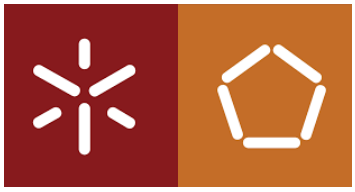


Universidade do Minho

Inteligência Artificial

Relatório do Trabalho Prático

Ano letivo 2023/2024

A100711, João Andrade Rodrigues

A100645, Mateus Lemos Martins

A100754, Rafael Vale da Costa Peixoto

A100713, Vicente Costa Martins

Avaliação pelos pares:

A100711 João DELTA = 0.5

A100645 Mateus DELTA = 0

A100754 Rafael DELTA = 0

A100713 Vicente DELTA = -0.5

Índice

1. Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Inteligência Artificial da Licenciatura em Engenharia Informática da Universidade do Minho, e teve como objetivo a formalização e resolução de problemas através da implementação de algoritmos de procura eficientes. Enquadrado na temática da sustentabilidade, este trabalho prático visa o desenvolvimento de algoritmos de procura aplicados à otimização do meio de entrega de encomendas pela empresa de distribuição Health Planet. Ao longo deste relatório, exploraremos a formalização do problema proposto, discutiremos a implementação de algoritmos de procura específicos e analisaremos as implicações dessas soluções no contexto da sustentabilidade ambiental.

2. Formulação do Problema

Uma vez que o problema se encontra num ambiente determinístico e completamente observável, em que o agente "sabe" exatamente o estado em que estará e as soluções pretendidas são sequências, podemos afirmar que este é um problema de estado único. Além disso, a sua formulação pode ser efetuada da seguinte forma:

- **Representação do estado:** Grafo não orientado, em que cada nodo representa uma posição possível no Grafo e cada aresta representa o trajeto entre posições.

- **Estado inicial**: A posição inicial representa a sede da Health Planet ("Rua da Fonte Longa, Mogege).

- **Estado/teste objetivo:** Todas as encomendas associadas a um estafeta devem ter sido entregues, não restando assim encomendas associadas a estafetas.

- **Operadores:** Deslocação de um Nodo para outro contando que o Grafo apresenta uma ligação entre os mesmos.

- **Solução:** Um caminho válido (sequência de posições percorridas) que comece na posição inicial e termine numa das posições finais (rua de uma das encomendas) contando que já tenha passado por todas as outras posições finais.

- **Custo da solução:** Distância total do percurso feito pelo agente (em metros).

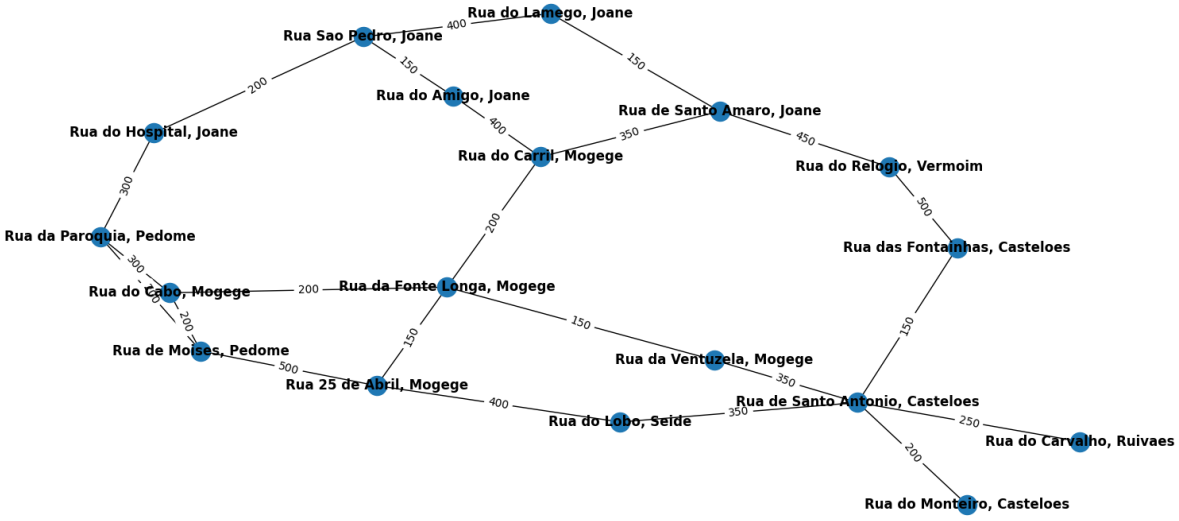
3. Criação do Grafo

O Grafo é criado através deste código que adiciona uma aresta (ligação) entre dois Nodos, acrescentando também o custo associado à passagem entre um Nodo e outro.

```
g = Grafo()

g.add_edge( node1: "Rua do Hospital, Joane", node2: "Rua da Paroquia, Pedome", weight: 300)
g.add_edge( node1: "Rua do Cabo, Mogege", node2: "Rua da Paroquia, Pedome", weight: 300)
g.add_edge( node1: "Rua de Moises, Pedome", node2: "Rua da Paroquia, Pedome", weight: 200)
g.add_edge( node1: "Rua do Cabo, Mogege", node2: "Rua de Moises, Pedome", weight: 200)
g.add_edge( node1: "Rua 25 de Abril, Mogege", node2: "Rua de Moises, Pedome", weight: 500)
g.add_edge( node1: "Rua 25 de Abril, Mogege", node2: "Rua da Fonte Longa, Mogege", weight: 150)
g.add_edge( node1: "Rua do Cabo, Mogege", node2: "Rua da Fonte Longa, Mogege", weight: 200)
g.add_edge( node1: "Rua da Ventuzela, Mogege", node2: "Rua da Fonte Longa, Mogege", weight: 150)
g.add_edge( node1: "Rua do Carril, Mogege", node2: "Rua da Fonte Longa, Mogege", weight: 200)
g.add_edge( node1: "Rua do Carril, Mogege", node2: "Rua do Amigo, Joane", weight: 400)
g.add_edge( node1: "Rua Sao Pedro, Joane", node2: "Rua do Amigo, Joane", weight: 150)
g.add_edge( node1: "Rua Sao Pedro, Joane", node2: "Rua do Hospital, Joane", weight: 200)
g.add_edge( node1: "Rua Sao Pedro, Joane", node2: "Rua do Lamego, Joane", weight: 400)
g.add_edge( node1: "Rua de Santo Amaro, Joane", node2: "Rua do Lamego, Joane", weight: 150)
g.add_edge( node1: "Rua de Santo Amaro, Joane", node2: "Rua do Carril, Mogege", weight: 350)
g.add_edge( node1: "Rua de Santo Amaro, Joane", node2: "Rua do Relogio, Vermoim", weight: 450)
g.add_edge( node1: "Rua das Fontainhas, Casteloes", node2: "Rua do Relogio, Vermoim", weight: 500)
g.add_edge( node1: "Rua das Fontainhas, Casteloes", node2: "Rua de Santo Antonio, Casteloes", weight: 150)
g.add_edge( node1: "Rua do Carvalho, Ruivaes", node2: "Rua de Santo Antonio, Casteloes", weight: 250)
g.add_edge( node1: "Rua da Ventuzela, Mogege", node2: "Rua de Santo Antonio, Casteloes", weight: 350)
g.add_edge( node1: "Rua do Lobo, Seide", node2: "Rua de Santo Antonio, Casteloes", weight: 350)
g.add_edge( node1: "Rua do Monteiro, Casteloes", node2: "Rua de Santo Antonio, Casteloes", weight: 200)
g.add_edge( node1: "Rua do Lobo, Seide", node2: "Rua 25 de Abril, Mogege", weight: 400)
```

Esta é uma representação gráfica do nosso circuito:



Como podemos observar, não há nenhum Nodo que não esteja ligado a outro o que torna possível alcançar qualquer um deles.

4. Criação dos Estafetas e das Encomendas

Tanto os Estafetas como as Encomendas são criadas através do parse a um ficheiro txt ("Estafetas.txt" e "Entregas.txt" respetivamente).

```
## parse aos estafetas
with open('Estafetas.txt', 'r') as arquivo:
    linhas = arquivo.readlines()

    # variavel que vai guardar a lista de encomendas
lista_estafetas = []

for linha in linhas:
    tokens = linha.split(';')
    id = int(tokens[0])
    transporte = tokens[1]
    sum_avaliacoes = float(tokens[2])
    nr_avaliacoes = int(tokens[3])
    estafeta = Estafeta(id,transporte,sum_avaliacoes,nr_avaliacoes,[])
    lista_estafetas.append(estafeta)

print("ESTAFETAS: ")
for estafeta in lista_estafetas:
    print(
        f"Id: {estafeta.id}, Transporte: {estafeta.transporte}")
```

Este excerto de código é o que cria os nossos Estafetas. Para serem criadas as encomendas é utilizada a mesma metodologia.

5. Atribuição de Encomendas a cada Estafeta

Para as Encomendas serem distribuídas elas devem ser atribuídas a um Estafeta que vai fazer essa entrega. Um dos requisitos do projeto era fazer as entregas de uma forma sustentável/ecológica. Então demos prioridade à atribuição de Encomendas a Estafetas que utilizassem o transporte bicicleta. Porém também não queremos que as encomendas cheguem atrasadas, assim só encomendas associadas à mesma freguesia onde é a sede da empresa devem ser atribuídas a bicicletas. De notar que cada bicicleta só pode levar no máximo 5kg e que por cada kg transportado sofre um decréscimo de 0.6 km/h na velocidade.

```
# Atribuir encomendas às bicicletas
for estafeta in lista_estafetas:
    if estafeta.transporte == "bicicleta":
        encomendas_a_remover = []  # Lista auxiliar para armazenar encomendas a serem removidas
        for encomenda in lista_encomendas:
            if "Mogege" in encomenda.rua_destino and estafeta.transporte_peso_atual + encomenda.peso <= 5:
                estafeta.transporte_peso_atual += encomenda.peso
                estafeta.encomendas.append(copy.copy(encomenda))
                # multiplica por 1000 e divide por 60 para passar de km/h para metros/minuto
                estafeta.transporte_velocidade = ((10 - (0.6 * estafeta.transporte_peso_atual)) * 1000) / 60
                encomendas_a_remover.append(encomenda)
                print(f"ENCOMENDA: {encomenda.rua_destino} ADICIONADA A ESTAFETA {estafeta.id} (BICICLETA)")

        # remove as encomendas que já foram atribuídas da lista de encomendas disponíveis
        for encomenda in encomendas_a_remover:
            lista_encomendas.remove(encomenda)
```

Com o restante das encomendas, distribuímo-las por Estafetas que possuem motas, visto que são mais económicas que os carros. De notar que cada mota só pode levar no máximo 20kg e que por cada kg transportado sofre um decréscimo de 0.5 km/h na velocidade.

```
# Atribuir encomendas às motas
for estafeta in lista_estafetas:
    if estafeta.transporte == "mota":
        encomendas_a_remover = []  # Lista auxiliar para armazenar encomendas a serem removidas
        for encomenda in lista_encomendas:
            if estafeta.transporte_peso_atual + encomenda.peso <= 20:
                estafeta.transporte_peso_atual += encomenda.peso
                estafeta.encomendas.append(encomenda)
                # multiplica por 1000 e divide por 60 para passar de km/h para metros/minuto
                estafeta.transporte_velocidade = ((35 - (0.5 * estafeta.transporte_peso_atual)) * 1000) / 60
                encomendas_a_remover.append(encomenda)
                print(f"ENCOMENDA: {encomenda.rua_destino} ADICIONADA A ESTAFETA {estafeta.id} (MOTA)")

        # remove as encomendas que já foram atribuídas da lista de encomendas disponíveis
        for encomenda in encomendas_a_remover:
            lista_encomendas.remove(encomenda)
```

Se ainda restarem encomendas, estas devem ser distribuídas pelos carros, que podem levar no máximo 100 kg e que por cada kg transportado sofre um decréscimo de 0.1 km/h na velocidade.

```
# Atribuir encomendas aos carros
for estafeta in lista_estafetas:
    if estafeta.transporte == "carro":
        encomendas_a_remover = []  # Lista auxiliar para armazenar encomendas a serem removidas
        for encomenda in lista_encomendas:
            if estafeta.transporte_peso_atual + encomenda.peso <= 100:
                estafeta.transporte_peso_atual += encomenda.peso
                estafeta.encomendas.append(encomenda)
                # multiplica por 1000 e divide por 60 para passar de km/h para metros/minuto
                estafeta.transporte_velocidade = ((50 - (0.1 * estafeta.transporte_peso_atual)) * 1000) / 60
                encomendas_a_remover.append(encomenda)
                print(f"ENCOMENDA: {encomenda.rua_destino} ADICIONADA A ESTAFETA {estafeta.id} (CARRO)")

        # remove as encomendas que já foram atribuídas da lista de encomendas disponíveis
        for encomenda in encomendas_a_remover:
            lista_encomendas.remove(encomenda)
```

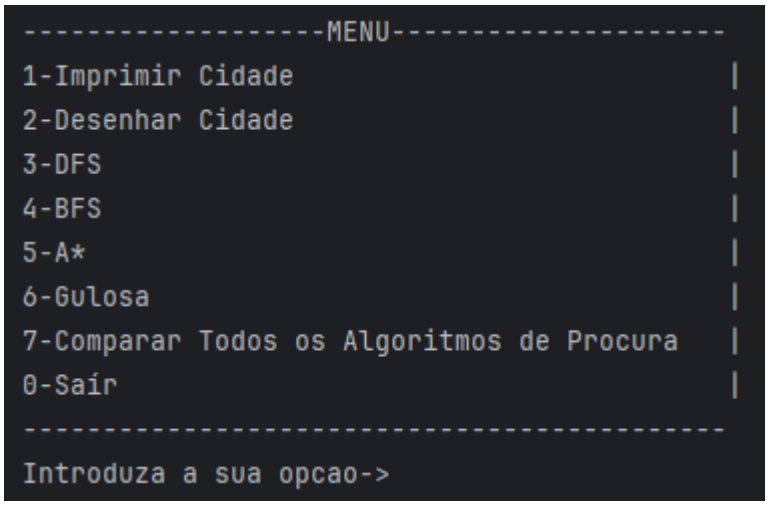
No final mostra as Encomendas restantes e as Encomendas associadas a cada Estafeta.

```
lista_enc_rest = []
for enc in lista_encomendas:
    lista_enc_rest.append(str(enc))
print(f"LISTA DE ENCOMENDAS RESTANTE: {lista_enc_rest}")

# Mostrar encomendas de cada estafeta
for estafeta in lista_estafetas:
    lista_enc = []
    for enc in estafeta.encomendas:
        lista_enc.append(str(enc))
    print(f"ESTAFETA: {estafeta.id}, ENCOMENDAS: {lista_enc}")
```

6. Funcionamento do programa

Ao compilar-mos o código é-nos apresentado o seguinte Menu:



Este Menu permite-nos:

1. **Imprimir Cidade:** Mostra todos os Nodos do Grafo e os possíveis próximos Nodos bem como o custo associado a cada travessia entre Nodos.
2. **Desenhar Grafo:** Exprime o Grafo de uma forma visual
3. **DFS:** Que aplica o algoritmo BFS (**Depth-First Search**), distribuindo todas as encomendas associadas a cada estafeta, no final retorna o caminho mais curto obtido pelo algoritmo que permite a entregas de todas as encomendas (ou seja, um caminho que passe por todas as ruas-destino), o custo associado a esse caminho e caminho percorrido ao longo da execução, a metodologia para encontrar a solução será mais tarde explicada.
4. **BFS:** Que aplica o algoritmo BFS (**Breadth-First Search**), distribuindo todas as encomendas associadas a cada estafeta, no final retorna o caminho mais curto obtido pelo algoritmo que permite a entregas de todas as encomendas (ou seja, um caminho que passe por todas as ruas-destino), o custo associado a esse caminho e caminho percorrido ao longo da execução, a metodologia para encontrar a solução será mais tarde explicada.
5. **A*:** Que aplica o algoritmo A* (**A-Star**), distribuindo todas as encomendas associadas a cada estafeta, no final retorna o caminho mais curto obtido pelo algoritmo que permite a entregas de todas as encomendas (ou seja, um caminho que passe por todas as ruas-destino), o custo associado a esse caminho e caminho percorrido ao longo da execução, a metodologia para encontrar a solução será mais tarde explicada.
6. **Gulosa:** Que aplica o algoritmo Gulosa (**Greedy**), distribuindo todas as encomendas associadas a cada estafeta, no final retorna o caminho mais curto obtido pelo algoritmo que permite a entregas de todas as encomendas (ou seja, um caminho que passe por todas as ruas-destino), o custo associado a esse caminho e caminho percorrido ao longo da execução, a metodologia para encontrar a solução será mais tarde explicada.
7. **Comparar Todos os Algoritmos de Procura:** O nome já é bastante explicativo por si só, porém o que faz é aplicar todos os algoritmos ao mesmo tempo o que nos permite uma fácil comparação para assim sabermos que algoritmo resulta melhor com base nas Encomendas e no Grafo que possuímos.
0. **Sair:** Permite-nos sair da execução.

7. Estratégia de procura

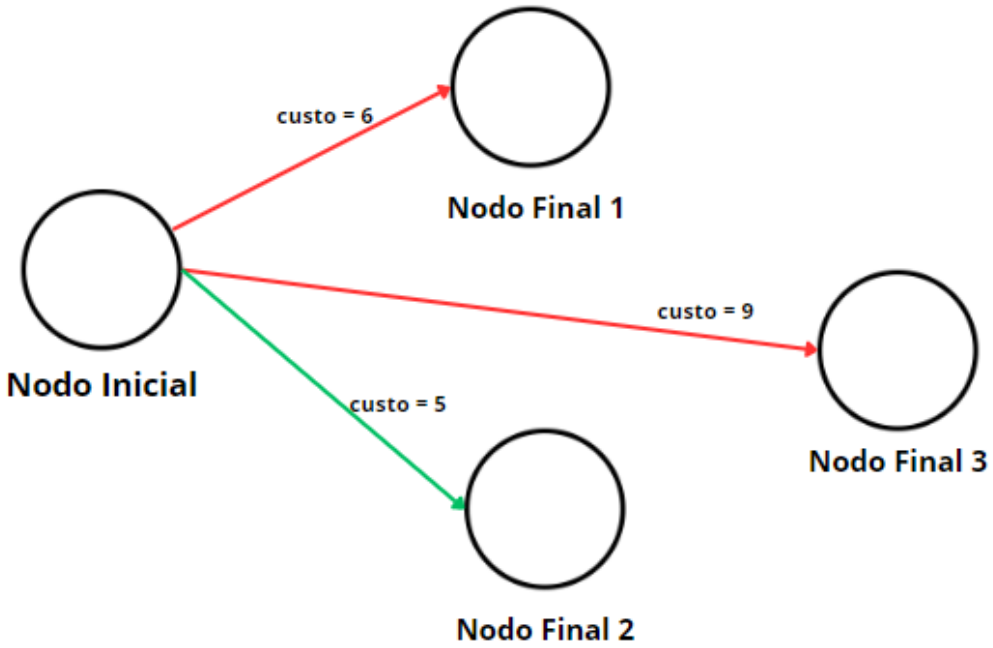
Visto que o nosso objetivo é encontrar uma solução de caminho que passe por um ou vários pontos, escolhemos utilizar a seguinte estratégia:

Imaginemos que um Estafeta tem associado a si 3 encomendas, com destino em: Nodo Final 1, Nodo Final 2 e Nodo Final 3. É aplicado o algoritmo em questão (pode ser: **BFS**, **DFS**, **A*** ou **Gulosa**) para cada Nodo Final e é guardado o caminho para chegar a esse Nodo e o custo num dicionário dedicado a guardar as informações de travessias através desse algoritmo (dicBFS, dicDFS, dicAstar, dicGulosa).

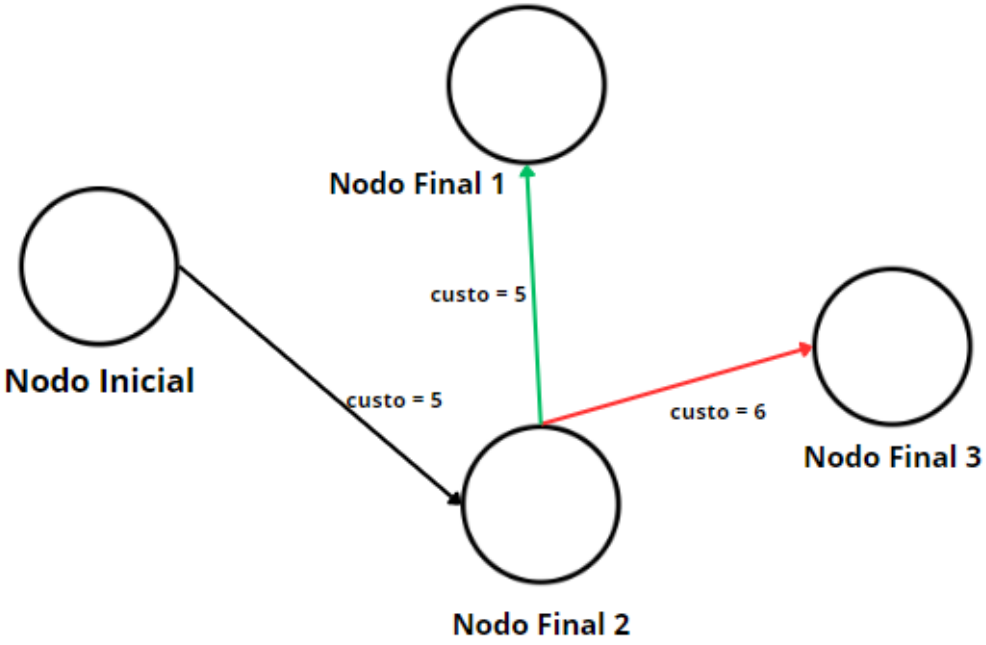
Depois, escolhemos a travessia que apresenta menor custo para ser a travessia que será executada. Em seguida faremos o mesmo processo, porém desta vez teremos menos um Nodo a ser encontrado, e o Nodo de onde vamos calcular a próxima travessia será o Nodo encontrado. Ao longo de todo o processo vamos guardando o caminho percorrido, o custo da solução e o caminho total percorrido que será posteriormente impresso para o usuário.

O desempenho do programa será melhor de cada vez que é executado uma vez que utiliza os dicionários, o que permite que nem sempre que seja preciso fazer uma comparação entre custos de caminhos os algoritmos sejam chamados, visto que já possui essa informação de travessias anteriores.

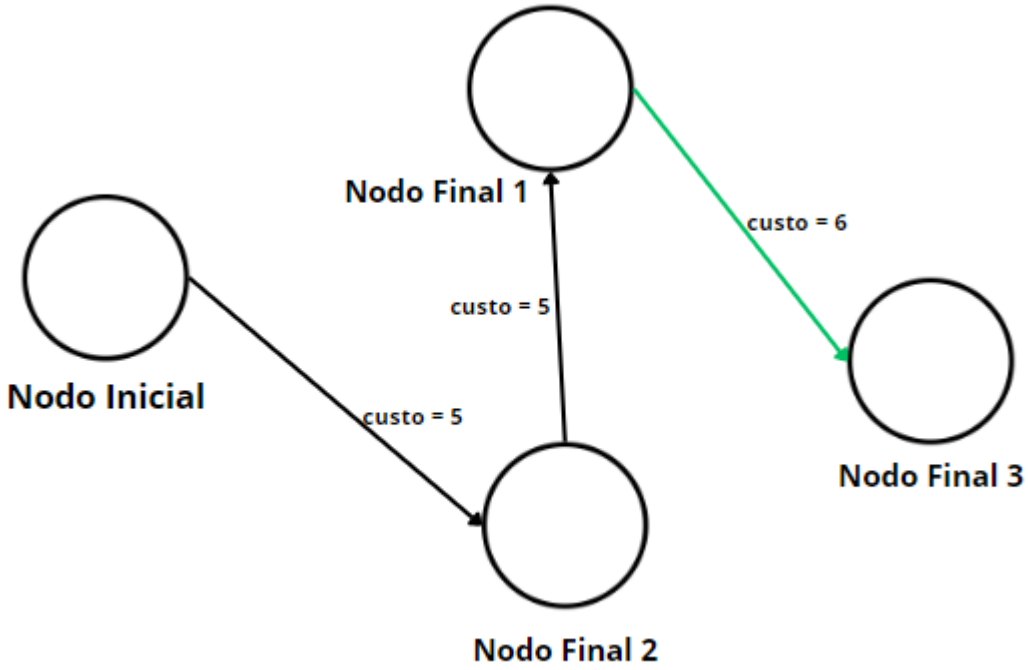
A seguir são apresentados alguns diagramas que pretendem demonstrar de uma forma gráfica a nossa estratégia de procura.



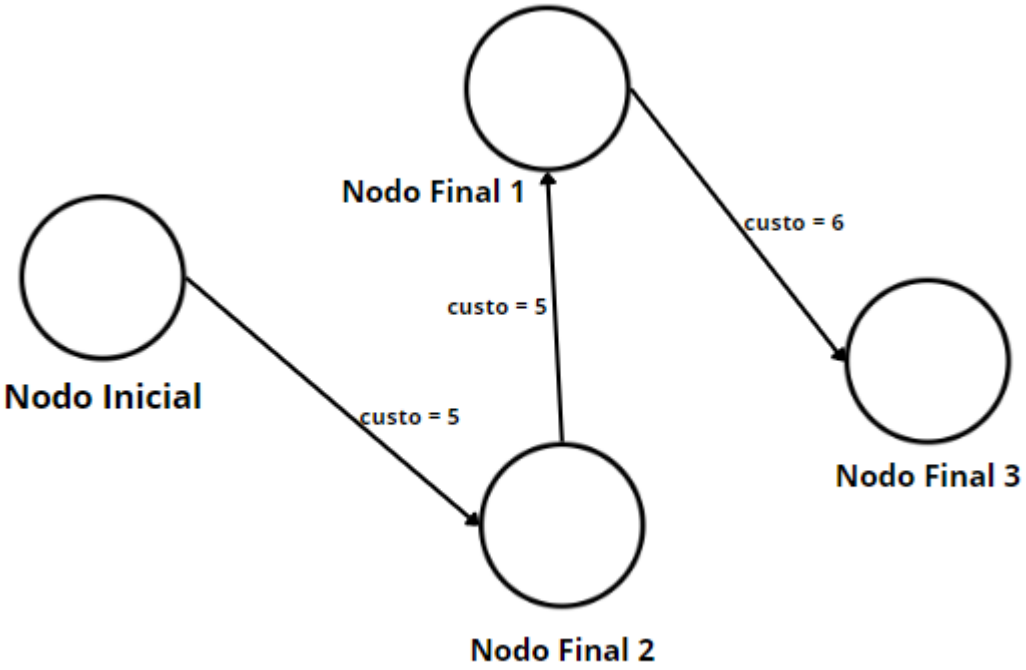
Neste primeiro diagrama podemos observar que existem 3 Nodes Finais a serem encontrados, assim se o custo para chegar a cada um deles não estiver disponível no dicionário do algoritmo ele calcula esse custo, os caminhos a vermelho representam caminhos calculados que não têm o menor custo, e o verde o com menor custo, sendo esse que deve ser seguido.



Neste diagrama já podemos observar que o caminho entre Node Inicial e Node Final 2 foi o caminho seguido. Foram guardadas informações sobre os Node onde passaram para chegar de um Node a outro (caminho) assim como o custo. E fez-se o mesmo processo apresentado no diagrama anterior, mas neste caso com o início a ser em Node Final 2 e os Nodes a serem encontrados com Node Final 1 e Node Final 3.



Semelhante ao processo dos outros diagramas apresentados, é guardado o caminho para chegar ao Node Final 1 a partir de Node Final 2 assim como o custo associado, e é calculado o próximo Nodo a ser encontrado, neste caso já só existe Node Final 3 para encontrar sendo então esse o próximo.



No final, teremos o caminho total que permite passar por todos os Nodos, o custo associado a este caminho e o caminho percorrido ao longo da execução que serão impressos para o utilizador.

8. Estratégias de procura não informada

As estratégias de procura não informadas que decidimos implementar neste projeto foram as seguintes: **Depth-First Search (DFS)** e **Breadth-First Search (BFS).**

8.1. Procura em profundidade (DFS)

A procura em profundidade, que consiste em expandir sempre um dos nodos mais profundos do grafo, destaca-se pelo uso eficiente de memória e adequação a problemas com múltiplas soluções. Contudo, mostra-se menos eficaz em grafos profundos com poucas soluções, e a solução obtida pode não ser a ótima. Numa comparação mais a frente poderemos observar que irá ser o algoritmo que normalmente apresenta as piores soluções. A implementação do algoritmo de procura em profundidade neste contexto impede a reexploração de nodos já visitados, evitando a ocorrência de loops.

8.1.1 Exemplo de procura em profundidade (DFS)

```
ENTREGAS A SEREM FEITAS EM: ['Rua do Cabo, Mogege', 'Rua do Carril, Mogege']
Encomenda Chegou ao Destino: Rua do Cabo, Mogege
Entrega não chegou a tempo!
Encomenda Chegou ao Destino: Rua do Carril, Mogege
Entrega Não Chegou a Tempo!
PATH: [['Rua da Fonte Longa, Mogege', 'Rua 25 de Abril, Mogege', 'Rua de Moises, Pedome', 'Rua da
CUSTO TOTAL: 2500
NODOS VISITADOS: [{'Rua do Carril, Mogege', 'Rua do Lobo, Seide', 'Rua da Paroquia, Pedome', 'Rua
Entregas Bem Sucedidas!
```

Inicialmente são impressos os Nodes onde devem ser feitas as entregas, neste caso “Rua do Cabo, Mogege” e “Rua do Carril, Mogege”. Quando chega a um dos destinos na encomenda, imprime que a Encomenda chegou ao destino e posteriormente se chegou ou não a tempo, neste caso ambas as encomendas não chegam a tempo (dentro do prazo de entrega).

Quando todas as Encomendas são entregues é apresentado o path (caminho) utilizado pelo Estafeta para as distribuir, o custo desse caminho (metros percorridos) e os Nodos visitados, que em certas situações podem até não existir, devido ao facto da utilização do dicionário de cada algoritmo que já foram apresentados anteriormente.

8.2. Procura em largura (BFS)

A procura em largura, prioriza a expansão dos nodos de menor profundidade no grafo, destaca-se pela obtenção de soluções ótimas quando todas as arestas têm custo 1. No entanto, apresenta desvantagens significativas, como um tempo de pesquisa elevado, percorrendo mais nodos do que necessário, e um consumo considerável de espaço em memória. No contexto do problema em questão, este algoritmo comporta-se bastante bem muitas vezes encontrando soluções ótimas.

8.1.1 Exemplo de procura em largura (BFS)

```
ENTREGAS A SEREM FEITAS EM: ['Rua do Cabo, Mogege', 'Rua do Carril, Mogege']
Encomenda Chegou ao Destino: Rua do Carril, Mogege
Entrega chegou a tempo!
Encomenda Chegou ao Destino: Rua do Cabo, Mogege
Entrega Chegou a Tempo!
PATH: [['Rua da Fonte Longa, Mogege', 'Rua do Carril, Mogege'], ['Rua do Carril, Mogege', 'Rua da Fonte
CUSTO TOTAL: 600
NODOS VISITADOS: [{'Rua da Fonte Longa, Mogege', 'Rua do Carril, Mogege', 'Rua de Moises, Pedome', 'Rua
Entregas Bem Sucedidas!
```

Foi corrido o algoritmo para o mesmo conjunto de dados onde foi corrido o algoritmo anterior (DFS) e é evidente uma diferença na eficácia. Desta vez, todas as encomendas chegaram a tempo ao seu destino e o caminho percorrido foi cerca de 4x mais curto.

9. Estratégias de procura informada

As estratégias de procura informada implementadas neste projeto foram as seguintes: **Gulosa (Greedy)** e a **Procura A***.

9.1. Heurística

Para a utilização destes algoritmos de procura precisávamos da implementação de uma heurística adequada. Como estávamos a falar de entregas entre um ponto e outro optamos por usar a distância em linha reta entre dois pontos.

```
# Heuristicas da Rua da Fonte Longa, Mogege
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua da Ventuzela, Mogege", 150)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua do Carril, Mogege", 200)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua do Cabo, Mogege", 200)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua 25 de Abril, Mogege", 150)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua do Amigo, Joane", 500)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua de Santo Amaro, Joane", 550)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua do Lamego, Joane", 700)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua Sao Pedro, Joane", 650)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua do Hospital, Joane", 1050)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua da Paroquia, Pedome", 550)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua de Moises, Pedome", 300)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua do Lobo, Seide", 400)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua de Santo Antonio, Casteloes", 450)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua do Monteiro, Casteloes", 700)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua do Carvalho, Ruivaes", 700)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua das Fontainhas, Casteloes", 650)
g.add_heuristica2("Rua da Fonte Longa, Mogege","Rua do Relogio, Vermoim", 650)
```

Na imagem anterior podemos ver como foram implementadas as heurísticas, fornecemos informação sobre a distância entre qualquer dois Nodos do Grafo (imagem só representa as heurísticas da “Rua da Fonte Longa, Mogege”). Agora com o projeto acabado, olhamos para trás e conseguimos concluir que foi uma estratégia muito pouco eficiente especialmente a nível de recursos humanos, visto que poderíamos simplesmente ter adicionado uma coordenada a cada Nodo e calcular assim a distância em linha reta entre dois deles (heurística).

9.2. Procura Gulosa (Greedy)

A procura Gulosa (greedy) expande para o nodo que aparenta estar mais próximo da solução, utilizando uma heurística. Apresenta a vantagem de reduzir o tempo de procura se a função heurística for eficaz. Contudo, as desvantagens incluem a possibilidade de não atingir a solução ótima e um consumo elevado de espaço em memória. No contexto do problema em análise, este algoritmo comporta-se bastante bem, encontrando a solução ótima grande parte das vezes.

9.2.1 Exemplo de utilização da procura Gulosa (greedy)

```
ENTREGAS A SEREM FEITAS EM: ['Rua da Ventuzela, Mogege', 'Rua da Paroquia, Pedome', 'Rua do Lobo, Seide']
Encomenda Chegou ao Destino: Rua da Ventuzela, Mogege
Entrega chegou a tempo!
Encomenda Chegou ao Destino: Rua da Paroquia, Pedome
Entrega chegou a tempo!
Encomenda Chegou ao Destino: Rua do Lobo, Seide
Entrega Chegou a Tempo!
PATH: [['Rua da Fonte Longa, Mogege', 'Rua da Ventuzela, Mogege'], ['Rua da Ventuzela, Mogege', 'Rua da Fonte Longa
CUSTO TOTAL: 1900
NODOS VISITADOS: [{'Rua de Santo Antonio, Casteloes', 'Rua de Santo Amaro, Joane', 'Rua do Cabo, Mogege', 'Rua 25 d
Entregas Bem Sucedidas!
```

Como podemos observar nesta imagem, o algoritmo encontrou um caminho muito próximo do caminho ideal, encontrou 1900 e o caminho ideal é 1850. Que será demonstrado no exemplo de procura A*.

9.3. Procura A*

Este tipo de procura evita expandir caminhos que são dispendiosos, combinando para isso o algoritmo de procura gulosa com o algoritmo de procura uniforme. Utiliza então a seguinte função para a escolha do nodo a ser explorado de seguida: $f(n) = g(n) + h(n)$

Onde g(n) é o custo acumulado e h(n) é custo estimado para chegar ao destino (heurística).

Este algoritmo fica substancialmente mais eficiente conforme a heurística utilizada.

9.3.1 Exemplo de utilização da procura A*

```
ENTREGAS A SEREM FEITAS EM: ['Rua da Ventuzela, Mogege', 'Rua da Paroquia, Pedome', 'Rua do Lobo, Seide']
Encomenda Chegou ao Destino: Rua da Ventuzela, Mogege
Entrega chegou a tempo!
Encomenda Chegou ao Destino: Rua da Paroquia, Pedome
Entrega chegou a tempo!
Encomenda Chegou ao Destino: Rua do Lobo, Seide
Entrega Chegou a Tempo!
PATH: [['Rua da Fonte Longa, Mogege', 'Rua da Ventuzela, Mogege'], ['Rua da Ventuzela, Mogege', 'Rua da Fonte Lo
CUSTO TOTAL: 1850
NODOS VISITADOS: [{'Rua de Santo Amaro, Joane', 'Rua de Moises, Pedome', 'Rua do Amigo, Joane', 'Rua do Carril,
Entregas Bem Sucedidas!
```

Como podemos observar na imagem anterior, o algoritmo encontra o caminho ideal. Sendo claramente este o melhor algoritmo que temos à disposição neste projeto.

10. Entregas

Depois de uma Encomenda ser entregue é calculada uma avaliação para a entrega, se tiver chegado em até metade do prazo de entrega é atribuída nota 5, se tiver chegado a tempo, mas não em metade do tempo é atribuída nota 3 e se a encomenda tiver chegado atrasada então é atribuída nota 1. E a entrega é escrita em "Entregas.txt", um arquivo que possui para cada entrega, rua destino, peso, volume, preço, avaliação, meio de transporte, prazo de entrega, tempo de entrega, id do estafeta e classificação. Este arquivo pode-se mostrar bastante vantajoso no futuro para fazermos algumas querys, como por exemplo mostrar a evolução da classificação de um estafeta ao longo de um percurso de entregas ou até mesmo para calcular a taxa de encomendas que chegam a tempo. A imagem a seguir mostra como foi implementado.

```
print(f"Encomenda Chegou ao Destino: {e.rua_destino}")
# calcular tempo de entrega da encomenda e atribui a classificação ao estafeta associada
distancia = custoTotal
prazo = e.prazo_entrega
velocidade = estafeta.transporte_velocidade
tempo_entrega = distancia/velocidade
# ver se a encomenda esta atrasada
if prazo > tempo_entrega:
    print("Entrega chegou a tempo!")
    # se a encomenda chegar antes de metade do prazo de entrega atribui nota 5
    if prazo > tempo_entrega/2:
        avaliacao = 5
    # se chegar a tempo mas não em metade do prazo atribui 3
    else:
        avaliacao = 3
if prazo < tempo_entrega:
    print("Entrega não chegou a tempo!")
    # se chegar fora do prazo de entrega atribui nota 1
    avaliacao = 1
# atualiza a avaliação do estafeta
estafeta.avaliacao_total += avaliacao
estafeta.nr_avaliacoes += 1
# escreve em ficheiro a entrega da encomenda
with open('Entregas.txt','a') as arquivo:
    arquivo.write(f"Rua Destino: {e.rua_destino}; Peso: {e.peso}kg; Volume: {e.volume}cm2;
# remove a encomenda da lista de encomendas do estafeta uma vez que já foi entregue
estafeta.encomendas.remove(e)
```

No final de uma entrega faz-se também a atualização da velocidade do veículo visto que como tem menos encomendas, tem menos peso e, portanto, maior velocidade. O excerto de código seguinte representa como foi implementada esta atualização.

```
# atualiza a  velocidade do transporte uma vez que há menos peso
estafeta.transporte_peso_atual -= e.peso
if estafeta.transporte == "carro":
    estafeta.transporte_velocidade = ((50 - (0.1 * estafeta.transporte_peso_atual)) * 1000) / 60
if estafeta.transporte == "mota":
    estafeta.transporte_velocidade = ((35 - (0.5 * estafeta.transporte_peso_atual)) * 1000) / 60
if estafeta.transporte == "bicicleta":
    estafeta.transporte_velocidade = ((10 - (0.6 * estafeta.transporte_peso_atual)) * 1000) / 60
```

11. Conclusões e trabalho futuro

Em suma, consideramos que este projeto foi bastante desafiador. O trabalho foi desenvolvido de uma forma que não estávamos habituados, onde houve bastante reaproveitamento de código desenvolvido nas aulas práticas especialmente dos algoritmos de pesquisa. Tivemos também bastantes problemas durante a implementação do código que sustenta este projeto. Um deles foi a remoção de objetos de uma lista enquanto a estávamos a iterar que apesar de não impedir a execução do programa fez aparecer resultados que não estávamos a espera especialmente na distribuição de encomendas aos estafetas, não foi um erro muito difícil de resolver, porém estava bastante bem escondido e foi demorada a sua resolução.

Finalmente, gostaríamos de falar sobre a implementação da heurística, que apesar de considerarmos que seja uma heurística adequada ao problema, a forma com que foi implementada não foi a melhor, como já tínhamos referido anteriormente. Se recomeçássemos o projeto hoje seria claramente uma mudança que faríamos.