Grafos com pesos não-negativos

Daniel Ferreira e Lucas Garcia

Linguagem Escolhida: Python

Orientação à Objetos

Facilidade na implementação

Conhecimento Prévio

Facilidade na leitura

Leitura do grafo

```
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)
```

Lista de Adjacência

Implementação

```
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))
```

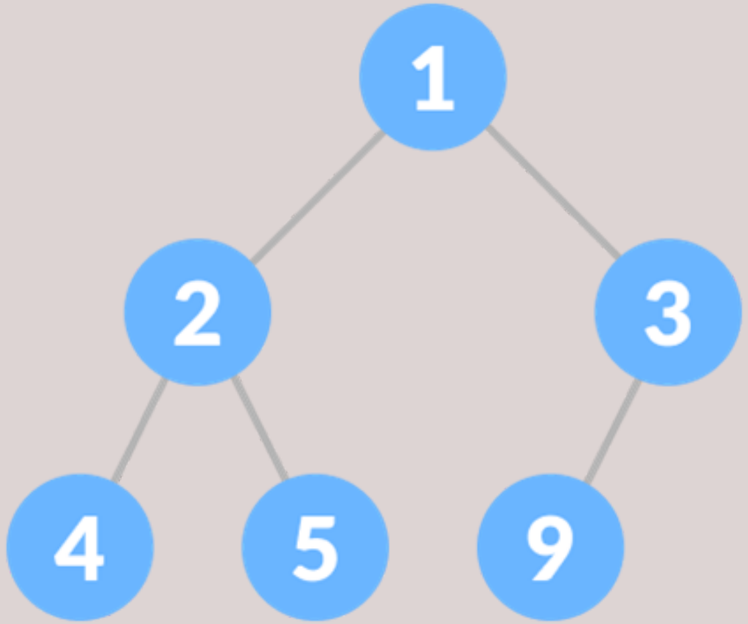
Tratamento de Pesos Negativos

```
for linha in arquivo:    #Adicionando vizinhos às listas
    split = linha.split()

    verticel = int(linha.split()[1])
    vertice2 = int(linha.split()[0])
    peso_aresta = float(linha.split()[2])

    if peso_aresta < 0:
        self._negativo = True
```

```
elif self._negativo == True:
    distancia = "Grafo apresenta peso negativo." #Dijkstra não é capaz de computar distancias para grafos negativos.
```

Implementação da Heap

Biblioteca Heapdict



Decrease-Key

Complexidade de Memória

PopItem

Facilidade na Implementação

Vetor X Heap

Comparação

Grafo	Tempo médio das distâncias Heap/Vetor (seg)	Distância (10, 20)	Distância (10, 30)	Distância (10, 40)	Distância (10, 50)	Distância (10, 60)	Peso da MST
grafo_W_1_1	0,09 / 0,9	1,52	1,48	1,52	1,39	1,39	220,11
grafo_W_2_1	1,1 / 102,9	2,08	1,92	1,61	1,34	1,69	2247,07
grafo_W_3_1	16,1 / -	1,98	2,08	2,14	1,82	2,23	22218,71
grafo_W_4_1	- / -	-	-	-	-	-	-
grafo_W_5_1	- / -	-	-	-	-	-	-

Algoritmo de Prim

Minumum Spanning Tree

```
peso_total = sum(pesos)
arquivo = open(txt_saida, "w")
for i in range(1,len(pais)+1):
    if i != partida:
        arquivo.writelines(f'{i} {pais[i-1]} {pesos[i-1]}\n')
arquivo.close()
return peso total
```

Rede de Colaboração

Resultados

Pesquisador	Distância (até Dijkstra)	Caminho mínimo (com os nomes)
Alan M. Turing	infinito	Não há caminho.
J. B. Kruskal	3,48	[Dijkstra, John R. Rice, Dan C. Marinescu, Howard Jay Siegel, Edwin K. P. Chong, Ness B. Shroff, R. Srikant, Albert G. Greenberg, Kruskal]
Jon M. Kleinberg	2,71	[Dijkstra, A. J. M. van Gasteren, Gerard Tel, Hans L. Bodlaender, Dimitrios M. Thilikos, Prabhakar Ragde, Avi Wigderson, Eli Upfal, Prabhakar Raghavan, Kleinberg]
Eva Tardos	2,75	[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, Tardos]
Daniel R. Figueiredo	2,94	[Dijkstra, John R. Rice, Dan C. Marinescu, Chuang Lin, Bo Li, Y. Thomas Hou, Zhi-Li Zhang, Donald F. Towsley, Ratton]