UNIVERSIDADE PRESBITERIANA MACKENZIE
Faculdade de Computação e Informática
Prof. Dr. Ivan Carlos Alcântara de Oliveira
Teoría dos Grafos

Relatório do Projeto – Parte 1

Nome do Integrante	TIA
Leonardo Pinheiro de Souza	32127391
Lucas Paulo da Rocha	32196628
Luiz Octavio Tassinari Saraiva	32030411
Thiago Aidar Figueiredo	32144547

Conteúdo do Relatório da aplicação Percursos Internacionais

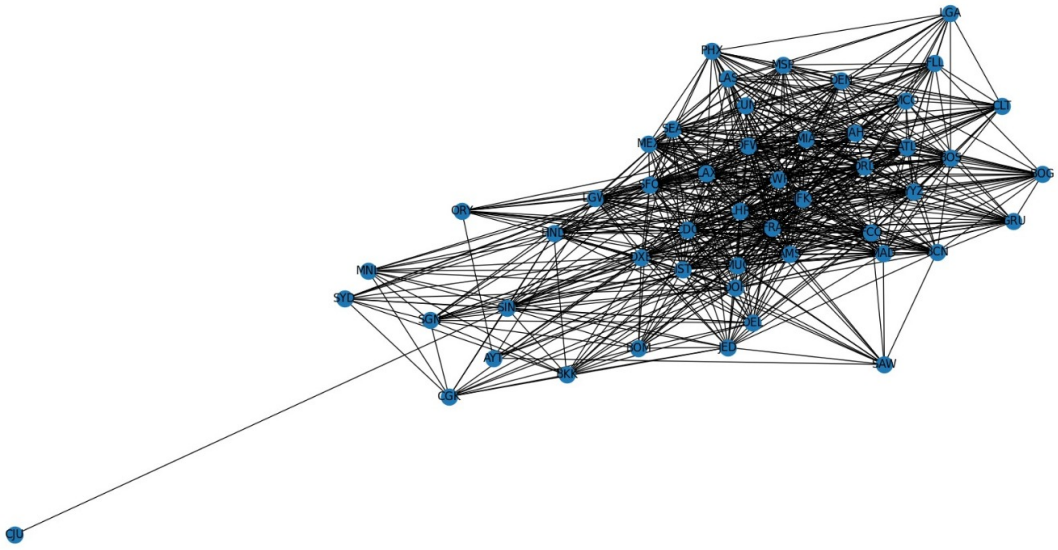
Introdução:

Dentre os 17 Objetivos de Desenvolvimento Sustentável, escolhemos desenvolver uma aplicação focada no objetivo 9 (Indústria, Inovação e Infraestrutura). Nosso projeto é uma aplicação que usa o sistema de grafos não direcionados e com pesos na arestas para representar as principais linhas aéreas entre os principais aeroportos internacionais do mundo.
A contribuição desse projeto para os objetivo citados está em promover uma forma de análise de quais rotas existem e analisar quais aeronaves podem ser usadas de forma mais efetiva entre as rotas

Dados coletados:

Para este projeto foram escolhidos os 50 aeroportos mais movimentados em passageiros, segundo o relatório de tráfego em aeroportos de 2022 divulgado pelo consórcio governamental americano *Port Authority of New York and New Jersey*. Utilizando o site “flightradar24.com”, foram coletados os seguintes dados para as arestas encontradas: distância da rota entre os aeroportos, companhias aéreas que atuam no trajeto e os modelos de avião utilizados.

Grafo não-direcionado não rotulado



Grafo criado pelo função desenharGrafo() da classe GrafoND

As informações obtidas foram salvas dentro do arquivo .csv “voos - Página1.csv”, disponível entre os arquivos na entrega. Inicialmente foi criada uma função de leitura deste arquivo .csv para armazenar as informações dentro do grafo, para em sequência realizar a gravação no arquivo de texto. A função de leitura do arquivo CSV está disponível no arquivo “utils.py”.

Função de leitura de arquivos .csv

```
def lerCSV():
  tabela = pd.read_csv("voos - Página1.csv")
  #CRIANDO TUPLAS COM SIGLA,NOME_AEROPORTO
  sigla_saida = tabela["Saida Sigla"]
  aeroporto_saida = tabela["Saida Nome Aeroporto"]
  saida1 = list(set(zip(sigla_saida, aeroporto_saida)))
  sigla_chegada = tabela["Chegada"]
  aeroporto_chegada = tabela["Chegada Nome Aeroporto"]
  chegada1 = list(set(zip(sigla_chegada, aeroporto_chegada)))
  aeroportos1= list(sorted(set(saida1 + chegada1)))

  mapa = {}
  for i, item in enumerate(aeroportos1):
      mapa[item[0]] = [i,item[1]]

  #criando grafo
  n = len(mapa)
  g1 = GrafoND(n, mapa)

  #criando arestas
  for index, row in tabela.iterrows():
    origem = row["Saida Sigla"]
    destino = row["Chegada"]
    distancia = row["Distancia (Km)"]
    modelos = row["ModelosAvioes"]
    comp = row["Airline"]
    dict = {
      "distancia": distancia,
      "modelos":modelos.split("-"),
      "airlines":comp.split("-")
    }

    g1.insereA(origem, destino, dict)
  return g1
```

Estrutura do Grafo e arquitetura do sistema:
Inicialmente o Grafo não possui nenhum vértice e aresta, sendo necessário executar a opção de leitura presente no menu para obter o Grafo salvo no arquivo de texto. Feita essa leitura inicial que já permite ao usuário interagir com as informações dos aeroportos, ele também pode fazer a gravação em um arquivo .txt, que dado o comando de salvar pelo usuário, irá salvar todas as modificações feitas pelo usuário.

O objeto grafo do projeto pertence a uma classe Python, que possui os seguintes atributos: “n”, “m”, “mapa”, “adj”. O atributo “n” é responsável por guardar o número de nós presentes no grafo e utilizamos ele para manter as dimensões de nossa matriz adjacente. O atributo “m” é responsável por guardar o número de arestas do grafo. O atributo “mapa” é um dicionário Python, que no lugar das chaves guarda a sigla de um aeroporto e no lugar dos valores guardar um número correspondente ao nó daquele aeroporto na matriz de adjacência e o nome completo do aeroporto. Por fim, o atributo “adj” é a nossa matriz adjacência que tem dimensão de “n” por “n”, e em cada posição guarda em um dicionário os seguintes valores: a distância em KM da rota, os modelos dos aviões que fazem essa rota e as companhias aéreas que fazem essa rota.

Função de inicialização de classe GrafoND

```python
class GrafoND:
  TAM_MAX_DEFAULT = 100
  MODELOS_DEFAULT = ()
  def __init__(self, n=TAM_MAX_DEFAULT, mapa={}):
    self.n = n  # número de vértices
    self.m = 0  # número de arestas
    self.mapa = mapa
    self.adj = [[float('inf') for i in range(n)] for j in range(n)]
```

Implementação:

1. **Menu**

O nosso Menu é uma função que permite ao usuário informar o que ele quer fazer com o grafo e em caso de alteração do grafo solicita informações adicionais sobre como e onde a alteração deve ocorrer. Um exemplo disso é adição de um novo nó, na qual será solicitada a sigla do aeroporto e o seu nome completo.

```
-------------- GRAFO PRINCIPAIS ROTAS AEREAS INTERNACIONAIS --------------

1 - Ler dados do arquivo grafo.txt
2 - Gravar dados no arquivo grafo.txt;
3 - Inserir vértice;
4 - Inserir aresta
5 - Remove vértice
6 - Remove aresta
7 - Mostrar conteúdo do arquivo
8 - Mostrar grafo
9 - Conexidade do grafo
10 - Encerrar a aplicação.
/>
```

2. Ler dados do arquivo grafo.txt

Essa função faz a leitura do arquivo. txt e carrega as informações presente nele no grafo. Esse arquivo guarda informações sobre a quantidade de nós, sigla dos aeroportos, nome completo dos aeroportos, número de arestas, e as informações de cada aresta, que são: a sigla do aeroporto de origem, a sigla do aeroporto de destino, a distância em KM da rota, os modelos dos aviões que fazem essa rota e as companhias aéreas que fazem essa rota.

Ao realizar a leitura, atribuímos o número de nós lido ao nosso atributo “n”, guardamos as siglas dos aeroportos no atributo “mapa” e atribuímos a cada sigla um valor identificador, atribuímos ao atributo “m” o número de arestas e por fim completamos nossa matriz adjacência com as informações correspondetes a sigla de origem e a sigla de destinos dos aeroportos, passando para cada posição da matriz as informações distância em KM da rota, os modelos dos aviões que fazem essa rota e as companhias aéreas que fazem essa rota. Esse correspondência é feita por meio do atributo mapa que guarda um valor do índice da matriz adjacência que corresponde apenas a sigla daquele aeroporto, assim se o mapa guardar GRU:0, a primeira linha corresponderá as arestas que têm como origem GRU e a primeira coluna corresponde as arestas que têm como destino GRU.

Formato do arquivo de texto:

```
2
50
AMS Amsterdam Airport Schiphol
ATL Hartsfield-Jackson Atlanta International
Airport
AYT Antalya Airport
BCN Josep Tarradellas Barcelona-El Prat Airport
BKK Bangkok Suvarnabhumi Airport
BOG El Dorado International Airport
BOM Chhatrapati Shivaji Maharaj International
Airport
BOS Logan International Airport
CDG Charles de Gaulle Airport
CGK Soekarno-Hatta International Airport
CJU Jeju International Airport
CLT Charlotte Douglas International Airport
CUN Cancún International Airport
DEL Indira Gandhi International Airport
DEN Denver International Airport
DFW Dallas Fort Worth International Airport
DOH Hamad International Airport
```

```
656
AMS ATL 7074.0;['Delta Air Lines', 'KLM'];['A333',
'333', 'A359', '359', '350', 'B773', '77W']
AMS AYT 2658.0;['Corendon Dutch Airlines',
'Corendon Airlines', 'TUI Airlines Netherlands',
'Pegasus Airlines'];['738', 'B738', '320', '788',
'B788', '76W', 'A20N']
AMS BCN 1243.0;['Transavia', 'KLM', 'Vueling'];
['73H', '73W', 'B78', '38', 'B739', '73J', '32A',
'320', 'A321', '321', 'A320', 'A20N', '32Q', '32N']
AMS BOG 8850.0;['KLM'];['B789', '789']
AMS BOM 6867.0;['KLM', 'China Airlines',
'Singapore Airlines'];['333', '74Y', '332']
AMS CDG 399.0;['Air France', 'KLM'];['320', '32A',
'223', 'A321', '321', '318', 'BCS3', '223', '319',
'A319', 'B738', '73H', '7S7', '73J']
AMS CGK 11366.0;['Garuda Indonesia'];['77W', '777']
```

Função leituraTXT()::

```python
def leituraTXT():
  mapa = {}
  with open("grafo.txt", "r") as arquivo:
    tipo = arquivo.readline().strip('\n')
    n = int(arquivo.readline())
    for i in range(n):
      linha = arquivo.readline().strip('\n')
      linha = linha.split(" ",1)
      mapa[linha[0]] = [i,linha[1]]
    Grafo = GrafoND(n,mapa)
    m = int(arquivo.readline())
    for w in range(m):
      linha = arquivo.readline().strip('\n')
      linha = linha.split(" ",2)
      linha[2] = linha[2].replace("'", '')
      linha[2] = linha[2].strip("'")
      linha[2] = linha[2].split(';')
      linha[2][1]=linha[2][1].replace("[", "")
      linha[2][1]=linha[2][1].replace("]", "")
      linha[2][2]=linha[2][2].replace("]", "")
      linha[2][2]=linha[2][2].replace("[", "")
      dict = {
        "distancia": linha[2][0],
        "modelos": linha[2][2].strip().split(", "),
        "airlines": linha[2][1].strip().split(", ")
      }
      Grafo.insereA(linha[0],linha[1],dict)
  return tipo, Grafo
```

Teste de execução da opção 1 do menu:

```
1 - Ler dados do arquivo grafo.txt
2 - Gravar dados no arquivo grafo.txt;
3 - Inserir vértice;
4 - Inserir aresta
5 - Remove vértice
6 - Remove aresta
7 - Mostrar conteúdo do arquivo
8 - Mostrar grafo
9 - Conexidade do grafo
10 - Encerrar a aplicação.
/> 1
Aresta inserida!
Aresta inserida!
Aresta inserida!
Aresta inserida!
Aresta inserida!
Aresta inserida!
Aresta inserida!
Aresta inserida!
Aresta inserida!
Aresta inserida!
Aresta inserida!
```

3. Gravar dados no arquivo grafo.txt

Essa função faz a escrita no arquivo. txt e carrega as informações do grafo nele. Esse arquivo guarda informações sobre a quantidade de nós, sigla dos aeroportos, nome completo dos aeroportos, número de arestas, e as informações de cada aresta, que são: a sigla do aeroporto de origem, a sigla do aeroporto de destino, a distância em KM da rota, os modelos dos aviões que fazem essa rota e as companhias aéreas que fazem essa rota.

Na primeira linha do arquivo nós escrevemos o número 2 que é correspondente ao tipo “grafo não orientado com peso na aresta”. Na linha seguinte escrevemos o número de nós e nas linhas em diante escrevemos a sigla do aeroporto e seu nome completo. Quando chegamos ao fim dos nós, escrevemos o número de arestas e nas linhas seguinte as informações para estruturar cada aresta, que são: a sigla do aeroporto de origem, a sigla do aeroporto de destino, a distância em KM da rota, os modelos dos aviões que fazem essa rota e as companhias aéreas que fazem essa rota.

Dados salvos no arquivo de texto:

```
2
50
AMS Amsterdam Airport Schiphol
ATL Hartsfield-Jackson Atlanta International
Airport
AYT Antalya Airport
BCN Josep Tarradellas Barcelona-El Prat Airport
BKK Bangkok Suvarnabhumi Airport
BOG El Dorado International Airport
BOM Chhatrapati Shivaji Maharaj International
Airport
BOS Logan International Airport
CDG Charles de Gaulle Airport
CGK Soekarno-Hatta International Airport
CJU Jeju International Airport
CLT Charlotte Douglas International Airport
CUN Cancún International Airport
DEL Indira Gandhi International Airport
DEN Denver International Airport
DFW Dallas Fort Worth International Airport
DOH Hamad International Airport
```

```
656
AMS ATL 7074.0;['Delta Air Lines', 'KLM'];['A333',
'333', 'A359', '359', '350', 'B773', '77W']
AMS AYT 2658.0;['Corendon Dutch Airlines',
'Corendon Airlines', 'TUI Airlines Netherlands',
'Pegasus Airlines'];['738', 'B738', '320', '788',
'B788', '76W', 'A20N']
AMS BCN 1243.0;['Transavia', 'KLM', 'Vueling'];
['73H', '73W', 'B78', '38', 'B739', '73J', '32A',
'320', 'A321', '321', 'A320', 'A20N', '32Q', '32N']
AMS BOG 8850.0;['KLM'];['B789', '789']
AMS BOM 6867.0;['KLM', 'China Airlines',
'Singapore Airlines'];['333', '74Y', '332']
AMS CDG 399.0;['Air France', 'KLM'];['320', '32A',
'223', 'A321', '321', '318', 'BCS3', '223', '319',
'A319', 'B738', '73H', '7S7', '73J']
AMS CGK 11366.0;['Garuda Indonesia'];['77W', '777']
```

Teste de execução da opção 2 do menu:

```
1 - Ler dados do arquivo grafo.txt
2 - Gravar dados no arquivo grafo.txt;
3 - Inserir vértice;
4 - Inserir aresta
5 - Remove vértice
6 - Remove aresta
7 - Mostrar conteúdo do arquivo
8 - Mostrar grafo
9 - Conexidade do grafo
10 - Encerrar a aplicação.
/> 2

Grafo salvo no arquivo.
```

Função gravarTXT():

```
  def gravarTXT(self):
    lista =(list(self.mapa))
    with open("grafo.txt", "w") as arquivo:
      arquivo.write("2\n")
      arquivo.write(f"{self.n:2d}\n")
      for key,value in self.mapa.items():
        arquivo.write(f"{key} {self.mapa[key][1]}\n")
      arquivo.write(f"{self.m:2d}\n")
      for i in range(self.n):
        for w in range(i+1,self.n):
          if isinstance(self.adj[i][w], dict):
            arquivo.write(f"{lista[i]} {lista[w]} {self.adj[i][w]['distancia']};
{self.adj[i][w]['airlines']};{self.adj[i][w]['modelos']}\n")
```

4. Inserir vértice

O método de inserir um vértice dentro do menu requer o preenchimento da sigla do aeroporto a ser adicionado, assim como o nome completo do aeroporto. Após receber as informações, é chamado o método “insereV” da classe “GrafoND”.

Opção 3 do Menu:

```
case 3:
  sigla = input("Sigla do aeroporto: ").strip().upper()
  nome = input("Nome do aeroporto: ").strip()
  g1.insereV(sigla, nome)
```

Método insereV():

```
def insereV(self, sigla, nome):
  if sigla not in self.mapa:
    self.mapa[sigla] = [len(self.mapa), nome]
    self.n = len(self.mapa)
    for i in range(self.n - 1):
      self.adj[i].append(float('inf'))
    self.adj.append([float('inf') for z in range(self.n)])
    print("Vértice inserido")
```

Teste de execução da opção 3 do menu:

```
1 - Ler dados do arquivo grafo.txt
2 - Gravar dados no arquivo grafo.txt;
3 - Inserir vértice;
4 - Inserir aresta
5 - Remove vértice
6 - Remove aresta
7 - Mostrar conteúdo do arquivo
8 - Mostrar grafo
9 - Conexidade do grafo
10 - Encerrar a aplicação.
/> 3
Sigla do aeroporto: DES
Nome do aeroporto: Destino
Vértice inserido
```

Na função, é verificado se já não existe um vértice de mesma sigla no grafo. Caso seja uma nova sigla, o novo vértice é adicionado ao dicionário de vértices e a quantidade de vértices no grafo é alterada. Em seguida, a matriz de adjacência é aumentada para suportar o novo vértice, inicializado com infinito. Por fim, é realizado um print informando que o vértice foi inserido ao grafo.

5. Inserir Aresta

O método insereA é responsável por adicionar uma nova aresta no grafo, realizando um incremento no atributo “m” e alterando o valor da posição correspondente do aeroporto de origem e do aeroporto de destino na matriz adjacência. Assim, se a pessoa quiser inserir uma aresta com origem GRU e destino ATL, a pessoa terá de informar a sigla de origem, a sigla de destino, a distância da rota, a quantidade de modelos de avião que realizam essa rota, os nomes dos modelos de aviões que realizam essa rota, a quantidade de empresas aéreas que realizam voos nessa rota, e os nomes das empresas aéreas que fazem voos nesta rota. A posição correspondente desses aeroportos será obtida por meio do atributo “mapa”, que fornecerá o índice correspondente a cada sigla e nessa posição da matriz adjacência guarda as informações de distância, modelos e empresas aéreas.

Opção 4 do menu:

```
case 4:
  sigla_saida = input("Sigla de saida: ").strip().upper()
  sigla_chegada = input("Sigla de chegada: ").strip().upper()
  distancia = float(input("Distancia: "))
  qtdModelos = int(input("Quantidade Modelos: ").strip())
  modelos = []
  for i in range(qtdModelos):
    modelos.append((input("Modelos: ")).strip())
  qtdAirlines = int(input("Quantidade de Airlines: ").strip())
  Airlines = []
  for i in range(qtdAirlines):
    Airlines.append((input("Airline: ")).strip())
  dict = {
    "distancia": distancia,
    "modelos":modelos,
    "airlines":Airlines
  }
  g1.insereA(sigla_saida,sigla_chegada, dict)
```

Método InsereA():

```
def insereA(self, v, w, dict):
  if v in self.mapa and w in self.mapa:
    if self.adj[self.mapa[v][0]][self.mapa[w][0]] == float('inf'):
      self.adj[self.mapa[v][0]][self.mapa[w][0]] = dict
      self.adj[self.mapa[w][0]][self.mapa[v][0]] = dict
      self.m += 1
      print("Aresta inserida!")
```

Teste de execução da opção 4 do menu:

```
1 - Ler dados do arquivo grafo.txt
2 - Gravar dados no arquivo grafo.txt;
3 - Inserir vértice;
4 - Inserir aresta
5 - Remove vértice
6 - Remove aresta
7 - Mostrar conteúdo do arquivo
8 - Mostrar grafo
9 - Conexidade do grafo
10 - Encerrar a aplicação.
/> 4
Sigla de saida: ORG
Sigla de chegada: DES
Distancia: 400
Quantidade Modelos: 3
Modelos: A20N
Modelos: 320
Modelos: A320
Quantidade de Airlines: 1
Airline: GOL
Aresta inserida!
```

6. Remove vértice

A opção 5 do menu, ao ser chamada, pede como entrada a sigla do vértice a ser removido e o método removeV da classe GrafoND é chamada no menu com a sigla informada como parâmetro. No método, se a sigla estiver no dicionário de vértices, a matriz de adjacência será diminuída de forma a retirar a linha e coluna do nó sendo removido. Por fim, o nó é removido do dicionário de aeroportos, o grafo tem seu total de vértices atualizado e é realizado um print informando que o vértice foi removido.

Opção 5 do menu:

```
case 5:
  nome = input("Sigla do aeroporto: ").strip().upper()
  g1.removeV(nome)
```

Método removeV():

```
def removeV(self, nome):
  if nome in self.mapa:
    for i in range(self.n):
      self.adj[i].pop(self.mapa[nome][0])
    self.adj.pop(self.mapa[nome][0])
    for i in self.mapa.keys():
      if (self.mapa[nome][0] < self.mapa[i][0]):
        self.mapa[i][0] = self.mapa[i][0] - 1
    self.mapa.pop(nome, True)
    self.n = len(self.mapa)
    print("Vértice removido!")
```

Teste de execução da opção 5 do menu:

```
1 - Ler dados do arquivo grafo.txt
2 - Gravar dados no arquivo grafo.txt;
3 - Inserir vértice;
4 - Inserir aresta
5 - Remove vértice
6 - Remove aresta
7 - Mostrar conteúdo do arquivo
8 - Mostrar grafo
9 - Conexidade do grafo
10 - Encerrar a aplicação.
/> 5
Sigla do aeroporto: gru
Vértice removido!
```

7. Remove aresta

O método removeA() é responsável por remover uma aresta do grafo não-direcionado. Para a remoção, é necessário informar as siglas dos aeroportos adjacentes a aresta a ser removida. No método removeA(), primeiro é verificado se os vértices informados estão no dicionário de aeroportos. Caso estejam e o valor na matriz de adjacência seja diferente de infinito, as arestas v→w e w→v são alteradas para infinito. Por fim, o total de arestas “m” é decrementado em 1 e é realizado um print informando que a aresta foi removida.

Opção de remover aresta do menu:

```
case 6:
  saida = input("Sigla do aeroporto de saida: ").strip().upper()
  chegada = input("Sigla do aeroporto de chegada: ").strip().upper()
  g1.removeA(saida, chegada)
```

Método removeA():

```
def removeA(self, v, w):
  if v in self.mapa and w in self.mapa:
    if self.adj[self.mapa[v][0]][self.mapa[w][0]] != float('inf'):
      self.adj[self.mapa[v][0]][self.mapa[w][0]] = float('inf')
      self.adj[self.mapa[w][0]][self.mapa[v][0]] = float('inf')
      self.m -= 1
      print("Aresta removida!")
```

Teste de execução da opção 6 do menu:

```
1 - Ler dados do arquivo grafo.txt
2 - Gravar dados no arquivo grafo.txt;
3 - Inserir vértice;
4 - Inserir aresta
5 - Remove vértice
6 - Remove aresta
7 - Mostrar conteúdo do arquivo
8 - Mostrar grafo
9 - Conexidade do grafo
10 - Encerrar a aplicação.
/> 6
Sigla do aeroporto de saida: ams
Sigla do aeroporto de chegada: ayt
Aresta removida!
```

8. Mostrar conteúdo do arquivo

A opção 7 do menu chama a função showTXT(), que abre o arquivo e realiza o print de todas as suas linhas.

Função showTXT():

```
def showTXT():
  with open("grafo.txt", "r") as arquivo:
    Linhas = arquivo.readlines()
    for linha in Linhas:
      print("{}".format(linha.strip()))
```

Teste de execução da opção 7 do menu:

```
1 - Ler dados do arquivo grafo.txt
2 - Gravar dados no arquivo grafo.txt;
3 - Inserir vértice;
4 - Inserir aresta
5 - Remove vértice
6 - Remove aresta
7 - Mostrar conteúdo do arquivo
8 - Mostrar grafo
9 - Conexidade do grafo
10 - Encerrar a aplicação.
/> 7
2
50
AMS Amsterdam Airport Schiphol
ATL Hartsfield-Jackson Atlanta International Airport
AYT Antalya Airport
BCN Josep Tarradellas Barcelona-El Prat Airport
BKK Bangkok Suvarnabhumi Airport
BOG El Dorado International Airport
BOM Chhatrapati Shivaji Maharaj International Airport
BOS Logan International Airport
CDG Charles de Gaulle Airport
CGK Soekarno-Hatta International Airport
CJU Jeju International Airport
CLT Charlotte Douglas International Airport
CUN Cancún International Airport
DEL Indira Gandhi International Airport
DEN Denver International Airport
DFW Dallas Fort Worth International Airport
DOH Hamad International Airport
DXB Dubai International Airport
EWR Newark Liberty International Airport
FCO Leonardo da Vinci-Fiumicino Airport
FLL Fort Lauderdale-Hollywood International Airport
FRA Frankfurt Airport
GRU São Paulo/Guarulhos International Airport
HND Tokyo Haneda Airport
IAH George Bush Intercontinental Airport
IST Istanbul Airport
JED King Abdulaziz International Airport
JFK John F. Kennedy International Airport
LAS Harry Reid International Airport
LAX Los Angeles International Airport
LGA LaGuardia Airport
LGW London Gatwick Airport
LHR Heathrow Airport
MAD Adolfo Suárez Madrid-Barajas Airport
MCO Orlando International Airport
```

9. Mostrar grafo

A opção 8 do menu chama o método showMin() da classe GrafoND, que realiza o print das ligações entre dois aeroportos no seguinte formato: “sigla do primeiro aeroporto” “silga do segundo aeroporto” “distância”.

Método showMin():

```
def showMin(self):
  lista =(list(self.mapa))
  print(f"\n n: {self.n:2d} ", end="")
  print(f"m: {self.m:2d}\n")
  for i in range(self.n):
      for w in range(i+1,self.n):
        if isinstance(self.adj[i][w], dict):
          print(f"{lista[i]} {lista[w]} {self.adj[i][w]['distancia']}")
  print("\nfim da impressao do grafo.")
```

Teste de execução da opção 8 do menu:

```
1 - Ler dados do arquivo grafo.txt
2 - Gravar dados no arquivo grafo.txt;
3 - Inserir vértice;
4 - Inserir aresta
5 - Remove vértice
6 - Remove aresta
7 - Mostrar conteúdo do arquivo
8 - Mostrar grafo
9 - Conexidade do grafo
10 - Encerrar a aplicação.
/> 8

 n: 50 m: 656

AMS ATL 7074.0
AMS AYT 2658.0
AMS BCN 1243.0
AMS BOG 8850.0
AMS BOM 6867.0
AMS CDG 399.0
AMS CGK 11366.0
AMS DEL 6371.0
```

10. Conexidade do grafo

Por se tratar de um grafo não-direcionado, essa opção apenas trata se o grafo é conexo ou desconexo. Para isso, o método conexo() da classe GrafoND verifica se o primeiro vértice possui um percurso para todos os outros vértices, utilizando do outro método atingivelPercurso(), um algoritmo de busca em profundidade que tenta alcançar o vértice de destino parâmetro da função. O método atingivelPercurso() se utiliza de uma lista numérica em que os vértices já navegados tem seu índice na lista alterados para 1, de forma a não realizar realizar chamadas recursivas de vértices já alcançados.

Método conexo():

```python
def conexo(self):  #C0
  bool = True
  for i in range(1,self.n):
    teste = self.atingivelPercurso(0, i, [0] * self.n)
    if not teste:
      bool = False
  return bool
```

Método atingivelPercurso():

```python
def atingivelPercurso(self, s, c, lista):
  if s == c:
    return True
  elif (self.adj[s][c] != float('inf') or self.adj[c][s] != float('inf')):
    return True
  for i in range(self.n):
    if lista[i] == 0:
      if (self.adj[s][i] != float('inf')):
        lista[i] = 1
        if self.atingivelPercurso(i, c, lista):
          return True
      elif (self.adj[i][s] != float('inf')):
        lista[i] = 1
        if self.atingivelPercurso(i, c, lista):
          return True
  return False
```

Teste de execução da opção 9 do menu:

```
1 - Ler dados do arquivo grafo.txt
2 - Gravar dados no arquivo grafo.txt;
3 - Inserir vértice;
4 - Inserir aresta
5 - Remove vértice
6 - Remove aresta
7 - Mostrar conteúdo do arquivo
8 - Mostrar grafo
9 - Conexidade do grafo
10 - Encerrar a aplicação.
/> 9
Conexo
```

11. Encerrar a aplicação

A última opção do menu serve para encerrar o programa. Nesta opção é apenas realizado um “continue” pois o loop do while é encerrado ao ler que “resp” tem valor 10. No final, é realizado um print informando que o menu foi encerrado.

Opção 10 do menu:

```
case 10:
  continue
```

Teste de execução da opção 10 do menu:

```
1 - Ler dados do arquivo grafo.txt
2 - Gravar dados no arquivo grafo.txt;
3 - Inserir vértice;
4 - Inserir aresta
5 - Remove vértice
6 - Remove aresta
7 - Mostrar conteúdo do arquivo
8 - Mostrar grafo
9 - Conexidade do grafo
10 - Encerrar a aplicação.
/> 10
Menu encerrado
```