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

RELATÓRIO DO PROJETO – PARTE 2

NOME DOS INTEGRANTES	TIA
MATHEUS HENRIQUE DA SILVA APOSTULO	32092921
VALDIR LOPES JUNIOR	32095971

METRÔGRAFOS: MAPEAMENTO DE TODAS AS LINHAS DE METRÔ DE SÃO PAULO

MODELAGEM

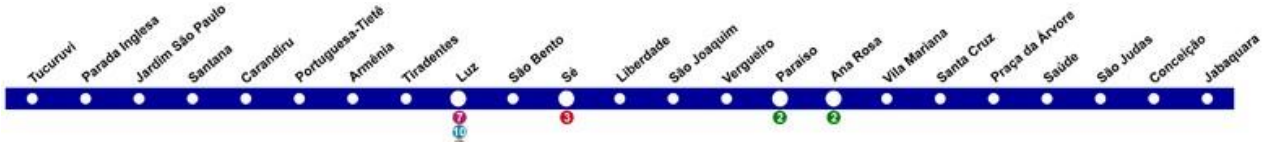
Para a modelagem do projeto, iniciamos na procura de um determinado problema para abordar, e nos encontramos na ideia de simular as linhas metroviárias de São Paulo utilizando um grafo não orientado com pesos nas arestas: comportamento observado para funcionamento da locomoção entre metrôs, pois você pode ir em dois sentidos, ou seja, ir e voltar na mesma "aresta". As linhas que escolhemos, totalizando o número de 98 vértices e 101 arestas, foram apenas as de metrô, sendo elas:

LINHA AMARELA – 11 ESTAÇÕES

LINHA VERDE – 14 ESTAÇÕES

LINHA VERMELHA – 18 ESTAÇÕES

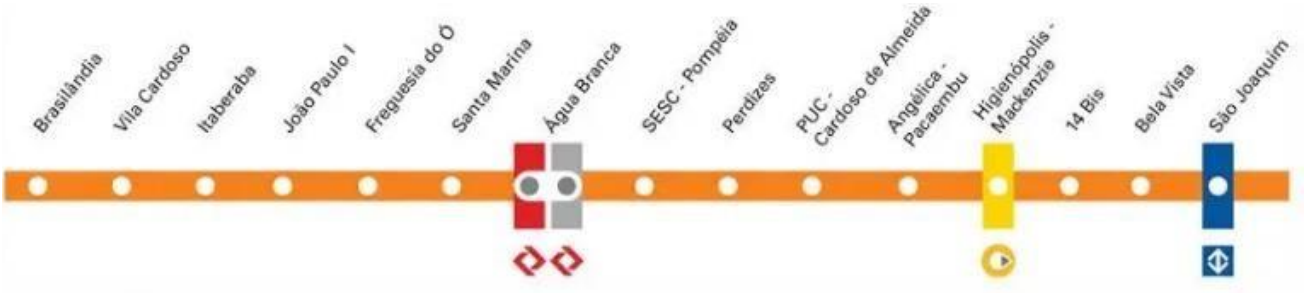
LINHA AZUL – 23 ESTAÇÕES



LINHA LILÁS – 17 ESTAÇÕES

LINHA PRATA – 11 ESTAÇÕES

LINHA LARANJA – 15 ESTAÇÕES



Após selecionar as linhas, pudemos modelar o grafo, já que cada vértice será uma estação e cada aresta o caminho a ser percorrido entre elas. Dessa forma, definimos:

VÉRTICE: NOME DA ESTAÇÃO

ARESTA: TEMPO EM MINUTOS DE DESLOCAMENTO ENTRE CADA UMA.

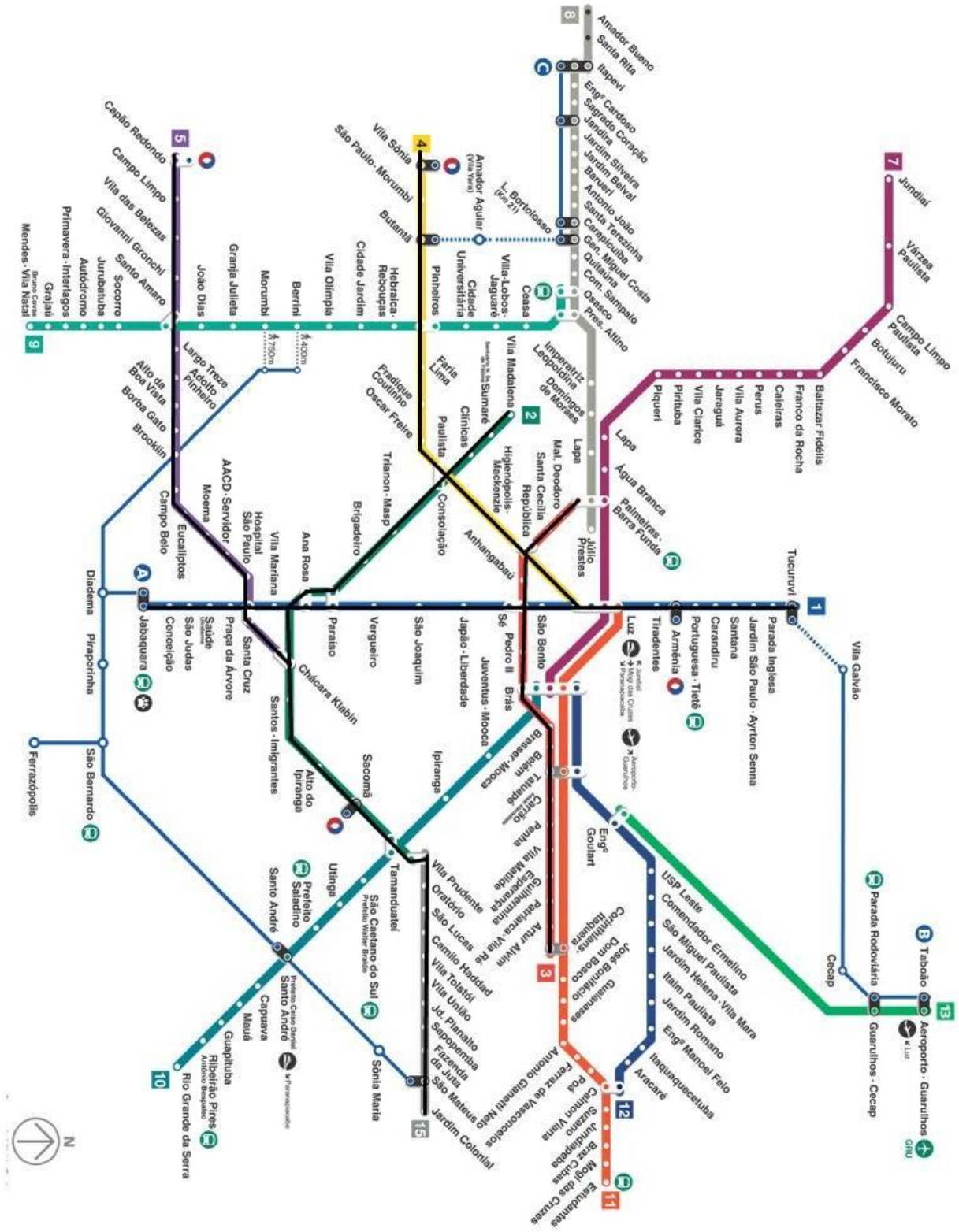
Com essas definições, realizamos a busca do tempo em minutos para atribuir em cada aresta. Para coletar essas informações, utilizamos o tempo estimado de uma estação a outra no sistema de rotas do Google Maps. Assim, nossas informações são o mais perto de precisas nesse problema. Seguindo nessa coleta, observamos que: a linha laranja, por ainda não estar em funcionamento, não nos é informado o tempo estimado. Buscamos a distância de cada estação da linha laranja para montar uma estimativa de quanto cada aresta iria receber de tempo. Utilizamos o mapa fornecido pelo site MetrôCPTM. Veja:

https://www.metrocptm.com.br/pela-6a-vez-governo-doria-adia-caducidade-da-linha-6-laranja/linha-6-laranja-mapa-geral/

Após realizar a análise, atribuímos os determinados pesos para cada aresta desta linha:

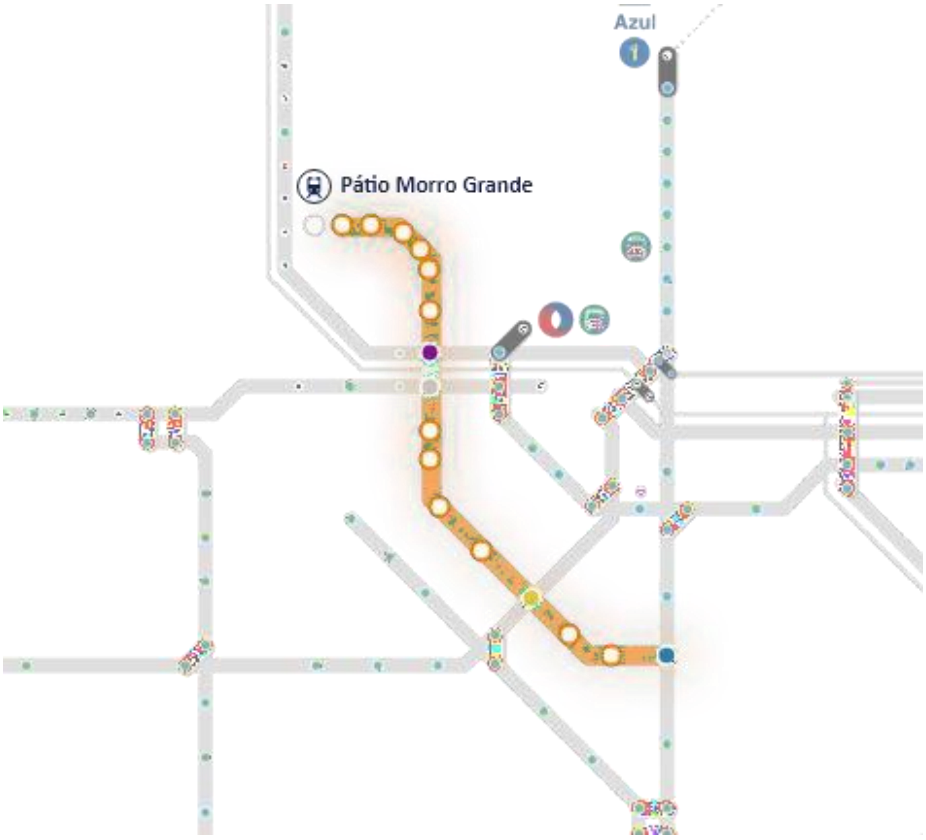
UNIVERSIDADE PRESBITERIANA MACKENZIE
Faculdade de Computação e Informática
Prof. Dr. Ivan Carlos Alcântara de Oliveira
Teoría dos Grafos

"GRAFO RELACIONADO" – MAPA DO METRÔ (SOMENTE AS LINHAS EM PRETO)



https://www.metrocptm.com.br/wp-content/uploads/2019/09/mapa -da-rede-metro-0522-abre-800x533.jpg

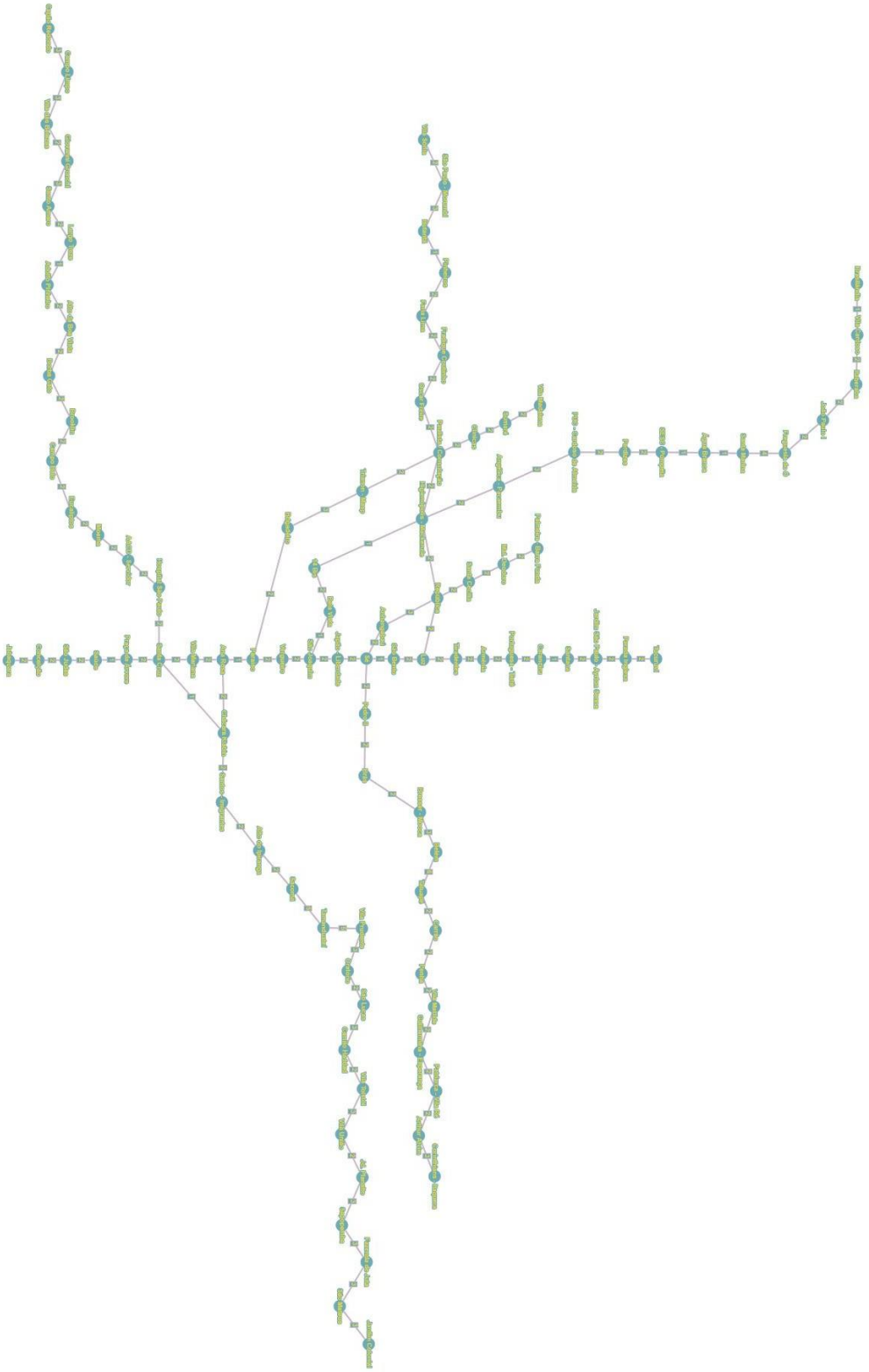
ADIÇÃO DA LINHA LARANJA:



https://www.linhauni.com.br/custom/site-linhauni-portal/img/map-internal.png

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

CRIAÇÃO DO GRAFO UTILIZANDO O GRAPH



https://graphonline.ru/en/?graph=pMTPSuhhWwfrhLJj

PARSING GRAPH ONLINE - CRIAÇÃO DO "GRAFO.TXT"

Após ter modelado o problema como grafo no Graph Online, de forma a gerar mais fácil o "grafo.txt" que será utilizado como forma de armazenamento do grafo como estrutura de dado, fizemos um *parsing* do arquivo XML que é gerado ao salvar um grafo no graph online, com esse *parsing*, conseguimos criar o "grafo.txt" de forma automatizada, assim, poupando o trabalho de fazer tudo a mão.

BEAUTIFUL SOUP

Para o auxílio da extração das informações referentes aos vértices e arestas presentas nas *tags* do XML do grafo que foi criado e gerado pelo Graph Online, utilizamos uma biblioteca em *python* própria para isso, então, demos um "*pip install beautifulsoup4*" no próprio *shell* do *replit* para baixar a *lib* e conseguir fazer nosso *parsing*.

```
~/pip-beaut$ pip install beautifulsoup4
Looking in indexes: https://package-proxy.replit.com/pypi/simple/
Collecting beautifulsoup4
  Downloading https://package-proxy.replit.com/pypi/packages/ee/a7/06b189a2e280e351adcef25df532af3c59442123187e2
28b960ab3238687/beautifulsoup4-4.12.0-py3-none-any.whl (132 kB)
     |                                | 10 kB 5.3 MB/s eta
     |                                | 20 kB 2.7 MB/s eta
     |                                | 30 kB 4.0 MB/s eta
     |                                | 40 kB 3.5 MB/s eta
     |                                | 51 kB 3.6 MB/s eta
     |                                | 61 kB 4.3 MB/s eta
     |                                | 71 kB 4.4 MB/s eta
     |                                | 81 kB 5.0 MB/s eta
     |                                | 92 kB 5.2 MB/s eta
     |                                | 102 kB 5.4 MB/s et
     |                                | 112 kB 5.4 MB/s et
     |                                | 122 kB 5.4 MB/s et
     |                                | 132 kB 5.4 MB/s
Collecting soupsieve>1.2
  Using cached soupsieve-2.4-py3-none-any.whl
WARNING: pip is using a content-addressable pool to install files from. This experimental feature is enabled thr
ough --use-feature=content-addressable-pool and it is not ready for production.
Installing collected packages: soupsieve, beautifulsoup4
Successfully installed beautifulsoup4-4.12.0 soupsieve-2.4
~/pip-beaut$
```

ESTRUTURA DO ARQUIVO TXT "GRAFO.TXT"

main.py × graph_biIyXIgpeHrnXudk.xml × grafo.txt × +

grafo.txt

```
1 1 "VILA SÔNIA"
2 2 "SÃO PAULO - MORUMBI"
3 3 "BUTANTÃ"
4 4 "PINHEIROS"
5 5 "FARIA LIMA"
6 6 "FRADIQUE COUTINHO"
```

main.py × graph_bilyXIgpeHrnXudk.xml × grafo.txt ×

grafo.txt

```
91  91 "ALTO DA BOA VISTA"
92  92 "ADOLFO PINHEIRO"
93  93 "LARGO TREZE"
94  94 "SANTO AMARO"
95  95 "GIOVANNI GRONCHI"
96  96 "VILA DAS BELEZAS"
97  97 "CAMPO LIMPO"
98  98 "CAPÃO REDONDO"
```

Optamos por colocar o tipo e tamanho do grafo (linha 1 e 2 como txt modelo pedido no documento do projeto) manualmente, pois acabaria sendo mais simples no fim das contas. Como podemos ver nas figuras acima, a partir do XML tirado do Graph Online, conseguimos extrair todos os vértices e numerá-los conforme eles foram aparecendo no arquivo que foi feito o *parsing*.

```
100  SÃO PAULO - MORUMBI, BUTANTÃ, 3
101  BUTANTÃ, PINHEIROS, 1
102  PINHEIROS, FARIA LIMA, 2
103  FARIA LIMA, FRADIQUE COUTINHO, 2
104  FRADIQUE COUTINHO, OSCAR FREIRE, 2
105  OSCAR FREIRE, PAULISTA / CONSOLAÇÃO, 2
106  PAULISTA / CONSOLAÇÃO, HIGIENÓPOLIS - MACKENZIE, 2
107  HIGIENÓPOLIS - MACKENZIE, REPÚBLICA, 2
108  REPÚBLICA, LUZ, 2
109  PAULISTA / CONSOLAÇÃO, TRIANON - MASP, 2
110  TRIANON - MASP, BRIGADEIRO, 3
111  BRIGADEIRO, PARAÍSO, 2
112  PARAÍSO, ANA ROSA, 1
113  ANA ROSA, CHÁCARA KLABIN, 2
114  CHÁCARA KLABIN, SANTOS - IMIGRANTES, 2
115  SANTOS - IMIGRANTES, ALTO DO IPIRANGA, 2
116  ALTO DO IPIRANGA, SACOMÃ, 2
117  SACOMÃ, TAMANDUATEÍ, 5
```

main.py × graph_bilyXIgpeHrnXudk.xml × grafo.txt ×

grafo.txt

```
193  ADOLFO PINHEIRO, ALTO DA BOA VISTA, 2
194  LARGO TREZE, ADOLFO PINHEIRO, 1
195  SANTO AMARO, LARGO TREZE, 2
196  GIOVANNI GRONCHI, SANTO AMARO, 3
197  VILA DAS BELEZAS, GIOVANNI GRONCHI, 2
198  CAMPO LIMPO, VILA DAS BELEZAS, 3
199  CAPÃO REDONDO, CAMPO LIMPO, 3
200
```

Acima, temos como ficou nossas Arestas. Optamos por separar as Estações/Vértices e peso por vírgula, para facilitar um pouco visualmente e ajudar posteriormente na leitura do arquivo já no projeto.

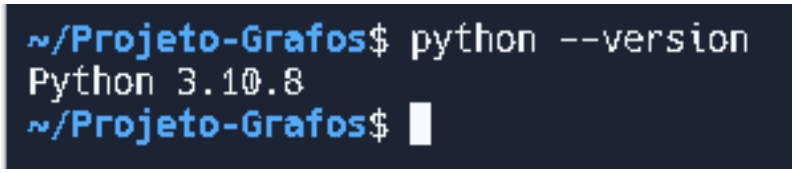
Portanto, temos nosso arquivo txt final com uma quantidade de 98 vértices e 100 arestas.

DESENVOLVIMENTO DA APLICAÇÃO

PLATAFORMA E VERSÃO

Podemos começar ressaltando que desenvolvemos o projeto na plataforma *replit*, devido à sua fácil forma de colaboração e simplicidade geral.

Além disso, devemos ressaltar que o projeto está rodando no *python 3.10.8*, que é o que o *replit* está suportando no momento. Essa versão é importante ser seguida, pois, por exemplo, a função "match", adicionado como case da *main* do grafo, só funciona nessa versão, ocasionando erro caso seja rodado em outras mais antigas.



IMPLEMENTAÇÕES IMPORTANTES

Julgamos importante listar alguns detalhes de implementação de forma prática, vamos explicar esses detalhes arquivo por arquivo:

grafoMatriz.py

- Na classe principal "Grafo", temos um array **"nomes_vertices"** referente a todos as estações/vértices dos nosso grafo, esse array irá possuir os nomes das estações, para, assim, podermos fazer as operações por nome, e não por posição na matriz.
- Ainda sobre a questão de fazer as operações por nome, temos a função **"getPosicaoNome"**, que quando chamada, descobre qual a posição de determinada estação na Matriz de Adjacência.
- O método **"show"** foi alterado para ser de fácil visualização do usuário, o método antigo, em que era mostrado toda a matriz de adjacência, poluía o terminal. Dito isso, mudamos o método para exibir apenas as posições da matriz que tem algum peso, ou seja, uma aresta entre dois vértices.
- A quantidade de vértice do grafo.txt (linha 2) é modificado de acordo com o self.n (quantidade de vértices do grafo).

main.py

- Para instanciar o grafo, enquanto liamos os dados do "grafo.txt", utilizamos um array "estacoes**"** (para os vértices) e uma a classe "**Aresta"** com um array **"arestas_listas"** do tipo **Aresta** (para as arestas). Assim, com essas estruturas, instanciamos o grafo mais facilmente.
- A maioria das opções (opções de manipulações), só é possível se lermos os dados do txt antes (instanciarmos o grafo), se tentar sem ler os dados do txt, terá um erro.

Tratamentos

Tratamos as coisas que mais julgamos importantes para o projeto nesse primeiro momento, que são:

- Uso do Upper (caixa alta) em todos os vértices e arestas do grafos.txt e em todos os inputs para evitar erros.
- Tratamento nos métodos, como por exemplo, erro caso o peso seja negativo e "vértice não existente no grafo" caso queira inserir uma aresta nova.

NOVAS FUNCIONALIDADES IMPLEMENTADAS NA PARTE 2 DO PROJETO

Análise da conexidade do grafo (opção 8)

Como requisito da segunda etapa, tivemos que analisar a conexidade do grafo criado.

De acordo com a teoria vista em aula, temos a seguinte definição:

Seja um grafo não direcionado G = (V, A)

- Realizar o percurso em largura ou profundidade partindo de um vértice v ∈ V qualquer .
- Contar a quantidade de vértices visitados durante o percurso.
- Se a quantidade visitada for igual ao número de vértices V do grafo G então o grafo não direcionado é conexo caso contrário é não conexo (desconexo).

Então, a partir disso, aplicamos o algoritmo de **percurso em largura** na opção 8 do código.

Obs.: O código pode ser encontrado no link do replit, do github ou nos apêndices desse documento, o nome do arquivo se encontra como "grafoMatriz.py". Além disso, podemos ver a execução dessa opção nos testes feitos posteriormente nesse mesmo documento.

Abaixo, podemos observar a comparação feita da quantidade de vértices no grafo “self.n” com a quantidade de vértices que foram visitados no algoritmo de percuso em profundidade.

```
# Tendo o percurso em profundidade, poderemos verificar a conexidade
print(f"Qtd de vértices = {self.n}. Qtd_vistiados = {quantidade_visitados}")
# Verifica se o grafo é conexo ou não
if(self.n == quantidade_visitados):
  print("O grafo é conexo!\n\n")
else:
  print("O grafo é não conexo!\n\n")
```

Menor caminho entre duas estações (opção 7)

De forma a tentar fornecer ao usuário uma rota de menor caminho entre duas estações do metrô que foram modelados em vértices, usamos um algoritmo de menor caminho, o algoritmo de Dijkstra.

Abaixo, podemos ver como manipulamos as estruturas obtidas na execução do algoritmo para exibir ao usuário as rotas e o TEMPO TOTAL DO PERCURSO.

```
    # Agora, iremos exibir quanto é a distância entre todos os vértice da partida ao
destino
    for i in range(len(rota_invertida)-1):
      #print("i = ",i, "i + 1 = ", i+1)
      print(f"{self.nomes_vertices[rota_invertida[i]]} --->
{self.nomes_vertices[rota_invertida[i+1]]} = {self.adj[rota_invertida[i]]
[rota_invertida[i+1]]} minutos\n" )

    # Por último, iremos printar o tempo total do percurso de acordo com o que foi
calculado no algoritmo
    print(f"\nTEMPO TOTAL DO PERCURSO = { d[indice_v2]} minutos")
```

Obs.: Da mesma forma da opção 8, para mais informações sobre o código da opção 7, ele pode ser encontrado no link do replit, do github ou nos apêndices desse documento, o nome do arquivo se encontra como “grafoMatriz.py”. Além disso, também podemos ver a execução dessa opção nos testes feitos posteriormente nesse mesmo documento.

Exibir adjacências de um vértice (opção 11)

Como não foi possível a implementação de uma exibição mais visual do grafo, adicionamos uma funcionalidade para visualizar as adjacência de um vértice específico. O funcionamento dessa opção pode ser observado nos testes da opção 11 posteriormente nesse mesmo documento.

Metrôgrafos e os Objetivos do Desenvolvimento Sustentável (ODS)

1. **ODS 8 - Trabalho Decente e Crescimento Econômico:**

(a) Emprego e produtividade: A aplicação pode ter um impacto positivo no mercado de trabalho, uma vez que um transporte público eficiente e confiável pode facilitar o acesso aos locais de trabalho, melhorando a mobilidade dos trabalhadores e contribuindo para a redução do tempo de deslocamento. Isso pode aumentar a produtividade, pois os trabalhadores chegam ao trabalho de forma mais rápida e com menos estresse.

2. **ODS 9 - Indústria, Inovação e Infraestrutura:**

(a) Eficiência do transporte público: A aplicação de navegação em metrô pode aprimorar a eficiência do transporte público, fornecendo rotas precisas e atualizadas aos usuários. Isso ajuda a otimizar o fluxo de passageiros, reduzindo atrasos e melhorando a pontualidade do serviço. Com uma infraestrutura mais eficiente, o sistema de transporte público pode atender melhor às necessidades dos usuários.

(b) Inovação tecnológica: A aplicação utiliza algoritmos avançados de grafos para calcular o menor caminho entre as estações de metrô. Essa abordagem inovadora demonstra como a tecnologia pode ser aplicada ao setor de transporte, melhorando a experiência dos usuários e impulsionando a inovação no campo da mobilidade urbana.

(c) Sustentabilidade: Ao incentivar o uso do transporte público, a aplicação contribui para a redução das emissões de gases de efeito estufa e a melhoria da qualidade do ar. Com uma infraestrutura de transporte público eficiente e informativo sobre quanto tempo pode durar um trajeto, as pessoas são encorajadas a deixar seus carros particulares em casa, diminuindo o congestionamento nas estradas e reduzindo a poluição ambiental.

3. **ODS 16 - Paz, Justiça e Instituições Eficazes:**

(a) Segurança no transporte: A aplicação de navegação em metrô pode melhorar a segurança do transporte público, fornecendo rotas seguras e confiáveis para os usuários. Isso é especialmente relevante para passageiros que não estão familiarizados com a cidade ou que utilizam o metrô em horários noturnos. O acesso a informações precisas e em tempo real pode contribuir para a sensação de segurança dos usuários.

(b) Eficiência e transparência: Ao fornecer informações claras sobre horários, rotas e atualizações do sistema, a aplicação aumenta a transparência do serviço de transporte público. Isso pode contribuir para a confiança dos usuários nas instituições responsáveis pelo transporte e garantir a eficiência dos serviços oferecidos.

APLICAÇÕES FUTURAS

Como o projeto tem caráter extensionista, segue abaixo ideias de aplicações futuras:

- Inserir elementos que aumentem a usabilidade do usuário. Ex.: Fornecer o mapa das estações para o usuário se situar.
- Exibir o grafo em alguma interface mais visual e de forma mais interativa e trivial ao usuário.
- Considerar, por meio de algum cálculo, o tempo de baldeação entre uma estação ou outra para gerar um caminho mais otimizado. Esse cálculo pode levar em consideração o tempo médio (de diferentes momentos do dia) necessário para sair de uma linha para outra em determinada estação que permita baldeação. Não conseguimos aplicar no projeto pois não existem dados sobre isso teríamos que fazer “manualmente”.
- Implementar algum tipo de atualização da duração do trajeto entre duas estações em tempo real, fazendo com que o algoritmo se adapte a diversos imprevistos que resultem em atrasos no percurso. Um exemplo de aplicação que faz algo semelhante é o Waze.
- Gerar um aplicativo que implemente o projeto feito, visando a utilização diária em situações reais.

LINKS IMPORTANTES

Link do Replit

https://replit.com/@mrRobotMackenzista/METROGRAFOS

Link do Github

https://github.com/matheusapostulo/METROGRAFOS

VÍDEO EXPLICATIVO

Abaixo, segue um vídeo disponibilizado na plataforma YouTube para mais informações referentes ao projeto, tal como a teoria se relacionando com a modelagem e as soluções.

Link do Vídeo no YouTube

https://www.youtube.com/watch?v=4C9rLMxCuf8

TESTES DA EXECUÇÕES

MENU

```
+----------------------------------------+
|              METRÔGRAFOS               |
+----------------------------------------+
| 1:  Ler dados do arquivo.              |
| 2:  Gravar os dados no arquivo.        |
| 3:  Inserir vértice.                   |
| 4:  Inserir aresta.                    |
| 5:  Remover vértice.                   |
| 6:  Remover aresta.                    |
| 7:  Menor caminho entre duas estações. |
| 8:  Mostrar conexidade do grafo.       |
| 9:  Mostrar conteúdo do arquivo.       |
| 10: Mostrar grafo.                     |
| 11: Exibir adjacências de um vértice.  |
| 12: Exibir objetivos da ODS do projeto.|
| 13: Encerrar a aplicação.              |
+----------------------------------------+
```

OPÇÃO 1:

LENDO OS DADOS...

```
Escolha uma opção: 1
Grafo criado com 98 vértices e 101 arestas!


+----------------------------------------+
|              METRÔGRAFOS               |
+----------------------------------------+
| 1:  Ler dados do arquivo.              |
| 2:  Gravar os dados no arquivo.        |
| 3:  Inserir vértice.                   |
| 4:  Inserir aresta.                    |
| 5:  Remover vértice.                   |
| 6:  Remover aresta.                    |
| 7:  Menor caminho entre duas estações. |
| 8:  Mostrar conexidade do grafo.       |
| 9:  Mostrar conteúdo do arquivo.       |
| 10: Mostrar grafo.                     |
| 11: Exibir adjacências de um vértice.  |
| 12: Exibir objetivos da ODS do projeto.|
| 13: Encerrar a aplicação.              |
+----------------------------------------+
```

CASO TENTAR OUTRA OPÇÃO SEM LER...

```
Escolha uma opção: 5
Por favor, leia os dados do arquivo (1) antes de selecionar essa opção!
```

OPÇÃO 2:

GRAVANDO OS DADOS...

```
Escolha uma opção: 2


+---------------------------------------+
|             METRÔGRAFOS               |
+---------------------------------------+
| 1:  Ler dados do arquivo.             |
| 2:  Gravar os dados no arquivo.       |
| 3:  Inserir vértice.                  |
| 4:  Inserir aresta.                   |
| 5:  Remover vértice.                  |
| 6:  Remover aresta.                   |
| 7:  Menor caminho entre duas estações.|
| 8:  Mostrar conexidade do grafo.      |
| 9:  Mostrar conteúdo do arquivo.      |
| 10: Mostrar grafo.                    |
| 11: Exibir adjacências de um vértice. |
| 12: Exibir objetivos da ODS do projeto.|
| 13: Encerrar a aplicação.             |
+---------------------------------------+
```

OPÇÃO 3:

INSERINDO UM VÉRTICE...

```
Escolha uma opção: 3
Insira um nome pra o vértice que você deseja inserir: COTIA
```

APÓS GRAVAR O CONTEÚDO, ESCOLHE-SE A OPÇÃO 7 – MOSTRAR CONTEÚDO DO ARQUIVO.

```
97 "CAMPO LIMPO"
98 "CAPÃO REDONDO"
99 "COTIA"
VILA SÔNIA, SÃO PAUL
```

```
Escolha uma opção: 3
Insira um nome pra o vértice que você deseja inserir: SOROCABA
```

APÓS GRAVAR O CONTEÚDO, ESCOLHE-SE A OPÇÃO 7 – MOSTRAR CONTEÚDO DO ARQUIVO.

```
99 "COTIA"
100 "SOROCABA"
VILA SONIA, SAO
```

OPÇÃO 4:

ADICIONANDO UMA ARESTA...

```
Escolha uma opção: 4
Diga o primeiro vértice que irá ter adjacência:COTIA
Diga o segundo vértice que irá ser adjacente ao anterior:SOROCABA
Digite o peso da adjacência entre os vértices inseridos:15
```

```
Escolha uma opção: 4
Diga o primeiro vértice que irá ter adjacência:MONTES CLAROS
Diga o segundo vértice que irá ser adjacente ao anterior:SÃO BERNARDO
Digite o peso da adjacência entre os vértices inseridos:30
```

VERIFICANDO SE AS ARESTAS FORAM CRIADAS...

```
VILA DAS BELEZAS, CAMPO LIMPO, 3
CAMPO LIMPO, CAPÃO REDONDO, 3
COTIA, SOROCABA, 15
MONTES CLAROS, SÃO BERNARDO, 30
```

OPÇÃO 5:

EXCLUINDO UM VÉRTICE...

```
Escolha uma opção: 5
Insira um nome do vértice que você deseja remover: VILA SÔNIA
```

VERIFICANDO A EXCLUSÃO...

```
2
101
1 "SAO PAULO - MORUMBI"
2 "BUTANTÃ"
3 "PINHEIROS"
4 "FARIA LIMA"
5 "FRADIQUE COUTINHO"
6 "OSCAR FREIRE"
7 "PAULISTA / CONSOLAÇÃO"
```

```
98 "COTIA"
99 "SOROCABA"
100 "MONTES CLAROS"
101 "SÃO BERNARDO"
SAO PAULO - MORUMBI, BUTANTÃ, 3
BUTANTÃ, PINHEIROS, 1
```

OBSERVA-SE QUE AS ARESTAS TAMBÉM FORAM REMOVIDAS.

EXCLUINDO UM VÉRTICE...

```
Escolha uma opção: 5
Insira um nome do vértice que você deseja remover: BUTANTÃ
```

VERIFICANDO A EXCLUSÃO...

```
2
100
1 "SÃO PAULO - MORUMBI"
2 "PINHEIROS"
3 "FARIA LIMA"
4 "FRADIQUE COUTINHO"
5 "OSCAR FREIRE"
```

```
99 "MONTES CLAROS"
100 "SÃO BERNARDO"
PINHEIROS, FARIA LIMA, 2
FARIA LIMA, FRADIQUE COUTINHO,
FRADIQUE COUTINHO, OSCAR FREIR
```

OBSERVA-SE QUE AS ARESTAS TAMBÉM FORAM REMOVIDAS.

OPÇÃO 6:

REMOVENDO UMA ARESTA...

```
Escolha uma opção: 6
Diga um vértice que você quer remover uma aresta:COTIA
Diga um vértice adjacente ao vértice anterior para remover a aresta:SOROCABA
```

VERIFICANDO A EXCLUSÃO...

```
LARGO TREZE, SANTO AMARO, 2
SANTO AMARO, GIOVANNI GRONCHI, 3
GIOVANNI GRONCHI, VILA DAS BELEZAS, 2
VILA DAS BELEZAS, CAMPO LIMPO, 3
CAMPO LIMPO, CAPÃO REDONDO, 3
MONTES CLAROS, SÃO BERNARDO, 30
```

```
95 "CAMPO LIMPO"
96 "CAPÃO REDONDO"
97 "COTIA"
98 "SOROCABA"
99 "MONTES CLAROS"
100 "SÃO BERNARDO"
PINHEIROS, FARIA LIM
```

OBSERVA-SE QUE OS VÉRTICES SE MANTÊM.

REMOVENDO UMA ARESTA...

```
Escolha uma opção: 6
Diga um vértice que você quer remover uma aresta:MONTES CLAROS
Diga um vértice adjacente ao vértice anterior para remover a aresta:SÃO BERNARDO
```

VERIFICANDO A EXCLUSÃO...

```
LARGO TREZE, SANTO AMARO, 2
SANTO AMARO, GIOVANNI GRONCHI, 3
GIOVANNI GRONCHI, VILA DAS BELEZAS, 2
VILA DAS BELEZAS, CAMPO LIMPO, 3
CAMPO LIMPO, CAPÃO REDONDO, 3
```

```
95 "CAMPO LIMPO"
96 "CAPÃO REDONDO"
97 "COTIA"
98 "SOROCABA"
99 "MONTES CLAROS"
100 "SÃO BERNARDO"
PINHEIROS, FARIA LIMA, 2
FARIA LIMA, FRADIQUE COUTIN
```

OBSERVA-SE QUE OS VÉRTICES SE MANTÊM.

OPÇÃO 7:

MENOR CAMINHO ENTRE DUAS ESTAÇÕES...

```
Escolha uma opção: 7
Diga o vértice origem: luz
Diga o vértice destino: saúde
```

VERIFICANDO O RESULTADO...

```
A seguir, temos a rota com o menor tempo entre LUZ e SAÚDE:

LUZ ---> SÃO BENTO = 2 minutos

SÃO BENTO ---> SÉ = 1 minutos

SÉ ---> JAPÃO - LIBERDADE = 2 minutos

JAPÃO - LIBERDADE ---> SÃO JOAQUIM = 2 minutos

SÃO JOAQUIM ---> VERGUEIRO = 2 minutos

VERGUEIRO ---> PARAÍSO = 2 minutos

PARAÍSO ---> ANA ROSA = 1 minutos

ANA ROSA ---> CHÁCARA KLABIN = 2 minutos

CHÁCARA KLABIN ---> SANTA CRUZ = 1 minutos

SANTA CRUZ ---> PRAÇA DA ÁRVORE = 2 minutos

PRAÇA DA ÁRVORE ---> SAÚDE = 1 minutos

TEMPO TOTAL DO PERCURSO = 18 minutos
```

PODEMOS OBSERVAR NO DESTAQUE QUE, ALÉM DO TEMPO ENTRE CADA ESTAÇÃO, TEMOS TAMBÉM O TEMPO TOTAL DO PERCURSO. TUDO EM MINUTOS.

OPÇÃO 8:

EXIBINDO CONEXIDADE DO GRAFO...

```
Escolha uma opção: 8
Qtd de vértices = 98. Qtd_vistiados = 98
O grafo é conexo!
```

PODEMOS OBSERVAR QUE A QUANTIDADE DE VÉRTICES E QUANTIDADE DE VISITADOS SENDO A MESMA, O GRAFO É CONEXO. VISTO QUE SE VISITARMOS TODOS OS VÉRTICES, É QUE TODOS SÃO CONECTADOS ENTRE SI!

```
Escolha uma opção: 3
Insira um nome pra o vértice que você deseja inserir: conexidade
```

```
Escolha uma opção: 8
Qtd de vértices = 99. Qtd_vistiados = 98
O grafo é não conexo!
```

AINDA, COMO PODEMOS OBSERVAR ACIMA, ADICIONANDO UM VÉRTICE SEM ADICIONAR ALGUMA ADJACÊNCIA, NÃO CONSEGUIMOS VISITAR ESSE VÉRTICE. ENTÃO, O GRAFO IRÁ SE TORNAR NÃO CONEXO.

OPÇÃO 9:

MOSTRANDO O CONTEÚDO...

```
Escolha uma opção: 9
Mostrando o conteúdo do arquivo:

2
98
1 "VILA SÔNIA"
2 "SÃO PAULO - MORUMBI"
3 "BUTANTÃ"
4 "PINHEIROS"
5 "FARIA LIMA"
6 "FRADIQUE COUTINHO"
7 "OSCAR FREIRE"
8 "PAULISTA / CONSOLAÇÃO"
9 "HIGIENÓPOLIS - MACKENZIE"
10 "REPÚBLICA"
11 "LUZ"
12 "VILA MADALENA"
13 "SUMARÉ"
14 "CLÍNICAS"
15 "TRIANON - MASP"
16 "BRIGADEIRO"
17 "PARAÍSO"
18 "ANA ROSA"
19 "CHÁCARA KLABIN"
20 "SANTOS - IMIGRANTES"
21 "ALTO DO IPIRANGA"
22 "SACOMÃ"
23 "TAMANDUATEÍ"
24 "VILA PRUDENTE"
25 "TIRADENTES"
```

...

```
FAZENDA DE JUTA, SÃO MATEUS, 3
SÃO MATEUS, JARDIM COLONIAL, 3
BELA VISTA, 14 BIS, 2
ANGÉLICA - PACAEMBU, PUC - CARDOSO DE ALMEIDA, 3
PUC - CARDOSO DE ALMEIDA, PERDIZES, 2
PERDIZES, SESC - POMPÉIA, 2
SESC - POMPÉIA, ÁGUA BRANCA, 1
ÁGUA BRANCA, SANTA MARINA, 1
SANTA MARINA, FREGUESIA DO Ó, 4
FREGUESIA DO Ó, JOÃO PAULO I, 2
JOÃO PAULO I, ITABERABA, 2
ITABERABA, VILA CARDOSO, 2
VILA CARDOSO, BRASILÂNDIA, 1
HOSPITAL SÃO PAULO, AACD - SERVIDOR, 2
AACD - SERVIDOR, MOEMA, 2
MOEMA, EUCALIPTOS, 2
EUCALIPTOS, CAMPO BELO, 2
CAMPO BELO, BROOKLIN, 2
BROOKLIN, BORBA GATO, 2
BORBA GATO, ALTO DA BOA VISTA, 2
ALTO DA BOA VISTA, ADOLFO PINHEIRO, 2
ADOLFO PINHEIRO, LARGO TREZE, 1
LARGO TREZE, SANTO AMARO, 2
SANTO AMARO, GIOVANNI GRONCHI, 3
GIOVANNI GRONCHI, VILA DAS BELEZAS, 2
VILA DAS BELEZAS, CAMPO LIMPO, 3
CAMPO LIMPO, CAPÃO REDONDO, 3


Conteúdo exibido!
```

OPÇÃO 10:

MOSTRANDO GRAFO...

```
Escolha uma opção: 10

 n: 98 m: 101

ESTAMOS NO VÉRTICE 'VILA SÔNIA':
 Adj[VILA SÔNIA, SÃO PAULO - MORUMBI] = 2


ESTAMOS NO VÉRTICE 'SÃO PAULO - MORUMBI':
 Adj[SÃO PAULO - MORUMBI, VILA SÔNIA] = 2
 Adj[SÃO PAULO - MORUMBI, BUTANTÃ] = 3


ESTAMOS NO VÉRTICE 'BUTANTÃ':
 Adj[BUTANTÃ, SÃO PAULO - MORUMBI] = 3
 Adj[BUTANTÃ, PINHEIROS] = 1


ESTAMOS NO VÉRTICE 'PINHEIROS':
 Adj[PINHEIROS, BUTANTÃ] = 1
 Adj[PINHEIROS, FARIA LIMA] = 2


ESTAMOS NO VÉRTICE 'FARIA LIMA':
 Adj[FARIA LIMA, PINHEIROS] = 2
 Adj[FARIA LIMA, FRADIQUE COUTINHO] = 2
```

...

```
ESTAMOS NO VÉRTICE 'CAMPO LIMPO':
 Adj[CAMPO LIMPO, VILA DAS BELEZAS] = 3
 Adj[CAMPO LIMPO, CAPÃO REDONDO] = 3


ESTAMOS NO VÉRTICE 'CAPÃO REDONDO':
 Adj[CAPÃO REDONDO, CAMPO LIMPO] = 3


ESTAMOS NO VÉRTICE 'COTIA':


ESTAMOS NO VÉRTICE 'SOROCABA':


ESTAMOS NO VÉRTICE 'MONTES CLAROS':


ESTAMOS NO VÉRTICE 'SÃO BERNARDO':


fim da impressao do grafo.
```

OPÇÃO 11:

MOSTRANDO ADJACÊNCIAS DE UM DETERMINADO VÉRTICE...

```
Escolha uma opção: 11
Diga o vértice que você deseja exibir as adjacências:luz

ADJACÊNCIAS NO VÉRTICE 'LUZ':
 Adj[LUZ, REPÚBLICA] = 2
 Adj[LUZ, TIRADENTES] = 2
 Adj[LUZ, SÃO BENTO] = 2
```

COMO PODEMOS OBSERVAR ACIMA, CONSEGUIMOS OBSERVAR TODAS AS ADJACÊNCIAS DE UM VÉRTICE DIGITADO!

OPÇÃO 12:

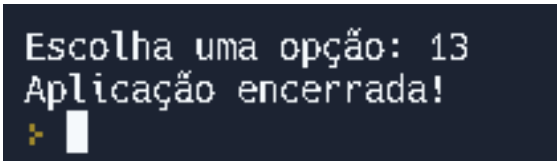
OBJETIVOS DA ODS...

```
Escolha uma opção: 12
+-----------------------------------------------------+
|                 OBJETIVOS DA ODS:                   |
+-----------------------------------------------------+
| ODS 8 - Trabalho Decente e Crescimento Econômico:   |
| (a)   Emprego e produtividade                       |
| (b)   Economia local                                |
|                                                     |
| ODS 9 - Indústria, Inovação e Infraestrutura:       |
| (a)   Eficiência do transporte público:             |
| (b)   Inovação tecnológica                          |
| (c)   Sustentabilidade                              |
|                                                     |
| ODS 16 - Paz, Justiça e Instituições Eficazes:      |
| (a)   Segurança no transporte                       |
| (b)   Eficiência e transparência                    |
|                                                     |
|   * PARA MAIS INFORMAÇÕES CONSULTE A DOCUMENTAÇÃO! * |
|   *    github.com/matheusapostulo/metrografos     * |
+-----------------------------------------------------+
```

COMO PODEMOS OBSERVAR, TEMOS OS TÓPICOS DA ODS QUE FORAM UTILIZADOS NA DOCUMENTAÇÃO DO PROJETO. COMO DESCRITO NA IMAGEM, PARA INFORMAÇÕES COMPLETAS E DETALHADAS O USUÁRIO DEVE CONSULTAR A DOCUMENTAÇÃO QUE ESTÁ NO REPOSITÓRIO DO GITHUB LINKADO.

OPÇÃO 13:

ENCERRANDO APLICAÇÃO...



APÊNDICES

GRAFO.TXT

98

1 "VILA SÔNIA"

2 "SÃO PAULO - MORUMBI"

3 "BUTANTÃ"

4 "PINHEIROS"

5 "FARIA LIMA"

6 "FRADIQUE COUTINHO"

7 "OSCAR FREIRE"

8 "PAULISTA / CONSOLAÇÃO"

9 "HIGIENÓPOLIS - MACKENZIE"

10 "REPÚBLICA"

11 "LUZ"

12 "VILA MADALENA"

13 "SUMARÉ"

14 "CLÍNICAS"

15 "TRIANON - MASP"

16 "BRIGADEIRO"

17 "PARAÍSO"

18 "ANA ROSA"

19 "CHÁCARA KLABIN"

20 "SANTOS - IMIGRANTES"

21 "ALTO DO IPIRANGA"

22 "SACOMÃ"

23 "TAMANDUATEÍ"

24 "VILA PRUDENTE"

25 "TIRADENTES"

26 "ARMÊNIA"

27 "PORTUGUESA - TIETÊ"

28 "CARANDIRU"

29 "SANTANA"

30 "JARDIM SÃO PAULO - AYRTON SENNA"

31 "PARADA INGLESA"

32 "TUCURUVI"

33 "SÃO BENTO"

34 "SÉ"

35 "JAPÃO - LIBERDADE"

36 "SÃO JOAQUIM"

37 "VERGUEIRO"

38 "VILA MARIANA"

39 "SANTA CRUZ"

40 "PRAÇA DA ÁRVORE"

41 "SAÚDE"

42 "SÃO JUDAS"

43 "CONCEIÇÃO"

44 "JABAQUARA"

45 "ANHANGABAÚ"

46 "SANTA CECILIA"

47 "MAL. DEODORO"

48 "PALMEIRA - BARRA FUNDA"

49 "PEDRO II"

50 "BRÁS"

51 "BRESSER - MOOCA"

52 "BELÉM"

53 "TATUAPÉ"

54 "CARRÃO"

55 "PENHA"

56 "VILA MATILDE"

57 "GUILHERMINA - ESPERANÇA"

58 "PATRIARCA - VILA RÉ"

59 "ARTHUR ALVIM"

60 "CORINTHIANS - ITAQUERA"

61 "ORATÓRIO"

62 "SÃO LUCAS"

63 "CAMILO HADDAD"

64 "VILA TOLSTÓI"

65 "VILA UNIÃO"

66 "JD. PLANALTO"

67 "SAPOPEMBA"

68 "FAZENDA DE JUTA"

69 "SÃO MATEUS"

70 "JARDIM COLONIAL"

71 "BELA VISTA"

72 "14 BIS"

73 "ANGÉLICA - PACAEMBU"

74 "PUC - CARDOSO DE ALMEIDA"

75 "PERDIZES"

76 "SESC - POMPÉIA"

77 "ÁGUA BRANCA"

78 "SANTA MARINA"

79 "FREGUESIA DO Ó"

80 "JOÃO PAULO I"

81 "ITABERABA"

82 "VILA CARDOSO"

83 "BRASILÂNDIA"

84 "HOSPITAL SÃO PAULO"

85 "AACD - SERVIDOR"

86 "MOEMA"

87 "EUCALIPTOS"

88 "CAMPO BELO"

89 "BROOKLIN"

90 "BORBA GATO"

91 "ALTO DA BOA VISTA"

92 "ADOLFO PINHEIRO"

93 "LARGO TREZE"

94 "SANTO AMARO"

95 "GIOVANNI GRONCHI"

96 "VILA DAS BELEZAS"

97 "CAMPO LIMPO"

98 "CAPÃO REDONDO"

VILA SÔNIA, SÃO PAULO - MORUMBI, 2

SÃO PAULO - MORUMBI, BUTANTÃ, 3

BUTANTÃ, PINHEIROS, 1

PINHEIROS, FARIA LIMA, 2

FARIA LIMA, FRADIQUE COUTINHO, 2

FRADIQUE COUTINHO, OSCAR FREIRE, 2

OSCAR FREIRE, PAULISTA / CONSOLAÇÃO, 2

PAULISTA / CONSOLAÇÃO, HIGIENÓPOLIS - MACKENZIE, 2

PAULISTA / CONSOLAÇÃO, CLÍNICAS, 2

PAULISTA / CONSOLAÇÃO, TRIANON - MASP, 2

HIGIENÓPOLIS - MACKENZIE, REPÚBLICA, 2

HIGIENÓPOLIS - MACKENZIE, 14 BIS, 1

HIGIENÓPOLIS - MACKENZIE, ANGÉLICA - PACAEMBU, 2

REPÚBLICA, LUZ, 2

REPÚBLICA, ANHANGABAÚ, 3

REPÚBLICA, SANTA CECILIA, 2

LUZ, TIRADENTES, 2

LUZ, SÃO BENTO, 2

VILA MADALENA, SUMARÉ, 2

SUMARÉ, CLÍNICAS, 2

TRIANON - MASP, BRIGADEIRO, 3

BRIGADEIRO, PARAÍSO, 2

PARAÍSO, ANA ROSA, 1

PARAÍSO, VERGUEIRO, 2

ANA ROSA, CHÁCARA KLABIN, 2

ANA ROSA, VILA MARIANA, 2

CHÁCARA KLABIN, SANTOS - IMIGRANTES, 2

CHÁCARA KLABIN, SANTA CRUZ, 1

SANTOS - IMIGRANTES, ALTO DO IPIRANGA, 2

ALTO DO IPIRANGA, SACOMÃ, 2

SACOMÃ, TAMANDUATEÍ, 5

TAMANDUATEÍ, VILA PRUDENTE, 5

VILA PRUDENTE, ORATÓRIO, 3

TIRADENTES, ARMÊNIA, 2

ARMÊNIA, PORTUGUESA - TIETÊ, 2

PORTUGUESA - TIETÊ, CARANDIRU, 2

CARANDIRU, SANTANA, 1

SANTANA, JARDIM SÃO PAULO - AYRTON SENNA, 2

JARDIM SÃO PAULO - AYRTON SENNA, PARADA INGLESA, 2

PARADA INGLESA, TUCURUVI, 2

SÃO BENTO, SÉ, 1

SÉ, JAPÃO - LIBERDADE, 2

SÉ, ANHANGABAÚ, 2

SÉ, PEDRO II, 2

JAPÃO - LIBERDADE, SÃO JOAQUIM, 2

SÃO JOAQUIM, VERGUEIRO, 2

SÃO JOAQUIM, BELA VISTA, 1

VILA MARIANA, SANTA CRUZ, 2

SANTA CRUZ, PRAÇA DA ÁRVORE, 2

SANTA CRUZ, HOSPITAL SÃO PAULO, 2

PRAÇA DA ÁRVORE, SAÚDE, 1

SAÚDE, SÃO JUDAS, 2

SÃO JUDAS, CONCEIÇÃO, 2

CONCEIÇÃO, JABAQUARA, 2

SANTA CECILIA, MAL. DEODORO, 2

MAL. DEODORO, PALMEIRA - BARRA FUNDA, 2

PEDRO II, BRÁS, 2

BRÁS, BRESSER - MOOCA, 2

BRESSER - MOOCA, BELÉM, 3

BELÉM, TATUAPÉ, 6

TATUAPÉ, CARRÃO, 2

CARRÃO, PENHA, 2

PENHA, VILA MATILDE, 2

VILA MATILDE, GUILHERMINA - ESPERANÇA, 2

GUILHERMINA - ESPERANÇA, PATRIARCA - VILA RÉ, 2

PATRIARCA - VILA RÉ, ARTHUR ALVIM, 2

ARTHUR ALVIM, CORINTHIANS - ITAQUERA, 3

ORATÓRIO, SÃO LUCAS, 2

SÃO LUCAS, CAMILO HADDAD, 3

CAMILO HADDAD, VILA TOLSTÓI, 2

VILA TOLSTÓI, VILA UNIÃO, 3

VILA UNIÃO, JD. PLANALTO, 2

JD. PLANALTO, SAPOPEMBA, 3

SAPOPEMBA, FAZENDA DE JUTA, 3

FAZENDA DE JUTA, SÃO MATEUS, 3

SÃO MATEUS, JARDIM COLONIAL, 3

BELA VISTA, 14 BIS, 2

ANGÉLICA - PACAEMBU, PUC - CARDOSO DE ALMEIDA, 3

PUC - CARDOSO DE ALMEIDA, PERDIZES, 2

PERDIZES, SESC - POMPÉIA, 2

SESC - POMPÉIA, ÁGUA BRANCA, 1

ÁGUA BRANCA, SANTA MARINA, 1

SANTA MARINA, FREGUESIA DO Ó, 4

FREGUESIA DO Ó, JOÃO PAULO I, 2

JOÃO PAULO I, ITABERABA, 2

ITABERABA, VILA CARDOSO, 2

VILA CARDOSO, BRASILÂNDIA, 1

HOSPITAL SÃO PAULO, AACD - SERVIDOR, 2

AACD - SERVIDOR, MOEMA, 2

MOEMA, EUCALIPTOS, 2

EUCALIPTOS, CAMPO BELO, 2

CAMPO BELO, BROOKLIN, 2

BROOKLIN, BORBA GATO, 2

BORBA GATO, ALTO DA BOA VISTA, 2

ALTO DA BOA VISTA, ADOLFO PINHEIRO, 2

ADOLFO PINHEIRO, LARGO TREZE, 1

LARGO TREZE, SANTO AMARO, 2

SANTO AMARO, GIOVANNI GRONCHI, 3

GIOVANNI GRONCHI, VILA DAS BELEZAS, 2

VILA DAS BELEZAS, CAMPO LIMPO, 3

CAMPO LIMPO, CAPÃO REDONDO, 3

MAIN.PY

```
"""
32092921 | MATHEUS HENRIQUE DA SILVA APOSTULO
32095971 | VALDIR LOPES JUNIOR
"""

from grafoMatriz import Grafo
# Classe para auxiliar nas ligações das arestas do grafo
class Aresta:
  def __init__(self,a1,a2,peso):
    self.aresta1 = a1
    self.aresta2 = a2
    self.peso = peso

# Lista global com todos os nomes, o tamanho da lista vai iniciar o grafo
estacoes = []
#Lista global de arestas que vai ajudar a popular o grafo
arestas_lista = []
# Controle do carregamento do item "a"
controle_carregamento = True

# MENU
while True:
  print("+----------------------------------------+")
  print("|          METRÔGRAFOS                   |")
  print("+----------------------------------------+")
  print("| 1:  Ler dados do arquivo.              |")
  print("| 2:  Gravar os dados no arquivo.        |")
  print("| 3:  Inserir vértice.                   |")
  print("| 4:  Inserir aresta.                    |")
  print("| 5:  Remover vértice.                   |")
  print("| 6:  Remover aresta.                    |")
  print("| 7:  Menor caminho entre duas estações. |")
  print("| 8:  Mostrar conexidade do grafo.       |")
  print("| 9:  Mostrar conteúdo do arquivo.       |")
  print("| 10: Mostrar grafo.                     |")
  print("| 11: Exibir adjacências de um vértice.  |")
  print("| 12: Exibir objetivos da ODS do projeto.|")
  print("| 13: Encerrar a aplicação.              |")
  print("+----------------------------------------+\n")

  opcao = int(input("Escolha uma opção: "))

  match opcao:
    case 1:
      if controle_carregamento:
        # a) Ler dados do arquivo grafo.txt;
        with open("grafo.txt", "r") as grf:
            lines = grf.readlines()
            for i, line in enumerate(lines): # i: percorre linhas txt, line: conteúdo da linha
              match i:
                case 0:
                  tipo_grafo = line
                case 1:
```

```
            tamanho_inicial = line
          case _:
            if "" in line: # Vértice com rótulo
              # Fazendo o split para pegar apenas o rótulo e jogar na lista global
              line_splitada = line.split("")
              line_splitada = line_splitada[1]
              estacoes.append(line_splitada)

            else: # Arestas e ligações
              # splitando linhas arestas e guardando dados em variáveis
              arestas_txt_split = line.split(", ")
              #print(arestas_txt_split)

              a_1 = arestas_txt_split[0]
              a_2 = arestas_txt_split[1]
              peso = arestas_txt_split[2]
              #print(f"a1: {a_1}, a2:{a_2}, peso: {peso}")

              # instanciando arestas e atribuindo os atributos
              aresta_obj = Aresta(a_1, a_2, int(peso))
              arestas_lista.append(aresta_obj)


        #CRIANDO GRAFO
        grafo = Grafo(len(estacoes)) # instancia pelo tamanho do vetor grafo

        # Inserindo vértices no grafo com a lista criada anteriormente
        for nome_estacao in estacoes:
          grafo.insereV_txt(nome_estacao)

        # Inserindo arestas no grafo
        for aresta in arestas_lista:
          grafo.insereA(aresta.aresta1, aresta.aresta2, aresta.peso)
        #Fechando o arquivo txt
        grf.close()

        # Mostrando informações iniciais e atestando que o grafo foi instanciado
        tamanho_vertices_split = tamanho_inicial.split("\n")
        print(f"Grafo criado com {tamanho_vertices_split[0]} vértices e {len(arestas_lista)} arestas!")

        # Mudando variável de controle para não entrar mais nessa opção
        controle_carregamento = False

      else: # Caso o arquivo já tenha sido carregado
        print("O ARQUIVO TXT JÁ FOI CARRREGADO NO GRAFO!")

    case 2:
      if controle_carregamento == False:
        # Chama o método de gravar dados
        grafo.gravarDados()
      else:
        print("Por favor, leia os dados do arquivo (1) antes de selecionar outras opções!")

    case 3:
      if controle_carregamento == False:
        inserir_vertice = input("Insira um nome pra o vértice que você deseja inserir: ")
        grafo.insereV(inserir_vertice.upper())
```

```
  else:
    print("Por favor, leia os dados do arquivo (1) antes de selecionar outras opções!")

case 4:
  if controle_carregamento == False:
    v1 = input("Diga o primeiro vértice que irá ter adjacência:")
    v2 = input("Diga o segundo vértice que irá ser adjacente ao anterior:")
    peso = int(input("Digite o peso da adjacência entre os vértices inseridos:"))
    grafo.insereA(v1.upper(), v2.upper(), peso)
  else:
    print("Por favor, leia os dados do arquivo (1) antes de selecionar outras opções!")

case 5:
  if controle_carregamento == False:
    remover_vertice = input("Insira um nome do vértice que você deseja remover: ")
    grafo.removeV(remover_vertice.upper())
  else:
    print("Por favor, leia os dados do arquivo (1) antes de selecionar essa opção!")

case 6:
  if controle_carregamento == False:
    a1 = input("Diga um vértice que você quer remover uma aresta:")
    a2 = input("Diga um vértice adjacente ao vértice anterior para remover a aresta:")
    grafo.removeA(a1.upper(), a2.upper())
  else:
    print("Por favor, leia os dados do arquivo (1) antes de selecionar essa opção!")

case 7:
  if controle_carregamento == False:
    v1 = input("Diga o vértice origem: ")
    v2 = input("Diga o vértice destino: ")
    grafo.menor_caminho(v1.upper(),v2.upper())
  else:
    print("Por favor, leia os dados do arquivo (1) antes de selecionar essa opção!")

case 8:
  grafo.conexidade()

case 9:
  print("Mostrando o conteúdo do arquivo:\n")
  with open("grafo.txt", "r") as conteudo:
      lines = conteudo.read()
      print(lines)
  print("\nConteúdo exibido!")

  conteudo.close()

case 10:
  if controle_carregamento == False:
    grafo.show()
  else:
    print("Por favor, leia os dados do arquivo (1) antes de selecionar essa opção!")

case 11:
  if controle_carregamento == False:
    vertice = input("Diga o vértice que você deseja exibir as adjacências:")
    grafo.mostrarAdjacenciaVertice(vertice.upper())
```

```
    else:
      print("Por favor, leia os dados do arquivo (1) antes de selecionar essa opção!")

  case 12:
    print("+-----------------------------------------------------+")
    print("|              OBJETIVOS DA ODS:                      |")
    print("+-----------------------------------------------------+")
    print("| ODS 8 - Trabalho Decente e Crescimento Econômico:    |")
    print("| (a) Emprego e produtividade                         |")
    print("| (b) Economia local                                  |")
    print("|                                                     |")
    print("| ODS 9 - Indústria, Inovação e Infraestrutura:       |")
    print("| (a) Eficiência do transporte público:               |")
    print("| (b) Inovação tecnológica                            |")
    print("| (c) Sustentabilidade                                |")
    print("|                                                     |")
    print("| ODS 16 - Paz, Justiça e Instituições Eficazes:      |")
    print("| (a) Segurança no transporte                         |")
    print("| (b) Eficiência e transparência                      |")
    print("|                                                     |")
    print("|   * PARA MAIS INFORMAÇÕES CONSULTE A DOCUMENTAÇÃO! * |")
    print("|   *    github.com/matheusapostulo/metrografos       * |")
    print("+-----------------------------------------------------+\n")



  case 13:
    print("Aplicação encerrada!")
    break

  case _:
    print("ERRO! ENTRADA INVÁLIDA!")
print("\n")
```

GRAFOMATRIZ.PY

```
"""
32092921 | MATHEUS HENRIQUE DA SILVA APOSTULO
32095971 | VALDIR LOPES JUNIOR
"""

from math import inf
import os
from copy import deepcopy # Usado nos métodos de cópia de lista. Opções 2 e 4!
from pilha import Pilha

infinito = inf

class Grafo:
  TAM_MAX_DEFAULT = 100  # qtde de vértices máxima default

  # construtor da classe grafo
  def __init__(self, n=TAM_MAX_DEFAULT):
    self.n = n  # número de vértices
    self.m = 0  # número de arestas
    # matriz de adjacência
    self.adj = [[infinito for i in range(n)] for j in range(n)]
    self.nomes_vertices = [] #Lista com nomes dos vértices, manipulação vai ser feita aqui

  # Atribui nome a um número de vértice
  def atribuiVertice(self, nome_vertice, numero_vertice):
    self.nomes_vertices[numero_vertice] = nome_vertice

  #Percorre a lista de nomes e retorna a posição que o nome está
  def getPosicaoNome(self, nome_vertice):
    for i in range(len(self.nomes_vertices)):
      if self.nomes_vertices[i] == nome_vertice:
```

```
      return i


    return -1

  def insereV_txt(self, nome_vertice): # Esse método é o usado para a leitura do txt apenas
    self.nomes_vertices.append(nome_vertice)

  # b) Gravar dados no arquivo grafo.txt;
  def gravarDados(self):
    # Apaga o arquivo pois o grafo pode estar totalmente diferenre do original
    os.remove("grafo.txt")
    # Abre o arquivo
    with open("grafo.txt", "w") as grf_r:
      # Laço que escreve os VÉRTICES EXISTENTES NO GRAFO
      for i in range(len(self.nomes_vertices)+2):# percorre linhas txt, line: conteúdo da linha
        match i:
          case 0:
            grf_r.write("2\n")
          case 1:
            grf_r.write(f"{str(self.n)}\n")
          case _:
            string_vertice = f'{i-1} "{self.nomes_vertices[i-2]}"\n'
            grf_r.write(string_vertice)
      # Laço para percorrer a matriz principal para pegar as arestas
      # Utilizaremos um matriz cópia auxiliar e iremos setando infinito no contrário da posição([i][j] ->
[j][i])
      matriz_aux = deepcopy(self.adj) # usando lib copy para criar uma cópia
      for i in range(self.n):
        for j in range(self.n):
          if matriz_aux[i][j] != infinito: # verifica se não é infinito
            # Guarda os nomes dos vértice pela posição e também o peso
            vertice_i = self.nomes_vertices[i]
            vertice_j = self.nomes_vertices[j]
            peso = self.adj[i][j]
```

```
            string_aresta = f'{vertice_i}, {vertice_j}, {peso}\n'
            grf_r.write(string_aresta)
            # Agora iremos setar infinito na posição inversa para não pegarmos a mesma relação
            matriz_aux[j][i] = infinito


        #print(f"ASSIM FICOU A MATRIZ AUX: {matriz_aux}")
        #print(f"ASSIM FICOU A LISTA ADJ: {self.adj}")
        grf_r.close()
```

c) Insere vértice na lista que representa o grafo. Irá inserir na última posição, não fará diferença na lógica geral pois pesquisamos por nomes para fazer as operações dos métodos

```
    def insereV(self, nome_vertice):
        # Se o vértice não existe, adiciona à última posição da lista de nomes e aumenta o tamanho de vértices do grafo
        if nome_vertice not in self.nomes_vertices:
            # Copia a matriz antiga para futuras operações no método
            matriz_copia = deepcopy(self.adj)
            # Add à lista de nomes de vértices
            self.nomes_vertices.append(nome_vertice)
            self.n += 1
            # Resetaremos a matriz do grafo com o novo tamanho
            self.adj = [[infinito for i in range(self.n)] for j in range(self.n)]
            # Repaseremos os dados anteriores para a lista nova (somente onde tinha aresta)
            for i in range(len(matriz_copia)):
                for j in range(len(matriz_copia)):
                    if matriz_copia[i][j] != infinito:
                        self.adj[i][j] = matriz_copia[i][j]

        else:
            print("ESSE VÉRTICE JÁ EXISTE NO GRAFO! Tente outro da próxima vez!")
```

d) Insere uma aresta no Grafo tal que v é adjacente a w (pega a posição pelo nome). Não precisamos saber a posição, apenas o nome para poder ligar dois vértices.

```
def insereA(self, vertice_i, vertice_j, peso):
  if peso > 0:
    pos_i = self.getPosicaoNome(vertice_i)
    pos_j = self.getPosicaoNome(vertice_j)
    if pos_i != -1 and pos_j != -1:
      if self.adj[pos_i][pos_j] == infinito and self.adj[pos_j][pos_i] == infinito:
        self.adj[pos_i][pos_j] = peso
        self.adj[pos_j][pos_i] = peso
        self.m += 1  # atualiza qtd arestas
      elif self.adj[pos_i][pos_j] != infinito and self.adj[pos_j][pos_i] != infinito: # Atualizar com um novo peso caso já tenha um peso, condicional para n mudar qtd aresta
        self.adj[pos_i][pos_j] = peso
        self.adj[pos_j][pos_i] = peso
    else:
      print("Algum dos vértices digitados não existe!\n")
  else:
    print("Insira um peso positivo maior que 0!")


 # e) Método para remover vértice do grafo ND
 def removeV(self, vertice):
  if vertice in self.nomes_vertices:
    posicao_vertice = self.getPosicaoNome(vertice)
    # Removendo as arestas com todos os outros vértices primeiro antes de apagar as posições do vetor
    for i in range(self.n):
      vertice_2 = self.nomes_vertices[i] # variável que pega todos os nomes dos vértices existentes na lista de nomes
      self.removeA(vertice,vertice_2) # remove as ligações existentes com os outros vértices
    # Atualiza a matriz (removendo os elementos)
    del self.adj[posicao_vertice] # Apaga o vértice
    self.n -= 1 # Atualiza a quantidade de vértices
    #Atualizando as colunas desalocando o espaço do antigo vértice
    for i in range(self.n):
      del self.adj[i][posicao_vertice]
```

```
    # Apagando o vértice da lista de nomes
    del self.nomes_vertices[posicao_vertice]
  else:
    print("Vértice não existente no grafo!")

# f) remove uma aresta v->w do Grafo utiliza nome para achar a posição
def removeA(self, vertice_i, vertice_j):
  pos_i = self.getPosicaoNome(vertice_i)
  pos_j = self.getPosicaoNome(vertice_j)
  if pos_i != -1 and pos_j != -1:
    # testa se temos a aresta
    if self.adj[pos_i][pos_j] != infinito and self.adj[pos_j][pos_i] != infinito:
      self.adj[pos_i][pos_j] = infinito
      self.adj[pos_j][pos_i] = infinito
      self.m -= 1
    # atualiza qtd arestas
  else:
    print("\nUm dos vértices digitados não existe!\n")

# g) Verifica se o grafo é conexo ou não-conexo
#PERGUNTAR PRO PROF SE PRECISA DO VERTICE INICIAL
def adjacenciasVertice(self, n, nosMarcados):
  vetorAdjacencias = []
  for i in range(self.n): #percorre linhas
    if self.adj[n][i] != infinito and i not in nosMarcados: # se o não estiver marcado e é adjacente a n
      vetorAdjacencias.append(i)
  return vetorAdjacencias

def adjacenciasV(self, n):
  vetorAdjacencias = []
  for i in range(self.n): #percorre linhas
    if self.adj[n][i] != infinito: # se é adjacente a n
      vetorAdjacencias.append(i)
  return vetorAdjacencias
```

```
 def conexidade(self):
   verticeInicio = 0
   # Cria a pilha e array de nós marcados e contador para marcar os nós visitados
   quantidade_visitados = 0
   nosMarcados = []
   pilha = Pilha()
   # Visita o Nó
   #print(f"Nó inicial visitado: {verticeInicio}")
   quantidade_visitados += 1
   # Marca o nó inicial
   nosMarcados.append(verticeInicio)
   # Empilha o nó
   pilha.push(verticeInicio)

   while not pilha.isEmpty():
     n = pilha.pop()
     #print("Pilha está assim = ", pilha)

     adjacentesDeN = self.adjacenciasVertice(n, nosMarcados)
     #print(f"adjacentes de {n} = ", adjacentesDeN)

     while len(adjacentesDeN) > 0: # Roda em todos os adjacentes de "n" que ainda não foram
marcados
       #print("Nó m visitado = ", chr((adjacentesDeN[0]+1) + 96))
       quantidade_visitados += 1 # incrementa visitados
       pilha.push(n)
       #print("n colocado na pilha = ", pilha)
       nosMarcados.append(adjacentesDeN[0]) # "m é marcado = ", nosMarcados)
       n = adjacentesDeN[0] #"Troca o valor de n para m (n <- m(atribuição)) = ", n, "\n")
       adjacentesDeN = self.adjacenciasVertice(n, nosMarcados)

   '''
```

```
  # mostrando Percurso em profundidade
  print("O percurso em profundidade foi:", end = " ")
  for i in nosMarcados:
    print(i, end = " ")
  print("\n\n")
  '''

  # Tendo o percurso em profundidade, poderemos verificar a conexidade
  print(f"Qtd de vértices = {self.n}. Qtd_vistiados = {quantidade_visitados}")
  # Verifica se o grafo é conexo ou não
  if(self.n == quantidade_visitados):
    print("O grafo é conexo!\n\n")
  else:
    print("O grafo é não conexo!\n\n")

# h) Menor caminho entre dois vértices utilizando o algoritmo de Dijkstra
def menor_caminho(self, v1, v2):
  # Recebe os vértices em nome e descobre se índice
  indice_v1 = self.getPosicaoNome(v1)
  indice_v2 = self.getPosicaoNome(v2)

  #print(f"indice 1 = {indice_v1}, indice 2 = {indice_v2}")
  if indice_v1 == -1 or indice_v2 == -1:
    print("Alguma vértice digitado não existe!")
  else:
    #print("Continua a lógica")
    # Criando o array d(distancias)
    d = [infinito] * self.n
    #print(d)

    # Atribuindo 0 no array d(distancias) na posição do indice de v1
    d[indice_v1] = 0

    # Criando array A(abertos), F(fechados), S(sucessores), k e rot(rotas)
```

```
  A = [i for i in range(0, self.n)]
  #print("A = ", A)
  F = [] # começa vazio
  S = [indice_v1] #começa S com o vértice inicial
  k = 0
  rot = [-1 for i in range(0, self.n)] #inicializando rot

  while len(A) != 0: #mudar aqui
    k += 1
    distanciasA = []
    indices = []

    for i in A:
      distanciasA.append(d[i])
      indices.append(i)

    r = indices[distanciasA.index(min(distanciasA))]
    F.append(r)
    A.remove(r)

    S = list(set(A) & set(self.adjacenciasV(r)))

    for i in S:
      p = min(d[i], d[r] + self.adj[r][i])
      if p < d[i]:
        d[i] = p
        rot[i] = r

#print("Terminou o algoritmo!")
print(f"\nA seguir, temos a rota com o menor tempo entre {v1} e {v2}:\n")
#Vetor = [i for i in range(0, self.n)]
#print("Vetor", Vetor)
#print("ROT:", rot)
#print("Distancias:", d)
```

```
    # Vamos pegar a rota que devemos fazer
    rota = []
    indice_aux = indice_v2
    while indice_aux != indice_v1:
      vertice_visitar = rot[indice_aux]
      rota.append(vertice_visitar)
      indice_aux = vertice_visitar

    # Vamos inverter a ordem para simular o caminho real a partir de v1
    rota_invertida = rota[::-1]
    rota_invertida.append(indice_v2) # Adicionamos o v2 ao array de rotas tmb, pois é o destino e não entrou no laço
    #print("Essa é a rota invertida: ", rota_invertida)

    # Agora, iremos exibir quanto é a distância entre todos os vértice da partida ao destino
    for i in range(len(rota_invertida)-1):
      #print("i = ",i, "i + 1 = ", i+1)
      print(f"{self.nomes_vertices[rota_invertida[i]]} ---> {self.nomes_vertices[rota_invertida[i+1]]} = {self.adj[rota_invertida[i]][rota_invertida[i+1]]} minutos\n" )

    # Por último, iremos printar o tempo total do percurso de acordo com o que foi calculado no algoritmo
    print(f"\nTEMPO TOTAL DO PERCURSO = { d[indice_v2]} minutos")


  # Método para exibir as adjacências apenas de um vértice
  def mostrarAdjacenciaVertice(self, vertice):
    #Vamos pesquisar se o vértice existe na nossa lista de estacoes
    if vertice in self.nomes_vertices:
      # Agora, iremos pegar a posição do vértice na matriz e percorrer apenas na linha da matriz referente a ele
      posicao_vertice = self.getPosicaoNome(vertice)
      print(f"\nADJACÊNCIAS NO VÉRTICE '{vertice}': ")
```

```
    for i in range(self.n):
      if self.adj[posicao_vertice][i] != infinito:
        print(f" Adj[{self.nomes_vertices[posicao_vertice]}, {self.nomes_vertices[i]}] =
{self.adj[posicao_vertice][i]}")
    else:
      print("VÉRTICE NÃO EXISTENTE NO GRAFO")


  # Exibe de forma reduzida devido a grande quantidade de vértices
  def show(self):
    print(f"\n n: {self.n:2d} ", end="")
    print(f"m: {self.m:2d}\n")
    for i in range(self.n):
      print(f"ESTAMOS NO VÉRTICE '{self.nomes_vertices[i]}': ")
      for w in range(self.n):
        if self.adj[i][w] != infinito:
          print(f" Adj[{self.nomes_vertices[i]}, {self.nomes_vertices[w]}] = {self.adj[i][w]}")
        '''
        Estamos exibindo apenas as arestas
        else:
          print(f" Adj[{self.nomes_vertices[i]}, {self.nomes_vertices[w]}] = {infinito}")
        '''
      print("\n")
    print("fim da impressao do grafo.\n\n")
```

PILHA.PY

```
"""
32092921 | MATHEUS HENRIQUE DA SILVA APOSTULO
32095971 | VALDIR LOPES JUNIOR
"""

# -*- coding: utf-8 -*-
"""
Created on Tue Feb 14 18:53:01 2023

@author: icalc
"""
class Pilha:
    TAM_DEFAULT = 1100
    def __init__(self, tamanho=TAM_DEFAULT):
        self.pilha = list(range(tamanho))
        self.topoPilha = -1

         #Verifica se a pilha
         #está vazia
    def isEmpty(self):
                return self.topoPilha == -1

    # Verifica se a pilha está
    # cheia
    def isFull(self):
        return self.topoPilha == (len(self.pilha) - 1)

         # insere um elemento e
         # no topo da pilha
    def push(self, e):
        if not self.isFull():
```

```
        self.topoPilha+=1
        self.pilha[self.topoPilha] = e
    else:
        print("overflow - Estouro de Pilha")


# remove um elemento
    # do topo da pilha
def pop(self):
    if not self.isEmpty():
        e = self.pilha[self.topoPilha]
        self.topoPilha-=1
        return e
    else:
        print("Underflow - Esvaziamento de Pilha")
        return -1;


    # Retorna o elemento que está
    # no topo da pilha
def topo(self):
    if not self.isEmpty():
        return self.pilha[self.topoPilha]
    else:
        print("Underlow - Esvaziamento de Pilha")
        return -1


    # obtém o total de elementos
    # armazenados na Pilha
def totalElementos(self):
    return self.topoPilha+1
```