Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Departamento de Informática

Relatório do Trabalho Final

Métodos de Procura

Inteligência Artificial

Realizado por

Daniel Tejo 17743

João Pinto 17744

Marcelo Silva 17724

Daniel Bernardo 17918

Ricardo Sá 17722

Orientador: Filipe Pinto

Viseu, 2022

Índice

Índice de figuras

1. Introdução

Neste relatório será apresentado o projeto que foi desenvolvido na cadeira de Inteligência Artificial da Licenciatura de Engenharia Informática, na Escola Superior de Tecnologia e Gestão de Viseu.

Este trabalho consiste no desenvolvimento, na linguagem Python, de um agente que resolva os problemas que possam aparecer ao procurar/encontrar um caminho entre duas cidades diferentes, sendo que este caminho deve ser elaborado de acordo com as diferentes estratégias/algoritmos.

No desenvolvimento deste relatório, iremos apresentar o desenvolvimento do nosso projeto em Python, em que consiste esta linguagem e algumas explicações do nosso trabalho.

Concluindo iremos apresentar uma conclusão/reflexão do nosso trabalho elaborado nesta cadeira do 3º ano da Licenciatura de Engenharia Informática

2. Tecnologias e Ferramentas

Neste capítulo irá ser abordado o estado de arte das tecnologias utilizadas. É realizada uma análise à arquitetura geral do projeto assim como as possíveis soluções que possam solucionar o problema apresentado.

2.1. Linguagens/Ferramentas Utilizadas

2.1.1. Python

Neste capítulo irá ser feito uma apresentação da tecnologia Python, utilizada ao longo do desenvolvimento do trabalho.

A linguagem de programação Python foi criada em 1991 por Guido van Rossum, tendo como principais objetivos a produtividade e a legibilidade, ou seja, foi criada de modo a produzir um bom código, sendo também fácil de manter de uma forma rápida.

Tem características como o uso reduzido de caracteres especiais e de palavras-chaves voltadas para a compilação, a marcação por blocos, o uso de gerenciador automático de memória, o suporte de diversos padrões de programação e a biblioteca padrão extensa.

É uma linguagem multiplataforma, sendo isto uma das suas principais vantagens, pois programas escritos numa plataforma poderão, na sua maioria e sem problemas, ser executados noutras diferentes.

2.1.2. Framework Tkinter

Tkinter é uma biblioteca da linguagem Python que acompanha a instalação padrão e permite desenvolver interfaces gráficas.

Isso significa que qualquer computador que tenha o interpretador Python instalado é capaz de criar interfaces gráficas usando o Tkinter, com exceção de algumas distribuições Linux, exigindo que seja feita o download do módulo separadamente.

Um dos motivos de estarmos usando o Tkinter como exemplo é a sua facilidade de uso e recursos disponíveis. Outra vantagem é que é nativo da linguagem Python, tudo o que precisamos fazer é importá-lo no momento do uso, ou seja, estará sempre disponível.

3. Desenvolvimento

Neste capítulo vai ser explicado a estrutura e desenvolvimento do trabalho. Foi estruturado em dois parâmetros, o tratamento dos métodos de procura e o tratamento das cidades e distâncias.

3.1. Cidades

Neste parâmetro foi criada uma classe Cidade onde se desenvolveram 3 dicionários, sendo eles: CidadesVizinhas, LinhaReta e DistanciaReta.

3.1.1. Cidades Vizinhas

A Figura 1 apresenta o desenvolvimento do dicionário CidadesVizinhas.

Já dentro do dicionário em si (CidadesVizinhas = {}), podem ver-se as várias linhas do mesmo, sendo que para cada uma delas, são indicadas as fronteiras do primeiro elemento da linha, bem como a distância (peso) da primeira cidade de cada linha às suas fronteiras, por exemplo, para a cidade Viseu tem-se como fronteiras as cidades Aveiro, Guarda, Coimbra, Porto e Vila Real e as distâncias 95, 85, 96, 133 e 110, respetivamente.

```
def CidadesVizinhas(self):
    vizinhos = {
        'Aveiro': {'Porto':68, 'Viseu': 95, 'Coimbra':68, 'Leiria': 115},
        'Braga': {'Viana do Castelo': 48, 'Vila Real': 106, 'Porto':53},
        'Braganca': {'Vila Real': 137, 'Guarda':202},
        'Beja': {'Evora': 78, 'Faro':152, 'Setubal':142},
        'Castelo Branco': {'Coimbra':159,'Guarda':106,'Portalegre':80,'Evora':203},
        'Coimbra': {'Viseu':96,'Leiria':67,'Aveiro':68,'Castelo Branco':159},
        'Evora': {'Lisboa':150,'Santarem':117,'Portalegre':131,'Setubal':103,'Beja':78,'Castelo Branco':203},
        'Faro':  {'Setubal': 249,'Lisboa':299,'Beja':152},
        'Guarda': {'Vila Real':157,'Viseu':85,'Braganca':202,'Castelo Branco':106},
        'Leiria': {'Lisboa':129,'Santarem':70,'Aveiro':115,'Coimbra':67},
        'Lisboa': {'Santarem':78,'Setubal':50,'Evora':150,'Faro':299,'Leiria':129},
        'Portalegre': {'Castelo Branco':80,'Evora':131},
        'Porto': {'Viana do Castelo':71,'Vila Real':116,'Viseu':133, 'Aveiro': 68,'Braga':53},
        'Santarem': {'Evora':117,'Leiria':70,'Lisboa':78},
        'Setubal': {'Beja':142,'Evora':103,'Faro':249,'Lisboa':50},
        'Viana do Castelo': {'Braga':48,'Porto':71},
        'Vila Real': {'Viseu':110,'Braga':106,'Braganca':137, 'Guarda':157,'Porto':116},
        'Viseu': {'Aveiro':95,'Coimbra':96,'Guarda':85,'Porto':133,'Vila Real':110}
    }
    return vizinhos
```

Figura 1 - Cidades Vizinhas

3.1.2. Linha Reta

Na Figura 2, pode ver-se o desenvolvimento do dicionário LinhaReta.

Funciona da mesma forma que o dicionário explicado anteriormente, tendo como única diferença a ausência das distâncias (pesos), pois nos métodos cegos não são necessários, onde este dicionário irá ser utilizado.

```
def LinhaReta(self):
    linha = {
        'Aveiro': ['Porto', 'Viseu', 'Coimbra', 'Leiria'],
        'Braga': ['Viana do Castelo', 'Vila Real', 'Porto'],
        'Braganca': ['Vila Real', 'Guarda'],
        'Beja': ['Evora', 'Faro', 'Setubal'],
        'Castelo Branco': ['Coimbra', 'Guarda', 'Portalegre', 'Evora'],
        'Coimbra': ['Viseu', 'Leiria', 'Aveiro', 'Castelo Branco'],
        'Evora': ['Lisboa', 'Santarem', 'Portalegre', 'Setubal', 'Beja', 'Castelo Branco'],
        'Faro': ['Setubal', 'Lisboa', 'Beja'],
        'Guarda': ['Vila Real', 'Viseu', 'Braganca', 'Castelo Branco'],
        'Leiria': ['Lisboa', 'Santarem', 'Aveiro', 'Coimbra'],
        'Lisboa': ['Santarem', 'Setubal', 'Evora', 'Faro', 'Leiria'],
        'Porto': ['Viana do Castelo', 'Vila Real', 'Viseu', 'Aveiro', 'Braga'],
        'Vila Real': ['Viseu', 'Braga', 'Braganca', 'Guarda', 'Porto'],
        'Viseu': ['Aveiro', 'Guarda', 'Porto', 'Vila Real', 'Coimbra'],
        'Setubal': ['Beja', 'Evora', 'Faro', 'Lisboa'],
        'Viana do Castelo': ['Braga', 'Porto'],
        'Santarem': ['Evora', 'Leiria', 'Lisboa'],
        'Portalegre': ['Castelo Branco', 'Evora']
    }
    return linha
```

Figura 2 - Linha Reta

3.1.3. Distância Reta

Este dicionário (Figura 3), foi desenvolvido com o intuito de indicar a distância (peso) entre as cidades apresentadas e Faro, bidirecionalmente, que é a cidade predefinida, que se assume como destino para os métodos de Procura Sôfrega e A* (Heurísticos). Podemos dizer que a distância (peso) entre Viseu e Faro é de 363.

```
def DistanciaReta(self):
    distancia = {
        ('Faro', 'Aveiro'): 366,
        ('Faro', 'Braga'): 454,
        ('Faro', 'Braganca'): 487,
        ('Faro', 'Beja'): 99,
        ('Faro', 'Castelo Branco'): 280,
        ('Faro', 'Coimbra'): 319,
        ('Faro', 'Evora'): 157,
        ('Faro', 'Faro'): 0,
        ('Faro', 'Guarda'): 352,
        ('Faro', 'Leiria'): 278,
        ('Faro', 'Lisboa'): 195,
        ('Faro', 'Portalegre'): 228,
        ('Faro', 'Porto'): 418,
        ('Faro', 'Santarem'): 231,
        ('Faro', 'Setubal'): 168,
        ('Faro', 'Viana'): 473,
        ('Faro', 'Vila Real'): 429,
        ('Faro', 'Viseu'): 363,
        ('Aveiro', 'Faro'): 366,
        ('Braga', 'Faro'): 454,
        ('Braganca', 'Faro'): 487,
        ('Beja', 'Faro'): 99,
        ('Castelo Branco', 'Faro'): 280,
        ('Coimbra', 'Faro'): 319,
        ('Evora', 'Faro'): 157,
        ('Faro', 'Faro'): 0,
        ('Guarda', 'Faro'): 352,
        ('Leiria', 'Faro'): 278,
        ('Lisboa', 'Faro'): 195,
        ('Portalegre', 'Faro'): 228,
        ('Porto', 'Faro'): 418,
        ('Santarem', 'Faro'): 231,
        ('Setubal', 'Faro'): 168,
        ('Viana do Castelo', 'Faro'): 473,
        ('Vila Real', 'Faro'): 429,
        ('Viseu', 'Faro'): 363

    }
    return distancia
```

Figura 3 - Distância Reta

3.2. Métodos de Procura

Neste parâmetro foi onde se desenvolveram os métodos de procura em si, sendo dois deles cegos (não informados) e os restantes dois heurísticos (informados). Irão ser explicados, primeiro, os métodos cegos (profundidade limitada e custo uniforme) e de seguida os métodos heurísticos (procura sôfrega e A*).

3.2.1. Profundidade Limitada

O método de procura Profundidade Limitada é semelhante ao método Profundidade Primeiro. Tem como princípio a expansão do nó o mais profundo possível, sendo que não é considerado um método completo (encontra a solução quando esta existe), nem ótimo (encontra a solução de melhor qualidade aquando da existência de várias), com a adição de um limite de profundidade. Na Figura 4 está representada como a função é usada recursivamente.

```
def profundidade_limitada(inicio, fim, caminho, iteracao, limite, is_first):
    if is_first==True:
        print("\n**********Profundidade Limitada**********")
        caminho.clear()
    else:
        pass
    print('\nIteração-->', iteracao)
    print('A testar', inicio)
    caminho.append(inicio)
    if inicio == fim:
        print('Chegou ao destino')
        print('Caminho: ',caminho)
        return caminho
    print('Não é o nó final')
    if iteracao == limite:
        return False
    print('\nExpandindo o nó atual', inicio)
    for no in cidadesLigadas[inicio]:
        if profundidade_limitada(no, fim, caminho, iteracao+1, limite, is_first=False):
            return caminho
        caminho.pop()
    print('\nNão consegue chegar ao destino com esse limite, defina um novo')
    return False
```

Figura 4 - Profundidade Limitada

3.2.2. Custo Uniforme

O método de procura Custo Uniforme é uma modificação da procura em largura que consiste em expandir primeiro o nó raiz, depois todos os nós gerados por esse e assim sucessivamente até chegar ao nó final.

O método Custo Uniforme diferencia-se ao expandir sempre o nó na fronteira da árvore com menor custo. Se forem cumpridas certas condições, a solução encontrada será a mais barata.

Este método minimiza os custos do caminho, é ótimo e completo, no entanto pode ser muito ineficiente. Encontra-se representado na Figura 5.

```
def custo_uniforme(graph, inicio, fim, peso):

    print("\n**********Custo Uniforme**********\n")
    fronteira = PriorityQueue()
    fronteira.put((0,inicio))
    print('Iteração 0')
    print(inicio)
    pos = 1
    while True:
        ucs_w, no_atual = fronteira.get()

        if no_atual == fim:
            print('-'*100)
            print('No escolhido ' + fim)
            print('FIM')
            return
        print('-'*100)
        print('Iteração '+str(pos))
        for no in graph[no_atual]:
            fronteira.put((ucs_w + return_peso(no_atual, no, peso), no))
        print(sorted(fronteira.queue))
        if fronteira.queue[0][1] is not fim:
            print('Para abrir ---> ' + str(fronteira.queue[0][1] + '.'))
        pos = pos+1
```

Figura 5 - Custo Uniforme

3.2.3. Procura Sôfrega

O método de procura sôfrega tem como objetivo minimizar o custo estimado até se atingir o destino pretendido, assim, expande-se o nó mais próximo do destino pretendido primeiro. É possível estimar o custo com este método, mas não é possível determiná-lo de modo preciso. O algoritmo escolhe o menor valor entre o nó atual e nó objetivo (Faro). Não é um método ótimo, nem completo. A Figura 6 representa este método de procura.

```
def sofrega(graph, inicio, fim,reta):

    print('\n**********Procura Sofrega**********\n')
    fronteira = PriorityQueue()
    fronteira.put((0,inicio))
    print('Iteração 0')
    print(inicio)
    pos = 1
    valor = False
    while True:

        ucs_w, no_atual = fronteira.get()
        if valor:
            ucs_w = return_peso(no_atual, fim, reta)

        if no_atual == fim:
            print('-'*100)
            print('No escolhido ' + fim)
            print('FIM')
            return

        print('-'*100)
        print('Iteração '+str(pos))
        for no in graph[no_atual]:
            fronteira.put((return_peso(no, fim, reta), no))
        print(sorted(fronteira.queue))
        if fronteira.queue[0][1] is not fim:
            print('Para abrir ---> ' + str(fronteira.queue[0][1] + '.'))
        pos = pos+1
        valor = True
```

Figura 6 - Procura Sôfrega

3.2.4. A*

O método A* é possível ser criado através da combinação do método de procura sôfrega e do método de custo uniforme, assim obtém-se um método completo e ótimo, usufruindo das vantagens de cada um dos algoritmos anteriores. Tem como principal objetivo tentar expandir o nó que pertence ao caminho com um menor custo associado.

Baseia-se em f(n) = g(n) + h(n), sendo f(n) o custo estimado da solução mais barata passado por n, g(n) o custo do caminho e h(n) o custo estimado para o objetivo. Assim a melhor solução passa por escolher primeiro o nó com o valor mais baixo de f.

Este método está representado na Figura 7.

```
def a_estrela(graph, inicio, fim, peso, reta):
    print('\n********** A* **********\n')
    fronteira = PriorityQueue()
    fronteira.put((0,inicio))
    print('Iteração 0')
    print(inicio)
    pos = 1
    valor = False
    while True:

        ucs_w, no_atual = fronteira.get()
        if valor:
            ucs_w = ucs_w - return_peso(no_atual, fim, reta)

        if no_atual == fim:
            print('-'*100)
            print('No escolhido ' + fim)
            print('FIM')
            return

        print('-'*100)
        print('Iteração '+str(pos))
        for no in graph[no_atual]:
            fronteira.put((ucs_w + return_peso(no_atual, no, peso)+return_peso(no, fim, reta), no))
        print(sorted(fronteira.queue))
        if fronteira.queue[0][1] is not fim:
            print('Para abrir ---> ' + str(fronteira.queue[0][1] + '.'))
        pos = pos+1
        valor = True
```

Figura 7 - A*

3.3. Menu

O código fonte do menu da aplicação encontra-se na Figura 8.

Aqui é configurada a interface gráfica. Para isso foi definido uma cor de fundo, três caixas de *input* de forma que fosse introduzido tanto o nó inicial como o final e o limite para o método Profundidade Limitada, quatro botões para selecionar qual o método de procura e um botão para sair da aplicação.

```
var1 = StringVar()
var2 = StringVar()
var3 = IntVar()

welcome ="Bem vindo aos métodos de procura."
labelwelcome = Label(root, text=welcome)
labelwelcome.configure(bg="#90dea4")
labelwelcome.pack(pady=(5,10))

label2 = Label(text="Origem")
label2.configure(bg="#90dea4")
label2.pack(pady=(2,0))
e1 = Entry(textvariable=var1, width=40)
e1.pack(ipady=5, pady=(10,0), padx=(20,20))

label3 = Label(text="Destino")
label3.configure(bg="#90dea4")
label3.pack(pady=(2,0))
e2 = Entry(textvariable=var2, width=40)
e2.pack(ipady=5,pady=5, padx=(20,20))

label4 = Label(text="Limite (Profundidade Limitada)")
label4.configure(bg="#90dea4")
label4.pack(pady=(2,0))
e3 = Entry(textvariable=var3, width=40)
e3.pack(ipady=5,pady=5, padx=(20,20))

a_fim_faro = "Faro"
path = list()

button1 = Button(root, text="Profundidade Limitada", command=lambda:profundidade_limitada(var1.get(), var2.get(),path, 0, var3.get(), True), height=1, width=35)
button1.pack(pady=(10,0), padx=(30,30))
button2 = Button(root, text="Custo Uniforme", command=lambda:custo_uniforme(cidadeReta, var1.get(), var2.get(), pesos), height=1, width=35)
button2.pack(pady=(10,0), padx=(30,30))
button3 = Button(root, text="Sôfrega", command=lambda:sofrega(cidadeReta, var1.get(), a_fim_faro,cidadeRetaFaro), height=1, width=35)
button3.pack(pady=(10,0), padx=(30,30))
button4 = Button(root, text="A Estrela (A*)", command=lambda:a_estrela(cidadesLigadas, var1.get(), a_fim_faro, pesos,cidadeRetaFaro), height=1, width=35)
button4.pack(pady=(10,15), padx=(30,30))
button5 = Button(root, text="Sair", command=lambda:root.destroy(), height=1, width=25)
button5.pack(pady=(15,15), padx=(50,50))

imagem = PhotoImage(file="Mapa_Portugal1.png")
w = Label(root, image=imagem)
w.imagem = imagem
w.pack(pady=(0,15))
```

Figura 8 - Menu

4. Conclusão

Finalizado o Projeto, será feito uma breve conclusão ao trabalho efetuado.

Concluído agora a realização do trabalho, podemos afirmar que temos um maior conhecimento nesta área de Inteligência Artificial e mais especificamente sobre os métodos de procura de procura abordados neste trabalho.

Para a realização deste trabalho recorremos a pesquisa na internet para duvidas que surgiriam durante a realização do mesmo, entreajuda entre os elementos do grupo e sempre que necessário ao professor durante as aulas práticas e teóricas.

Finalizando, concluímos o trabalho proposto nesta disciplina com sucesso, numa área bastante interessante e com grande futuro na área da informática.

Bibliografia

http://www.google.pt/

https://www.devmedia.com.br/tkinter-interfaces-graficas-em-python/33956

https://pt.stackoverflow.com/

https://moodle.estgv.ipv.pt/course/view.php?id=4547