

**Universidade do Minho**  
Escola de Engenharia  
Licenciatura em Engenharia Informática

## **Unidade Curricular de Inteligência Artificial**

Ano Letivo de 2022/2023

### **Relatório do Projeto - Fase 1 Resolução de Problemas - Algoritmos de procura**

#### **Grupo 21**

Luís Alberto Barreiro Araújo	A96351
Pedro Miguel da Cruz Pacheco	A61042
Pedro Calheno Pinto	A87983
Rafael Arêas	A86817

# Index

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Descrição do problema</b>	<b>1</b>
<b>3</b>	<b>Formulação do problema</b>	<b>1</b>
3.1	Circuitos Usados . . . . .	2
<b>4</b>	<b>Descrição de todas as tarefas realizadas, bem como de todas as decisões tomadas pelo grupo de trabalho</b>	<b>4</b>
4.1	Construtores e criação de Estruturas . . . . .	4
4.2	Algoritmos Implementados . . . . .	6
4.3	Menu Interativo . . . . .	7
<b>5</b>	<b>Sumário e discussão dos resultados obtidos</b>	<b>9</b>
<b>6</b>	<b>Conclusão</b>	<b>13</b>
<b>7</b>	<b>Bibliografia</b>	<b>13</b>

# 1 Introdução

Com a realização deste trabalho prático é pretendido que se desenvolvam diversos algoritmos de procura para a resolução de um jogo, nomeadamente o VectorRace, também conhecido como RaceTrack. O VectorRace é um jogo de simulação de carros simplificado, que contém um conjunto de movimentos e regras associadas. Neste relatório mostram-se apenas as estratégias usadas na primeira fase de resolução, nomeadamente a implementação de algoritmos de pesquisa não informada para procura num grafo.

## 2 Descrição do problema

Foi nos proposto a realização do *VectorRace game*, um jogo baseado num conjunto de movimentos e acelerações numa determinada direção escolhida pelo jogador. Temos como objetivo implementar algoritmos de procura, que encontram uma sequência de ações que nos levam ao estado final desejado, que no caso do *VectorRace*, se trata de descobrir o melhor caminho para percorrer no percurso do estado inicial até o estado objetivo proposto. A solução ideal para este problema, o melhor caminho para percorrer o percurso nesta fase do projeto, será sempre aquele que envolva um menor custo total.

## 3 Formulação do problema

Para a formulação do problema consideramos a notação  $l$ , que representa a linha e  $c$  a coluna para os vetores. Num determinado instante o carro pode avançar para qualquer peça adjacente, incluindo as peças diagonais e não pode avançar para fora da pista. Para a representação das peças no circuito declaramos da seguinte forma:

- As peças válidas do circuito são representadas por -
- As peças que representam as barreiras do circuito são os X
- A peça que representa o ponto de partida de um carro no circuito é o P
- A peça que representa a chegada, o ponto final do circuito é o F

O Estado inicial é a peça 'P' e o estado objetivo é a peça 'F'.

Já o carro terá uma velocidade que é calculada com base na sua velocidade atual e a aceleração a que está sujeito. Desta forma, a posição final de um veículo é calculada com base na sua posição atual e a velocidade que sofre.

### 3.1 Circuitos Usados

Em seguida apresentamos os circuitos que criamos para elaboração deste trabalho e nos quais foram testados os algoritmos de procura implementados.

```

XXXXXXXXXXXXX
XP-----XX
X---XXX---X
X--XXXX---X
X---XXX---X
X----XX---X
X-----X---X
XXX-----XX
XX-----FFXX
XXXXXXXXXXXXX

```

Figure 1: Circuito 1

**Posição inicial : (1, 1, 'P') Posição final : (8, 8, 'F')**

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XX-----XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX-XXXXX
XXXXXXXXXXXXX-----XXXXXXXXXXXXXXXXXXXXXXXXXXXXX-
XX-----XXXXX-----XXXXXXXXXXXXXXXXXXXXXXXXXXXX-XX
X-----XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX-XXXX
X-P-----XXXXXXX-----XXXXXXXXXXXXXXXXXXXX-XXXXXX
XXXX-----XXXXXXXXXXXXXXXXXXXXXXXXXXXX-XX
XXXXX---XXXX-----XXXXXXXXXXXXXXXXXXXX-XXX
XXXXXXXXXXXXX-----XXXXXX-----XXXXX-XX
XX-----XXXXXXXXXXXXXXXXXXXXXXXXXXXX-XXXXX
XXX-----XXXXXXXXXXXX-XXXXXXXXXXXX-XXX
XXXXXXXXXX-----XXX-----XXXXXX-----XX
XXXXXXXXXXXXX-----XX-----XXXXX
XXXXX-----XXXX-----X-----F
XXXXXX-----XXXXXXXXXXXXXXXXXXXXXXXXXXXXX|
XXX-----XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

Figure 2: Circuito 2

**Posição inicial : (5, 2, 'P') Posição final : (13, 44, 'F')**

[illegible]

Figure 3: Circuito 3

**Posição inicial : (1, 2, 'P') Posição final : (12, 44, 'F')**

[illegible]

Figure 4: Circuito 4

**Posição inicial : (21, 34, 'P') Posição final : (3, 15, 'F')**

## 4 Descrição de todas as tarefas realizadas, bem como de todas as decisões tomadas pelo grupo de trabalho

### 4.1 Construtores e criação de Estruturas

A resolução do problema proposto começou por se fazer a partir da leitura dos ficheiros de texto que contêm os vários circuitos para uma matriz (para cada circuito) de tuplos de três posições - (x, y, peça). Cada tuplo representa uma posição em que o veículo pode estar, sendo x e y as suas coordenadas na matriz e peça, o tipo de coordenada que é (uma casa livre, uma parede, uma posição final ou a posição inicial).

Posteriormente houve a necessidade de criar um grafo com esta informação. Com esse intuito, foi desenvolvida uma classe 'Nodo' com a seguinte implementação:

```
def __init__(self, no):
    self.m_name = "(" + str(no[0]) + ", " + str(no[1]) + ") : " + str(no[2])
    self.x = no[0]
    self.y = no[1]
    self.peca = no[2]
```

Listing 1: Construtor da classe Nodo

O construtor do Nodo usa o tuplo da matriz para criar os seus campos. Estes nodos serão a parte central de um grafo que vai representar todas as possíveis deslocações no circuito.

A estrutura 'Graph' é então criada à custa do método 'addedge', que tem como objetivo adicionar as arestas entre os nodos, método este que é chamado dentro de um outro - 'createGraph'. O 'createGraph', percorre a matriz inicial, verifica quais as deslocações válidas, ou seja, impossibilita a deslocação para uma peça 'X' (parede), e passa então os dois tuplos ao método 'addedge' (nodo origem e nodo destino) e os custos das ligações nos dois sentidos. Aqui é chamado o construtor da classe 'Nodo' e preenchidos os campos da classe 'Graph'.

De notar que para as classes 'Graph' e 'Nodo', um ponto de partida para a nossa implementação foi parte do código utilizado nas aulas práticas da UC de Inteligência Artificial.

```
def __init__(self, directed=False):
    self.m_nodes = []
    self.m_directed = directed
    self.m_graph = {}
    self.m_h = {}
```

Listing 2: Construtor da classe Graph

```

with open("circuit01.txt") as file1:
    matrix4 = ([list(line.strip().replace(" ", "")) for line in file1.readlines()
])

def createGraph(g, mat):
    for index, node in enumerate(mat):
        for node2 in mat[index:]:
            if (node2[0] == (node[0] + 1) and node2[1] == node[1]) or (
                node2[0] == (node[0] - 1) and node2[1] == node[1]) or (
                node2[0] == node[0] and node2[1] == (node[1] + 1)) or (
                node2[0] == node[0] and node2[1] == (node[1] - 1)) or (
                node2[0] == (node[0] - 1) and node2[1] == (node[1] - 1)) or (
                node2[0] == (node[0] + 1) and node2[1] == (node[1] - 1)) or (
                node2[0] == (node[0] - 1) and node2[1] == (node[1] + 1)) or (
                node2[0] == (node[0] + 1) and node2[1] == (node[1] + 1)):
                if (node[2] != "X" and node2[2] != "X"):
                    g.addedge(node, node2, 1, 1)

# CIRCUITO 1

# criacao da matriz mat1
mat1 = []
for x, line in enumerate(matrix1):
    for y, piece in enumerate(line):
        mat1.append((x, y, piece))

g1 = Graph()

grafo1 = createGraph(g1, mat1)

# Nota : addedge localiza-se no graph.py e nao no main.py

def addedge(self, node1, node2, weight1, weight2):
    n1 = Node(node1)
    n2 = Node(node2)

    if n1 not in self.m_nodes:
        self.m_nodes.append(n1)
        self.m_graph[node1] = set()
    else:
        n1 = self.get_node_by_name(node1)

    if n2 not in self.m_nodes:
        self.m_nodes.append(n2)
        self.m_graph[node2] = set()
    else:
        n2 = self.get_node_by_name(node2)

    self.m_graph[node1].add((node2, weight1))

    if not self.m_directed:
        self.m_graph[node2].add((node1, weight2))

```

Listing 3: Exemplo da transformação do circuito.txt num grafo através dos vários métodos usados.

## 4.2 Algoritmos Implementados

Para a implementação dos algoritmos de procura propostos para esta fase do projeto, utilizamos como estratégia o tipo de procura não-informada (cega). Para este tipo de procura, onde não é levada em conta uma heurística auxiliar, foram implementados dois tipos de algoritmos: o de procura DFS (Depth-first Search) e o de Procura BFS (Breadth-first search).

Primeiramente apresentamos o algoritmo de procura BFS, que consiste num algoritmo de procura em largura e tem como estratégia expandir todos os nós com a menor profundidade. Uma das mais valias deste algoritmo é a capacidade que ele tem de calcular o melhor caminho entre dois nodos (leia-se, o de menor custo), mas este benefício é contraposto com a sua necessidade de um maior tempo de execução.

```
def procura_BFS(self, start, end):
    visited = set()
    fila = Queue()

    fila.put(start)
    visited.add(start)

    parent = dict()
    parent[start] = None

    path_found = False
    while not fila.empty() and path_found == False:
        nodo_atual = fila.get()
        if nodo_atual == end:
            path_found = True
        else:
            for (adjacente, peso) in self.m_graph[nodo_atual]:
                if adjacente not in visited:
                    fila.put(adjacente)
                    parent[adjacente] = nodo_atual
                    visited.add(adjacente)

    path = []
    if path_found:
        path.append(end)
        while parent[end] is not None:
            path.append(parent[end])
            end = parent[end]
        path.reverse()
        custo = self.calcula_custo(path)
    return path, custo
```

Listing 4: Algoritmo de procura BFS.



O segundo algoritmo de procura a ser desenvolvido foi o de procura DFS, que se baseia na estratégia de uma procura em profundidade que consiste na expansão recursiva de um dos nós de uma árvore de nós, até que seja encontrado o mais profundo. Embora com a possibilidade de ser muito mais rápido que o algoritmo BFS, é demarcado pelo facto de o resultado desta procura ser o primeiro caminho encontrado entre o nodo origem e o nodo destino, não sendo esse caminho necessariamente o mais curto.

```
def procura_DFS(self, start, end, path=[], visited=set()):
    path.append(start)
    visited.add(start)

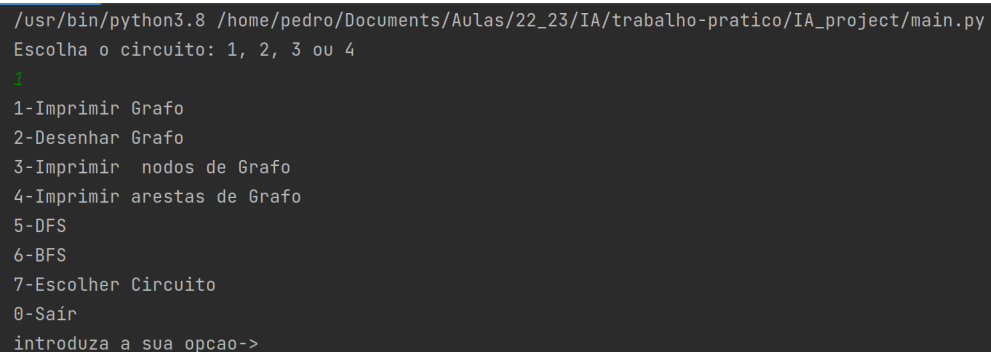
    if start == end:
        custoT = self.calcula_custo(path)
        return path, custoT
    for (adjacente, peso) in self.m_graph[start]:
        if adjacente not in visited:
            resultado = self.procura_DFS(adjacente, end, path, visited)
            if resultado is not None:
                return resultado
    path.pop()
    return None
```

Listing 5: Algoritmo de procura BFS.

### 4.3 Menu Interativo

Já numa parte terminal do desenvolvimento desta fase do projeto, decidimos implementar um Menu de Usuário como o que se pode ver na figura 5, para melhor estruturar as funcionalidades do programa.

Inicialmente, o programa pede ao usuário que insira qual o circuito que quer resolver e é seguidamente apresentado com as opções de imprimir grafo, desenhar grafo, imprimir nodos do grafo, imprimir arestas de grafo, algoritmo DFS, algoritmo BFS, escolher um circuito diferente e finalmente sair e terminar a execução.



```
/usr/bin/python3.8 /home/pedro/Documents/Aulas/22_23/IA/trabalho-pratico/IA_project/main.py
Escolha o circuito: 1, 2, 3 ou 4
1
1-Imprimir Grafo
2-Desenhar Grafo
3-Imprimir nodos de Grafo
4-Imprimir arestas de Grafo
5-DFS
6-BFS
7-Escolher Circuito
0-Sair
introduza a sua opcao->
```

Figure 5: Menu de Usuário

Eis algumas funcionalidades implementadas:

```
((1, 1, 'P'), ((2, 1, '-'), 1), ((2, 2, '-'), 1), ((1, 2, '-'), 1)))
((1, 2, '-'), ((2, 2, '-'), 1), ((2, 1, '-'), 1), ((1, 3, '-'), 1), ((1, 1, 'P'), 1)))
((2, 1, '-'), ((2, 2, '-'), 1), ((3, 1, '-'), 1), ((3, 2, '-'), 1), ((1, 1, 'P'), 1)))
((2, 2, '-'), ((2, 1, '-'), 1), ((3, 1, '-'), 1), ((3, 2, '-'), 1), ((2, 3, '-'), 1), ((1, 2, '-'), 1), ((1, 1, 'P'), 1)))
((1, 3, '-'), ((1, 4, '-'), 1), ((2, 2, '-'), 1), ((2, 3, '-'), 1), ((1, 2, '-'), 1)))
((2, 3, '-'), ((2, 2, '-'), 1), ((1, 4, '-'), 1), ((1, 3, '-'), 1), ((3, 2, '-'), 1), ((1, 2, '-'), 1)))
((1, 4, '-'), ((1, 3, '-'), 1), ((2, 3, '-'), 1), ((1, 5, '-'), 1)))
((1, 5, '-'), ((1, 6, '-'), 1), ((1, 4, '-'), 1)))
((1, 6, '-'), ((1, 7, '-'), 1), ((2, 7, '-'), 1), ((1, 5, '-'), 1)))
((1, 7, '-'), ((1, 8, '-'), 1), ((2, 7, '-'), 1), ((1, 6, '-'), 1), ((3, 8, '-'), 1), ((1, 7, '-'), 1)))
((2, 7, '-'), ((3, 7, '-'), 1), ((2, 8, '-'), 1), ((1, 6, '-'), 1), ((1, 8, '-'), 1)))
((1, 8, '-'), ((2, 7, '-'), 1), ((2, 8, '-'), 1), ((1, 7, '-'), 1)))
((2, 8, '-'), ((3, 7, '-'), 1), ((2, 7, '-'), 1), ((1, 8, '-'), 1), ((2, 9, '-'), 1), ((3, 8, '-'), 1), ((1, 7, '-'), 1), ((3, 9, '-'), 1)))
((2, 9, '-'), ((1, 8, '-'), 1), ((2, 8, '-'), 1), ((3, 9, '-'), 1)))
((3, 1, '-'), ((2, 2, '-'), 1), ((4, 3, '-'), 1), ((4, 1, '-'), 1), ((3, 2, '-'), 1), ((4, 2, '-'), 1)))
((3, 2, '-'), ((2, 7, '-'), 1), ((2, 8, '-'), 1), ((4, 8, '-'), 1), ((3, 8, '-'), 1), ((4, 7, '-'), 1)))
((3, 7, '-'), ((2, 7, '-'), 1), ((4, 9, '-'), 1), ((2, 9, '-'), 1), ((4, 7, '-'), 1), ((3, 9, '-'), 1)))
((3, 8, '-'), ((3, 7, '-'), 1), ((4, 9, '-'), 1), ((2, 9, '-'), 1), ((4, 8, '-'), 1), ((3, 9, '-'), 1)))
((3, 9, '-'), ((4, 9, '-'), 1), ((2, 8, '-'), 1), ((4, 8, '-'), 1), ((2, 9, '-'), 1), ((3, 8, '-'), 1)))
((4, 1, '-'), ((3, 1, '-'), 1), ((5, 1, '-'), 1), ((3, 2, '-'), 1), ((4, 2, '-'), 1), ((5, 2, '-'), 1)))
((4, 2, '-'), ((3, 2, '-'), 1), ((4, 1, '-'), 1), ((5, 1, '-'), 1), ((5, 2, '-'), 1), ((5, 3, '-'), 1)))
((4, 3, '-'), ((3, 2, '-'), 1), ((4, 2, '-'), 1), ((5, 4, '-'), 1), ((5, 2, '-'), 1), ((5, 3, '-'), 1)))
((4, 7, '-'), ((3, 7, '-'), 1), ((5, 8, '-'), 1), ((4, 8, '-'), 1), ((3, 8, '-'), 1), ((5, 7, '-'), 1)))
((4, 8, '-'), ((3, 7, '-'), 1), ((4, 9, '-'), 1), ((5, 8, '-'), 1), ((4, 7, '-'), 1), ((5, 9, '-'), 1), ((3, 9, '-'), 1)))
((4, 9, '-'), ((5, 8, '-'), 1), ((4, 8, '-'), 1), ((6, 2, '-'), 1), ((4, 2, '-'), 1), ((5, 2, '-'), 1)))
((5, 1, '-'), ((6, 1, '-'), 1), ((4, 3, '-'), 1), ((4, 1, '-'), 1), ((6, 2, '-'), 1), ((5, 2, '-'), 1)))
((5, 2, '-'), ((6, 1, '-'), 1), ((4, 3, '-'), 1), ((4, 1, '-'), 1), ((6, 3, '-'), 1), ((5, 3, '-'), 1), ((5, 1, '-'), 1), ((4, 2, '-'), 1), ((5, 4, '-'), 1), ((5, 2, '-'), 1)))
((5, 3, '-'), ((4, 3, '-'), 1), ((6, 4, '-'), 1), ((6, 2, '-'), 1), ((4, 2, '-'), 1), ((5, 4, '-'), 1), ((5, 3, '-'), 1)))
((5, 4, '-'), ((4, 3, '-'), 1), ((6, 3, '-'), 1), ((6, 5, '-'), 1), ((6, 4, '-'), 1), ((5, 3, '-'), 1)))
((5, 7, '-'), ((5, 8, '-'), 1), ((4, 8, '-'), 1), ((4, 7, '-'), 1), ((6, 7, '-'), 1), ((6, 9, '-'), 1), ((5, 9, '-'), 1), ((6, 8, '-'), 1)))
((5, 8, '-'), ((4, 9, '-'), 1), ((4, 8, '-'), 1), ((4, 7, '-'), 1), ((6, 7, '-'), 1), ((6, 9, '-'), 1), ((5, 9, '-'), 1), ((6, 8, '-'), 1), ((5, 7, '-'), 1)))
((5, 9, '-'), ((4, 9, '-'), 1), ((5, 8, '-'), 1), ((4, 8, '-'), 1), ((6, 9, '-'), 1), ((6, 8, '-'), 1)))
((6, 1, '-'), ((6, 2, '-'), 1), ((5, 1, '-'), 1), ((6, 3, '-'), 1), ((5, 2, '-'), 1), ((5, 3, '-'), 1), ((7, 3, '-'), 1)))
((6, 2, '-'), ((6, 1, '-'), 1), ((6, 3, '-'), 1), ((5, 2, '-'), 1), ((5, 3, '-'), 1), ((7, 3, '-'), 1)))
((6, 3, '-'), ((6, 4, '-'), 1), ((7, 4, '-'), 1), ((6, 2, '-'), 1), ((7, 5, '-'), 1), ((5, 3, '-'), 1), ((7, 3, '-'), 1)))
((6, 4, '-'), ((7, 4, '-'), 1), ((6, 5, '-'), 1), ((7, 5, '-'), 1), ((5, 4, '-'), 1), ((7, 5, '-'), 1)))
((6, 5, '-'), ((7, 6, '-'), 1), ((6, 4, '-'), 1), ((7, 4, '-'), 1), ((5, 4, '-'), 1), ((7, 5, '-'), 1)))
((6, 7, '-'), ((7, 6, '-'), 1), ((6, 8, '-'), 1), ((7, 7, '-'), 1), ((6, 6, '-'), 1), ((7, 7, '-'), 1)))
((6, 8, '-'), ((5, 8, '-'), 1), ((6, 9, '-'), 1), ((5, 9, '-'), 1), ((7, 7, '-'), 1), ((7, 8, '-'), 1)))
((6, 9, '-'), ((7, 8, '-'), 1), ((5, 9, '-'), 1), ((6, 8, '-'), 1), ((5, 8, '-'), 1)))
((7, 1, '-'), ((8, 1, '-'), 1), ((7, 2, '-'), 1), ((6, 1, '-'), 1), ((8, 2, '-'), 1), ((7, 3, '-'), 1), ((6, 2, '-'), 1), ((8, 3, '-'), 1)))
((7, 2, '-'), ((8, 3, '-'), 1), ((7, 4, '-'), 1), ((6, 3, '-'), 1), ((8, 4, '-'), 1), ((7, 5, '-'), 1), ((6, 4, '-'), 1), ((8, 5, '-'), 1), ((7, 3, '-'), 1)))
((7, 3, '-'), ((8, 6, '-'), 1), ((7, 6, '-'), 1), ((8, 5, '-'), 1), ((7, 4, '-'), 1), ((8, 4, '-'), 1), ((6, 5, '-'), 1), ((6, 4, '-'), 1)))
((7, 6, '-'), ((8, 6, '-'), 1), ((8, 5, '-'), 1), ((8, 7, '-'), 1), ((6, 7, '-'), 1), ((6, 8, '-'), 1), ((8, 8, '-'), 1), ((8, 7, '-'), 1)))
((7, 7, '-'), ((8, 6, '-'), 1), ((7, 6, '-'), 1), ((8, 7, '-'), 1), ((6, 7, '-'), 1), ((6, 8, '-'), 1), ((8, 8, '-'), 1), ((8, 7, '-'), 1)))
((7, 8, '-'), ((8, 8, '-'), 1), ((8, 7, '-'), 1), ((6, 7, '-'), 1), ((6, 9, '-'), 1), ((6, 8, '-'), 1), ((7, 7, '-'), 1)))
((8, 1, '-'), ((8, 2, '-'), 1), ((7, 3, '-'), 1)))
((8, 2, '-'), ((8, 3, '-'), 1), ((7, 4, '-'), 1), ((7, 5, '-'), 1), ((7, 3, '-'), 1)))
((8, 3, '-'), ((8, 4, '-'), 1), ((7, 4, '-'), 1), ((8, 5, '-'), 1), ((7, 5, '-'), 1), ((7, 6, '-'), 1)))
((8, 4, '-'), ((8, 5, '-'), 1), ((7, 6, '-'), 1), ((8, 6, '-'), 1), ((7, 7, '-'), 1), ((7, 8, '-'), 1)))
((8, 5, '-'), ((8, 6, '-'), 1), ((7, 6, '-'), 1), ((8, 7, '-'), 1), ((7, 7, '-'), 1), ((7, 8, '-'), 1)))
((8, 6, '-'), ((8, 7, '-'), 1), ((8, 8, '-'), 1), ((7, 8, '-'), 1), ((7, 7, '-'), 1), ((7, 8, '-'), 1)))
((8, 7, '-'), ((8, 8, '-'), 1), ((7, 8, '-'), 1), ((7, 7, '-'), 1), ((7, 8, '-'), 1)))
((8, 8, '-'), ((7, 8, '-'), 1), ((7, 7, '-'), 1), ((7, 8, '-'), 1)))
```

Figure 6: Circuito1 representado num Grafo

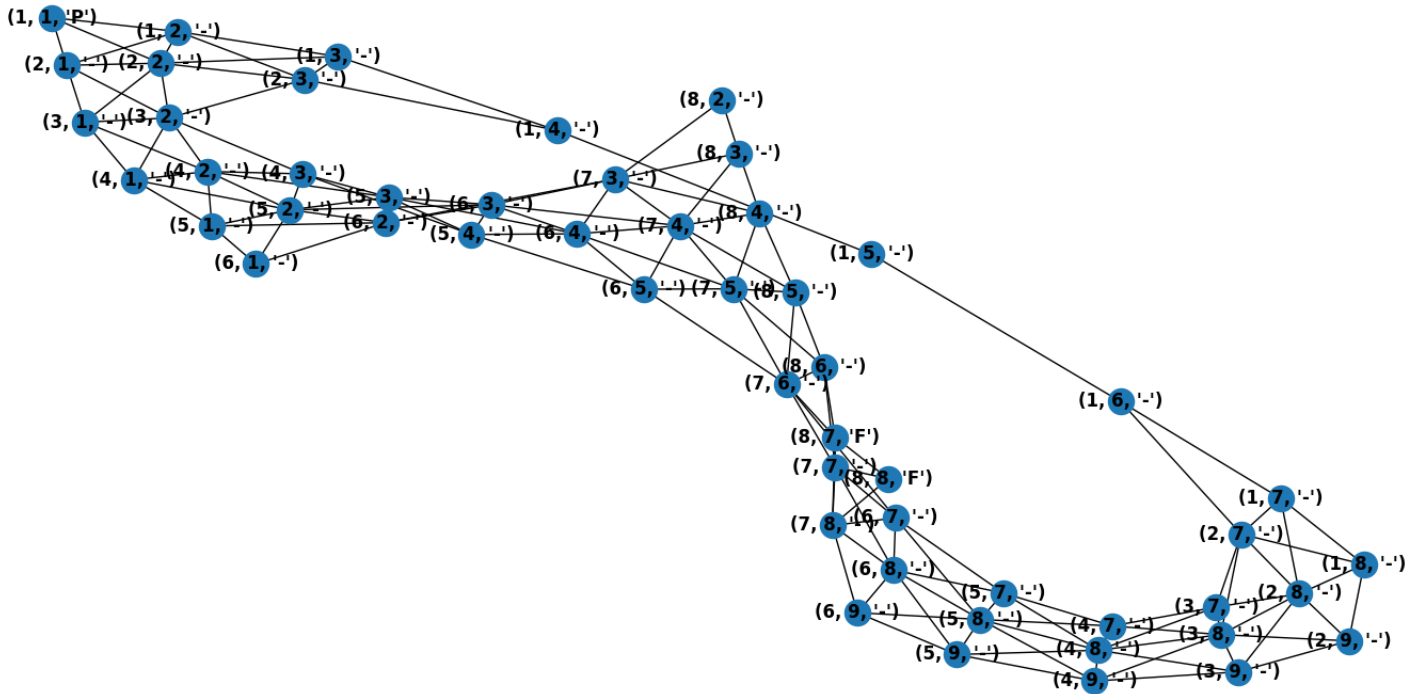


Figure 7: Representação Gráfica do Grafo do Circuito 1

## 5 Sumário e discussão dos resultados obtidos

```
Escolha o circuito: 1, 2, 3 ou 4
1
1-Imprimir Grafo
2-Desenhar Grafo
3-Imprimir nodos de Grafo
4-Imprimir arestas de Grafo
5-DFS
6-BFS
7-Escolher Circuito
0-Sair
introduza a sua opcao-> 6
caminho percorrido: [(1, 1, 'P'), (2, 1, '-'), (3, 2, '-'), (4, 3, '-'), (5, 4, '-'), (6, 5, '-'), (7, 6, '-'), (8, 7, 'F'), (8, 8, 'F')]
custo total: 8
```

Figure 8: Solução do Algoritmo BFS para o circuito 1

**Custo Total = 8**

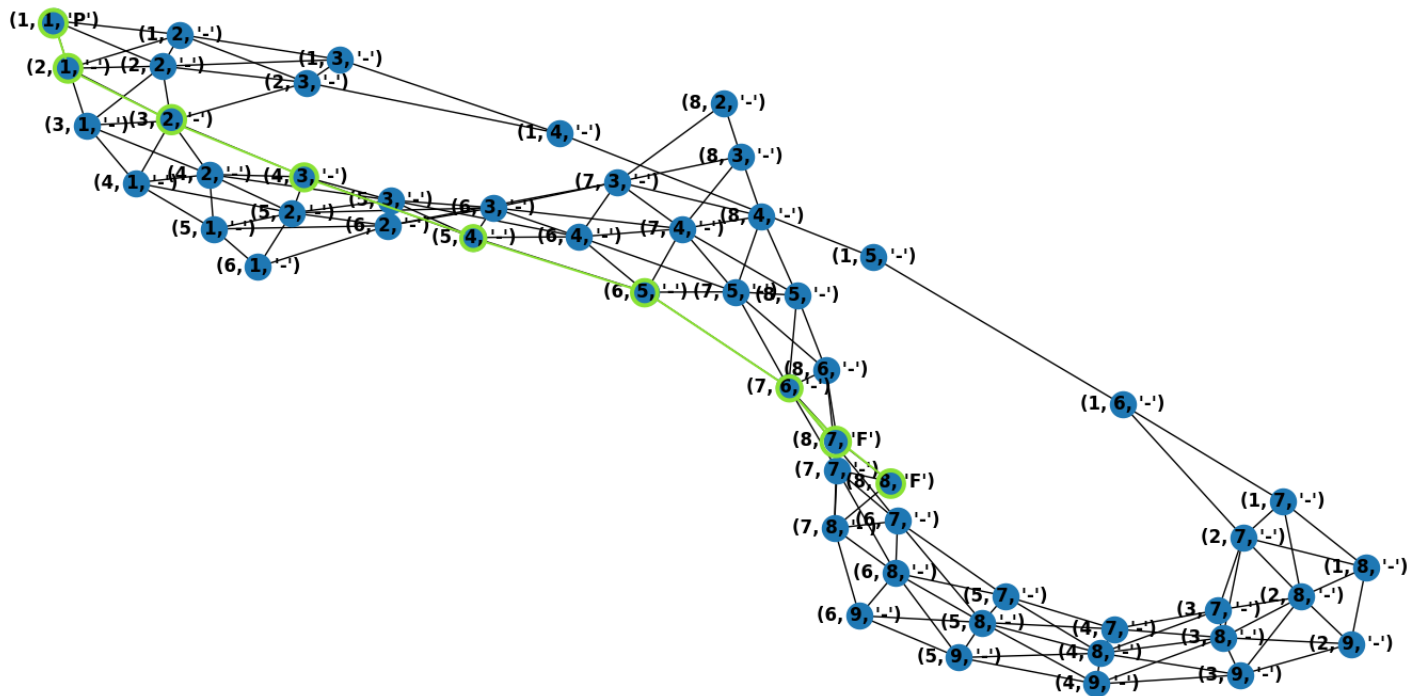


Figure 9: Representação Gráfica do caminho percorrido pela BFS no Grafo do Circuito 1

```

1-Imprimir Grafo
2-Desenhar Grafo
3-Imprimir nodos do Grafo
4-Imprimir arestas do Grafo
5-BFS
6-BFS
7-Escolher Circuito
8-Sair
introduza a sua opcao->
caminho percorrido: [(1, 1, 'P'), (2, 1, '-'), (2, 2, '-'), (2, 3, '-'), (3, 2, '-'), (4, 3, '-'), (5, 3, '-'), (6, 4, '-'), (7, 5, '-'), (7, 6, '-'), (6, 7, '-'), (7, 7, '-'), (8, 8, 'F')]
custo total: 12

```

Figure 10: Solução do Algoritmo DFS para o circuito 1

**Custo Total = 12**

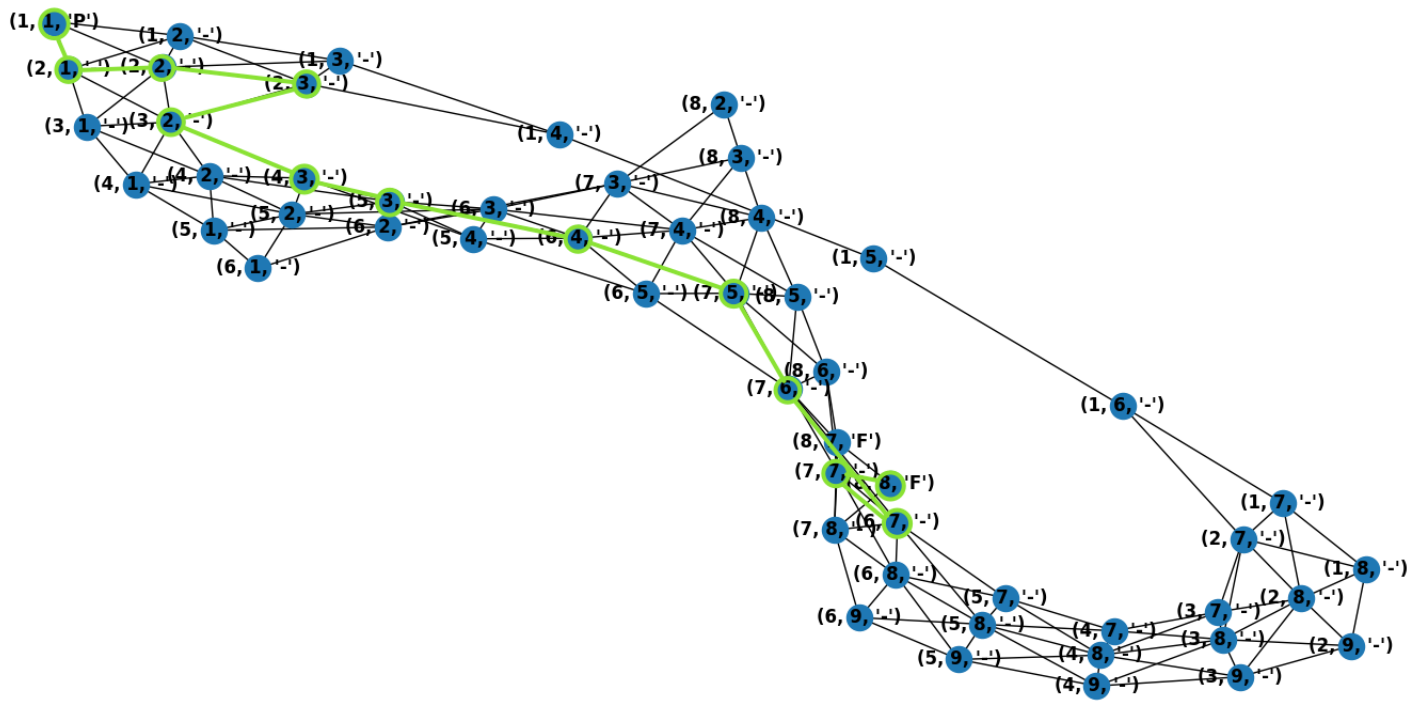


Figure 11: Representação Gráfica do caminho percorrido pela DFS no Grafo do Circuito 1

```
Escolha o circuito: 1, 2, 3 ou 4

1-Imprimir Grafo
2-Desenhar Grafo
3-Imprimir nodos de Grafo
4-Imprimir arestas de Grafo
5-DFS
6-BFS
7-Escolher Circuito
8-Sair
introduza a sua opcao-> 2
caminho percorrido: [(5, 2, 'P'), (5, 1, '-'), (4, 2, '-'), (3, 3, '-'), (3, 4, '-'), (4, 4, '-'), (5, 4, '-'), (6, 5, '-'), (5, 6, '-'), (5, 7, '-'), (6, 8, '-'), (6, 7, '-'), (6, 6, '-'), (7, 5, '-'), (7, 6, '-'), (7, 7, '-'), (7, 8, '-'), (6, 9, '-'),
custo total: 103
prima enter para continuar
```

Figure 12: Solução do Algoritmo DFS para o circuito 2

**Custo Total = 103**

```
1-Imprimir Grafo
2-Desenhar Grafo
3-Imprimir nodos de Grafo
4-Imprimir arestas de Grafo
5-DFS
6-BFS
7-Escolher Circuito
8-Sair
introduza a sua opcao-> 2
caminho percorrido: [(5, 2, 'P'), (4, 3, '-'), (5, 4, '-'), (6, 5, '-'), (5, 6, '-'), (5, 7, '-'), (6, 8, '-'), (6, 9, '-'), (6, 10, '-'), (6, 11, '-'), (6, 12, '-'), (6, 13, '-'), (7, 14, '-'), (6, 15, '-'), (7, 16, '-'), (7, 17, '-'), (6, 18, '-'), (6,
custo total: 42
prima enter para continuar
```

Figure 13: Solução do Algoritmo BFS para o circuito 2

**Custo Total = 42**

```
1-Imprimir Grafo
2-Desenhar Grafo
3-Imprimir nodos de Grafo
4-Imprimir arestas de Grafo
5-DFS
6-BFS
7-Escolher Circuito
8-Sair
introduza a sua opcao-> 3
caminho percorrido: [(1, 2, 'P'), (1, 3, '-'), (2, 4, '-'), (3, 3, '-'), (4, 2, '-'), (4, 1, '-'), (3, 2, '-'), (2, 3, '-'), (3, 4, '-'), (4, 4, '-'), (5, 4, '-'), (6, 5, '-'), (5, 6, '-'), (4, 7, '-'), (3, 6, '-'), (2, 5, '-'), (1, 5, '-'), (2, 6, '-'), (
custo total: 129
prima enter para continuar
```

Figure 14: Solução do Algoritmo DFS para o circuito 3

**Custo Total = 129**

```
prima enter para continuar
1-Imprimir Grafo
2-Desenhar Grafo
3-Imprimir nodos de Grafo
4-Imprimir arestas de Grafo
5-DFS
6-BFS
7-Escolher Circuito
8-Sair
introduza a sua opcao-> 3
caminho percorrido: [(1, 2, 'P'), (2, 2, '-'), (3, 3, '-'), (4, 4, '-'), (5, 5, '-'), (6, 6, '-'), (7, 7, '-'), (8, 8, '-'), (9, 9, '-'), (10, 10, '-'), (11, 11, '-'), (12, 12, '-'), (13, 13, '-'), (14, 14, '-'), (15, 15, '-'), (14, 16, '-'), (15, 17, '-'),
custo total: 43
prima enter para continuar
```

Figure 15: Solução do Algoritmo BFS para o circuito 3

**Custo Total = 43**

```

1-Imprimir Grafo
2-Desenhar Grafo
3-Imprimir nodos de Grafo
4-Imprimir arestas de Grafo
5-BFS
6-BFS
7-Escolher Circuito
8-Sair
introduza a sua opcao-> 4
caminho percorrido: [(21, 34, 'P'), (20, 34, '-'), (19, 35, '-'), (19, 34, '-'), (20, 33, '-'), (21, 33, '-'), (21, 32, '-'), (21, 31, '-'), (20, 31, '-'), (19, 32, '-'), (19, 33, '-'), (20, 32, '-'), (19, 31, '-'), (20, 30, '-'), (20, 29, '-'), (20, 28, '-')]
custo total: 99
prima enter para continuar

```

Figure 16: Solução do Algoritmo DFS para o circuito 4

**Custo Total = 99**

```

1-Imprimir Grafo
2-Desenhar Grafo
3-Imprimir nodos de Grafo
4-Imprimir arestas de Grafo
5-BFS
6-BFS
7-Escolher Circuito
8-Sair
introduza a sua opcao-> 4
caminho percorrido: [(21, 34, 'P'), (21, 33, '-'), (21, 32, '-'), (21, 31, '-'), (20, 30, '-'), (20, 29, '-'), (20, 28, '-'), (20, 27, '-'), (19, 26, '-'), (18, 25, '-'), (19, 24, '-'), (19, 23, '-'), (19, 22, '-'), (19, 21, '-'), (19, 20, '-'), (18, 19, '-')]
custo total: 43
prima enter para continuar

```

Figure 17: Solução do Algoritmo BFS para o circuito 4

**Custo Total = 43**

Como é possível observar através da comparação de custos entre o Algoritmo de Pesquisa em Profundidade (DFS) e o Algoritmo de Pesquisa em Largura (BFS), assim como o caminho calculado por cada um, o segundo tipo é muito mais eficaz a encontrar o caminho mais curto que o primeiro. Isso deve-se, tal como explicado anteriormente neste texto, à forma como cada um faz a procura pelo nodo destino.

Nota: De forma a tornar este relatório mais conciso, não foram incluídas as representações gráficas dos outros circuitos, pois elas tornavam-se demasiado complexas e extensas para a compreensão do leitor. Devido às limitações do *Networkx*, é também impossível fazer a representação gráfica do Grafo4 pois este tem demasiados nodos e arestas.

## 6 Conclusão

Depois desta primeira fase do trabalho prático podemos dizer que aprendemos a utilizar e implementar algoritmos de procura num grafo para descobrir o melhor caminho entre dois nodos. Foi nos possível também consolidar melhor os conhecimentos acerca da linguagem de programação *Python*. Como a decisão de algoritmos implementados caiu no uso de procuras não informadas, não fazia sentido o uso do cálculo da velocidade para deslocação do veículo. Como tal será uma das tarefas que irá ser implementada numa segunda fase do trabalho. Ademais, iremos apresentar algoritmos de pesquisa informada fazendo o uso de heurísticas para aprimorar a solução de atingir um estado final que dê resposta ao problema.

## 7 Bibliografia

Código base retirado de : **Licenciatura em Engenharia Informática - Mestrado Integrado em Engenharia Informática - Inteligência Artificial - Ficha Prática n.º 3 - Tema: Formulação de Problemas de Pesquisa e Resolução de Problemas utilizando Pesquisa Não Informada**

Boschloo, H. (no date) Vector racer, harmmade.com. Available at: <https://harmmade.com/vectorracer/> (Accessed: November 30, 2022).

Adjacency matrix (no date) Programiz. Available at: <https://www.programiz.com/dsa/graph-adjacency-matrix> (Accessed: November 30, 2022).

Python - convert matrix to coordinate dictionary (2020) GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/convert-matrix-to-coordinate-dictionary/> (Accessed: November 30, 2022).

Julie RaswickJulie Raswick 20.4k33 gold badges1515 silver badges33 bronze badges et al. (1957) How to read a file line-by-line into a list?, Stack Overflow. Available at: <https://stackoverflow.com/questions/327750/to-read-a-file-line-by-line-into-a-list> (Accessed: November 30, 2022).