CENTRO DE ESTUDOS E SISTEMAS AVANÇADOS DO RECIFE

ARTHUR SILVA CAPISTRANO

ÉRICO CHEN

GABRIEL RAMOS CORRÊA TABOSA

GHEYSON YARCONI CHAVES MELO

JOÃO ANTÔNIO MEDEIROS LUZ PARENTE

RELATÓRIO DO PROJETO DE TEORIA DOS GRAFOS

Recife

2025

ARTHUR SILVA CAPISTRANO
ÉRICO CHEN
GABRIEL RAMOS CORRÊA TABOSA
GHEYSON YARCONI CHAVES MELO
JOÃO ANTÔNIO MEDEIROS LUZ PARENTE

RELATÓRIO DO PROJETO DE TEORIA DOS GRAFOS

Relatório técnico-científico do projeto como requisito de avaliação da disciplina de Teoria dos Grafos, no CESAR School, da turma 2025.2 de Ciência da Computação.
Profª. Mr. Laura Pacífico

Recife
2025

RESUMO

O presente relatório evidencia a explicação de termos técnicos a respeito de Teoria dos Grafos em Geral, bem como aos utilizados no mesmo, explica minuciosamente o passo a passo da construção do projeto aqui relatado, além de discutir os resultados obtidos.

SUMÁRIO

1. INTRODUÇÃO 5
1.1 Principais Propriedades e Tipos de Grafos 5
1.2 Medidas e Algoritmos Essenciais 7
1.3 Algoritmos de Exploração e Otimização 7
1.4 Grafos Espaciais e Análise de Dados 8
1.5 Ferramentas Conceituais 8
2. OBJETIVOS 9
3. PROCEDIMENTO EXPERIMENTAL 10
3.1. ARQUIVOS/DOCUMENTOS/FERRAMENTAS IMPORTANTES 10
3.2. METODOLOGIA 10
4. RESULTADOS E DISCUSSÃO 21
4.1. PARTE 1: GRAFO DE BAIRROS DO RECIFE 21
4.2. PARTE 2: GRAFO MÚSICAS 21
5. CONCLUSÃO 23
5.1 Adequação e Precisão da Modelagem Espacial (Recife) 23
5.2 Desempenho Algorítmico e Otimização (Grafo Músicas) 23
5.3 Limitações e Recomendações Futuras 24

1. INTRODUÇÃO

Esta introdução visa apresentar, de maneira simples e didática, alguns conceitos de Teoria dos Grafos bem como as ferramentas de análise de redes que foram aplicadas neste projeto para modelar e analisar a infraestrutura urbana do Recife, assim como as músicas mais ouvidas no ano de 2025.

Um **grafo** é uma estrutura matemática fundamental, definida por um conjunto de elementos que representam objetos e um conjunto que representa as relações entre esses objetos. Formalmente, um grafo **G** é geralmente expresso como um par (**V, E**):

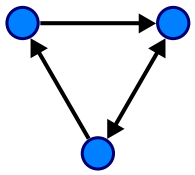
- **V** é o conjunto de **vértices** (ou **nós**) — que são as entidades ou pontos de interesse em uma rede (ex: uma cidade em um mapa de rotas ou um usuário em uma rede social).
- **E** é o conjunto de arestas (ou ligações) — que são as conexões ou relações que ligam um par de nós (exemplo: uma rodovia entre duas cidades ou uma amizade mútua).

Essa simplicidade estrutural torna os grafos uma ferramenta natural e poderosa para a modelagem de sistemas onde a interconexão e a dependência mútua são elementos-chave. Ao mapear objetos como vértices e suas interações ou conexões como arestas, qualquer sistema de relações pode ser abstrato e analisado sob uma perspectiva unificada, desde redes sociais e sistemas de computação até fenômenos biológicos e, como veremos adiante, estruturas espaciais complexas. A relevância do grafo reside na sua capacidade de modelar relações, fornecendo o arcabouço para entender a estrutura e o comportamento de sistemas interconectados.

1.1 Principais Propriedades e Tipos de Grafos

A utilidade da teoria dos grafos na modelagem de sistemas reais reside na capacidade de refinar a abstração por meio de propriedades e tipos específicos. As principais distinções que conferem flexibilidade à modelagem são a direcionalidade, o peso e a multiplicidade das arestas. Um grafo é direcionado (ou dígrafo – figura 1) quando as arestas possuem um sentido definido, modelando relações assimétricas (como um fluxo unidirecional ou uma hierarquia).

Figura 1: Exemplo de dígrafo



Fonte: https://pt.wikipedia.org/wiki/Grafo_orientado

Por outro lado, um grafo é não-direcionado quando as arestas representam relações simétricas, como a simples existência de uma conexão. A escolha entre eles é crucial para garantir que o modelo reflita a assimetria (direcionado) ou a simetria (não-direcionado) da

relação real. Adicionalmente, um multigrafo permite a existência de múltiplas arestas (paralelas) entre o mesmo par de vértices, o que é essencial para modelar sistemas onde diferentes tipos de relações coexistem (e.g., uma conexão rodoviária e uma ferroviária entre duas cidades). A implicação teórica do multigrafo é que a estrutura de rede é mais complexa, exigindo algoritmos que considerem todas as conexões simultaneamente.

Além disso, as arestas podem ser ponderadas, o que significa que um valor numérico (o peso) é associado a cada conexão. O peso geralmente representa uma métrica da ligação, como o custo, a qualidade, a distância ou a força da relação. A diferença entre um grafo ponderado e um grafo simples (não ponderado) é que o primeiro incorpora esse atributo quantitativo, o que é de importância fundamental para tarefas de otimização, como o cálculo de rotas eficientes em navegação ou logística.

Assim, a escolha entre grafos direcionados, não-direcionados, multigrafos e ponderados permite que o modelo matemático reflita com precisão a natureza e as restrições específicas do sistema físico ou conceitual que está sendo analisado.

Tabela 1 – Explicação dos diferentes tipos de grafos.

Tipo de Grafo	Estrutura Modelada	Vértices (Objetos)	Arestas (Relações)	Importância da Distinção
Não-Direcionado Simples	Rede de Amizade em Redes Sociais	Usuários	Amizades mútuas (relação simétrica)	Foca apenas na existência da conexão e na simetria da relação.
Direcionado (Dígrafo)	Estrutura Organizacional	Cargos (CEO, Gerente, Funcionário)	Relação hierárquica (aponta do superior para o subordinado)	Modela relações assimétricas (a aresta define a cadeia de comando).
Multigrafo	Conexões de Transporte Multimodal	Cidades/Hubs de Logística	Múltiplas arestas paralelas: Rodovia, Ferrovia, Hidrovia (entre as mesmas cidades)	Permite modelar a coexistência de diferentes tipos de conexões entre os mesmos pontos.
Ponderado Não-Direcionado	Rede Ferroviária	Estações de Trem	Trechos de trilho	O peso incorpora um atributo à ligação simétrica.
Ponderado Direcionado	Cadeia de Suprimentos	Fábricas, Armazéns, Lojas	Fluxo de produtos (sentido definido)	O peso permite otimizar rotas com base em custo/tempo em relações assimétricas.

Fonte: Dos autores (2025)

1.2 Medidas e Algoritmos Essenciais

Para transformar a estrutura abstrata dos grafos em análise quantitativa de sistemas interconectados, o foco se volta para medidas e algoritmos específicos. A conectividade da rede refere-se ao quão bem os nós estão ligados; a identificação de componentes conectados (subgrafos onde todos os nós estão ligados entre si) é vital para entender a integridade da rede, pois a presença de muitos componentes pequenos indica fragmentação da rede.

A distância entre dois vértices pode ser avaliada de duas formas principais: a distância topológica (o número mínimo de arestas entre eles, relevante para medir a "proximidade relacional" pura) ou a distância ponderada (a soma dos pesos das arestas, refletindo o custo ou o tempo de deslocamento real). A distinção importa porque a distância topológica é puramente estrutural, enquanto a ponderada incorpora atributos de custo. Um caminho mínimo é o caminho entre dois nós que minimiza a soma dos pesos das arestas (distância ponderada), ao contrário de um caminho simples, que é qualquer caminho que não repete nós, ou um caminho geográfico, que se refere a uma rota no espaço físico, geralmente coincidindo com o caminho mínimo ponderado em grafos espaciais.

1.3 Algoritmos de Exploração e Otimização

O estudo da rede depende fundamentalmente de como ela é percorrida. Os algoritmos de Busca em Largura (BFS) e Busca em Profundidade (DFS) são usados para exploração sistemática: o BFS expande a busca camada por camada (usando o conceito de Fila), sendo ideal para encontrar caminhos com o menor número de arestas (distância topológica). Por outro lado, o DFS vai o mais fundo possível em um caminho antes de retroceder (usando o conceito de Pilha), sendo útil para detectar ciclos e verificar a conectividade completa.

Para encontrar o caminho mínimo ponderado, são utilizados algoritmos de otimização:

- **O Algoritmo de Dijkstra:** tem como objetivo encontrar o caminho de menor custo em grafos ponderados, sendo aplicado em cenários onde todos os pesos de arestas são não-negativos (exemplo: GPS, redes de telecomunicações).
- **O Algoritmo de Bellman-Ford:** cumpre o mesmo objetivo, mas é essencial por sua capacidade de lidar com grafos que contenham pesos de arestas negativos. Essa robustez é crítica, pois permite ao algoritmo detectar a existência de ciclos de peso negativo, o que tornaria o caminho mínimo indefinido.

Além desses algoritmos, medidas de centralidade quantificam a importância de um nó na rede. A centralidade de grau (*degree centrality*) é simplesmente o número de arestas ligadas a um nó (o que indica o quão ativo ele é), enquanto a centralidade de intermediação (*betweenness centrality*) mede a frequência com que um nó aparece nos caminhos mínimos entre outros pares de nós (o que indica sua função de gargalo ou *hub* de fluxo).

1.4 Grafos Espaciais e Análise de Dados

Georreferencia é o processo de atribuir coordenadas geográficas a um local ou objeto no globo terrestre. Em grafos, o conceito de georreferencia entra em grafos espaciais, que são aqueles em que a geometria importa. Nesses grafos, as coordenadas espaciais dos vértices e o comprimento físico das arestas influenciam diretamente as análises. A implicação teórica é que, ao contrário de um grafo puramente abstrato, a análise métrica deve considerar as distâncias geográficas reais. Isso se contrasta com a topologia da rede, que se refere puramente a quem toca quem (a conectividade), desconsiderando o espaço. A geometria, por outro lado, refere-se às distâncias reais e posições no espaço.

Relacionado a isso, a planaridade de um grafo é o conceito de que ele pode ser desenhado em um plano sem que suas arestas se cruzem. A relevância teórica dos grafos planares é alta para o projeto de mapas, circuitos impressos e infraestrutura de rede, onde cruzamentos físicos de conexões são indesejáveis ou caros.

A validade da análise de grafos, especialmente em estudos interdisciplinares que envolvem dados geográficos, depende muito da qualidade dos dados. A validação e curadoria de dados é o processo de garantir que os dados sejam precisos, consistentes e bem estruturados, sendo necessária para eliminar erros e harmonizar formatos de dados de múltiplas fontes. Da mesma forma, registrar a proveniência de dados e versionamento a origem dos dados e as modificações aplicadas ao longo do tempo é crítico para a reprodutibilidade da pesquisa e para a rastreabilidade de resultados. É importante também reconhecer as limitações e incertezas na modelagem de redes, que podem surgir de dados faltantes, erros de representação (como simplificações da realidade) ou do nível de agregação dos dados.

1.5 Ferramentas Conceituais

- **Geopandas:** Estende a biblioteca Pandas, combinando as estruturas de dados desta com operações geoespaciais para facilitar o trabalho com **geo-dataframes** (tabelas de dados com uma coluna de geometria).
- **Pyvis:** É uma biblioteca Python que facilita a visualização interativa e dinâmica de redes complexas, permitindo a exploração visual da estrutura do grafo em um navegador web.

2. OBJETIVOS

Este artigo tem como foco o desafio de modelar a rede de infraestrutura urbana da cidade do Recife. A abordagem adotada combina a aplicação da Teoria dos Grafos com a análise comparativa de algoritmos fundamentais. A fim de detalhar este escopo, podemos apresentar os seguintes objetivos gerais e específicos.

2.1. OBJETIVOS GERAIS

- **Modelagem de Rede Urbana:** Desenvolver uma solução computacional completa da rede de bairros da cidade do Recife, utilizando como conceito base grafos ponderados, para analisar sua estrutura topológica e a qualidade de suas conexões.
- **Análise de Algoritmos:** Implementar e conduzir uma análise comparativa de desempenho dos algoritmos fundamentais (BFS, DFS, Dijkstra e Bellman-Ford). Num grafo das músicas mais ouvidas mundialmente no ano de 2025.

2.2. OBJETIVOS ESPECÍFICOS

- **Modelagem de Grafos.**
  - **Processamento de Dados**: Realizar ETL (Extract, Transform e Load) dos dados ofertados pela prefeitura do Recife.
  - **Construção do Grafo:** Modelar a rede de adjacências dos bairros e das músicas, definindo arestas com base em logradouros coletados e tratados – no caso dos bairros e validando manualmente as conexões.
  - **Definição de Pesos:** Identificar métrica ideal de custo para cada aresta baseada em atributos da infraestrutura da via no grafo dos bairros e de acordo com a similaridade das músicas, no grafo do Spotify.
  - **Análise de Centralidade:** Identificar bairros de destaque através de rankings de grau e de densidade da rede.
  - **Visualização de Dados:** Gerar visualizações para comunicar os insights.
- **Implementação e Comparação de Algoritmos**
  - **Implementação:** Desenvolver implementações próprias para algoritmos de BFS, DFS, Dijkstra e Bellman-Ford.
  - **Aplicação:** Utilizar a implementação própria do Dijkstra para calcular os caminhos de menor custo entre diferentes endereços no grafo.
  - **Análise Comparativa:** Medir e comparar o tempo de execução de cada algoritmo para diferentes cenários no grafo dos bairros.

3. PROCEDIMENTO EXPERIMENTAL

3.1. ARQUIVOS/DOCUMENTOS/FERRAMENTAS IMPORTANTES

- Manual do projeto (classroom)
- CSV com todos os bairros de Recife – dado pela professora.
- Bairros geojson – dataset dos bairros do Recife (Prefeitura).
- Logradouros geojason – dataset dos trechos de logradouros (Prefeitura).
- Face da quadra csv – dataset dos logradouros por face de quadra (Prefeitura).
- Google collab – para ETL da parte 1 do projeto.
- Google drive – para armazenar arquivos temporários.
- Google sheets – para filtragem manual dos datasets.
- Visual studio code – para desenvolvimento.
- Dataset do spotify – para parte 2 do projeto.

3.2. METODOLOGIA

Para realização desse projeto foi feito inicialmente a criação da estrutura do projeto em um repositório github para cumprir com os requisitos impostos no manual do projeto, aqui foi criado um repositório em uma organização que já tem todos os integrantes da equipe como donos de maneira a facilitar a ambientação dos membros. A arquitetura é padrão e está detalhada no manual supracitado, após isso iniciou-se o processo de derretimento do arquivo *bairros_recife.csv* que consiste num arquivo com todos os bairros de recife dado no lançamento do projeto. O processo de derretimento desse csv foi feito em código python e está presente no repositório, mais especificamente no arquivo *io.py*, nele há duas funções principais:

1. def detect_melt_columns(csv_path):

Essa função basicamente recebe o caminho do csv a ser derretido e identifica colunas desse csv que possuem o formato ‘α.α’ sendo α um número inteiro positivo qualquer e retorna o nome dessas colunas em uma lista de strings.

2. def melt_bairros_csv(csv_path: str) -> pd.DataFrame:

Já essa função, começa lendo o csv, recebe os nomes das colunas usando a função anterior e, em seguida, aplica o processo de melt nativo do pandas, criando uma tabela com a informação do nome do bairro e da microrregião em que ele está inserido. Após isso faz uma limpeza nos dados através da remoção de duplicatas, valores nulos, vazios ou similares, normaliza acentuação e ordena alfabeticamente pela microrregião. Por último, salva o resultado em *data/bairros_unique.csv* que é um dos entregáveis do projeto.

Tendo codificadas ambas as funções acima, através de um terminal bash comum é feito o seguinte comando:

```
python -c "from src.graphs.io import melt_bairros_csv; melt_bairros_csv({caminho_para_o_arquivo})"
```

E com isso o arquivo *bairros_unique.csv* foi gerado e inserido no repositório. Com o fim dessa etapa, iniciou-se o processo criativo para o segundo passo que consistiu na produção do arquivo *adjacências_bairros.csv* onde a equipe teve total liberdade criativa para construir um grafo onde os bairros de recife são os nós e as arestas são as vias entre esses bairros, devendo existir conexões apenas entre bairros vizinhos. Também era necessário que o csv tivesse um formato padrão descrito no manual onde basicamente existiam as colunas de bairro de origem, destino, a via de ligação entre os bairros e os pesos associados a essas vias.

O processo produtivo desse csv foi bem extenso, inicialmente buscou-se dados que foram encontrados em um portal de dados oficial da Prefeitura do Recife [1], lá existiam alguns conjuntos de dados referentes aos mais variados assuntos, mas para esse projeto foram utilizados apenas 3:

1. Bairros do Recife

   Esse conjunto de dados está no formato geojson e basicamente identifica os bairros do Recife, com localização geográfica e a microrregião onde o bairro está inserido. Perceba que não foi utilizado o arquivo *bairros_unique.csv* nessa parte pois foi necessário usar geolocalização e ele não possuía esses dados.

2. Trechos de Logradouros

   Esse conjunto de dados identifica os trechos de logradouros (vias) que compõem os respectivos logradouros do Recife e está no formato geojson, possuindo informações geográficas valiosas para a realização dessa etapa.

3. Logradouros por face de quadra

   Já esse conjunto de dados está no formato csv e possui detalhamento minucioso sobre cada logradouro de Recife como por exemplo, descrição sobre o tipo de coleta de lixo, limpeza urbana, rede de água, rede de esgoto, iluminação, emplacamento, dentre outros. Essas informações nos ajudaram a definir os pesos associados a cada via, de forma que a qualidade da via foi o critério adotado.

Com esses conjuntos de dados (datasets) selecionados, o processo de tratamento deles se deu através do Google Colab, que pode ser acessado ou pelo Read.me do repositório ou direto pelo Google Drive. Então, após a realização do upload desses arquivos na pasta do drive do projeto, a importação deles foi feita no ambiente do Colab. Como se trata de arquivos geojson, a biblioteca geopandas foi utilizada para leitura dos dados. Também por isso, foi feita uma "reprojeção" dos dados de graus para metros. A partir daí viu-se que cada rua presente no dataset eram muito finas, naturalmente, então elas foram "engrossadas" em 5 metros, para melhor assertividade, e depois a junção espacial dos bairros com as vias foi realizada.

Em seguida foi feito um agrupamento por rua, caso ela tocasse 2 bairros (ou mais) significa que ela é uma fronteira, aqui já podemos chamar atenção para a primeira filtragem feita: no dataset foi incluído os Rios como vias, então todas as vias que começam com a palavra Rio foram removidas, para além disso, quando uma rua toca 3 bairros então ela é fronteira em comum dos 3, exemplo, se uma rua toca os bairros A, B e C, então existem 3 pares de bairro adjacentes (A,B), (A,C) e (B,C). Também deve ser mencionado um "erro" de execução do projeto: no processo de filtragem, percebeu-se que o limite do município foi considerado uma rua e acabou atrapalhando a equipe numa etapa posterior, então foi visto que deveria ter sido filtrada já aqui como foi a questão dos rios.

O resultado desse agrupamento foi um dataframe pandas com as colunas *bairro_origem*, *bairro_destino* e *logradouro* que são autoexplicativos, aqui já foi feita mais uma filtragem: como um logradouro pode tocar 2 bairros em várias interseções, removemos todas as linhas duplicadas, tendo agora ligações únicas apenas. Perceba que nesse dataset ainda existem várias ligações entre os mesmos bairros, exemplo: entre o bairro A e o bairro B existem 50 vias, então nesse dataset estão as 50 vias. Nós salvamos esse dataset em um csv e mais de 3.700 linhas existem nele, sendo necessário ainda um grande refinamento.

Assim sendo, definimos uma ordem de prioridade para que a conexão entre os bairros fosse apenas por uma única via:

1. Avenida
2. Estrada
3. Ponte
4. Via
5. Rodovia
6. Rua
7. Travessa
8. Viela
9. Beco

Em seguida, uma coluna foi adicionada no dataset justamente a respeito do tipo da via, de tal forma que pudemos ordenar os pares de bairros por essa prioridade e realizar mais uma filtragem, mantendo apenas a primeira aparição do par. Dessa feita, reduziu-se o dataset para 782 linhas, cerca de 3 mil a menos. Mas, ainda havia um problema nítido: algumas dessas ruas por serem muito grandes conectavam bairros que não eram vizinhos. Então, manualmente, abrimos esse dataset em uma planilha e conferimos, através do mapa oficial de cada bairro, se os pares existentes de fato eram pares que representavam bairros adjacentes (vizinhos) foram longas horas de conferência e o resultado foi uma redução de mais de 500 linhas, resultando em 244 pares. Essa etapa foi necessária apenas pelo fato de ser um requisito do projeto que as ligações fossem apenas entre os bairros adjacentes

Nesse momento, salvamos o novo dataset filtrado no drive do projeto e fizemos a importação dele no ambiente do Colab. A partir disso, o processo de definição dos pesos que essas vias teriam foi iniciado. Primeiro, foi feita uma análise aprofundada do dataset dos logradouros por face da quadra, onde removemos cerca de 30 colunas que não consideramos relevantes para essa etapa. Depois disso, unimos os dois datasets (o que tinha sido resultado da filtragem manual e o com o detalhamento dos logradouros) e exportamos isso para um csv para análise. Durante essa análise percebeu-se que os nomes dos logradouros de um csv não necessariamente eram iguais aos do outro, fazendo com que o dataset da união tivesse algumas colunas nulas, então mais uma vez manualmente buscou-se no dataset com o detalhamento dos logradouros o nome dos logradouros que tinham ficado com as colunas nulas. Diferenças como "... de Melo" e "...de Mello" foram encontradas e com isso preencheu-se manualmente as colunas dos logradouros com caso similar ao exemplo. Uma observação importante é que todas as pontes ficaram sem atributos pois elas não existiam no dataset do detalhamento, então o peso das pontes foi padrão. Como o gerado automaticamente não foi assertivo suficiente, foi feito o upload do arquivo modificado no drive e a importação no ambiente do Colab.

Enfim, havia um dataset com os pares de bairros adjacentes, a via principal que une esses bairros e várias informações sobre essa via. Nesse momento, definiu-se a maneira de calcular o peso associado a cada via:

$$peso = Tipo\ da\ via + Penalidades$$

E a tabela de valores, tendo como critério a qualidade do logradouro:

Tipo da via:

- Avenida: 1.0
- Viaduto: 1.1
- Ponte: 1.2
- Estrada: 1.4
- Rua: 2.0

Penalidades/Benefícios pelo tipo de iluminação:

- Fluorescente: -0.05
- Comum: 0.0
- Vapor Mercúrio: 0.0
- Sem Iluminação: 0.30

Penalidades/Benefícios pelo tipo de pavimentação:

- Concreto: -0.10
- Asfalto: -0.05
- Paralelo: 0.10
- Poliedro: 0.20
- Outros: 0.10
- Escadaria: 0.50
- Sem pavimentação: 0.70

Penalidades/Benefícios pelo tipo de coleta de lixo:

- Convencional diária com seletiva: -0.05
- Convencional diária sem seletiva: 0.00
- Convencional alternada com seletiva: 0.00
- Convencional alternada sem seletiva: 0.05
- Manual diária: 0.00
- Manual alternada: 0.05
- Sem coleta de lixo: 0.50

Penalidades/Benefícios pelo tipo de limpeza da via:

- Regular diária: -0.05
- Regular alternada: 0.00
- Programada semanal: 0.05
- Programada mensal: 0.15
- Sem limpeza pública: 0.50

Penalidades/Benefícios pela presença de arborização:

- Com arborização: -0.05
- Sem arborização: 0.05

Penalidades/Benefícios pela presença de rede de água:

- Com rede de água: 0.0
- Sem rede de água: 0.20

Penalidades/Benefícios pela presença de rede de esgoto:

- Com rede de esgoto: 0.0
- Sem rede de esgoto: 0.30

Penalidades/Benefícios pela presença de emplacamento:

- Com emplacamento: -0.05
- Sem emplacamento: 0.05

É válido mencionar que os critérios foram definidos pelos integrantes do grupo e respeitou-se o critério de não haver pesos negativos através de duas particularidades: havia um peso mínimo de 0.5 por via e, além disso, se uma rua contasse com todos os melhores benefícios, ainda assim não era suficiente para que ela negativasse seu peso. A escolha de benefícios diminuírem o peso da via e penalidades aumentarem significa dizer que **quanto maior o peso da via, menor é sua qualidade.**

Em sucessão, o algoritmo do cálculo do peso foi realizado (ainda dentro do Colab). Basicamente, criou-se uma função que pega o tipo de cada atributo e cria uma linha com a penalidade associada. Exemplo, criou-se uma coluna *pen_esgoto*, então se uma rua tem rede de esgoto, essa coluna foi preenchida com 0.0, mas se não tem, foi preenchida com 0.30. Isso para cada penalidade que descrevemos acima, ao fim disso, uma coluna com a soma das penalidades foi criada para ter o valor da soma de todas as penalidades por via.

O peso foi calculado de maneira similar, criou-se uma coluna no dataset e preencheu-a utilizando a fórmula supracitada, além disso, aproveitou-se das descrições das penalidades para preencher a coluna observações do dataset de saída. Tendo todas essas informações, esse dataset foi exportado para o drive em um arquivo csv que foi adicionado no repositório do projeto e a partir dele que os entregáveis relativos a parte 1 foram realizados.

A essa altura, decidimos criar uma API de maneira que fosse o mais automatizado possível todo o processo de geração dos entregáveis, mas para além disso, existia o interesse de desenvolver um pequeno frontend e ter uma API ajudaria muito no desenvolvimento desse frontend. Então nós desenvolvemos usando os conceitos de FastAPI os arquivos que se dividem basicamente em: api.py (responsável pela implementação dos endpoints), solve.py (responsável pelas funções que os endpoints utilizam), algorithms.py (onde os algoritmos estão implementados), graph.py e music_graph.py (cada um é uma classe que representa, respectivamente, o grafo da parte 1 e o da parte 2), io.py, part2_build.py e part2_io.py

(responsáveis pelo tratamento e filtragem dos dados e geração dos *csvs* que são a fonte de dados que usa-se para construção dos grafos.

Com tudo implementado e rodando o projeto, bastou acessar no navegador localhost:3000/docs que se tem acesso a um *swagger* onde se pôde testar cada endpoint. Nesse ponto se evoluiu muito o projeto porque inicialmente havíamos feito uma classe Graph que era geral tanto para parte 1 do projeto quanto para a parte 2, mas percebeu-se que, para além de fazerem coisas bem distintas, o nível de complexidade para reduzir o acoplamento era tão alto que não valia a pena, então dividimos nas duas classes citada previamente e os endpoints sim se tornaram passíveis de reuso através de uma flag na query que se chama graph com a qual é possível fazer requisições aos endpoints tanto para o grafo dos bairros (parte 1) quanto para o grafo das músicas.

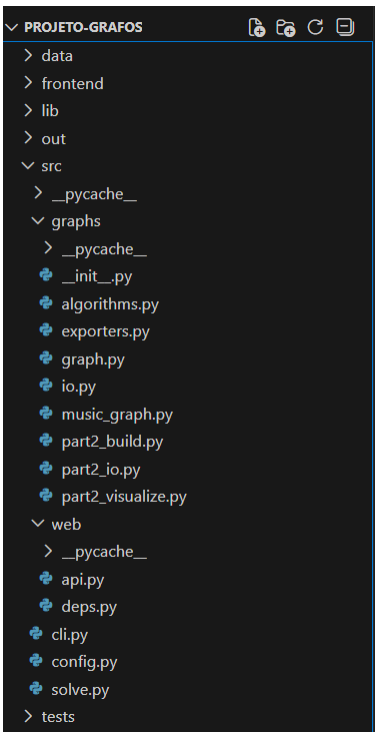
Dentre os endpoints existem endpoints para listagem de nós, de aresta, endpoints para cada um dos algoritmos, os de gerar os entregáveis, dentre outros. Mencionando, especialmente aqui o */generate/all* e o */bench* que, respectivamente, geram todos os entregáveis da parte 1 e da parte 2 do projeto, logo bastou apenas fazer a requisição (via swagger, postman ou similar) para esses endpoints passando a flag graph com o grafo desejado e todos os entregáveis estavam prontos. Os detalhes de implementação de cada um dos métodos necessários para tamanha feita podem ser vistos no github do projeto.

Agora que foi mencionado, a parte 1 desse projeto se dá por finalizada apenas na criação do frontend do projeto que devemos destacar mais abaixo. É relevante, nesse momento, explicar o passo a passo da construção do csv e dos entregáveis referentes à parte 2 do projeto. Nessa parte deveríamos buscar um dataset, tratá-lo, filtrá-lo e rodar alguns algoritmos para comparar métricas de tempo de execução. Escolhemos desde o princípio trabalhar com algo referente ao mundo da música. Então, após uma busca extensa na plataforma Kaggle, encontramos um dataset com dados do Spotify das músicas mais tocadas mundialmente no ano de 2025. Com o dataset em mãos, o processo analítico desses dados determinou que os nós seriam as músicas e a existência das arestas entre os nós se daria pela pergunta “os artistas dessas músicas compartilham o mesmo gênero?” caso afirmativo, a aresta existe, se não essas músicas não se conectariam.

Antes de partir para o cálculo dos pesos, tentamos gerar a visualização desse grafo e percebeu-se que, pela quantidade de dados, e a regra de existência da aresta ser simples de

ser satisfeita, uma quantidade descomunal de arestas eram geradas (cerca de 43 mil) e isso fazia com que o poder computacional das máquinas do grupo fosse limitado, dificultando essa visualização. Assim, tomou-se a decisão de filtrar as músicas de um mesmo artista (Ariana Grande), com a intenção de reduzir a quantidade de nós e a visualização ser possível de ser vista. Mas, mesmo com poucos nós, se comparado com o total (mais de 8 mil) a visualização é pouco representativa, como ilustrado na figura 2 a seguir e essa dificuldade é explicada pelo grafo ser completo, fazendo com que todos os nós se conectam com todos os outros, visto que se são músicas de um mesmo artista, obviamente todos os gêneros dos artistas seriam compartilhados.

Figura 2: Arquitetura do projeto.



Fonte: dos autores (2025)

Daí, mesmo com essa limitação, seguimos mais uma vez para determinar a régua de peso das arestas onde éramos livres e para isso, utilizou-se um modelo baseado em similaridade, no qual o peso da aresta representa o “custo” ou “distância” entre duas músicas. Assim como no cálculo das vias da parte 1, definiu-se uma régua de critérios que funcionam como benefícios e penalidades: quanto maior o peso, **menor a similaridade** entre as faixas; quanto menor o peso, maior sua proximidade musical. O cálculo seguiu a seguinte lógica geral:

$$peso = 1 - similaridade\ total$$

A similaridade total é composta por diversas dimensões musicais e contextuais, ponderadas de acordo com a relevância de cada uma para diferenciar ou aproximar duas músicas. Tudo descrito na tabela 2 apresentado abaixo com os critérios empregados.

Tabela 2 – Critérios do cálculo de peso para grafo da parte 2 – Spotify.

Critério	**Interpretação**	**Peso**
Gêneros em comum (Jaccard)	Mede a sobreposição entre listas de gêneros dos artistas das músicas	4.0
Mesmo artista	1.0 se as faixas são do mesmo artista, 0 se não	3.0
Mesmo álbum	1.0 se pertencem ao mesmo álbum, 0 se não	2.0
Popularidade da faixa	Diferença de popularidade das músicas	1.0
Popularidade do artista	Diferença de popularidade dos artistas	0.5
Seguidores do artista	Diferença logarítmica de seguidores (evita extremos)	0.5
Duração da faixa	Diferença de duração normalizada	0.3
Recência (ano do álbum)	Diferença de anos de lançamento dos álbuns	0.5
Explícita ou não	1.0 se ambas são explícitas, 0.7 caso contrário	0.3

Fonte: Dos autores (2025)

A soma ponderada desses critérios gera um valor de similaridade entre 0 e 1. Em seguida, a conversão para peso é realizada subtraindo a similaridade do valor 1.

Os critérios foram definidos pelo grupo e refletem a seguinte lógica:

- **Critérios de grande impacto** (gêneros, artista e álbum) aproximam drasticamente duas músicas.
- **Critérios de impacto moderado**, como popularidade, ano e seguidores, refinam o valor final sem dominá-lo.
- **Critérios sutis**, como diferenças de duração ou marcação explícita, atuam como ajustes finos.

Além disso, garantindo um comportamento consistente do grafo:

- O peso **nunca se torna negativo**, a menos que o usuário ative explicitamente a funcionalidade de arestas negativas para experimentos.
- Quanto **menor** o peso, maior a afinidade musical entre as duas faixas.
- Quanto **maior** o peso, menor o grau de semelhança.

Essa abordagem permitiu capturar nuances musicais de forma equilibrada e forneceu uma régua de pesos coerente para a construção do grafo final. Com o cálculo do peso definido o processo de geração do csv foi parecido com o da parte 1, mas dessa vez, mais automatizado. Após o download do csv e upload na pasta /data do projeto três comandos foram feitos via terminal na raiz do projeto:

1. python -c "from src.graphs.part2_io import prepare_spotify; prepare_spotify('data/spotify.csv')"
2. python -c "from src.graphs.part2_build import build_edges_from_spotify; build_edges_from_spotify('data/spotify_filtered.csv','data/parte2_adjacencias.csv', verbose=True)"
3. python -m src.graphs.part2_visualize --csv data/parte2_adjacencias.csv --mode largest_component --max-nodes 90 --max-edges-per-node 1000000 --out out/parte2_interactive.html

Esse arquivo chamado no terceiro passo é o arquivo responsável pela geração da amostra do grafo da parte 2, com isso tivemos o resultado da imagem anterior além do csv pronto com as colunas adequadas que são basicamente: música A, música B, gêneros em comum, número de gêneros em comum, peso e similaridade.

Desta feita, é válido retornar à menção do endpoint /bench da api que ao ser requisitado faz uma sequência de 35 cálculos de distância, sendo 10 vezes os algoritmos BFS, DFS e Dijkstra a partir de fontes distintas e diferentes pares e 5 vezes bellman-ford, incluindo nessas 5 uma inserção de pesos negativos + ciclo negativo e uma de pesos negativos apenas. No final de sua execução, o arquivo com as métricas de desempenho

(parte2_report.json) é gerado automaticamente, ele foi usado para debater a respeito dos algoritmos na seção seguinte.

Finalmente, foi desenvolvido o frontend do projeto através de uma aplicação react, usando a tecnologia tailwind. A escolha de fazer uma aplicação “single-page” foi para simplificar ao máximo o desempenho visto que nenhum membro da equipe tem qualificação de frontend. Então existe um arquivo App.jsx que faz toda a lógica da troca de páginas e sua componentização, não nos preocupamos a respeito de boas práticas nessa parte... Nesse front foram desenvolvidas 3 páginas, uma home, uma sobre os bairros e uma sobre as músicas. Na página relativa aos bairros o grafo é carregado usando uma biblioteca javascript e nele é possível selecionarmos dois bairros distintos e vermos o caminho melhor (via Dijkstra) entre eles ser destacado. Já no grafo das músicas tivemos um insight interessante: como o grafo é completo, aplicar Dijkstra e BF é pouco representativo para esse grafo, mas percebemos que, aplicando BFS ou DFS poderíamos ter as n músicas mais próximas a música “origem” do algoritmo. Logo, fizemos uma funcionalidade de gerar lista de reprodução (playlist) onde o usuário escolhe uma música, uma quantidade n de músicas a ter na playlist e um algoritmo e a playlist é gerada.

4. RESULTADOS E DISCUSSÃO

4.1. PARTE 1: GRAFO DE BAIRROS DO RECIFE

O projeto teve início com a modelagem de um grafo baseado nos bairros da cidade do Recife–PE, no qual as arestas representam conexões reais entre regiões adjacentes. Os pesos foram definidos considerando critérios como proximidade geográfica, acessibilidade viária e fluidez prática do deslocamento entre os bairros. Essa escolha de modelagem mostrou-se extremamente fiel à dinâmica urbana recifense: em múltiplos testes, os caminhos mínimos retornados pelos algoritmos coincidiam justamente com as rotas cotidianas escolhidas por moradores locais. Esse resultado evidencia que a lógica de pesos adotada não apenas refletiu a estrutura espacial da cidade, mas também capturou decisões reais de mobilidade, reforçando a adequação e a precisão do modelo inicial.

4.2. PARTE 2: GRAFO MÚSICAS

Na segunda etapa, foram realizados dez experimentos independentes para cada algoritmo (BFS, DFS e Dijkstra), avaliando tempo de execução e completude da busca. Os resultados revelaram comportamentos distintos conforme a natureza de cada algoritmo. O BFS apresentou o melhor desempenho médio, com tempos entre 0,00384 s e 0,00473 s, alcançando consistentemente os 79 nós do grafo. A estrutura completamente conectada favoreceu seu padrão de exploração por camadas, reduzindo encadeamentos profundos e maximizando eficiência.

O DFS, embora também tenha percorrido todos os vértices, registrou tempos maiores, variando de 0,00620 s a 0,00818 s. Sua estratégia de aprofundamento prolonga o percurso em grafos densos, aumentando o custo computacional. Esse resultado confirma que, apesar de funcional, o DFS é menos adequado para estruturas altamente conectadas.

O Dijkstra exibiu tempos bastante baixos, entre 0,00022 s e 0,0040 s, exibindo caminhos mínimos com valores de distância diferentes (variando entre 0.0317 e 0.2305). Ainda assim, a maioria das buscas envolveu poucas atualizações de prioridade devido à densidade do grafo, resultando em execuções muito rápidas.

Já para o algoritmo Bellman-Ford foi executado em cinco experimentos, incluindo testes com e sem injeção de pesos negativos e com presença ou ausência de ciclos negativos. Os tempos variaram entre 0,014 s (controle sem pesos negativos) e aproximadamente 0,255 s (iterações com ajustes fracionários e ciclos negativos).

Nos cenários com injeção de pesos negativos, mas sem ciclo negativo, o algoritmo retornou distâncias reduzidas, todas positivas, por exemplo, 0.2341, evidenciando seu funcionamento correto mesmo diante de pesos negativos isolados. Por outro lado, nos cenários contendo ciclos negativos, o algoritmo detectou corretamente sua presença, retornando valores de distância altamente negativos, como -185.96, este último correspondente ao ciclo negativo mais extenso inserido. Sendo assim, notamos que Bellman-Ford permanece estável com pesos negativos, desde que não formem ciclo.

Ademais, reconhecendo a função de peso estabelecida para este cenário, sendo peso = 1 - similaridade, pudemos notar uma certa variedade de pesos, o que nos trouxe trajetórias mais expressivas para Dijkstra. Além disso, foi visto que a ausência de pesos negativos no modelo original garante compatibilidade com Dijkstra, mas limita o potencial de representar relações de “afinidade extrema”. A inserção experimental de pesos negativos no Bellman-Ford evidenciou que o modelo é sensível a essas perturbações: mesmo pequenos ajustes fracionários alteraram significativamente os menores caminhos.

A respeito da geração de playlists, escolheu-se uma amiga do grupo fã de Ariana Grande que apontou as playlists geradas a partir do algoritmo BFS como playlists que “fazem mais sentido” do que as geradas a partir do algoritmo DFS, esse resultado é esperado porque em teoria, as músicas em largura são mais próximas que em profundidade para grandes grafos como o representado.

Por fim, podemos notar que o principal limite permanece: o modelo não incorpora aspectos contextuais, como época de lançamento, colaboração entre artistas, ou semelhança harmônica, isto ocasionado pela forma de coleta de dados escolhida. A decisão por um modelo de coleta em tempo real de dados poderia enriquecer as nuances do grafo. Apesar disto, os resultados mostram que as métricas adotadas foram suficientes para capturar proximidade estilística e estrutural entre faixas, contemplando o escopo requisitado ao projeto.

5. CONCLUSÃO

A execução deste trabalho permitiu consolidar os resultados e reflexões finais acerca do uso da Teoria dos Grafos e dos algoritmos de busca aplicados à modelagem de sistemas interconectados, validando a abordagem em dois contextos distintos: infraestrutura urbana e relações de dados.

5.1 Adequação e Precisão da Modelagem Espacial (Recife)

O projeto demonstrou a adequação e precisão da modelagem em grafos para sistemas espaciais complexos, como a rede de bairros do Recife. A definição de pesos baseada em proximidade e fluidez viária foi crucial, pois os caminhos mínimos identificados pelos algoritmos coincidiram consistentemente com as rotas cotidianas escolhidas por moradores locais. Esse resultado é de grande relevância, pois confirma que o modelo matemático não apenas representou a estrutura espacial, mas também capturou a lógica real de mobilidade urbana, reforçando a fidelidade do grafo inicial e sua aplicabilidade em planejamento urbano.

5.2 Desempenho Algorítmico e Otimização (Grafo Músicas)

Os experimentos com o Grafo Músicas forneceram *insights* claros sobre o desempenho algorítmico em estruturas densas e altamente conectadas foram apresentados na tabela a seguir:

Tabela 3: Dados de desempenho dos algoritmos com amostragem no grafo das músicas – Spotify.

Algoritmo	Desempenho Médio (Tempo)	Estratégia e Uso Ideal
BFS (Busca em Largura)	0,00384 s a 0,00473 s	Mais eficiente em grafos densos; ideal para encontrar a proximidade topológica (menor número de arestas).
DFS (Busca em Profundidade)	0,00620 s a 0,00818 s	Mais lento em grafos densos devido ao aprofundamento; útil para detecção de ciclos ou exploração completa de componentes.
Dijkstra	0,00022 s a 0,0040 s	Extremamente rápido neste cenário devido à densidade do grafo e à variedade de pesos (0.0317 a 0.2305), que validou a busca de caminhos minimamente ponderados.
Bellman-Ford	0,014 s a 0,255$ s	Mais lento, mas essencial para lidar com pesos negativos.

Fonte: Dos autores (2025)

A. Robustez do Bellman-Ford

O teste com o **Bellman-Ford** foi vital, confirmando sua **estabilidade** ao retornar distâncias positivas e corretas com a **injeção isolada de pesos negativos** (exemplo: 0.2341). Por outro lado, sua capacidade de **detectar ciclos de peso negativo** (retornando valores altamente negativos, como -185.96) demonstrou sua função de **garantia de integridade** em cenários de otimização onde a assunção de pesos não-negativos é insegura.

B. Impacto dos Pesos

A variedade de pesos no modelo original ($peso = 1 - similaridade$) foi suficiente para enriquecer as trajetórias do Dijkstra, distinguindo-o da busca puramente topológica. A ausência de pesos negativos no modelo original garante compatibilidade com Dijkstra, mas a experimentação com o Bellman-Ford evidenciou que o modelo é sensível a essas perturbações, e que sua inclusão, se justificada, poderia representar relações de "afinidade extrema".

5.3 Limitações e Recomendações Futuras

Os resultados mostram que as métricas de proximidade adotadas foram suficientes para capturar a proximidade estilística e estrutural entre as faixas, contemplando o escopo requisitado.

Contudo, o principal limite do modelo permanece sendo a ausência de aspectos contextuais (como época de lançamento, colaboração entre artistas ou semelhança harmônica) devido à forma de coleta de dados escolhida. Para enriquecer as nuances do grafo e aumentar sua capacidade preditiva, recomenda-se:

1. **Enriquecimento de Dados:** Mudar para um modelo de coleta de dados em tempo real ou complementar a base de dados com metadados contextuais para criar pesos mais dinâmicos e multifacetados.
2. **Exploração de Grafos Híbridos:** Investigar a aplicação de algoritmos em grafos híbridos onde diferentes tipos de arestas (representando similaridade musical, colaboração e proximidade temporal) possam coexistir.

Em suma, este estudo reforça a importância de um alinhamento rigoroso entre o algoritmo, a natureza da rede e o objetivo da análise, sendo um ponto de partida sólido para futuras investigações em sistemas de recomendação e planejamento urbano baseados em grafos.

REFERÊNCIA

[1] Prefeitura do Recife, BAIRROS DO RECIFE. Dados.pe.recife. Disponível em: <http://dados.recife.pe.gov.br/>. Acesso em 3 de novembro de 2025.

[2] Prefeitura do Recife, PERFIL DOS BAIRROS DE RECIFE. recife.pe.gov. Disponível em: <<https://www2.recife.pe.gov.br/servico/aflitos?op=NzQ0MQ==#>. Acesso em 3 de novembro de 2025.

[3] Spotify Songs for ML & Analysis (8700 + tracks). Disponível em: < https://www.kaggle.com/datasets/alyahmedts13/spotify-songs-for-ml-and-analysis-over-8700-tracks>. Acesso em 18 de novembro de 2025.