DOCUMENTAÇÃO

~ OBJETIVO:

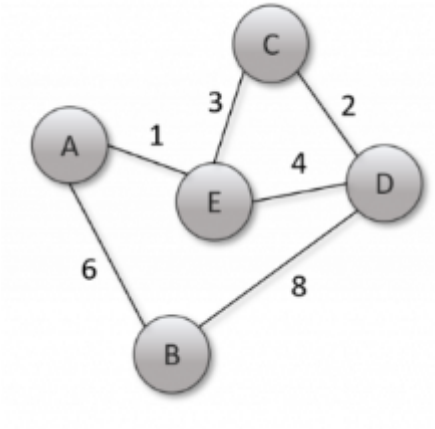
Este trabalho tem como objetivo o estudo e a aplicação de grafos em código, o desenvolvimento das habilidades em técnicas de programação e a utilização de bibliotecas externas.

~ FERRAMENTAS:

A linguagem utilizada foi python, para o desenvolvimento do código utilizamos o Google Colab, o GitHub para o versionamento e para a documentação, foi utilizado o CODA.

~ ESTRUTURA DO GRAFO:

Neste grafo, os vértices são aeródromos brasileiros aptos para voos comerciais e as arestas são aviões comerciais fazendo voos nacionais. Essa estrutura permite visualizar quais aeródromos estão conectados e quais são os de mais relevância.



No nosso caso, os vértices seriam os aeródromos, as arestas seriam os voos nacionais e o peso seria a quantidade de vezes que esse voo foi feito.

~ ARQUITETURA:

A arquitetura utilizada foi dividida em 3 partes:

- Métodos leitores
- Métodos interpretadores
- Métodos expositores

Desse modo a divisão de tarefas de cada método ficou mais clara e legível, facilitando então a produção do código.

~ BIBLIOTECAS:

Bibliotecas utilizadas para a execução da aplicação:

- Networkx (manipulação de estruturas em python)
- Pandas (manipulação de dados em python)
- Matplotlib.pyplot (adição de métodos matemáticos)
- Google.colab (integração com a rede Google)

~ MÉTODOS:
Estes métodos leitores servem para ler a base de dados em formato csv e armazenar as informações em variáveis para que os dados possam ser manipulados.

```
fluxo_aereo = pd.read_csv("/content/fluxoAereo.csv", sep = ";")

aero_publicos = pd.read_csv('/content/AeroPublicos.csv', sep=';', encoding='Latin-1')
```

O método .drop() foi utilizado para filtrar apenas as colunas que serão analisadas no projeto. A informação filtrada foi armazenada na própria variável.

```
fluxo_aereo = fluxo_aereo.drop(columns=['TARIFA', 'EMPRESA', 'ASSENTOS', 'ANO', 'MES'])

aero_publicos = aero_publicos.drop(columns=['Município Servido', 'UF Servido', 'LATGEOPOINT', 'LONGEOPOINT', 'Latitude', 'Longitude', 'Altitude', 'Operação Diurna', 'Operação Noturna', 'Designação 1', 'Comprimento 1', 'Largura 1', 'Resistência 1', 'Superfície 1', 'Designação 2', 'Comprimento 2', 'Largura 2', 'Resistência 2', 'Superfície 2', 'Situação', 'Validade do Registro', 'Portaria de Registro', 'Link Portaria'])
```

Utilizamos o método .merge() para relacionar as colunas em uma única variável.

```
fluxo_aerodromos_origem = fluxo_aereo.merge(aero_publicos, left_on="ORIGEM", right_on="Código OACI", how="outer", suffixes=("_aeroPOrigem", "_fluxoAOrigem"))

fluxo_aerodromos = fluxo_aerodromos_origem.merge(aero_publicos, left_on="DESTINO", right_on="Código OACI", how="outer", suffixes=("_Origem", "_Destino"))
```

.dropna() serve para remover as eventuais linhas e colunas que não tem valor. Essa técnica é importante para que o grafo possa ser criado sem nenhum erro de valor inexistente.

```
fluxo_aerodromos = fluxo_aerodromos.dropna(axis = 0)
```

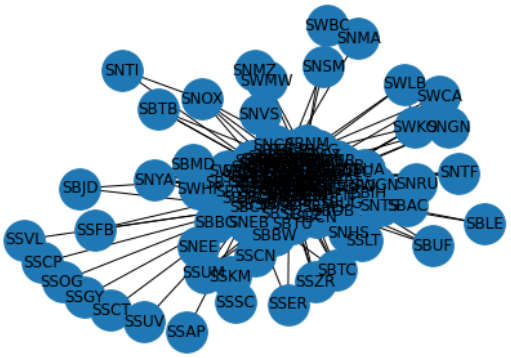
Criando os grafos linkando a origem e o destino e exibindo

```
G = nx.from_pandas_edgelist(
    fluxo_aerodromos,
```

```
    "ORIGEM",
    "DESTINO",
    create_using=nx.Graph()
)

nx.draw(G, with_labels=True,  node_size = 1000)
```

Resultado inicial:



Adicionando peso as arestas:

```
G = nx.Graph()

for index, row in fluxo_aerodromos.iterrows():
  G.add_edge(row['ORIGEM'], row['DESTINO'], weight = row['VEZES'])
```

Definindo centralidades:

```
nx.closeness_centrality(G)
nx.betweenness_centrality(G, weight='weight')
centralidade = nx.betweenness_centrality(G, weight='weight')
total_centralidade = []

for valor in centralidade.values():
  total_centralidade.append(valor)

total_centralidade.sort()
```

Separando os aeródromos mais influentes do menos influentes, assim podemos descartar aqueles que não interferem no grafo e trabalhamos apenas com os que são de interesse.

```
aeros_nao_influentes = [k for k, v in nx.betweenness_centrality(G,
weight='weight').items() if v == 0]

aeros_mais_influentes = [k for k, v in nx.betweenness_centrality(G,
weight='weight').items() if v >= total_centralidade[-10]]
```

Customizando o grafo para que a apresentacao dele seja mais bonita e compreensivel.

```
fig, ax = plt.subplots(1,1,figsize=(20,16))

pos = nx.spring_layout(G, seed=123456789, k=3, weight='weight')

# color of nodes
color = list(dict(nx.degree(G)).values())

# draw edges
nx.draw_networkx_edges(G, pos=pos, alpha=0.4, ax=ax)

# draw nodes
nodes = nx.draw_networkx_nodes(G, pos=pos, node_size=1000, node_color =
color, cmap = plt.cm.jet, ax=ax)

# draw labels
nx.draw_networkx_labels(G, pos=pos,font_color='white', ax=ax)


plt.axis("off")
plt.colorbar(nodes)
plt.savefig('degree_centrality.png', transparent=True,dpi=600)
plt.show()
```

~ TESTES:

Para testar nosso projeto, criamos grafos menores com informações reduzidas retiradas de bancos mockados para que o processo de debug seja mais simplificado.

Desse modo, foi possível criar os métodos de interpretadores e expositores de forma mais simples. Para criar os grafos finais, ajustes mínimos foram feitos, mas o grosso foi replicado para uma escala maior.

~ COMO EXECUTAR:

Para executar o projeto, basta ter o python instalado, baixar os bancos de dados em formato csv (caso necessário, ajustar a path do código para encontrar o banco de forma correta) e rodar a aplicação. As dependências serão instaladas automaticamente e o resultado do projeto será exibido.

~ RESULTADO:

Assim ficou o resultado final do grafo: