n")
        print(29 * "-", " Disciplina Inteligência Artificial", 25 * "-")
        print("\n")
        print("1. Método procura profundidade primeiro \n ")
        print("2. Método procura Custo Uniforme \n")
        print("3. Método procura Sôfrega \n")
        print("4. Método procura A* \n")
        print("5. Sair do script \n")
        print(90 * "*")
        print("\n")

    loop = True
    visitado = []

    while loop:
        origem=''
        destino=''
        print_menu()
        choice = input("Selecione uma opção [1-4]: ")

        if choice == '1':
            encontrei = 0
            contador = 0
            print("\nInsira a Origem: ")
            origem = input()
            print("\nInsira Destino: ")
            destino = input()
            visitado=[]
            profundidadePrimeiroRecursiva(GrafoComCustosdict,origem,destino,visitado)

            input("\n ENTER para continuar ...")

```

Figura 6 – Menu parcial, evidenciando as variáveis que devem ser reiniciadas para a procura em profundidade

---

```
def profundidadePrimeiroRecursiva(Grafo,no,destino,visitado):
    global encontrei, contador

    if(no not in Grafo or destino not in Grafo):
        print('\nCidade Origem ou Destino não existe ')
        return

    if encontrei == 1:
        return

    if(no == destino):
        visitado.append(no)
        print("\nIteracao " + str(contador) + ": ")
        print(", ".join(visitado))
        encontrei = 1
        return visitado

    if no not in visitado:
        visitado.append(no)
        print("\nIteracao " + str(contador) + ": ")
        print(", ".join(visitado))
        contador += 1
        for n in Grafo[no]:
            profundidadePrimeiroRecursiva(Grafo,n,destino,visitado)
```

Figura 7 – Implementação, de forma recursiva

## 4. Custo Uniforme

---

O método de procura custo uniforme (*uniform cost*) é considerado o melhor algoritmo de pesquisa que não envolve uso de heurísticas.

Este método analisa, um a um, todos os caminhos disponíveis, levando em consideração, tanto caminhos com baixo número de nós, mas também os caminhos com muitas “paragens”. Analisa os pesos de cada ligação e compara, de forma a obter o custo mais baixo.

A grande desvantagem do algoritmo é a morosidade e elevados custos de memória, uma vez que a quantidade de caminhos a analisar aumenta exponencialmente, quando aumenta o número de nós.[4][5][6]

Aplicando no nosso caso concreto, vamos conseguir o caminho de menor distância entre a origem e o destino.

Implementámos este algoritmo recorrendo a uma *PriorityQueue*, que pertence à biblioteca *queue* do python. Esta fila tem a vantagem de ordenar automaticamente os resultados, ficando o de menor custo em índice zero.

A origem é inserida na queue, sendo adicionados os caminhos em estudo e “retirado”, após cada ciclo, o caminho que se encontra no índice zero. O ponto de terminação é quando a queue ficar vazia e tivermos obtido a nossa cidade destino – aí, obtivemos o nosso caminho ótimo.

```
Linha de comandos - python trabalhofinal.py

Iteração 6472:
(580, ['Viseu', 'Coimbra', 'Aveiro', 'Porto', 'Braga', 'Porto', 'Braga', 'Porto', 'Aveiro', 'Porto'])

Iteração 6473:
(580, ['Viseu', 'Coimbra', 'Aveiro', 'Porto', 'Vila Real', 'Porto', 'Vila Real'])

Iteração 6474:
(580, ['Viseu', 'Coimbra', 'Leiria', 'Aveiro', 'Leiria', 'Santarém', 'Évora'])

Iteração 6475:
(580, ['Viseu', 'Coimbra', 'Leiria', 'Coimbra', 'Castelo Branco', 'Guarda', 'Viseu'])

Iteração 6476:
(580, ['Viseu', 'Coimbra', 'Leiria', 'Lisboa', 'Santarém', 'Leiria', 'Santarém', 'Leiria'])

Iteração 6477:
(580, ['Viseu', 'Coimbra', 'Leiria', 'Santarém', 'Leiria', 'Lisboa', 'Santarém', 'Leiria'])

Iteração 6478:
(580, ['Viseu', 'Coimbra', 'Leiria', 'Santarém', 'Leiria', 'Santarém', 'Lisboa', 'Leiria'])

Iteração 6479:
(580, ['Viseu', 'Coimbra', 'Leiria', 'Santarém', 'Lisboa', 'Leiria', 'Santarém', 'Leiria'])

Iteração 6480:
Viseu (580, ['Viseu', 'Coimbra', 'Leiria', 'Santarém', 'Évora', 'Beja', 'Faro'])

ENTER para continuar ...
```

**Figura 8 – Aplicação de custo uniforme entre Viseu e Faro, evidenciando o elevado número de iterações.**

## 5. Procura sôfrega

---

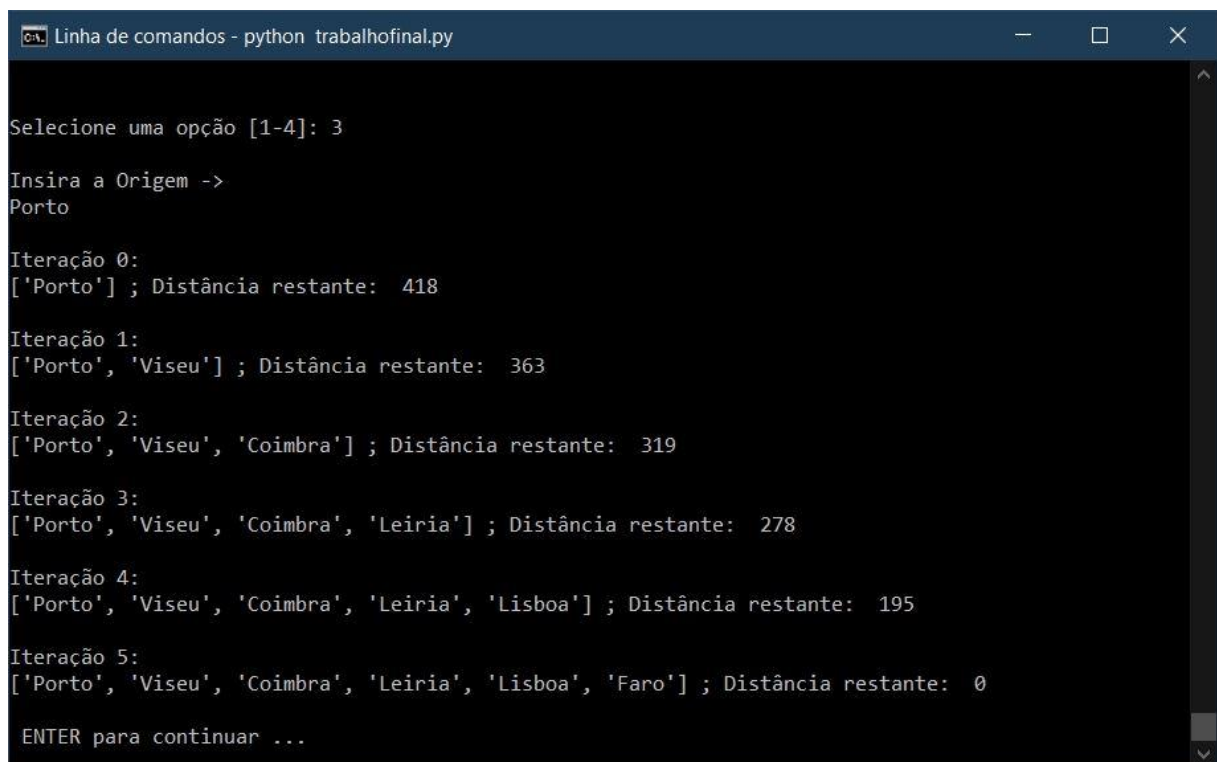
A procura sôfrega (*greedy search*) é um método heurístico, desenhado para analisar o caminho apenas iteração a iteração, supondo que a melhor solução local é também a que vai dar origem à melhor solução final.[7][8]

Assim, é escolhido como nó seguinte aquele que estiver mais perto do objetivo.

A grande vantagem é ser um método extremamente eficiente, minimizando a memória necessária para chegar ao resultado. A desvantagem é aumentar a probabilidade de o caminho escolhido não ser ótimo.

Implementámos este algoritmo de forma simples, recorrendo ao grafo distanciasFaro anteriormente exposto. No entanto também usamos o grafo com as relações entre as cidades.

O caminho vai sendo inserido numa list, tendo em conta a menor distância entre a cidade atual e a cidade no grafo. Essa lista vai sendo mostrada a cada iteração (ver Figura 9).



```
Linha de comandos - python trabalhofinal.py

Selecione uma opção [1-4]: 3

Insira a Origem ->
Porto

Iteração 0:
['Porto'] ; Distância restante: 418

Iteração 1:
['Porto', 'Viseu'] ; Distância restante: 363

Iteração 2:
['Porto', 'Viseu', 'Coimbra'] ; Distância restante: 319

Iteração 3:
['Porto', 'Viseu', 'Coimbra', 'Leiria'] ; Distância restante: 278

Iteração 4:
['Porto', 'Viseu', 'Coimbra', 'Leiria', 'Lisboa'] ; Distância restante: 195

Iteração 5:
['Porto', 'Viseu', 'Coimbra', 'Leiria', 'Lisboa', 'Faro'] ; Distância restante: 0

ENTER para continuar ...
```

**Figura 9 – Pesquisa sôfrega**

## 6. A\* (a-Star)

---

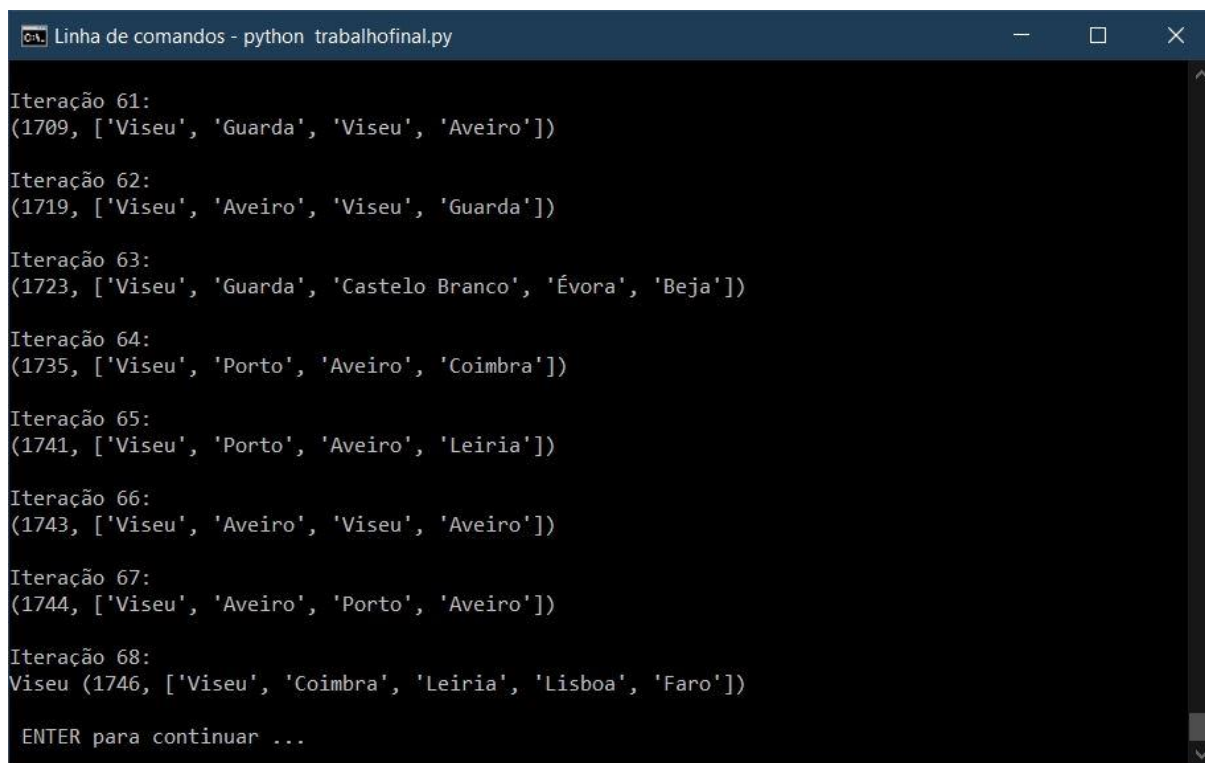
Este algoritmo heurístico combina as estratégias de custo uniforme e de procura sôfrega, obtendo um método que é simultaneamente mais eficiente, mas também mais próximo de obter o resultado ótimo. [9][10]

De forma muito geral, a distância entre um nó e o nó seguinte é somada à distância dos nós ao destino.

Assim, para implementar no nosso caso, consideremos o exemplo da origem Viseu. À partida, o peso é igual à distância de Viseu a Faro. De seguida, se o ponto de passagem for Aveiro, a distância Viseu – Aveiro é somada à distância de Aveiro a Faro. Esse é o valor que entra em comparação. E assim sucessivamente nos caminhos analisados.

Tal como no caso do custo uniforme, utilizámos uma PriorityQueue para ordenar as listas obtidas nas iterações, sendo tirada a cada ciclo a lista em posição zero.

Abaixo o resultado obtido (Figura 10).



```
Linha de comandos - python trabalhofinal.py

Iteração 61:
(1709, ['Viseu', 'Guarda', 'Viseu', 'Aveiro'])

Iteração 62:
(1719, ['Viseu', 'Aveiro', 'Viseu', 'Guarda'])

Iteração 63:
(1723, ['Viseu', 'Guarda', 'Castelo Branco', 'Évora', 'Beja'])

Iteração 64:
(1735, ['Viseu', 'Porto', 'Aveiro', 'Coimbra'])

Iteração 65:
(1741, ['Viseu', 'Porto', 'Aveiro', 'Leiria'])

Iteração 66:
(1743, ['Viseu', 'Aveiro', 'Viseu', 'Aveiro'])

Iteração 67:
(1744, ['Viseu', 'Aveiro', 'Porto', 'Aveiro'])

Iteração 68:
Viseu (1746, ['Viseu', 'Coimbra', 'Leiria', 'Lisboa', 'Faro'])

ENTER para continuar ...
```

**Figura 10 – Algoritmo A\***

---

## Conclusões

---

O software vai ao encontro dos objetivos delineados, permitindo uma comparação rápida do número de iterações que cada método executa.

Tal como esperado, o método de custo uniforme é aquele que apresenta maior número de iterações. No caso de Viseu – Faro, por exemplo, o método fez 6480 iterações.

Já os outros métodos não apresentam muita diferença em termos do tempo global de utilização, mas temos que levar em conta que apenas temos 18 cidades em estudo, sendo que nem todas as relações estão explicitadas nos grafos.

O software evidencia a aprendizagem que o sistema consegue fazer, mediante os dados que lhe são fornecidos, e como os resultados podem variar conforme as estratégias de pesquisa seguidas.



## 7. Referências

---

- [1] «Depth First Search». [Em linha]. Disponível em: <http://intelligence.worldofcomputing.net/ai-search/depth-first-search.html>. [Acedido: 24-Mai-2019].
- [2] «Depth First Search or DFS for a Graph - GeeksforGeeks». [Em linha]. Disponível em: <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>. [Acedido: 24-Mai-2019].
- [3] «Depth First Search Tutorials & Notes | Algorithms | HackerEarth». [Em linha]. Disponível em: <https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/>. [Acedido: 24-Mai-2019].
- [4] «Artificial Intelligence – Uniform Cost Search(UCS) | Algorithmic Thoughts - Artificial Intelligence | Machine Learning | Neuroscience | Computer Vision». [Em linha]. Disponível em: <https://algorithmicthoughts.wordpress.com/2012/12/15/artificial-intelligence-uniform-cost-searchucs/>. [Acedido: 24-Mai-2019].
- [5] «Search Algorithms Part 2: Uninformed Search Algorithms-1». [Em linha]. Disponível em: <https://medium.com/kredo-ai-engineering/search-algorithms-part-2-uninformed-search-algorithms-1-be5583a2f1e1>. [Acedido: 24-Mai-2019].
- [6] «Uniform-Cost Search (Dijkstra for large Graphs) - GeeksforGeeks». [Em linha]. Disponível em: <https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/>. [Acedido: 24-Mai-2019].
- [7] «Basics of Greedy Algorithms Tutorials & Notes | Algorithms | HackerEarth». [Em linha]. Disponível em: <https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/>. [Acedido: 24-Mai-2019].
- [8] «Greedy Algorithms | Brilliant Math & Science Wiki». [Em linha]. Disponível em: <https://brilliant.org/wiki/greedy-algorithm/>. [Acedido: 24-Mai-2019].
- [9] «Easy A\* (star) Pathfinding – Nicholas Swift – Medium». [Em linha]. Disponível em: <https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2>. [Acedido: 24-Mai-2019].
- [10] «Implementation of A\*». [Em linha]. Disponível em: <https://www.redblobgames.com/pathfinding/a-star/implementation.html>. [Acedido: 24-Mai-2019].

## **8. Bibliografia**

---

Slides e documentação disponibilizada em [moodle.estgv.ipv.pt](http://moodle.estgv.ipv.pt)