grafos.py testes.py

```
Biblioteca ‘grafos’ {
  [Manipulando Grafos]

     < Trabalho Prático 2 - Teoria dos Grafos >

}
```

Pedro Glaser de Senna e Rafaella Lenzi Romano

grafos.py

testes.py

01 {

[Leitura do Grafo]

< Tipo do Grafo: Matriz e Lista >

}

Pedro Glaser de Senna e Rafaella Lenzi Romano

grafos.py

testes.py

```python
class Grafo():
    def __init__(self, tipo_grafo, peso, txt):
        if tipo_grafo == "matriz":
            self.grafo = GrafoMatriz(txt)
            if peso == True:
                print("Grafo com matriz apena para grafo sem peso")
                self.grafo = GrafoLista(peso, txt)
        if tipo_grafo =="lista":
            self.grafo = GrafoLista(peso, txt)
```

Pedro Glaser de Senna e Rafaella Lenzi Romano

grafos.py

testes.py

02 {

[Lista de Adjacência]

< Implementação para lidar com grafos com pesos não negativo>

}

Pedro Glaser de Senna e Rafaella Lenzi Romano

grafos.py

testes.py

```python
class ListaVizinhos:
    #Cada lista dessa classe representa a lista de vizinhos de um vértice
    #Cada lista dessas será um elemento do vetor de vértices da lista de adjacência
    def __init__(self):
        self.head = None
        self.size = 0
    def add(self, vizinho):
        vizinho.proximo = self.head
        self.head = vizinho
        self.size += 1

class Vizinho:
    #Cada vizinho dessa classe será um nó da lista de vizinhos
    def __init__(self, vizinho: int, peso: int):
        self.vizinho = vizinho
        self.peso = peso
        self.proximo = None
    def __repr__(self):
        return (str(self.peso))
```

Pedro Glaser de Senna e Rafaella Lenzi Romano

grafos.py

testes.py

03 {

[Pesos Negativos]

< Como lidar com grafos com pesos negativos? >

}

Pedro Glaser de Senna e Rafaella Lenzi Romano

grafos.py

testes.py

```
1
2
3   for linha in arquivo:   #Adicionando vizinhos às listas
4
5       vertice1 = int(linha.split()[1])
6       vertice2 = int(linha.split()[0])
7       peso_aresta = float(linha.split()[2])
8
9       if peso_aresta < 0:
10          self._negativo = True
11
12
13  elif self._negativo == True:
14      caminho = "Grafo apresenta peso negativo." #Dijkstra não é capaz de computar distancias para grafos negativos.
```

Pedro Glaser de Senna e Rafaella Lenzi Romano

grafos.py

testes.py

04 {

[HEAP]

< Para implementar o Heap, utilizamos a biblioteca ‘heapdict’>

}

Pedro Glaser de Senna e Rafaella Lenzi Romano

testes.py

grafos.py

Rodando; {

‘DISTÂNCIA’

<p Conseguimos calcular de todos os grafos menos do 5>

</p>

}

Grafo	20	30	40	50	60
1	2.38	1.72	2.05	1.20	1.66
2	1.81	1.72	1.92	1.57	1.58
3	0.8	0.89	0.84	0.87	0.97
4	2.66	2.02	2.52	2.69	2.41
5	-	-	-	-	-

Pedro Glaser de Senna e Rafaella Lenzi Romano

testes.py

grafos.py

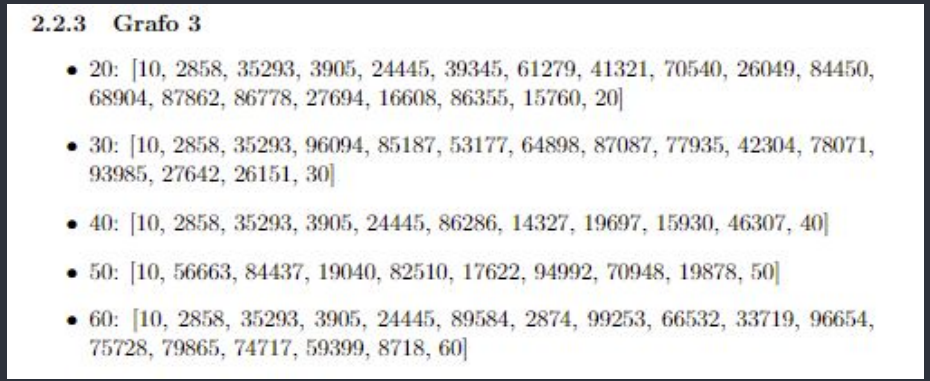
Rodando; {

‘CAMINHO MÍNIMO’

<p Conseguimos calcular de todos os grafos menos do 5>

</p>

}



2.2.3 Grafo 3

- 20: [10, 2858, 35293, 3905, 24445, 39345, 61279, 41321, 70540, 26049, 84450, 68904, 87862, 86778, 27694, 16608, 86355, 15760, 20]
- 30: [10, 2858, 35293, 96094, 85187, 53177, 64898, 87087, 77935, 42304, 78071, 93985, 27642, 26151, 30]
- 40: [10, 2858, 35293, 3905, 24445, 86286, 14327, 19697, 15930, 46307, 40]
- 50: [10, 56663, 84437, 19040, 82510, 17622, 94992, 70948, 19878, 50]
- 60: [10, 2858, 35293, 3905, 24445, 89584, 2874, 99253, 66532, 33719, 96654, 75728, 79865, 74717, 59399, 8718, 60]

Pedro Glaser de Senna e Rafaella Lenzi Romano

testes.py

grafos.py

Rodando; {

‘TEMPO (HEAP X VETOR)’

<p Os grafos 1 e 2, que eram menores, foram os únicos que conseguimos rodar das duas formas >

</p>

}

Grafo	Dijkstra Heap (s)	Dijkstra Vetor (s)
1	0.586122190952301	60.435516269207
2	2.568475213050842	822.950772075653
3	20.433632123470307	-
4	386.41773231744764	-
5	-	-

Pedro Glaser de Senna e Rafaella Lenzi Romano

testes.py

grafos.py

Rodando; {

‘REDE DE COLABORAÇÃO’

<p Calculamos a distância e o caminho mínimo>

</p>

}

Conexão	Distância
Dijkstra-Turing	infinito
Dijkstra-Kruskal	3.48036845488905
Dijkstra-Kleinberg	2.7069936175564644
Dijkstra-Tardos	2.7535141793573357
Dijkstra-Ratton	2.9428308695367855

Pedro Glaser de Senna e Rafaella Lenzi Romano

testes.py

grafos.py

- Dijkstra-Turing: Não há caminho.
- Dijkstra-Kruskal: [2722, 9490, 7200, 10343, 646765, 490368, 10746, 3655, 471365]

  [Edsger W. Dijkstra, John R. Rice, Dan C. Marinescu, Howard Jay Siegel, Edwin K. P. Chong, Ness B. Shroff, R. Srikant, Albert G. Greenberg, J. B. Kruskal]
- Dijkstra-Kleinberg: [2722, 217250, 11456, 768, 11448, 101826, 12242, 11834, 9608, 5709]

  [Edsger W. Dijkstra, A. J. M. van Gasteren, Gerard Tel, Hans L. Bodlaender, Dimitrios M. Thilikos, Prabhakar Ragde, Avi Wigderson, Eli Upfal, Prabhakar Raghavan, Jon M. Kleinberg]
- Dijkstra-Tardos: [2722, 217250, 11456, 768, 6479, 8528, 10572, 357587, 11649, 3694, 318911, 11386]

  [Edsger W. Dijkstra, A. J. M. van Gasteren, Gerard Tel, Hans L. Bodlaender, Jan van Leeuwen, Mark H. Overmars, Micha Sharir, Haim Kaplan, Robert Endre Tarjan, Andrew V. Goldberg, Serge A. Plotkin, Éva Tardos]
- Dijkstra-Ratton: [2722, 9490, 7200, 391667, 371226, 4379, 68773, 11466, 343930]

  [Edsger W. Dijkstra, John R. Rice, Dan C. Marinescu, Chuang Lin, Bo Li, Y. Thomas Hou, Zhi-Li Zhang, Donald F. Towsley, Daniel R. Figueiredo]

Pedro Glaser de Senna e Rafaella Lenzi Romano

testes.py

grafos.py

```
Rodando; {

    'BÔNUS'

        Ratton-Cukierman:
        6.589831236023287

        Ratton-Cukierman:
        [343930, 11466, 10421, 9478, 13111, 6601, 13131, 649176,
        1260, 90639, 600265]

}
```

Pedro Glaser de Senna e Rafaella Lenzi Romano

grafos.py testes.py

```
1
2
3   Biblioteca ‘grafos’ {
4
5       [Agradecemos a Atenção!]
6
7
8           < FIM >
9
10
11
12  }
13
14
```

Pedro Glaser de Senna e Rafaella Lenzi Romano