
UNIVERSIDADE FEDERAL DE PERNAMBUCO

Centro de Informática
UFPE

**Universidade Federal de Pernambuco**
**Centro de Informática - CIn**

**Disciplina:** Algoritmos e Estrutura de Dados
**Docente:** Sérgio Ricardo de Melo Queiroz
**Discente:**
Maria Clara Falcão Guerra Barretto
Mariana Rego Barros Santos Amorim

# Relatório do projeto: Grupo #4.2

Recife, 17 de abril de 2023

**Contexto do problema**

A base de dados contém informações sobre rotas de transporte público (ônibus) na cidade de Adelaide, localizada na costa sul da Austrália.

Cada linha, representante de uma tabela, possui dados classificados em seis categorias e que aparecem na seguinte ordem:

*TripID* – número de identificação do caminho (refere-se ao local de origem), varia de 79 a 65535; *RouteID* – número de identificação da rota, varia de 100 a 996; *StopID* – número de identificação do local de chegada, varia de 10001 a 18715; *StopName* – endereço do local de chegada, há 4165 endereços distintos; *WeekBeginning* – informam a data da primeira viagem da rota, variam de 29 de junho de 2013 até 5 de julho de 2014; *NumberOfBoardings* – informam a quantidade de pessoas que embarcaram na primeira viagem da rota, variam entre um e 40 pessoas;

No total, o dataset conta com cerca de 10.9 milhões de conjuntos (as 6 categorias citadas acima) de dados, dos quais foram utilizados uma fração, 5000 dados, para auxiliar na visualização das propriedades do código.

**Implementação**

**Algoritmo utilizado.**

O algoritmo escolhido foi o Kruskal de árvore geradora mínima

**Desenvolvimento.**

A base de dados escolhida foi uma *Public Bus Transport Dataset* retirada do site Kaggle. Procuramos especificamente por datasets de transporte público pois seriam os mais propensos a estarem estruturados como grafos, com nós com entrada e saída. A base poderia ter sido usada como direcionada inclusive, mas não foi necessário para este projeto.

Utilizamos ambos os IDE Colab (online e compartilhável) e VSCode (acesso ao github) para a criação dos códigos. Testamos separadamente cada etapa do código (formatação da entrada dos dados para adequar-se ao algoritmo, e o algoritmo em si) para facilitar a correção de bugs.

**Bibliotecas utilizadas.**

networkx e matplotlib (para visualização dos grafos)

**Conclusão**

A primeira parte contém a formatação do dataset (que seria a entrada) para que o mesmo se adeque às necessidades do algoritmo de kruskal, que vem logo em seguida. Ele utiliza duas listas oriundas do dataset, uma de vértices – strings – e outra de arestas (tuplas que contém o peso da aresta – int – e a dupla de vértices conjugados). Os dados utilizados como vértices são representantes do *TripID* e do *StopID*, enquanto o peso é dado pelo *NumberOfBoardings.*

A saída do programa é uma MST de kruskal do tipo lista de tuplas, com cada item da tupla sendo, respectivamente, o peso da aresta e a dupla de vértices integrantes. A segunda parte, configura a seção de ponto extra, em que é possível visualizar o grafo da mst.

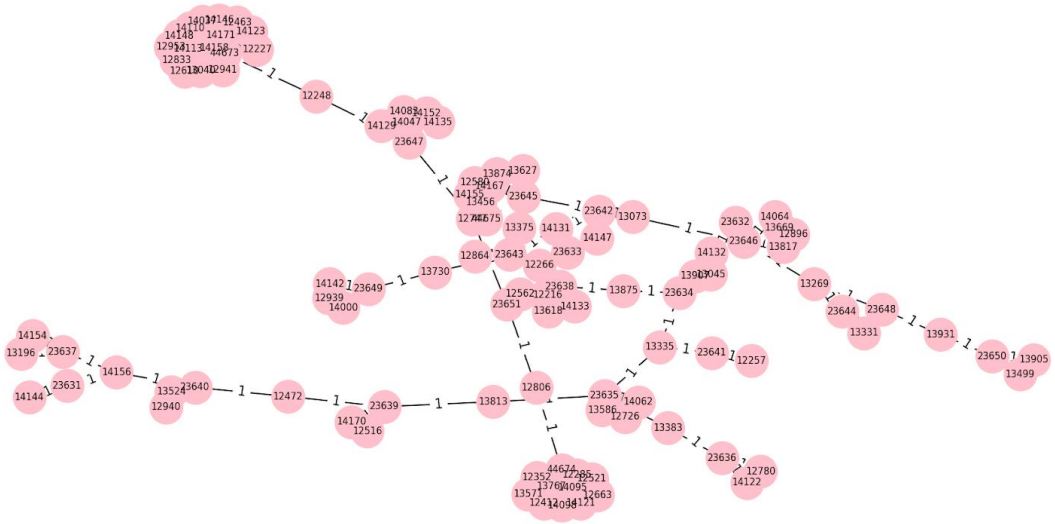
Imagens do programa e seu funcionamento:

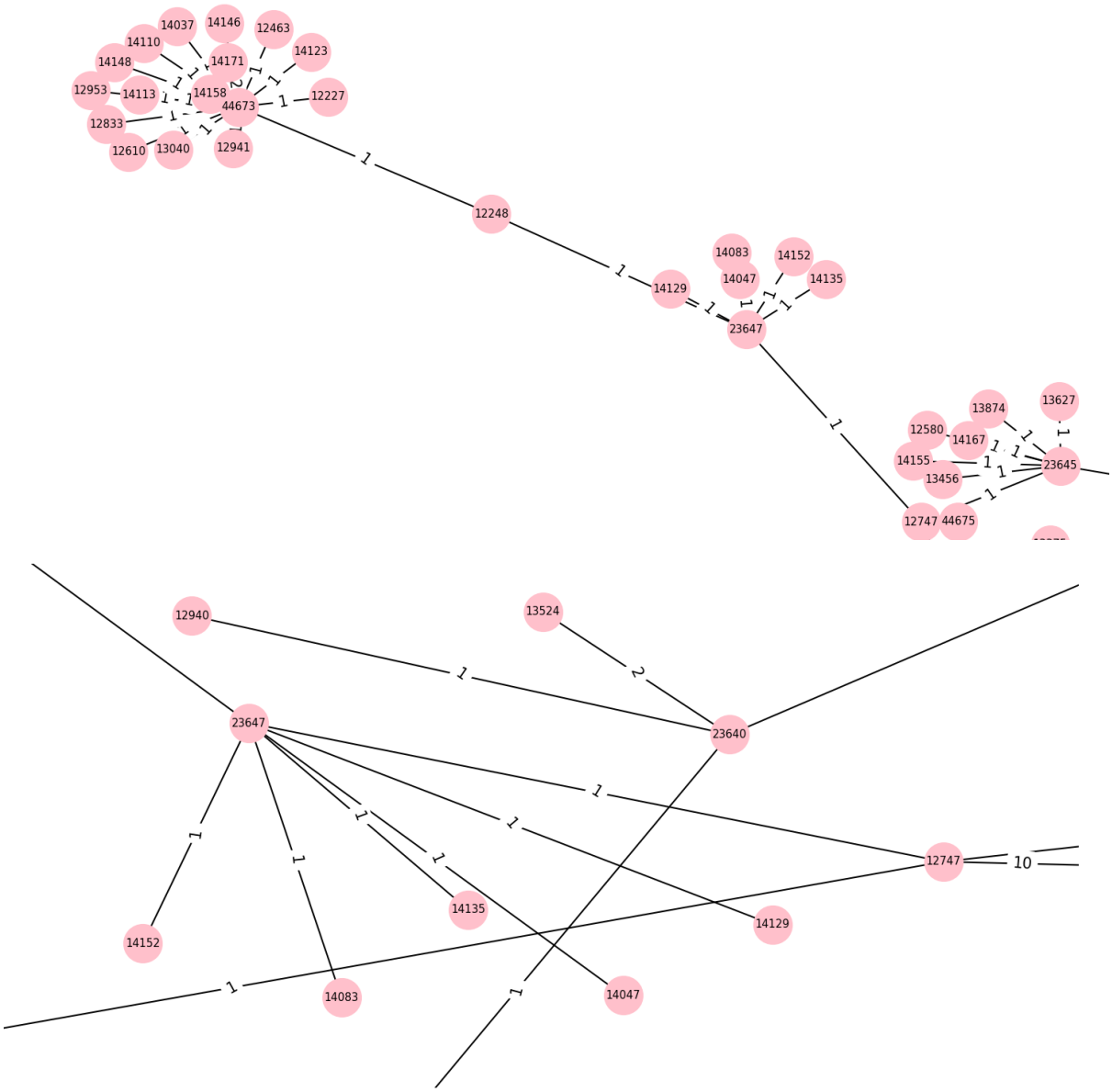
```python
adelaide_data = open('adelaide.CSV', 'r').readlines()[1:201] # a primeira linha é o título e não faz parte dos dados
adelaide_data

# Formatação do dataset
i = 0

v = []
e = []

for data in adelaide_data:
  data_set = adelaide_data[i]
  info = data_set.split(',')

  i+=1

  trip_id = info[0]
  stop_id = info[2]
  n_boarding = info[-1]

  if trip_id not in v:
    v.append(trip_id)

  if stop_id not in v:
    v.append(stop_id)

  p =  n_boarding
  remove = ' "'
  peso = int(p.translate(str.maketrans('', '', remove)))  # transforma em valor inteiro
  v1, v2 = info[0].translate(str.maketrans('', '', remove)), info[2].translate(str.maketrans('', '', remove))
  aresta = (peso, v1, v2)

  e.append(aresta)

v = [elem.replace('"', '') for elem in v]
```

```python
# Classes e métodos
class KruskalMST:  # Minimum Spanning Tree (Kruskal)

    def __init__(self, vertices, edges):  # método construtor
        self.vertices = vertices
        self.edges = edges  # arestas
        self.parent = {v: v for v in vertices}  # nó pai (família/subconjunto); cada nó é pai dele mesmo originalmente (várias árvores unitárias)
        self.rank = {v: 0 for v in vertices}  # classificação (quantidade de itens que fazem parte da família/subconjunto)

    def find(self, v):  # retorna a raiz de um nó v
        if self.parent[v] != v:
            self.parent[v] = self.find(self.parent[v])

        return self.parent[v]

    def union(self, v1, v2):  # une dois vértices que não pertencem a um(a) mesmo(a) subconjunto/família
        root1 = self.find(v1)
        root2 = self.find(v2)
        if root1 != root2:
            if self.rank[root1] > self.rank[root2]:
                self.parent[root2] = root1
                self.rank[root1] += self.rank[root2]

            else:
                self.parent[root1] = root2
                self.rank[root2] += self.rank[root1]

    def kruskal(self):  # retorna o grafo da árvore geradora mínima
        Tree = []
        e = sorted(self.edges)  # organiza arestas pelo peso ascendentemente
        while len(e) > 0:
            aresta = e.pop(0)  # aresta de menor peso
            if aresta not in Tree:
                peso, v1, v2 = aresta

                if self.find(v1) != self.find(v2):
                    self.union(v1, v2)
                    Tree.append(aresta)

        return Tree

# Saída da árvore geradora mínima de Kruskal
mst = KruskalMST(v, e).kruskal()
print(mst)
```

```python
print(mst)

# Pontuação extra
# Função para a vizualização do grafo
import networkx as nx
import matplotlib.pyplot as plt

# Criando o grafo
G = nx.Graph()
for aresta in mst:
    peso, v1, v2 = aresta
    G.add_edge(v1, v2, weight=peso)

# Desenhando o grafo
pos = nx.spring_layout(G)  # posição dos vértices
labels = nx.get_edge_attributes(G, 'weight')  # rótulos das arestas

nx.draw_networkx_nodes(G, pos, node_color='pink', node_size=600)
nx.draw_networkx_edges(G, pos, edge_color='black')
nx.draw_networkx_labels(G, pos, font_size = 7)
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.axis('off')

plt.show()  # exibir o gráfico

```

Saída e visualização do grafo:






```
(1, '23634', '14132'), (1, '23635', '12726'), (1, '23635', '13335'), (1, '23635', '13383'), (1, '23635', '13813'), (1, '23635', '14062'), (1, '23636', '12780'), (1, '23636', '13383'),
(1, '23636', '14122'), (1, '23637', '13196'), (1, '23637', '14154'), (1, '23637', '14156'), (1, '23638', '12562'), (1, '23638', '13618'), (1, '23638', '13875'), (1, '23638', '14133'),
(1, '23639', '12472'), (1, '23639', '12516'), (1, '23639', '13813'), (1, '23639', '14170'), (1, '23640', '12472'), (1, '23640', '12940'), (1, '23640', '14156'), (1, '23641', '12257'),
(1, '23641', '13335'), (1, '23642', '13073'), (1, '23642', '14131'), (1, '23643', '12864'), (1, '23643', '13375'), (1, '23643', '13730'), (1, '23643', '14131'), (1, '23644', '13269'),
(1, '23644', '13331'), (1, '23645', '12580'), (1, '23645', '12747'), (1, '23645', '13073'), (1, '23645', '13456'), (1, '23645', '13627'), (1, '23645', '13874'), (1, '23645', '14155'),
(1, '23645', '14167'), (1, '23646', '12896'), (1, '23646', '13073'), (1, '23646', '13269'), (1, '23646', '13669'), (1, '23646', '13817'), (1, '23646', '14064'), (1, '23647', '12248'),
(1, '23647', '12747'), (1, '23647', '14047'), (1, '23647', '14083'), (1, '23647', '14129'), (1, '23647', '14135'), (1, '23647', '14152'), (1, '23648', '13269'), (1, '23648', '13931'),
(1, '23649', '12939'), (1, '23649', '13730'), (1, '23649', '14000'), (1, '23649', '14142'), (1, '23650', '13499'), (1, '23650', '13905'), (1, '23650', '13931'), (1, '23651', '12747'),
(1, '23651', '12806'), (1, '44673', '12227'), (1, '44673', '12248'), (1, '44673', '12463'), (1, '44673', '12610'), (1, '44673', '12833'), (1, '44673', '12941'), (1, '44673', '12953'),
(1, '44673', '13040'), (1, '44673', '14037'), (1, '44673', '14110'), (1, '44673', '14113'), (1, '44673', '14123'), (1, '44673', '14146'), (1, '44673', '14148'), (1, '44674', '12412'),
(1, '44674', '12663'), (1, '44674', '12806'), (1, '44674', '13571'), (1, '44674', '14098'), (1, '44674', '14121'), (2, '23633', '12266'), (2, '23635', '13586'), (2, '23638', '12216'),
(2, '23640', '13524'), (2, '44673', '14171'), (2, '44674', '12352'), (2, '44674', '12521'), (2, '44674', '13767'), (2, '44674', '14095'), (3, '23638', '12266'), (3, '44674', '12285'),
(4, '23643', '12266'), (5, '44673', '14158'), (10, '44675', '12747')]
```

**Referências**

CHAURASIA, P. **networkx**. Disponível em: <https://github.com/networkx/networkx/tree/main/examples/graph>. Acesso em: 19 abr. 2023.

FERREIRA, M. **Utilizando Python para tratar base de dados com muitos arquivos**. Disponível em: <https://www.linkedin.com/pulse/utilizando-python-para-tratar-base-de-dados-com-muitos-mauro-ferreira/?originalSubdomain=pt>. Acesso em: 19 abr. 2023.

REDNIVRUG, O. **Public Bus Transport Dataset**. Disponível em: <https://www.kaggle.com/datasets/rednivrug/unisys>. Acesso em: 19 abr. 2023.

SILVA, R. S. **Clustering-Prim--Kruskal**. Disponível em: <https://github.com/rsousas/Clustering-Prim--Kruskal>. Acesso em: 19 abr. 2023.