Uma Implementação dos Algoritmos para resolução do Problema do Caminho Mínimo

Fábio Cáritas Barrionuevo da Luz
Sistemas de Informação - Faculdade Católica do Tocantins

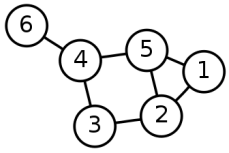
25 de maio de 2012

Sumário

Resumo

Este trabalho tem por objetivo relatar a experiência com implementação do Algoritmo de Dijkstra e Algoritmo de Bellman-Ford

1 Introdução



> Uma transportadora planeja fazer uma promoção de 50% de desconto para aumentar seus negócios. Só que para essa promoção não dar prejuízo, os caminhões devem passar somente pelas estradas de menor extensão. Para tanto, a equipe de negócios da transportadora deve descobrir quais são as rotas(percurso por estradas) de menor extensão que interligam todas as suas filiais.
> Mas há um problema, a transportadora possui 500 filiais, e existem mais de 5000 estradas diferentes que interligam direta e indiretamente todas essas 500 filiais. Como descobrir quais as melhores Rotas?

Esse tipo de problema pode ser resolvido com a utilização de Grafos. Um grafo é representado como um conjunto de pontos (vértices) que possuem ligação (as arestas).

Grafos podem ser utilizados para a resolução de diversos tipos de problemas, como por exemplo, o Grafo do Conhecimento[1][2], uma implementação do gigante de Buscas Google que visa melhorar a relevância do resultado das buscas relacionada a palavra chave pesquisada ou para o auxilio na gestão de tempo de projetos [3].

No caso do problema da transportadora, cada uma das 500 filiais representaria um vértice, e cada uma das estradas que interligam as filiais, representariam as arestas.

Problemas desse tipo são classificados como **"Problema do Caminho Mínimo"**. O problema do caminho mínimo consiste na minimização do custo de travessia de um grafo entre dois vértices, custo este dado pela soma dos pesos ou custo de cada aresta percorrida.

Uma aresta representa a ligação entre dois vértices, esta ligação por ter custo e sentido. Um Vértice pode representar o ponto de encontro de duas arestas.

Os algoritmos mais conhecidos para resolução do "Problema do Caminho Mínimo" são o Algoritmo de Dijkstra[4] e o algoritmo de Bellman-Ford[5]

2 Algoritmos de Dijkstra

Algoritmos de Dijkstra[4], foi concebido pelo cientista da computação holandês Edsger Dijkstra[7] em 1956 e publicado em 1959, servindo para localizar todos os caminhos mínimos de determinado vértice origem para todos os outros vértices alcançáveis do Grafo, somente se não houver caminhos(arestas) de custo negativo.

O Algoritmos de Dijkstra pode ser descrito da seguinte forma:

1. *Inicialização:*
   Criar dois vetores, "distancia" e "anterior", cujo o tamanho seja o numero

de vértices do Grafo G e uma lista “q”, que contenha todos os vértices pertencentes ao Grafo G.
Em seguida para todo vértice v pertencente ao Grafo G, fazer com que o vetor “distancia” na “posição v” receba um valor muito alto, fazer com que “anterior” na “posição v” receba valor nulo. Fazer com que “distancia” na posição “vértice origem” receba o valor 0.

2. *Processamento de Caminhos mínimos:*
Enquanto lista “q” não estiver vazia, obter do vetor “distancia” o vértice “u” com menor custo somente se o vértice “u” estiver contido na lista “q”.
Remover o vértice “u” da lista “q”.
Para cada “visinho” do vértice “u” (vértice em que “u” possui ligação), obter a soma ‘soma‘ de vetor “distancia” na posição “u” mais o custo da ligação de “u” com “visinho”.
Se “soma” for menor do que vetor “distancia” na posição “visinho”, fazer com que vetor “distancia” na posição “visinho” receba “soma” e vetor “anterior” na posição “visinho” receba “u”.

Para se obter o o percurso mínimo de um vértice origem até o vértice destino, basta iterar sobre o vetor “anterior”, começando da posição “destino” até quando vetor “anterior”, na posição “destino” for nulo,

3 *Chegar ao destino*
criar uma lista “percurso”,
Enquando vetor “anterior” na posição “destino” não for nulo, adicionar vértice “destino” ao final da lista “percurso”.
Fazer com que “destino” receba vetor “anterior” na posição “destino”
Quando o vetor “anterior” na posição “destino” for nulo, adicionar vértice origem” ao final da lista “percurso”.

Ao final da execução, a lista “percurso” conterá o caminho minimo partindo do vértice origem ate o vértice destino.

```
funcao Dijkstra(Grafo, origem):
    // Inicializacao
    para todo vertice v em Grafo:
        distancia[v] := Numero_Muito_Grande ;
        anterior[v] := undefined ;
    fim para ;
    distancia[origem] := 0;
    Q :=lista de todos os vertices do Grafo ;
    //Inicio do processamento
    enquanto Q nao eh vazio:
        u := vertice u de menor custo em distancia[] se u estiver
            contido em Q;
        se distancia[u] = Numero_Muito_Grande:
            break ; // Todos os vertices que sobraram em q sao
                inacessiveis
        fim se ;
        remova u de Q ;
        para todo visinho v of u://para cada visinho de u que esta
            contido em Q
            soma := distancia[u] + distancia_entre(u, v) ;
            se soma < distancia[v]:
                distancia[v] := soma ;
                anterior[v] := u ;
            fim se ;
```

```
        fim para ;
    fim enquanto ;
    retorne distancia[] e anterior[] ;
fim Dijkstra.
```

Obtendo o percurso entre o vértice origem e o vértice destino

```
funcao caminho_minimo_entre_vertices(Grafo, origem, destino):
    distancia[], anterior[] = Dijkstra(Grafo, origem)
    lista percurso
    enquanto anterior[destino] nao for nulo, faca:
        adicione anterior[destino] ao final da lista percurso
        destino = anterior[destino]
    fim enquanto
    adicione origem ao final da lista percurso
    retorne percurso

fim funcao caminho_minimo_entre_vertices
```

3 Algoritmo de Bellman-Ford

Algoritmos de Bellman-Ford[5], foi concebido pelo matemático americano Richard Ernest Bellman [9], servindo para localizar todos os caminhos mínimos de determinado vértice origem para todos os outros vértices alcançáveis do Grafo, incluindo os **caminhos(arestas) de custo negativo**.

```
funcao Bellman-Ford(Grafo, origem):

    // Inicializacao
    para todo vertice v em Grafo:
        distancia[v] := Numero_Muito_Grande ;
        anterior[v] := undefined ;
    fim para ;
    distancia[origem] := 0 ;

    //Inicio do processamento
    para i =1 ate numero total de vertices:
        para todo vertice u em Grafo:
            para todo vertice v em Grafo:
                soma := distancia[u] + distancia_entre(u, v) ;
                se distancia[v] > soma:
                    distancia[v] := soma ;
                    anterior[v] := u ;
                fim se ;
            fim para ;
        fim para ;
    fim para
    para todo vertice u em Grafo:
        para todo vertice v em Grafo:
            soma := distancia[u] + distancia_entre(u, v) ;
            se distancia[v] > soma:
                retorne Falso
            fim se ;
        fim para ;
    fim para ;
    retorno Verdadeiro
```

```
31 fim Bellman-Ford.
```

4 Testes

A validação da implementação ocorreu com conjuntos de testes contendo 8, 50, 250, 500, 750 e 1000 nós, e mais um com apenas 4 para verificar o tratamento de ciclo negativo. O tempo de execução para cada algoritmo, métrica adotada e número de nós é exibido nas tabelas a seguir. Os pontos de origem e destino são aleatórios para cada conjunto de nós.

A tabela 1 exibe os tempos de execução do algoritmo Bellman Ford e o caminho encontrado para cada conjunto de nós.

Numero de Vertices	Algoritmo de Bellman-Ford			
	1/C		Salto	
	Tempo	Caminho	Tempo	Caminho
8	0.000223875045776s	[1, 8, 7, 6]	0.000180959701538s	[1, 6]
50	0.0286750793457s	[10, 42, 43, 8, 39, 40]	0.0450778007507s	[10, 42, 43, 8, 39, 40]
250	4.9877448082s	[10, 24, 171, 220]	5.09902095795s	[10, 24, 171, 220]
500	46.3791968822s	[1, 33, 87, 133, 176, 350]	45.6265640259s	[1, 33, 87, 133, 176, 350]
750	179.693883181s	[1, 444, 2, 166, 181, 510, 368, 292, 700]	178.138207197s	[1, 444, 2, 166, 181, 510, 368, 292, 700]
1000	440.029853106s	[1, 624, 88, 823, 812, 318, 552, 29, 900]	438.587266207s	[1, 624, 88, 823, 812, 318, 552, 29, 900]

A tabela 2 exibe os tempos de execução do algoritmo Dijkstra e o caminho encontrado para cada conjunto de nós.

Numero de Vertices	Algoritmo de Dijkstra			
	1/C		Salto	
	Tempo	Caminho	Tempo	Caminho
8	0.00005006790161132812 5s	[1, 8, 7, 6]	0.00006103515625s	[1, 6]
50	0.000576972961426s	[10, 42, 43, 8, 39, 40]	0.000621795654297s	[10, 42, 43, 8, 39, 40]
250	0.0111138820648s	[10, 24, 171, 220]	0.0116069316864s	[10, 24, 171, 220]
500	0.0370750427246s	[1, 33, 87, 133, 176, 350]	0.0344271659851s	[1, 33, 87, 133, 176, 350]
750	0.270087957382s	[1, 444, 2, 166, 181, 510, 368, 292, 700]	0.27127289772s	[1, 444, 2, 166, 181, 510, 368, 292, 700]
1000	0.484181880951s	[1, 624, 88, 823, 812, 318, 552, 29, 900]	0.586689949036s	[1, 624, 88, 823, 812, 318, 552, 29, 900]

5 Conclusão

Uma das principais diferenças entre os algoritmos estudados nessa disciplina é o fato de o Dijkstra não realizar todas as comparações como é feito no Bellman-Ford. O primeiro realiza apenas o número de arestas, enquanto o segundo realiza o número de todas as arestas por arestas.

6 Bibliografia

Referências

[1] *http://googlediscovery.com/2012/05/16/google-anuncia-grafo-do-conhecimento/*, acessado em 24/05/2012

[2] *Video apresentando o Grafo do Conhecimento:*, http://www.youtube.com/watch?v=mmQl6VGvX-c acessado em 24/05/2012

[3] *Fernando Nogueira- Notas de Aula - Disciplina de Pesquisa Operacional*, - Estimativa de Três pontos - PERT/CMP - UFRRJ - Universidade Federal Rural do Rio de Janeiro ,acessado em 24/05/2012

[4] *Edsger W. Dijkstra. "A note on two problems in connection with graphs". [S.l.]:. Numerische Mathematik 1, 1959. 83–89 p.*,

[5] *Bellman, Richard (1958). "On a routing problem".* ,Quarterly of Applied Mathematics 16: 87–90.

[6] *Algoritmo de Dijkstra* , - Wikipedia, acessado em 24/05/2012

[7] *Edsger Wybe Dijkstra*,- Wikipedia, acessado em 24/05/2012

[8] *Algoritmo de Bellman-Ford*,- Wikipedia, acessado em 24/05/2012

[9] *Richard Ernest Bellman* ,- Wikipedia, acessado em 24/05/2012

7 Implementação em Python

Implementação

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
# Testado unicamente no Ubuntu 12.04 64bits, Python 2.7.3

from __future__ import print_function #import da nova versao do
    metodo print, compativel com python 3

import argparse
import random

import timeit
import time
import math

import pickle
import sys
import os


pygraph_instalado = True
try:
#    import gv
#
#    from pygraph.classes.graph import graph
#    from pygraph.classes.digraph import digraph
#    from pygraph.algorithms.searching import breadth_first_search
#    from pygraph.readwrite.dot import write


    #necessario para o metodo gerar_visualizacao()
    import pydot
    from PIL import Image # eh necessario instalacao do PIL com
        suporte a libjpej e libpng

except :
    print('pygraph , Python Imaging Library(PIL), pydot pode nao
        estar instalado')
    pygraph_instalado = False

class SupressorImpressao:

    def write(self, _in):
        pass



class Caminho(object):
    global ma
    global t

    #inicializador, em python voce nao precisa utilizar o
        construtor diretamente
    #voce simplesmente sobrescreve o metodo inicializador
    def __init__(self, numero_vertices=10, matriz={}, debug=False,
        iniciar_indices_em_zero=False, gerar_visul=False):
```

```
51
        self.iniciar_indices_em_zero = iniciar_indices_em_zero
53      if self.iniciar_indices_em_zero is False:
            #faz indices comecarem em 1 e nao em zero
55          self.numero_vertices = numero_vertices + 1
            self.indice_inicial = 1
57      else:
            self.numero_vertices = numero_vertices
59          self.indice_inicial = 0

61#         print('teset: ',self.numero_vertices)
        self.infinito = sys.maxint
63      self.lista_de_vertices = xrange(self.indice_inicial, self.
            numero_vertices)
        self.matriz = matriz
65      self.debug = debug
        self.ha_aresta_negativa = False
67      self.gerar_visul = gerar_visul


69
#
71#    def gerar_img(self):
#
73#        if pygraph_instalado:
#            # Graph creation
75#            gr = graph()
#
77#
#            # Add nodes and edges
79#            gr.add_nodes(["Portugal","Spain","France","Germany","
    Belgium","Netherlands","Italy"])
#            gr.add_nodes(["Switzerland","Austria","Denmark","
    Poland","Czech Republic","Slovakia","Hungary"])
81#            gr.add_nodes(["England","Ireland","Scotland","Wales"])
#
83#            gr.add_edge(edge=("Portugal", "Spain"), wt=22)
#            gr.add_edge(("Spain","France"), 50)
85#            gr.add_edge(("France","Belgium"))
#            gr.add_edge(("France","Germany"))
87#            # Draw as PNG
#            dot = write(gr)
89#            gvv = gv.readstring(dot)
#            gv.layout(gvv,'dot')
91#            gv.render(gvv,'png','europe.png')

93    #utiliza o pydot para gerar um grafo, utilizando a notacao do
          Graphviz
      def gerar_visualizacao(self, nome_arq=None, gerar_arq_dot=False
          , mater_nos_nao_utilizados=False):
95        if self.gerar_visul is True:
              if pygraph_instalado:
97                grafo = pydot.Dot(
                      'Grafo Direcionado',
99                    graph_type='digraph',
                      simplify=False,
101                   layout="dot",
                      #layout="fdp",
103                   #overlap=None,
                      overlap="scale",
105                   splines=True,
                      #splines=True,
107                   #size=10000,
```

```
                    #nojustify=True,
                    #repulsiveforce=0.5,
                    #target=4
                    #overlap="prism"
                    #sep=True
                )

                nos = {}
                arestas = []
                qnt = len(self.lista_de_vertices)
                if mater_nos_nao_utilizados:
                    for verticeX in self.lista_de_vertices:
                        nos[verticeX] = pydot.Node(name=verticeX, #
                            shape='doublecircle'
                        )


                for verticeX in self.lista_de_vertices:
                    for verticeY in self.lista_de_vertices:
                        if self.matriz[verticeX,verticeY] != 0:
                            if mater_nos_nao_utilizados:

                                aresta = pydot.Edge(src=nos[
                                    verticeX], dst=nos[verticeY],
                                    label=self.matriz[verticeX,
                                    verticeY],
                                    labelfontcolor='green', color='
                                        red', fontcolor ='green' ,
                                    repulsiveforce=0.5)
                            else:
                                aresta = pydot.Edge(src=str(
                                    verticeX), dst=str(verticeY),
                                    label=self.matriz[verticeX,
                                    verticeY],
                                    labelfontcolor='green', color='
                                        red', fontcolor ='green' ,
                                    repulsiveforce=0.5)

                            arestas.append(aresta)
                if mater_nos_nao_utilizados:
                    for no in nos.itervalues():
                        grafo.add_node(no)

                for aresta in arestas:
                    grafo.add_edge(aresta)
                n_arq = ''
                if nome_arq is None:
                    n_arq = ('%s.png' % len(self.lista_de_vertices)
                        )

                if '.png' not in n_arq:
                    n_arq = n_arq.join('.png')

                grafo.write_png(n_arq)
                if gerar_arq_dot:
                    grafo.write_dot(n_arq.replace('.png', '.dot') )
                #a = grafo.read_dot(('%s.dot' % len(self.
                    iterador_lista_vertices)) )
                #print(a)
                #im=Image.open( ('%s.png' % len(self.
                    lista_de_vertices)))
                #im.show()
```

```
159
    #gera um Grafo com numero de vertices
161     def gerarGrafoAleatorio(self, tamanho=None, nao_direcionado=
            True, ):
            if tamanho is not None:
163                 self.__init__(numero_vertices = tamanho, gerar_visul=
                        self.gerar_visul)

165             for i in self.lista_de_vertices:
                    for j in self.lista_de_vertices:
167                     self.matriz[i,j] = 0
            num_criacoes = 0
169             max_criacoes = len(self.lista_de_vertices) + len(self.
                    lista_de_vertices) + (len(self.lista_de_vertices)) /2
            for i in self.lista_de_vertices:
171                 for j in self.lista_de_vertices:
                        if i != j:
173                         numero_aleatorio = int(random.randrange(0, 100)
                                )

175                         if num_criacoes <  max_criacoes:

177                             if (numero_aleatorio % 3) ==0:#print
                                    numero_aleatorio
                                    if nao_direcionado is True:
179                                     self.matriz[i,j] = numero_aleatorio
                                        self.matriz[j,i] = numero_aleatorio
181                                 else:
                                        self.matriz[i,j] = numero_aleatorio
183                                 #matriz[j,i] = numero_aleatorio
                                    if self.debug is True:
185                                     print( i,j,':', numero_aleatorio)
                                else:
187                                 if nao_direcionado is True:
                                        self.matriz[i,j] = 0
189                                     self.matriz[j,i] = 0
                                    else:
191                                     self.matriz[i,j] = 0
                                    #matriz[j,i] = numero_aleatorio
193                                 if self.debug is True:
                                        print( i,j,':', 0)
195
                            num_criacoes += 1
197                 else:
                        self.matriz[i,j] = 0
199                     self.matriz[j,i] = 0

201         if self.debug is True:
                print( self.matriz.keys())
203             print(self.matriz.keys()[1][1])

205     def comparar_algoritmos(self,origem, destino,
            metrica_utilizar_saltos=False):
            #suprimindo impressao
207         e = SupressorImpressao()
            saida_anterior = sys.stdout
209         sys.stdout = e
            percurso, todos_os_caminhos, tempo_exec = self.
                caminho_mais_curto(origem, destino)
211
```

```
        percurso_bellman, todos_os_caminhos_bellman,
            tempo_exec_bellman = self.caminho_mais_curto(origem,
            destino, utilizar_bellman_ford=True)

        percurso_salto, todos_os_caminhos_salto, tempo_exec_salto =
             self.caminho_mais_curto(origem, destino,
            metrica_utilizar_saltos=True)

        percurso_bellman_salto, todos_os_caminhos_bellman_salto,
            tempo_exec_bellman_salto = self.caminho_mais_curto(
            origem,destino, metrica_utilizar_saltos=True,
            utilizar_bellman_ford=True)


        #restaurando stdout para o padrao do sistema
        #sys.stdout = sys.__stdout__
        sys.stdout = saida_anterior
        print('dijkstra      : %s, executado em %ss' % (percurso,
            tempo_exec))
        print('bellman_ford : %s, executado em %ss' % (
            percurso_bellman, tempo_exec_bellman))
        print('dijkstra metrica salto     : %s, executado em %ss' %
            (percurso_salto, tempo_exec_salto))
        print('bellman_ford metrica salto: %s, executado em %ss' %
            (percurso_bellman_salto, tempo_exec_bellman_salto))


    #serializa em arquivo a instancia das variaveis matriz e
        numero_vertices
    def gravar_matriz_dat(self):
        file_name = 'backup_matriz.dat'
        the_file = open(file_name, 'wb')
        pickle.dump(self.matriz, the_file)
        the_file.close()

        file_name2 = 'backup_num_vertices.dat'
        the_file2 = open(file_name2, 'wb')
        pickle.dump(self.numero_vertices, the_file2)
        the_file2.close()
    #le arquivos serializados
    def ler_matriz_dat(self):
        file_name = 'backup_matriz.dat'
        the_file = open(file_name)
        self.ma = pickle.load(the_file)
        self.matriz = self.ma
        file_name2 = 'backup_num_vertices.dat'
        the_file2 = open(file_name2)
        self.t = pickle.load(the_file2)
        self.numero_vertices = self.t

    #imprime a matriz, caso seja passado parametro nome_arq, ele
        gera um arquivo com a matriz
    def imprimeMatriz(self):

        for i in self.lista_de_vertices:
            print('[', end='')
            for j in self.lista_de_vertices:
                print('[', self.matriz[i,j] , ']','',sep='', end =
                    '')
            print(']')

```

```
261
    #imprime a matriz
263 def imprimeMatriz_adjacencia(self):

265     print(self.numero_vertices)
        for i in self.lista_de_vertices:
267         for j in self.lista_de_vertices:
                print(self.matriz[i,j] , ' ','',sep='', end = '')
269         print('')

271 def salvarMatrizAdjacencia(self, nome_arq):
        print(nome_arq)
273     arq = file(nome_arq, 'w')


275
        arq.write(str('%s\n' % (self.numero_vertices -1)))
277     for i in self.lista_de_vertices:
            for j in self.lista_de_vertices:
279             arq.write( '%s ' % self.matriz[i,j])
            arq.write('\n')
281     arq.close()

283 def le_arquivo(self,nome_do_arquivo):

285     self.arq = open(nome_do_arquivo)
        tamanho_matriz = int(self.arq.readline())
287     print('Quantidade de Vertices:', tamanho_matriz)
        matriz = []
289     ma ={}

291     linha = self.indice_inicial
        coluna = self.indice_inicial
293     nome_arq = nome_do_arquivo.split(os.path.sep)[-1]
        print('Iniciando leitura do arquivo', nome_arq)
295     t_inicial = timeit.default_timer()
        aresta_negativa = False
297     while 1:
            coluna = self.indice_inicial
299         linha_arq = self.arq.readline()

301         if linha_arq == "":
                break
303         pos = linha_arq.replace('\r\n', '').split()
            for a in pos:
305
                ma[linha,coluna] = int(a)
307             if ma[linha,coluna] < 0:
                    aresta_negativa = True
309             coluna += 1

311         linha += 1
        t_final = timeit.default_timer()
313     print('Leitura do arquivo %s terminada em %ss' % (nome_arq,
             (t_final - t_inicial)))

315     self.__init__(tamanho_matriz, ma, self.debug, self.
            iniciar_indices_em_zero)
        self.ha_aresta_negativa = aresta_negativa
317
        return [tamanho_matriz,ma]
319
```

```
    def adicionar_aresta(self, vertice_origem, vertice_destino,
        custo, direcionada=True):
321         if vertice_destino not in self.lista_de_vertices or
                vertice_destino not in self.lista_de_vertices:
                print('Vertice Origem ou destino fora dos vertices
                    cadastrados')
323             return None

325         if direcionada:
                self.matriz[vertice_origem, vertice_destino] = custo
327         else:
                self.matriz[vertice_origem, vertice_destino] = custo
329             self.matriz[vertice_destino, vertice_origem] = custo

331     def remover_aresta(self, vertice_origem, vertice_destino, custo
        , direcionada=True):
            if vertice_destino not in self.lista_de_vertices or
                vertice_destino not in self.lista_de_vertices:
333             print('Vertice Origem ou destino fora dos vertices
                    cadastrados')
                return None
335
            if direcionada:
337             self.matriz[vertice_origem, vertice_destino] = 0
            else:
339             self.matriz[vertice_origem, vertice_destino] = 0
                self.matriz[vertice_destino, vertice_origem] = 0
341


343     #http://www.php2python.com/wiki/function.microtime/
        def microtime(self, get_as_float = False) :
345         if get_as_float:
                return time.time()
347         else:
                return '%f %d' % math.modf(time.time())
349     #http://code.activestate.com/recipes/358361-non-exponential-
        floating-point-representation/
        def non_exp_repr(self,x):
351
            """Return a floating point representation without
                exponential notation.
353
            Result is a string that satisfies:
355             float(result)==float(x) and 'e' not in result.

357         >>> non_exp_repr(1.234e-025)
            '0.00000000000000000000000012339999999999999'
359         >>> non_exp_repr(-1.234e+018)
            '-1234000000000000000.0'
361
            >>> for e in xrange(-50,51):
363             ...     for m in (1.234, 0.018, -0.89, -75.59, 100/7.0,
                    -909):
                ...         x = m * 10 ** e
365             ...         s = non_exp_repr(x)
                ...         assert 'e' not in s
367             ...         assert float(x) == float(s)

369         """
            s = repr(float(x))
371         e_loc = s.lower().find('e')
            if e_loc == -1:
```

```
373             return s
375         mantissa = s[:e_loc].replace('.', '')
            exp = int(s[e_loc+1:])
377
            assert s[1] == '.' or s[0] == '-' and s[2] == '.', "
                Unsupported format"
379         sign = ''
            if mantissa[0] == '-':
381             sign = '-'
                mantissa = mantissa[1:]
383
            digitsafter = len(mantissa) - 1     # num digits after the
                decimal point
385         if exp >= digitsafter:
                return sign + mantissa + '0' * (exp - digitsafter) + '
                    .0'
387         elif exp <= -1:
                return sign + '0.' + '0' * (-exp - 1) + mantissa
389         ip = exp + 1                        # insertion point
            return sign + mantissa[:ip] + '.' + mantissa[ip:]
391 #
    ------------------------------------------------------------------------------


393     def menor_indice_de_distancia(self,q, dist):
            menor = self.infinito
395         verti = -1

397         for i in q:
                if dist[i] < menor:
399                 menor = dist[i]
                    verti = i
401
            return verti
403


405
    def dijkstra(self,origem, metrica_utilizar_saltos=False):
407     #inicilizacao
        distancia = {}
409     anterior = {}

411     for vertice in self.lista_de_vertices:
            distancia[vertice] = self.infinito
413         anterior[vertice] = None
        distancia[origem] = 0
415
        q = [i for i in self.lista_de_vertices]
417
        t_inicial = timeit.default_timer()
419     #Processamento de Caminhos minimos
        while len(q) > 0:
421         u = self.menor_indice_de_distancia(q, distancia)
            if u == -1:
423             break
            if distancia[u] == self.infinito:
425             break
            q.remove(u)
427         visinhos = [i for i in self.lista_de_vertices if self.
                matriz[u,i] > 0 and i in q]
```

```
429             for visinho in visinhos:
                    if metrica_utilizar_saltos is False:
431                     alt = distancia[u] + self.matriz[u, visinho]
                    else:
433                     alt = distancia[u] + 1

435                 if alt < distancia[visinho]:
                        distancia[visinho] = alt
437                     anterior[visinho] = u

439         t_final = timeit.default_timer()

441         return distancia, anterior, (t_final - t_inicial)


443

445     def bellman_ford(self,origem, metrica_utilizar_saltos=False):
            #inicilizacao
447         distancia = {}
            anterior = {}
449
            for vertice in self.lista_de_vertices:
451             distancia[vertice] = self.infinito
                anterior[vertice] = None
453         distancia[origem] = 0
            ciclo_negativo = False
455         q = [i for i in self.lista_de_vertices]
            #print('iniciando bellman_ford...')
457         t_inicial = self.microtime(True)

459         #Processamento de Caminhos minimos
            for cada_vertice in self.lista_de_vertices:
461             for u in self.lista_de_vertices:
                    for v in self.lista_de_vertices:
463                     if self.matriz[u,v] != 0:
                            distan = None
465                         if metrica_utilizar_saltos is False:
                                distan = distancia[u] + self.matriz[u,v
                                    ]
467                         else:
                                distan = distancia[u] + 1
469                         if distancia[v] > distan:
                                distancia[v] = distan
471                             anterior[v] = u
            #Checando ciclo negativo
473         print('Checando ciclo negativo')
            for u in self.lista_de_vertices:
475             for v in self.lista_de_vertices:
                    if self.matriz[u,v] != 0:
477                     distan = None
                        if metrica_utilizar_saltos is False:
479                         distan = distancia[u] + self.matriz[u,v]
                        else:
481                         distan = distancia[u] + 1
                        if distancia[v] > distan:
483                         t_final = self.microtime(True)
                            #print('terminando bellman_ford...')
485                         ciclo_negativo = True
                            return distancia, anterior, (t_final -
                                t_inicial), ciclo_negativo
487         t_final = self.microtime(True)
            #print('terminando bellman_ford...')
```

```
489         return distancia, anterior, (t_final - t_inicial),
                ciclo_negativo

491
    def caminho_mais_curto(self,origem, destino,
        metrica_utilizar_saltos=False, utilizar_bellman_ford=False)
        :
493         if origem < self.indice_inicial or origem > self.
                numero_vertices -1 \
            or destino < self.indice_inicial or destino > self.
                numero_vertices-1:
495
                print('Origem ou Destino nao sao vertices validos')
497             return 'nao validao', 'nao validao','nao validao'
            ciclo_negativo = False
499         percurso = []

501         if metrica_utilizar_saltos:
                print('Utilizando Metrica de Saltos')
503
            if self.ha_aresta_negativa is True or utilizar_bellman_ford
                 is True:
505             print('Utilizando bellman_ford')
                distancia, anterior, tempo_total , ciclo_negativo= self
                    .bellman_ford(origem, metrica_utilizar_saltos)
507         else:
                print('Utilizando dijkstra')
509             distancia, anterior, tempo_total = self.dijkstra(origem
                    , metrica_utilizar_saltos)
            print('Trassando caminho..')
511         #obtendo percurso
            if destino is not None:
513             if ciclo_negativo is not True:
                    while anterior[destino] is not None :
515                     percurso.append(destino)
                        destino = anterior[destino]
517             else:
                    print('Ha ciclo negativo')
519
            percurso.append(origem)
521         percurso.reverse()
            if 'e-' in str(tempo_total):
523             tempo_total = (self.non_exp_repr(tempo_total))

525         print('Executado em: %ss\nCaminho:\n%s' % (tempo_total,
                percurso))
            return percurso, distancia, tempo_total
527


529
if __name__ == '__main__':
531
    parser = argparse.ArgumentParser(description='Grafos e
        Algoritmos\nCriado por Fabio C. Barrionuevo')
533 parser.add_argument('-a', action='store', dest='arquivo',
        #default=[],
535     help='Arquivo com matriz de adjacencia',
    )
537 parser.add_argument('-o', action='store', dest='v_origem', type
        =int,
        #default=[],
539     help='Vertice origem',
```

```
    )
    parser.add_argument('-d', action='store', dest='v_destino',
        type=int,
        #default=[],
        help='Vertice destino',
    )
    parser.add_argument('-s', action='store', dest='arq_dest',
        #default=[],
        help='Salvar saida no arquivo destino',
    )
    parser.add_argument('-g', action='store', dest='num_vertices',
        type=int,
        #default=[],
        help="Gerar grafo aleatorio baseano no em num_vertices, e
            salvar matriz de adjacencia em arquivo nomeado como o
            matriz_adcancencia_'num_vertices'.txt",
    )
    parser.add_argument('-nd', action='store', dest='direcionado',
        type=int,
        #default=[],
        help="0 para nao direcionado, 1 para direcionado, default =
             0",
    )
    parser.add_argument('-v', action='store', dest='visul', type=
        int,
        #default=[],
        help="0 para nao gerar arquivo png com a visualizacao do
            grafo , 1 para gerar arquivo png com a visualizacao do
            grafo, default = 0",
    )
    result = parser.parse_args()

    DIRETORIO_ARQUIVO = os.path.abspath(os.path.dirname(__file__))

    nome_saida = result.arq_dest
    gerar_visul = False
    if result.visul is not None:
        gerar_visul = True

    if result.arq_dest is not None:
        if '.txt' not in result.arq_dest:
            nome_saida.join('.txt')
        #redirecionando saida STDOUT, que eh utilizada pelo comando
             print, para um arquivo
        print('Processando caminhos e salvando no arquivo %s\
            nDependendo do numero de vertices e arestas do isso
            pode demorar muito tempo.\nAguarde...' % nome_saida)
        sys.stdout = file(nome_saida, 'w')

    if result.arquivo is not None and result.v_origem is not None
        and result.v_destino is not None:
        caminho = Caminho(debug=False, gerar_visul=gerar_visul)
        caminho.le_arquivo(result.arquivo)
        caminho.caminho_mais_curto(result.v_origem, result.
            v_destino)
        caminho.gerar_visualizacao()
    elif result.num_vertices is not None:
        caminho = Caminho(numero_vertices=result.num_vertices,
            debug=False, gerar_visul=gerar_visul)
        if result.direcionado is not None:
            caminho.gerarGrafoAleatorio(nao_direcionado=False)
        else:
```

```
587             caminho.gerarGrafoAleatorio()
                caminho.salvarMatrizAdjacencia(str('matriz_adcancencia_%s.
                    txt' % result.num_vertices))
589             caminho.gerar_visualizacao()
                del caminho
591         else:
                print("Execute 'python %s -h', para verificar mais opcoes\
                    nNao foram passados todos os parametros, executando o
                    padrao..." % __file__)
593             time.sleep(3)

595             #-------------------------------------------------------
                print('
                    #-------------------------------------------------------')
597             caminho = Caminho(debug=False, gerar_visul=gerar_visul)
                ARQ = os.path.join(DIRETORIO_ARQUIVO, 'matrix.txt')
599             caminho.le_arquivo(ARQ)
                #caminho.caminho_mais_curto(1,6)
601             caminho.comparar_algoritmos(1,6)
                caminho.gerar_visualizacao()
603             del caminho
                #-------------------------------------------------------
605             print('
                    #-------------------------------------------------------')
                caminho = Caminho(debug=False, gerar_visul=gerar_visul)
607             ARQ = os.path.join(DIRETORIO_ARQUIVO, '50.txt')
                caminho.le_arquivo(ARQ)
609             #caminho.caminho_mais_curto(10,40)
                caminho.comparar_algoritmos(10,40)
611             caminho.gerar_visualizacao()
                del caminho
613             #-------------------------------------------------------
                print('
                    #-------------------------------------------------------')
615             caminho = Caminho(debug=False, gerar_visul=gerar_visul)
                ARQ = os.path.join(DIRETORIO_ARQUIVO, '250.txt')
617             caminho.le_arquivo(ARQ)
                #caminho.caminho_mais_curto(10,220)
619             caminho.comparar_algoritmos(10,220)
                caminho.gerar_visualizacao()
621             del caminho
                #-------------------------------------------------------
623             print('
                    #-------------------------------------------------------')
                caminho = Caminho(debug=False, gerar_visul=gerar_visul)
625             ARQ = os.path.join(DIRETORIO_ARQUIVO, '500.txt')
                caminho.le_arquivo(ARQ)
627             #caminho.caminho_mais_curto(1,350)
                caminho.comparar_algoritmos(1,350)
629             caminho.gerar_visualizacao()
                del caminho
631             #-------------------------------------------------------
                print('
                    #-------------------------------------------------------')
633             caminho = Caminho(debug=False, gerar_visul=gerar_visul)
                ARQ = os.path.join(DIRETORIO_ARQUIVO, '750_2.txt')
635             caminho.le_arquivo(ARQ)
                #caminho.caminho_mais_curto(1,700)
637             caminho.comparar_algoritmos(1,700)
                caminho.gerar_visualizacao()
639             del caminho
                #-------------------------------------------------------
```

```
641         print('
                #----------------------------------------------------------')
            caminho = Caminho(debug=False, gerar_visul=gerar_visul)
643         ARQ = os.path.join(DIRETORIO_ARQUIVO, '1000.txt')
            caminho.le_arquivo(ARQ)
645         #caminho.caminho_mais_curto(1,900)
            caminho.comparar_algoritmos(1,900)
647         caminho.gerar_visualizacao()
            del caminho
```

caminho_minimo.py