# Cálculo de Rotas Urbanas com Algoritmo de Dijkstra Aplicado a Dados do OpenStreetMap

**Isadora Maria de Araujo Cathalat, Leticia Amaral Figueiredo, Thiago Borges Laass**

[1]Instituto de Ciências Exatas e Informática (ICEI) – Pontifícia Universidade Católica de Minas Gerais (PUC Minas) – CEP 30535-901 – Belo Horizonte – MG – Brasil

imcathatal@sga.pucminas.br

leticia.figueiredo.1417260@sga.pucminas.br, tblaass@sga.pucminas.br

***Abstract.*** *This work presents the development of a geographic routing system based on Dijkstra's algorithm, applied to OpenStreetMap data. The application enables the calculation of the shortest path between two points on urban maps of Brazilian cities, by converting geographic data into weighted graphs whose structure accounts for the geometry of the streets. The client-server architecture integrates a Python-based backend with an interactive frontend, allowing users to select points directly on the map. The system builds the graph, identifies the nodes closest to the selected coordinates, and executes Dijkstra's algorithm to determine the optimal route.*

***Resumo.*** *Este trabalho descreve o desenvolvimento de um sistema de roteamento geográfico baseado no algoritmo de Dijkstra, aplicado a dados do OpenStreetMap. A aplicação permite o cálculo do menor caminho entre dois pontos em mapas urbanos brasileiros, convertendo dados geográficos em grafos ponderados, cuja estrutura considera a geometria das vias. A arquitetura cliente-servidor integra um backend em Python com um frontend interativo, permitindo ao usuário selecionar pontos diretamente sobre o mapa. O sistema realiza a construção do grafo, identifica os nós mais próximos aos pontos selecionados e executa o algoritmo de Dijkstra para determinar a rota ótima.*

## 1. Contextualização

A crescente urbanização tem trazido desafios significativos à mobilidade nas cidades. Entre esses desafios está a necessidade de determinar rotas eficientes entre dois pontos em meio a malhas viárias cada vez mais densas e complexas. A análise de caminhos possíveis, quando feita manualmente, se torna inviável devido ao elevado número de alternativas.

Nesse contexto, este projeto propõe o desenvolvimento de um sistema computacional capaz de calcular a menor rota entre dois pontos em uma cidade, utilizando dados do OpenStreetMap (OSM) e aplicando o algoritmo de Dijkstra, uma vez que este se mostra eficiente para grafos com arestas de peso não negativo — caso das distâncias percorridas nas ruas.

O sistema é dividido em duas partes: o front-end, que permite a interação com o usuário por meio de um mapa dinâmico, e o back-end, responsável pelo processamento dos dados espaciais e cálculo do menor caminho.

## 2. Modelagem

### 2.1. Representação do Grafo Urbano

O sistema adota um modelo de grafo orientado $G = (V, E)$ onde:

- **Vértices (V)**: Pontos geográficos ao longo das vias, com densidade variável conforme a curvatura das ruas (Figura 1).
- **Arestas (E)**: Conexões viárias ponderadas pela distância geodésica (Haversine):

$$w(u, v) = 2R \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_u \cos\phi_v \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \quad (1)$$

  onde:
  - $R = 6371000\,m$ (raio da Terra)
  - $\phi_u, \lambda_u$: latitude e longitude de $u$ em radianos
  - $\Delta\phi = \phi_v - \phi_u$, $\Delta\lambda = \lambda_v - \lambda_u$


**Figure 1. Segmentação de vias sinuosas com maior densidade de vértices**

O processo inicia-se com o download dos dados da cidade selecionada pelo usuário, extraídos do OpenStreetMap. Esses dados são então convertidos em um grafo interno utilizando estruturas de dados apropriadas. As curvaturas das ruas são consideradas na segmentação, resultando em grafos com maior densidade de vértices em regiões mais sinuosas.

O algoritmo de Dijkstra foi escolhido por sua adequação a grafos com pesos não negativos [Dijkstra 1959], como é o caso das distâncias das ruas. Ele é utilizado para determinar o menor caminho entre dois nós, que são selecionados com base nas coordenadas geográficas fornecidas pelo usuário.

### 2.2. Arquitetura do Sistema

#### 2.2.1. Frontend

- Tecnologias: HTML5/CSS3, Leaflet.js (visualização), Fetch API
- Responsabilidades:
  - Interação com usuário
  - Renderização do mapa e rotas
  - Comunicação assíncrona com backend

### 2.2.2. Backend

- **Tecnologias**: Python 3.10+, Flask, Geopy
- **Módulos**:
  - `Data.py`: Aquisição e cache de dados OSM
  - `Graph.py`: Manipulação do grafo e algoritmos

O algoritmo implementado segue o modelo clássico de Dijkstra [Dijkstra 1959] com as seguintes adaptações:

1. **Representação do Grafo**: Dicionário de adjacências para eficiência em grafos esparsos:

```
grafo = {
    nó_1: [(nó_2, 150.2), (nó_3, 80.7)],
    ...
}
```

2. **Relaxamento de Arestas**: Atualização condicional das distâncias mínimas:

$$\text{Se } d[u] + w(u,v) < d[v] \Rightarrow d[v] \leftarrow d[u] + w(u,v) \tag{2}$$

3. **Fila de Prioridade**: Implementada via `heapq` (binary heap), com complexidade $O(\log n)$ por operação.
4. **Condição de Término**: Interrupção quando o nó destino é extraído da fila, otimizando o tempo de execução.
5. **Complexidade**:

$$O(|E| + |V| \log |V|) \quad \text{(para } |E| \text{ arestas e } |V| \text{ vértices)} \tag{3}$$

   - **Tempo**: Dominado pelo custo da fila de prioridade (heap).
   - **Espaço**: $O(|V|)$ para armazenar distâncias e predecessores.

**Otimizações Futuras**:

- Substituição por A* com heurística geodésica para grafos grandes.
- Indexação espacial (ex: *R-trees*) para consultas repetidas.

Para uma compreensão detalhada da estrutura e funcionamento do sistema, apresentam-se a seguir diversos diagramas que ilustram os aspectos principais da modelagem. Esses diagramas abrangem as interações do usuário, a arquitetura do sistema e o fluxo das operações internas, proporcionando uma visão clara e organizada dos componentes envolvidos e suas relações.

## 2.3. Casos de Uso

A Figura 2 apresenta o diagrama de casos de uso que descreve as principais interações do usuário com o sistema, como selecionar pontos no mapa e visualizar a menor rota entre dois locais.


**Figure 2. Diagrama de Casos de Uso do sistema**

## 2.4. Histórias de Usuário

**HU01 – Seleção de cidade** Como *usuário*, eu quero selecionar uma cidade brasileira a partir de uma lista ou busca por nome, para que o sistema carregue o mapa correspondente.

**HU02 – Seleção de pontos no mapa** Como *usuário*, eu quero clicar em dois pontos no mapa (origem e destino), para que o sistema calcule a rota mais curta entre eles.

**HU03 – Cálculo da menor rota** Como *usuário*, eu quero que o sistema calcule automaticamente a menor rota entre os pontos selecionados, para planejar trajetos eficientes.

**HU04 – Visualização da rota** Como *usuário*, eu quero visualizar a rota destacada no mapa com vértices (bolinhas vermelhas) e arestas (linhas vermelhas), para compreender o caminho sugerido.

**HU05 – Detalhes da rota** Como *usuário*, eu quero ver um painel com distância total (em km), número de vértices e arestas, para analisar a complexidade do trajeto.

**HU06 – Exibição de coordenadas** Como *usuário*, eu quero visualizar as coordenadas geográficas (latitude/longitude) da origem e destino, para referência espacial precisa.

**HU07 – Reiniciar busca** Como *usuário*, eu quero um botão disponível para remover a rota atual e iniciar nova consulta sem recarregar a página.

**HU08 – Troca de cidade** Como *usuário*, eu quero alternar minha busca entre cidades, para comparar rotas em diferentes localidades.

**HU09 – Feedback visual** Como *usuário*, eu quero ver uma animação de carregamento durante o processamento da rota, para saber que o sistema está respondendo.

**HU10 – Controle de zoom** Como *usuário*, eu quero ajustar o zoom do mapa manualmente, para melhor visualização.

## 2.5. Diagrama de Componentes

A Figura 3 representa a divisão dos componentes principais do sistema, separando responsabilidades entre frontend, backend e bibliotecas externas.

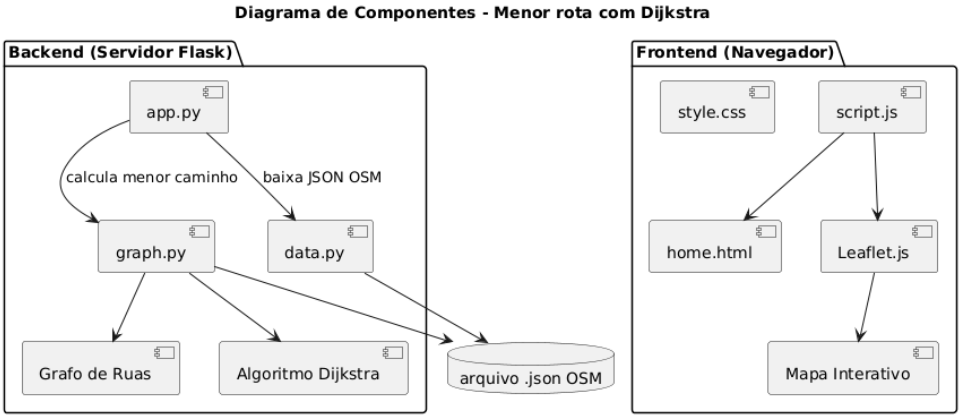


**Figure 3. Diagrama de Componentes do sistema**

## 2.6. Diagrama de Classes

O diagrama de classes, ilustrado na Figura 4, detalha a estrutura interna do sistema, com foco nas principais classes utilizadas no backend.

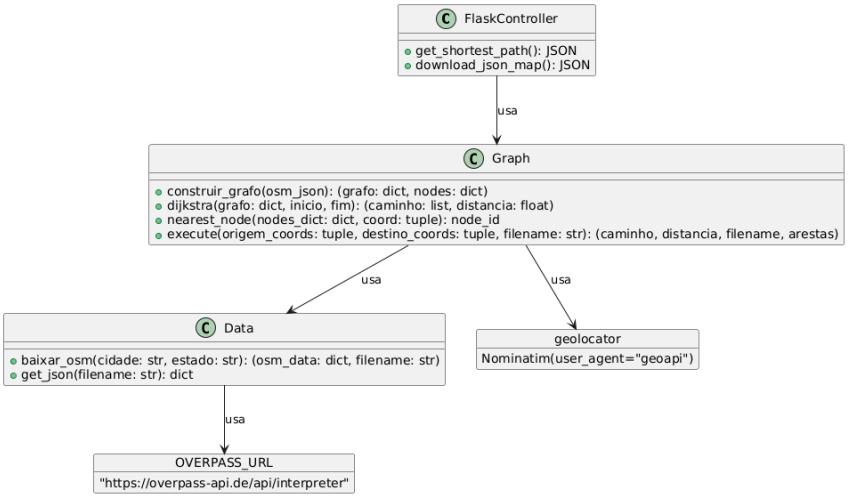


**Figure 4. Diagrama de Classes do backend**

## 2.7. Diagrama de Sequência

A Figura 11 ilustra a interação entre os componentes do sistema durante a solicitação de cálculo do menor caminho.

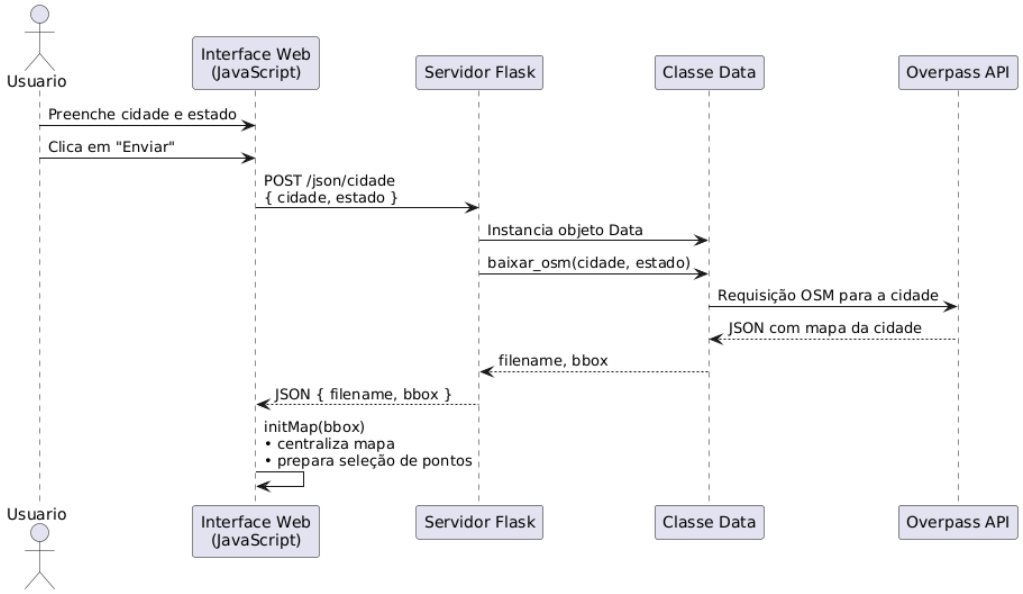


**Figure 5. Diagrama de sequência para baixar mapa**

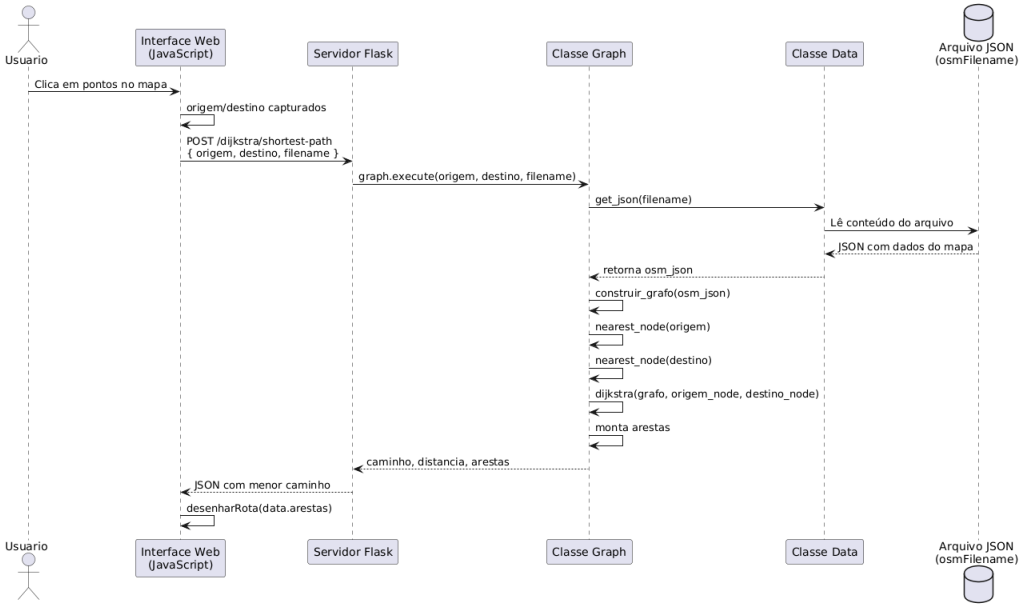


**Figure 6. Diagrama de sequência para requisição de rota**

## 3. Telas Produzidas

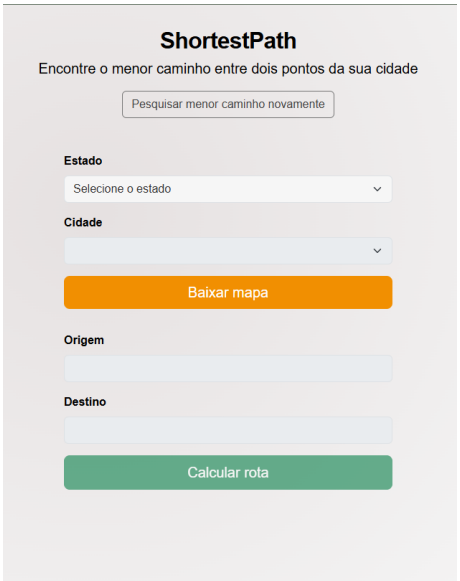


**Figure 7. Tela inicial**

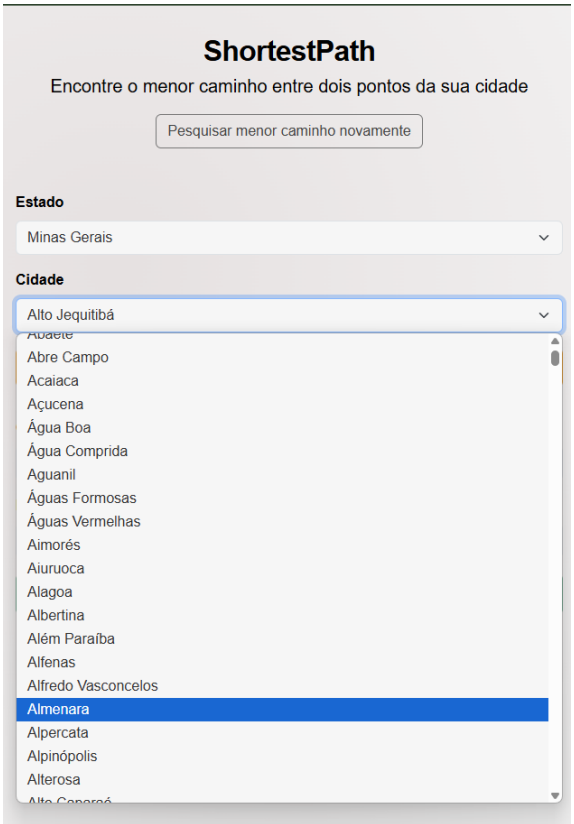


**Figure 8. Escolha da cidade**

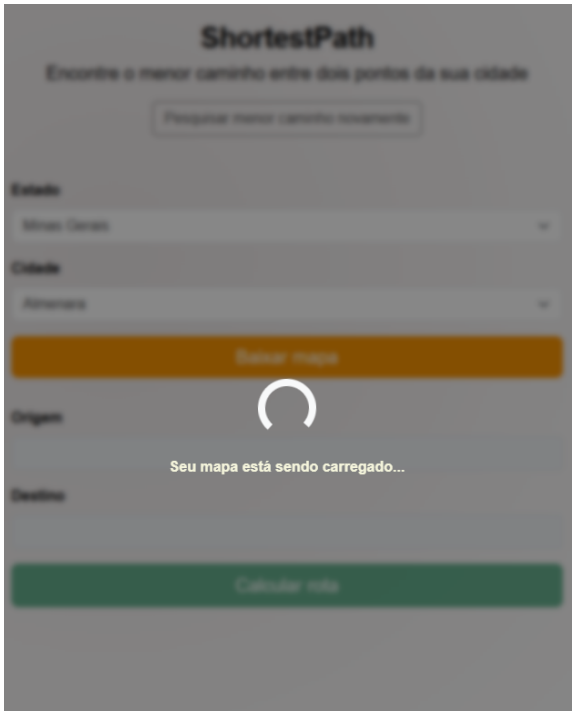


**Figure 9. Sinalização visual do carregamento do mapa**

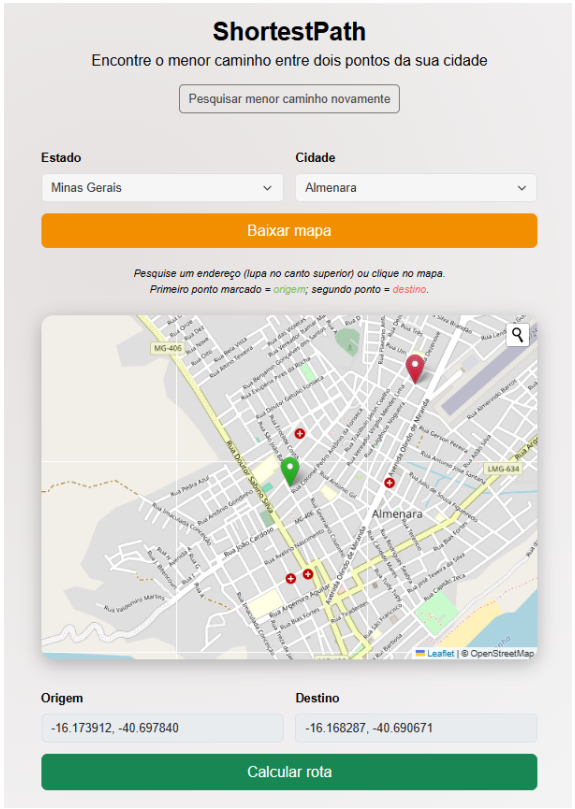


**Figure 10. Escolha dos pontos de origem e destino**

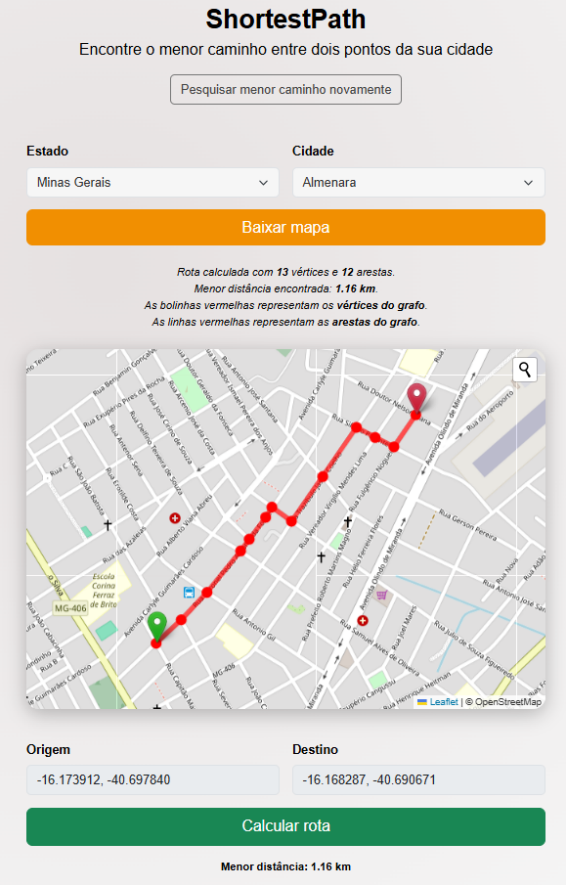


**Figure 11. Resultado do menor caminho no mapa**

# 4. Referências e Padrões

## 4.1. Documentação Técnica

- **OpenStreetMap API**: A documentação oficial da API Overpass utilizada para obtenção dos dados geográficos está disponível em `https://wiki.openstreetmap.org/wiki/Overpass_API` [OpenStreetMap Contributors 2023b].
- **Formato Bounding Box**: As coordenadas seguem a ordem padrão OSM:

$$\text{bbox} = (\text{minlon}, \text{minlat}, \text{maxlon}, \text{maxlat}) \tag{4}$$

  Exemplo: (-46.825, -23.704, -46.365, -23.395) delimita a região central de São Paulo.

# 5. Glossário

**Bounding Box** Retângulo delimitador definido por pares de coordenadas geográficas (longitude, latitude) que engloba uma área de interesse. Matematicamente representado como:

$$B = \{(x, y) \in R^2 | x_{min} \leq x \leq x_{max}, y_{min} \leq y \leq y_{max}\} \tag{5}$$

**Geocodificação** Processo de conversão entre coordenadas geográficas (lat/lon) e endereços físicos, implementado via API Nominatim [OpenStreetMap Contributors 2023a] neste trabalho.

**Nó mais próximo** Algoritmo que encontra o vértice $v \in V$ em $G = (V, E)$ minimizando a distância euclidiana plana entre coordenadas geográficas, conforme implementado:

$$v^* = \arg\min_{v \in V} \text{hypot}(lat_v - lat_p, lon_v - lon_p) \tag{6}$$

**Detalhes de implementação**:

- **Métrica**: Distância euclidiana (via `math.hypot`), válida para áreas $<$ 50 km² (erro $< 0.3\%$).
- **Complexidade**: $O(n)$ com busca linear em todos os nós.
- **Input flexível**: Aceita coordenadas como tupla `(lat, lon)` ou string `"lat,lon"`.

**Nota**: Para aplicações que exigem maior precisão ou grafos muito grandes, recomenda-se:

- Substituir a métrica por *Haversine* (já implementada no cálculo de arestas).
- Adicionar estruturas de indexação espacial (ex: *k-d trees* [Bentley 1975]).

## 6. Análise dos Resultados Obtidos

O sistema foi testado em diferentes cidades e demonstrou funcionar corretamente ao identificar os menores caminhos entre dois pontos selecionados. Os caminhos retornados são realistas, respeitando a malha urbana e oferecendo trajetos coerentes com os mapas reais.

Os dados retornados incluem a sequência de vértices e arestas de forma visual (mapa) e a quantidade total de cada um deles, além da menor distância percorrida. A renderização das rotas no mapa mostra consistência entre os pontos geográficos e os caminhos gerados, mesmo em cidades com traçados complexos.

Em regiões com ruas sinuosas, o grafo apresenta maior densidade de vértices, o que aumenta a precisão da rota ao representar melhor a geometria da via. Já em áreas mais retilíneas, a quantidade de vértices é menor, tornando o grafo mais enxuto.

## 7. Contribuições dos Integrantes

As responsabilidades foram distribuídas conforme as especialidades de cada membro da equipe, conforme detalhado nas subseções abaixo.

### 7.1. Isadora Maria de Araujo Cathalat

Encarregada do desenvolvimento frontend, com destaque para:

- Implementação da interface com Leaflet.js
- Integração com a API IBGE para seleção de cidades
- Desenvolvimento do sistema de geocodificação visual
- Criação de componentes reativos em JavaScript
- Design responsivo com CSS3
- Consumo da API backend via Fetch API

### 7.2. Letícia Amaral Figueiredo

Responsável pela documentação acadêmica e técnica do sistema, com as seguintes contribuições específicas:

- Elaboração da modelagem teórica do grafo urbano
- Desenvolvimento da documentação em LaTeX
- Criação de diagramas UML e fluxogramas
- Formatação de equações matemáticas
- Revisão bibliográfica

### 7.3. Thiago Borges Laass

Responsável pela implementação do backend, incluindo:

- Desenvolvimento do algoritmo de Dijkstra otimizado
- Criação da API REST com Flask
- Implementação da estrutura de dados do grafo
- Integração com a Overpass API (OpenStreetMap)
- Sistema de cache de dados geográficos
- Tratamento de exceções e códigos de erro HTTP

### 7.4. Integração e Trabalho Colaborativo

A equipe adotou as seguintes práticas para integração:

- Reuniões pontuais para sincronização
- Versionamento com Git
- Documentação compartilhada via Overleaf
- Testes cruzados entre frontend e backend

## 8. Conclusões

O sistema desenvolvido atingiu seu objetivo de oferecer uma solução eficiente para o cálculo de rotas urbanas curtas utilizando dados abertos e algoritmos clássicos de teoria dos grafos. A divisão entre front-end e back-end permitiu modularidade e escalabilidade, facilitando a manutenção e possível expansão do sistema.

A escolha do algoritmo de Dijkstra provou-se adequada ao domínio do problema, e a representação do grafo baseada na geometria real das vias garantiu maior precisão no cálculo das rotas.

Como trabalhos futuros, propõe-se a implementação de novos critérios de otimização, como tempo de percurso ou condições de tráfego, além da incorporação de algoritmos mais avançados como A* [Hart et al. 1968] e o uso de estruturas de dados otimizadas para buscas espaciais.

## References

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

OpenStreetMap Contributors (2023a). Nominatim – openstreetmap's geocoding api. Acessado em: 10 Out. 2023.

OpenStreetMap Contributors (2023b). Overpass api documentation.