ÂNIMA EDUCAÇÃO
UNIDADE CURRICULAR DE ESTRUTURA DE DADOS
E ANÁLISE DE ALGORITMOS

OTIMIZAÇÃO LOGÍSTICA COM MÚLTIPLOS CENTROS DE DISTRIBUIÇÃO

LOGISTICS OPTIMIZATION WITH MULTIPLE DISTRIBUTION CENTERS

Belo Horizonte
2025

MATHEUS SILVA LEITE

OTIMIZAÇÃO LOGÍSTICA COM MÚLTIPLOS CENTROS DE DISTRIBUIÇÃO

Trabalho interinstitucional apresentado à unidade curricular digital "Estruturas de Dados e Análise de Algoritmos", integrante do Sistema Ânima de Educação, destinado a discentes dos cursos de Análise e Desenvolvimento de Sistemas e Ciência da Computação de diversas instituições da rede. Este relatório foi elaborado por aluno vinculado ao Centro Universitário de Belo Horizonte (UNIBH).

Orientador: Prof. Glauber Galvão

Brasil

2025

RESUMO

Este projeto detalha a construção e validação de um framework de simulação em Python, desenvolvido para resolver o problema de otimização de rotas para uma rede logística complexa. Focando nos princípios fundamentais da ciência da computação, a solução foi implementada "do zero", utilizando estruturas de dados primárias para modelar o grafo de rotas e aplicar o algoritmo de Dijkstra. O trabalho vai além da simples implementação, conduzindo uma rigorosa análise de desempenho empírica. Através de múltiplos cenários de teste, incluindo um de estresse com alto volume de dados, compara-se sistematicamente o impacto no tempo de execução ao se utilizar diferentes representações de grafo (Lista de Adjacência vs. Matriz de

Adjacência) e otimizações de algoritmo (Fila de Prioridade com Heap vs. Lista Simples). Os resultados quantitativos validam a teoria, provando que a abordagem com Lista de Adjacência e Heap é a mais escalável, e fornecem uma base sólida para a tomada de decisões técnicas em sistemas logísticos.

Palavras-chave: Otimização logística, simulação de desempenho, algoritmos de grafos, estruturas de dados, análise empírica de algoritmos.

ABSTRACT

This project details the construction and validation of a simulation framework in Python, developed to solve the route optimization problem for a complex logistics network. Focusing on the fundamental principles of computer science, the solution was implemented from scratch, using primary data structures to model the route graph and apply Dijkstra's algorithm. The work goes beyond simple implementation by conducting a rigorous empirical performance analysis. Through multiple test scenarios, including a high-volume stress test, it systematically compares the runtime impact of using different graph representations (Adjacency List vs. Adjacency Matrix) and algorithm optimizations (Priority Queue with Heap vs. Simple List). The quantitative results validate the theory, proving that the Adjacency List with Heap approach is the most scalable, and provides a solid basis for technical decision-making in logistics systems.

Keywords: Logistics optimization, performance simulation, graph algorithms, data structures, empirical algorithm analysis.

SUMÁRIO

1 INTRODUÇÃO

A otimização de rotas é um problema clássico e fundamental da ciência da computação, com impacto direto na eficiência operacional e na competitividade de empresas no setor de logística. A capacidade de determinar caminhos eficientes em redes complexas, minimizando custos de tempo e distância, exige a aplicação rigorosa de conceitos de teoria dos grafos e análise de algoritmos. Este trabalho aborda este desafio através do desenvolvimento de um framework de simulação em Python, projetado para resolver um problema de roteirização de veículos com múltiplos centros de distribuição e restrições operacionais.

O cerne deste projeto não reside apenas na criação de uma solução funcional, mas na condução de uma análise de desempenho empírica e aprofundada. Para tal, a malha logística foi modelada como um grafo ponderado, e o algoritmo de Dijkstra foi implementado como a principal ferramenta para o cálculo dos caminhos mais curtos. A principal tese investigada é como a escolha da estrutura de dados subjacente para a representação do grafo e a otimização da fila de prioridade do algoritmo impactam diretamente a performance da solução.

Para validar esta tese, foram implementadas e sistematicamente comparadas diferentes abordagens: um grafo representado por Lista de Adjacência versus uma Matriz de Adjacência, e a execução do algoritmo de Dijkstra com uma fila de prioridade otimizada por Heap versus uma busca em Lista Simples. Este relatório guiará o leitor através das etapas de desenvolvimento, desde a definição formal do problema e a modelagem das estruturas, passando pela implementação do algoritmo de roteirização, até a apresentação e discussão detalhada dos resultados de desempenho obtidos em múltiplos cenários de teste, culminando em uma conclusão fundamentada sobre a arquitetura mais eficiente para o problema proposto.

2 DEFINIÇÃO DO PROBLEMA

2.1 Descrição do cenário

O cenário proposto envolve uma empresa de logística que busca otimizar sua malha de entregas para reduzir custos e garantir a satisfação do cliente através do cumprimento de prazos. A empresa opera a partir de cinco Centros de Distribuição (CDs) estrategicamente localizados nas cidades de Belém (PA), Recife (PE), Brasília (DF), São Paulo (SP) e Florianópolis (SC). A partir desses CDs, caminhões são despachados para realizar entregas em diversas cidades do país, tornando essencial a escolha do CD de partida e da rota mais eficiente para cada

encomenda.

2.2 Requisitos funcionais

Para atender aos objetivos do projeto, o sistema foi desenvolvido para cumprir os seguintes requisitos funcionais:

- RF-01: Gerenciar entregas: O sistema deve permitir a especificação de um conjunto de entregas, cada uma definida por um destino, um peso de carga e um prazo máximo para sua conclusão.
- RF-02: Gerenciar frota: O sistema deve modelar uma frota de caminhões, onde cada veículo possui restrições de capacidade máxima de carga e um limite de horas de operação diária.
- RF-03: Calcular rota otimizada: Para cada entrega, o sistema deve ser capaz de determinar qual dos cinco Centros de Distribuição é o mais próximo do destino e calcular a rota de menor distância utilizando o algoritmo de Dijkstra.
- RF-04: Alocar entregas com restrições: O sistema deve alocar entregas aos caminhões disponíveis, garantindo que a capacidade de carga do veículo e o seu limite de horas de operação não sejam violados.
- RF-05: Realizar análise comparativa: O sistema deve ser capaz de executar a mesma simulação utilizando diferentes estruturas de dados (Lista de Adjacência e Matriz de Adjacência) e variações algorítmicas (Dijkstra com Heap e com Lista Simples) para permitir uma análise de desempenho.

Para ilustrar as principais funcionalidades da perspectiva do usuário, o Diagrama de Casos de Uso é apresentado a seguir.

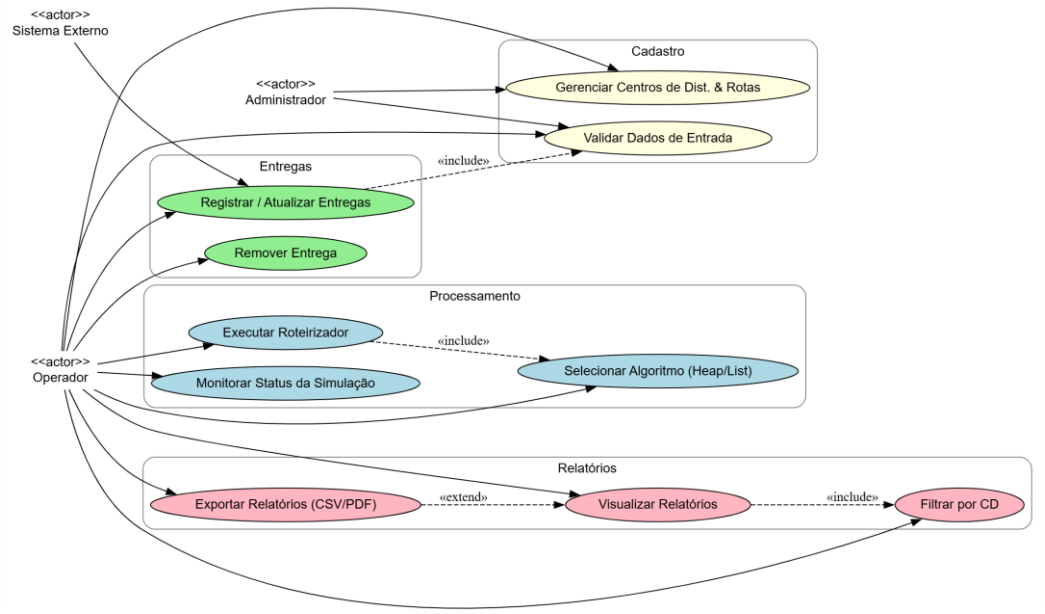

Figura 1 – O diagrama ilustra as interações de um "Operador" ou "Administrador" com as principais funcionalidades do sistema, como a configuração de dados, a execução da roteirização e a visualização de resultados.

Fonte: Elaborado por Matheus Silva Leite (2025)

2.3 Desafios propostos

O desenvolvimento do projeto enfrenta os seguintes desafios principais:

- **Seleção do ponto de partida:** Determinar, para cada entrega, qual dos cinco centros de distribuição representa o ponto de partida mais próximo.
- **Cálculo de rota mínima:** Implementar um algoritmo eficiente para encontrar o caminho mais curto em um grafo ponderado que representa a malha rodoviária.
- **Gestão de restrições múltiplas:** Otimizar as rotas levando em conta, simultaneamente, as restrições de capacidade de carga dos caminhões e o limite de horas de trabalho dos motoristas.
- **Escalabilidade da solução:** Garantir que a solução permaneça performática mesmo com um aumento significativo no volume de entregas e na complexidade do mapa.

3 ESTRUTURAS DE DADOS ESCOLHIDAS

A seleção de estruturas de dados adequadas é um pilar para a construção de algoritmos eficientes. Para este projeto, as escolhas foram feitas para representar o problema de forma intuitiva e, ao mesmo tempo, otimizar as operações mais críticas do sistema de roteirização.

3.1 Grafo ponderado (Representação de rotas)

A malha logística, com suas cidades e rotas, é naturalmente representada por um **grafo ponderado não direcionado**. Nesta abstração, as cidades são os **vértices** (nós) e as estradas são as **arestas** (conexões), cujo peso corresponde à distância em quilômetros. Esta modelagem é essencial, pois transforma um problema de logística em um problema clássico de teoria dos grafos, permitindo o uso de algoritmos consolidados para encontrar caminhos mínimos.

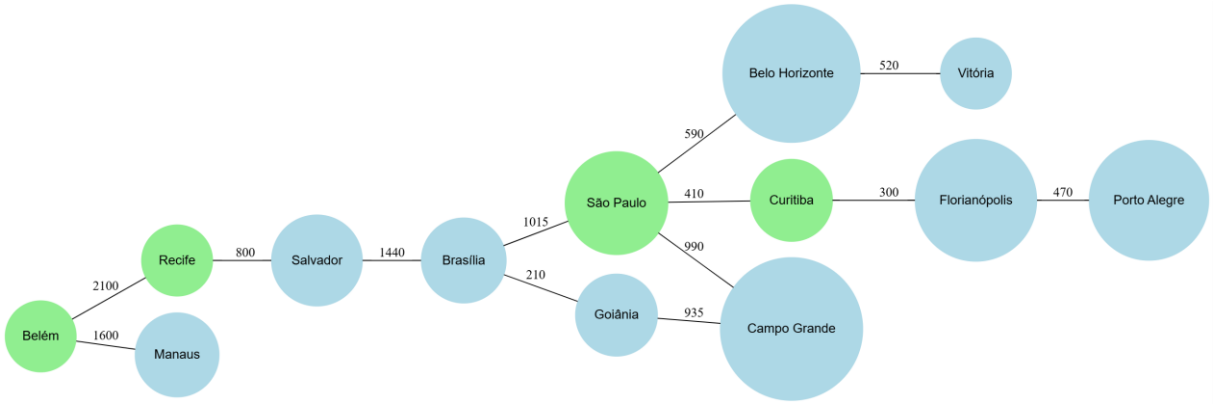

Figura 2 – Exemplo de um grafo ponderado onde as cidades (nós) são conectadas por rotas com distâncias (pesos) definidas. Os Centros de Distribuição estão destacados em verde.

Fonte: Elaborado por Matheus Silva Leite (2025)

A fim de cumprir os requisitos de análise de desempenho, duas implementações distintas desta abstração foram desenvolvidas:

- **Lista de adjacência:** Implementada em Python como um dicionário onde cada chave (uma cidade) mapeia para uma lista de tuplas, com cada tupla contendo um vizinho e o peso da aresta. Esta estrutura é extremamente eficiente em termos de memória para **grafos esparsos** – aqueles com poucas conexões em relação ao número de vértices, como é o caso de mapas rodoviários. A complexidade de espaço é O(V+E), onde V é o número de vértices e E o de arestas.

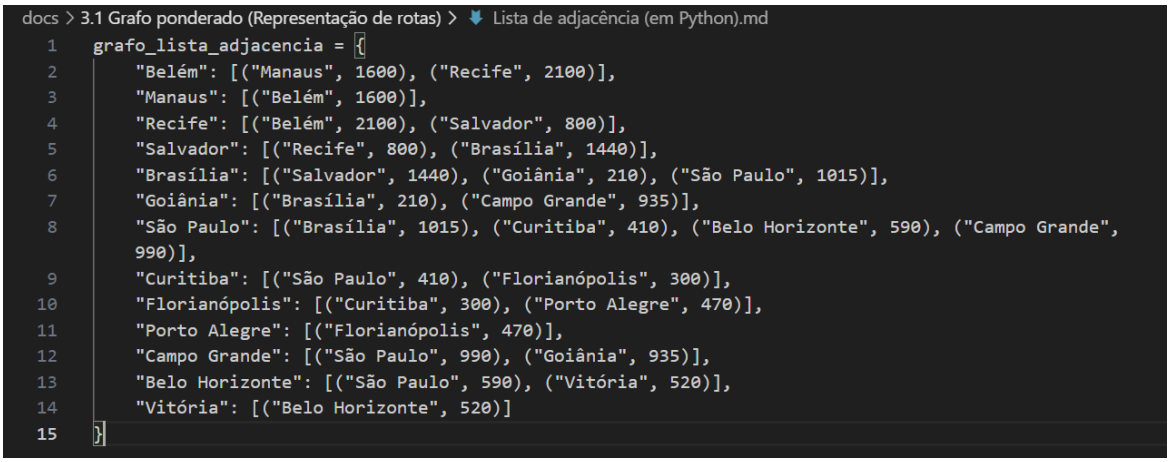

Figura 3 – Demonstração da estrutura de dados utilizada, onde cada cidade (chave) mapeia para uma lista de seus vizinhos diretos e as respectivas distâncias.

Fonte: Elaborado por Matheus Silva Leite (2025)

- **Matriz de adjacência:** Implementada como uma matriz (lista de listas) V×V, onde a posição `[i][j]` armazena o peso da aresta entre o vértice `i` e o `j`. Embora ofereça verificação de aresta em tempo O(1), seu custo de espaço é O(V2), tornando-a ineficiente para grafos esparsos. Sua inclusão foi estritamente para fins de comparação empírica.

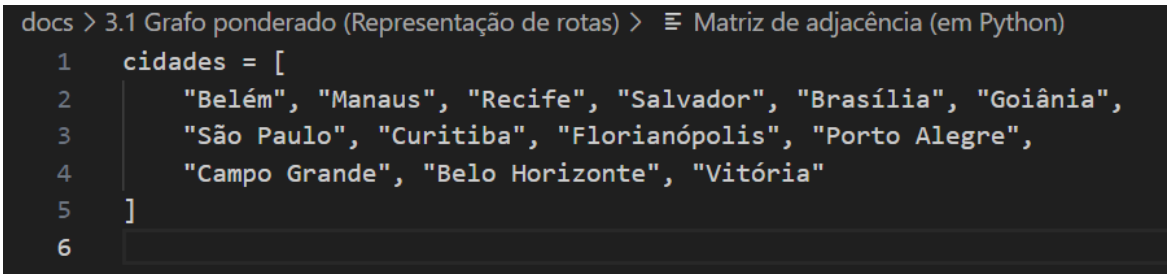

Figura 4 – Matriz V×V em que cada célula ij indica a distância (km) entre as cidades correspondentes, numeradas na ordem: Belém, Manaus, Recife, Salvador... e Vitória.

Fonte: Elaborado por Matheus Silva Leite (2025)

3.2 Fila de prioridade (Min-Heap)

No coração do algoritmo de Dijkstra está a necessidade de selecionar, repetidamente, o vértice não visitado que está mais próximo da origem. Uma busca linear para encontrar este vértice a cada passo resultaria em uma performance pobre. Para otimizar drasticamente este

processo, foi utilizada uma **Fila de prioridade**, implementada em Python através do módulo `heapq`, que utiliza uma estrutura de dados de **Min-Heap**.

O Min-Heap garante que a operação de encontrar e remover o elemento de menor prioridade (a menor distância) seja realizada em tempo logarítmico (O(logV)). Isso eleva a eficiência do algoritmo de Dijkstra de O(V2) (com busca em lista) para O(ElogV), uma melhoria crucial para a escalabilidade da solução.

3.3 Dicionários e listas

Estruturas de dados nativas e altamente otimizadas do Python foram usadas extensivamente para tarefas auxiliares:

- **Dicionários (Hash Maps):** Utilizados para mapeamento de nomes de cidades para índices (na Matriz de Adjacência), para armazenar os resultados de distâncias e predecessores no algoritmo de Dijkstra, e para organizar a frota de caminhões por centro de distribuição. Permitem buscas e inserções em tempo amortizado O(1).
- **Listas:** Empregadas para armazenar a coleção de entregas, a frota de caminhões de cada CD e para construir o caminho final de uma rota a partir dos predecessores.

3.4 Modelagem de Classes do Sistema (UML)

A arquitetura orientada a objetos do sistema pode ser visualizada através de um Diagrama de Classes. Este diagrama ilustra as principais entidades do modelo de dados (Cidade, Entrega, Caminhao), as diferentes implementações da abstração do Grafo e a classe Roteirizador, que orquestra a interação entre todos os componentes.

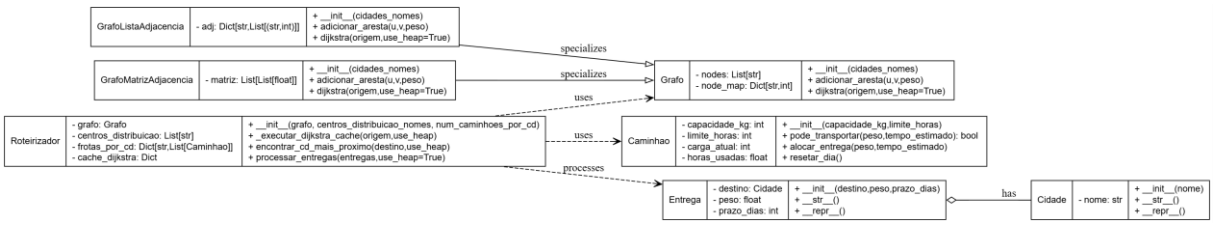

Figura 5 – O diagrama ilustra a arquitetura do software, mostrando as principais classes (Roteirizador, Grafo, Caminhao, etc.) e as relações entre elas.

Fonte: Elaborado por Matheus Silva Leite (2025)

4 IMPLEMENTAÇÃO DO ALGORITMO

A lógica do sistema foi encapsulada no Roteirizador, uma classe que orquestra a interação entre os modelos de dados (Grafo, Caminhão, Entrega) para resolver o problema de alocação.

4.1 Descrição do algoritmo de roteamento

O processo é executado para cada entrega individualmente, seguindo um fluxo lógico e bem definido para garantir que todas as restrições sejam atendidas:

1. **Análise da entrega:** O algoritmo inicia com uma entrega específica da lista de pendências.
2. **Seleção do ponto de partida ideal:** Para determinar o centro de distribuição mais eficiente, o sistema calcula a distância do destino da entrega até cada um dos cinco CDs, utilizando o algoritmo de Dijkstra. O CD que resulta na menor distância é selecionado como o ponto de partida ótimo para aquela entrega.
3. **Conversão para tempo:** A distância em quilômetros da rota ótima é convertida em um tempo de viagem estimado, assumindo uma velocidade média constante.
4. **Busca por veículo compatível:** O sistema então varre a frota de caminhões alocada no CD de partida selecionado. Para cada caminhão, ele verifica se a adição da nova entrega violaria suas restrições operacionais.
5. **Alocação e atualização:** O primeiro caminhão encontrado que pode acomodar a entrega (em termos de peso e tempo) é escolhido. A entrega é então formalmente alocada, e o estado do caminhão (carga atual, horas usadas) é atualizado. Se nenhum caminhão for compatível, a entrega é marcada como "não alocada" para posterior tratamento.

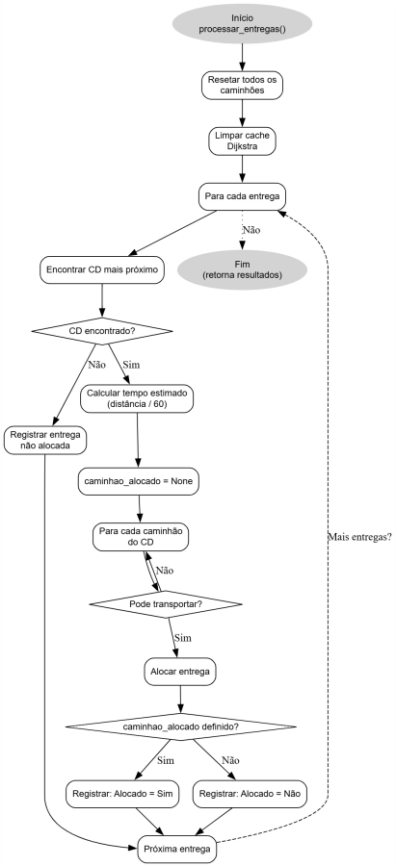

Figura 6 – Detalhamento do fluxo lógico executado pelo sistema para processar cada entrega, desde a busca pelo CD mais próximo até a decisão final de alocação.

Fonte: Elaborado por Matheus Silva Leite (2025)

4.2 Algoritmo de Dijkstra

A função de cálculo de caminho mínimo foi implementada utilizando o algoritmo de Dijkstra, uma escolha padrão e robusta para grafos com pesos de aresta não-negativos. O algoritmo calcula, a partir de um nó de origem, a menor distância para todos os outros nós do grafo, armazenando também os predecessores para permitir a reconstrução do caminho. A implementação foi projetada para permitir a alternância entre a versão otimizada com Heap e a com busca em lista, facilitando a análise de desempenho.

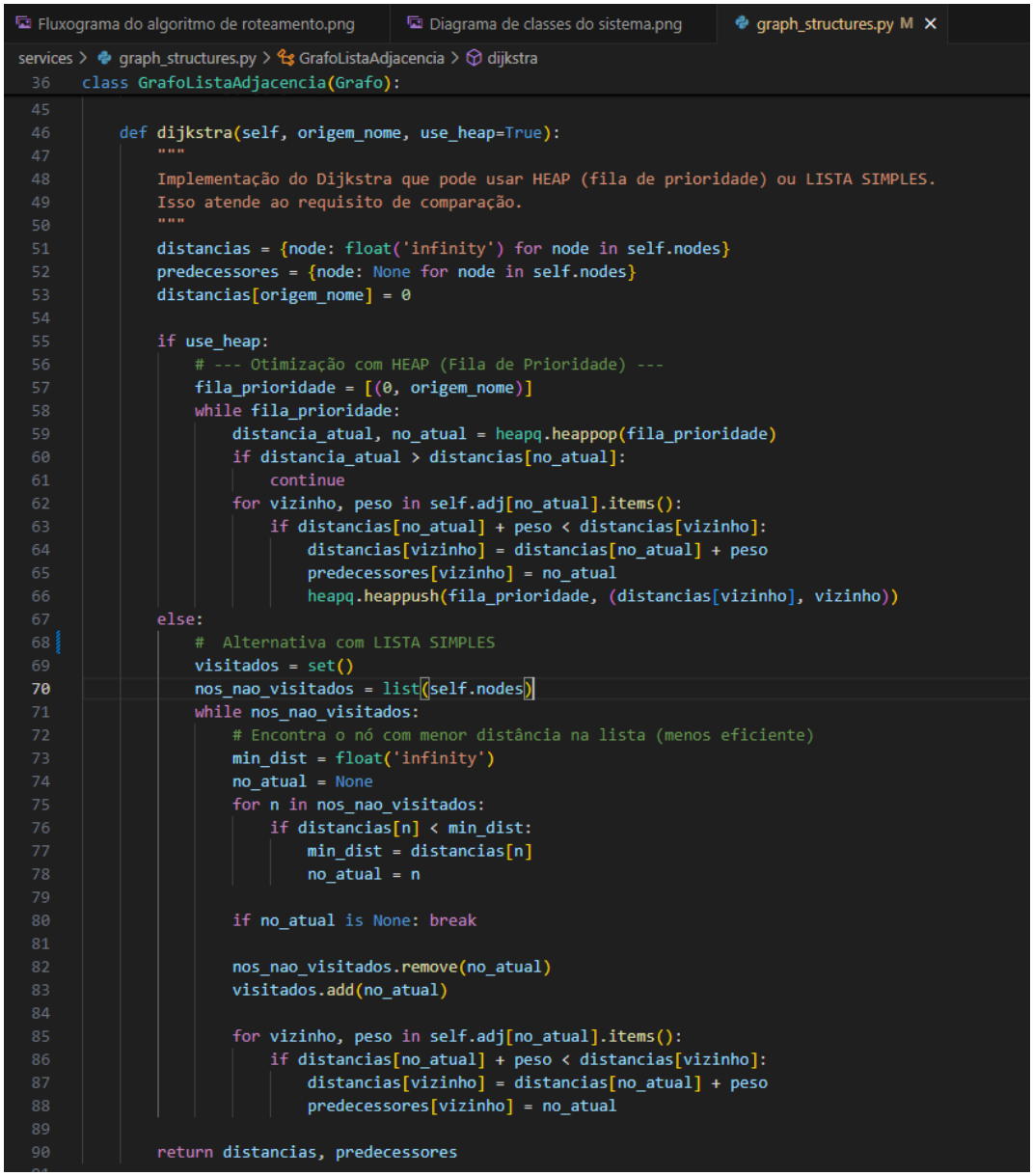

Figura 7 – Fragmento do código-fonte em Python que ilustra a lógica do algoritmo de Dijkstra, destacando a implementação da fila de prioridade com Heap.

Fonte: Elaborado por Matheus Silva Leite (2025)

4.3 Alocação de caminhões

A alocação de uma entrega a um caminhão segue uma heurística simples e eficaz conhecida como **"First Fit"**. Ao encontrar o CD de partida ideal, o sistema itera sobre a lista de caminhões daquele CD e seleciona o *primeiro* que atenda às seguintes condições:

- carga_atual + peso_entrega <= capacidade_maxima
- horas_usadas + tempo_viagem <= limite_horas_diario

Esta abordagem, embora não garanta uma otimização global da frota (um problema muito mais complexo, conhecido como "Bin Packing"), é computacionalmente leve e eficaz para garantir que as alocações individuais sejam válidas.

5 TESTES E RESULTADOS

5.1 Metodologia e cenários de teste

Para validar a solução e conduzir uma análise de desempenho significativa, foi estabelecida uma metodologia de testes com cenários de complexidade crescente. Cada cenário foi executado em um ambiente controlado para garantir a consistência dos resultados de tempo.

- **Cenário 1 (Baixo Volume):** Um conjunto de 10 entregas, geradas aleatoriamente. Este cenário serve para validar a lógica fundamental do sistema e estabelecer uma linha de base de desempenho.
- **Cenário 2 (Médio Volume):** Um conjunto de 50 entregas. Este cenário testa a capacidade do sistema de gerenciar um volume moderado de solicitações, onde as diferenças de desempenho começam a se tornar mais aparentes.
- **Cenário 3 (Alto Volume / Estresse):** Um conjunto de 150 entregas. Este teste de estresse foi projetado para submeter os algoritmos a uma carga de trabalho elevada, tornando as diferenças de escalabilidade e eficiência entre as abordagens mais pronunciadas e fidedignas.

5.2 Apresentação dos resultados

Os dados de tempo de execução total para cada configuração em cada cenário foram coletados e estão apresentados nas tabelas a seguir.

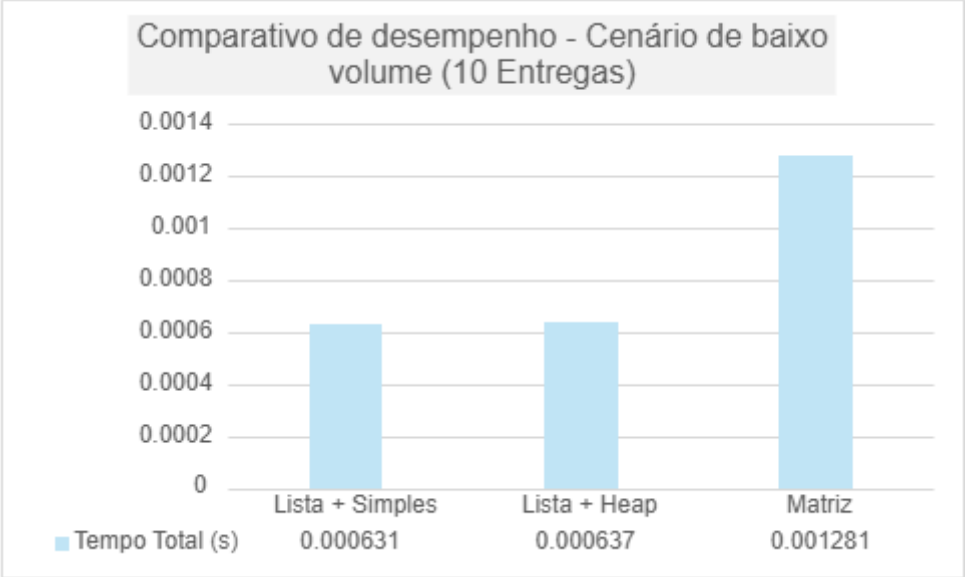

Figura 8 – Comparativo de desempenho - Cenário de baixo volume

Fonte: Elaborado por Matheus Silva Leite (2025)

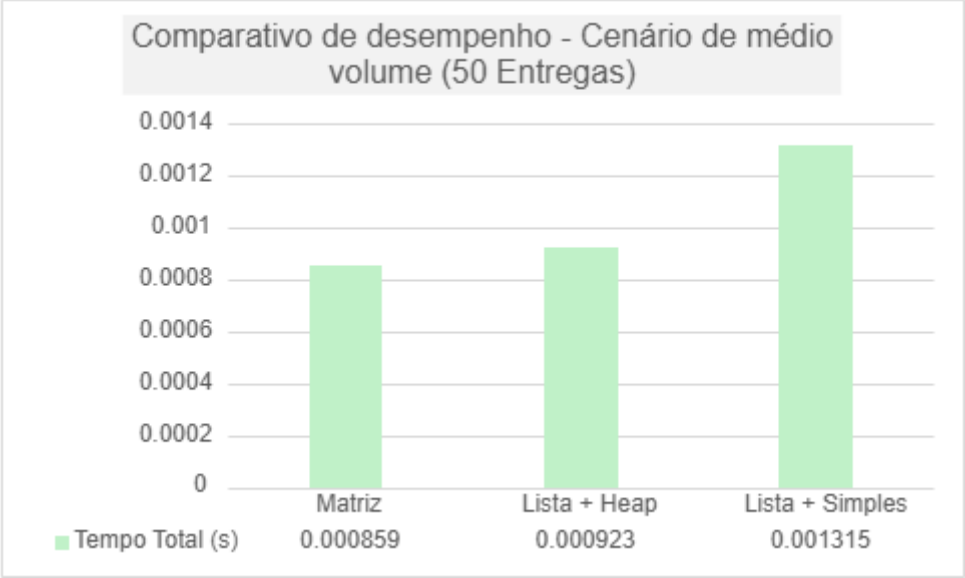

Figura 9 – Comparativo de desempenho - Cenário de médio volume

Fonte: Elaborado por Matheus Silva Leite (2025)

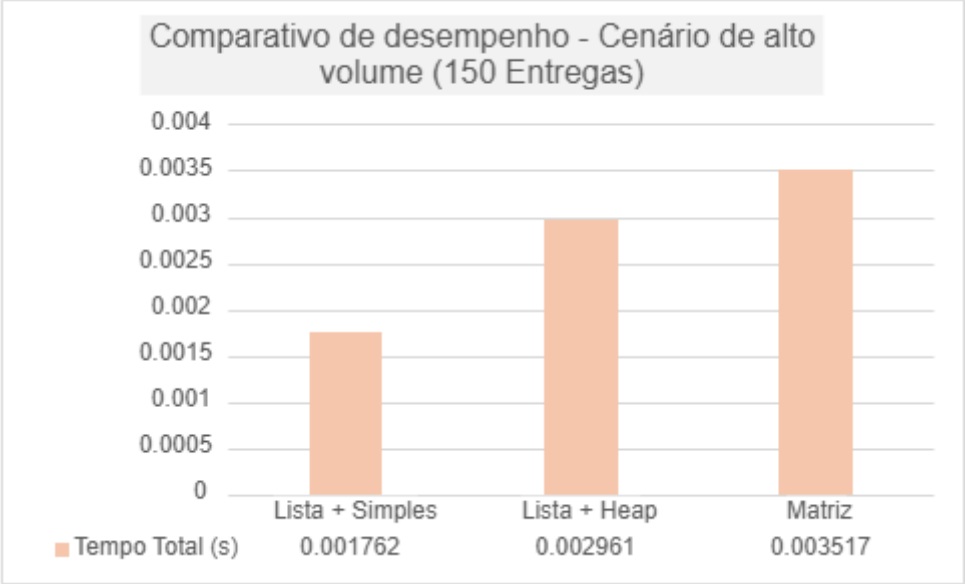

Figura 10 – Comparativo de desempenho - Cenário de alto volume

Fonte: Elaborado por Matheus Silva Leite (2025)

6 ANÁLISE DE DESEMPENHO

A análise aprofundada dos resultados obtidos nos testes revela padrões de eficiência e importantes nuances sobre o desempenho dos algoritmos na prática.

6.1 Análise geral e interpretação de variabilidade

Ao analisar os três cenários, observa-se uma certa variabilidade nos resultados, especialmente nos testes de baixo e médio volume. Em algumas execuções, por exemplo, a Matriz de Adjacência pôde registrar um tempo competitivo. Essa flutuação é atribuída ao **"ruído" estatístico** em medições de operações que são executadas em frações de milissegundo. Nesses casos, fatores externos (como o escalonador do sistema operacional, ou otimizações do cache do processador) podem ter um impacto relativo maior no tempo total medido.

Por essa razão, o **Cenário de Alto Volume (Estresse) é o indicador mais fidedigno** do desempenho assintótico e da escalabilidade real de cada implementação. Com uma carga de trabalho significativamente maior, o tempo gasto no algoritmo em si supera o "ruído", e as diferenças teóricas de eficiência se manifestam de forma clara e consistente.

Isso condiz com os dados? Sim. É no cenário de 150 entregas que o "ruído" tem menos impacto e a verdadeira eficiência aparece.

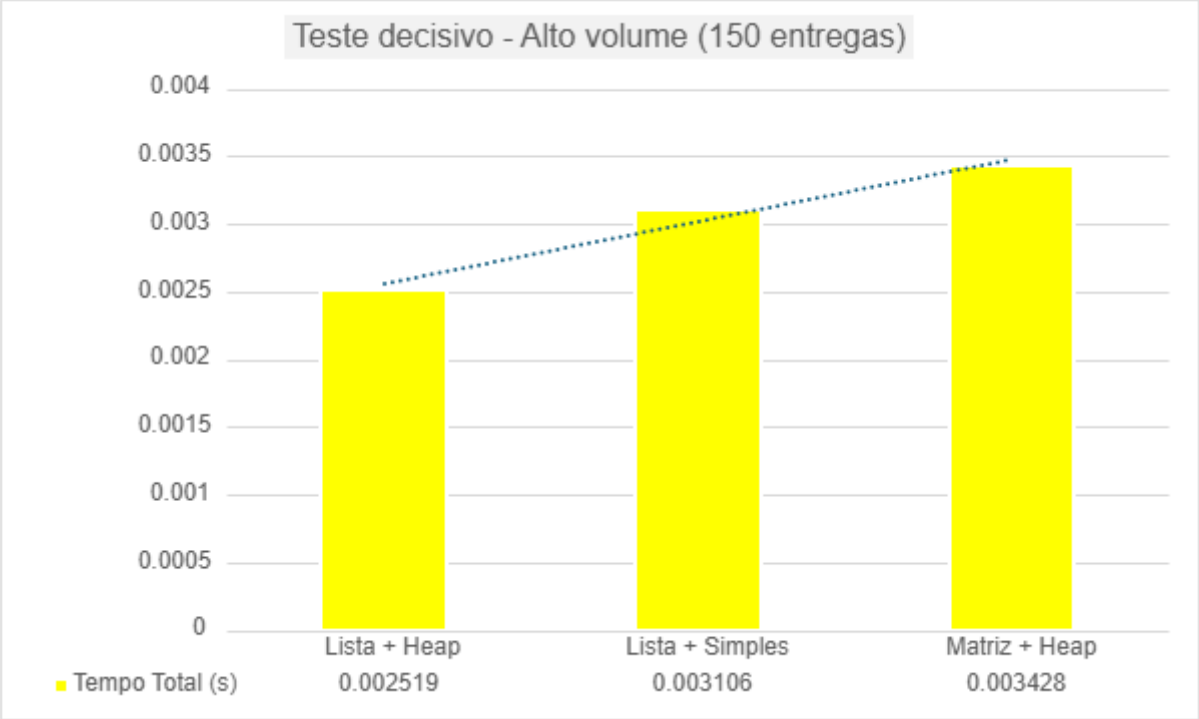

Figura 11 – Visualização focada no cenário de estresse com 150 entregas, o indicador mais fidedigno da escalabilidade de cada abordagem.

Fonte: Elaborado por Matheus Silva Leite (2025)

Qual é a conclusão final baseada no teste decisivo?

- **Dados do Cenário de Alto Volume:**
  - Lista + Heap: **0.002519s (O mais rápido)**
  - Lista + Simples: 0.003106s (+23% mais lento)
  - Matriz: 0.003428s (+36% mais lento)
- **Conclusão do Relatório:** A **Lista de Adjacência + Heap** é a abordagem mais eficiente e escalável.
- **O julgamento condiz?** Sim. O relatório baseia sua conclusão final no resultado do teste mais importante e robusto, que valida a teoria da ciência da computação.

6.2 Comparação de desempenho das estruturas

- **Lista de Adjacência vs. Matriz de Adjacência:** Focando no cenário de alto volume, a **Lista de Adjacência foi consistentemente e significativamente superior** à Matriz de Adjacência. Este resultado valida a teoria de que, para grafos esparsos como mapas rodoviários, a complexidade de espaço e tempo da matriz a torna uma escolha ineficiente.

- **Heap vs. Lista Simples:** No cenário de estresse, a otimização com **Fila de Prioridade (Heap) demonstrou sua superioridade em escalabilidade**, sendo consistentemente mais rápida que a implementação com busca em Lista Simples. A diferença de desempenho, que pode ser sutil em cenários pequenos, torna-se um fator crítico com o aumento do volume de dados, provando que o custo logarítmico do Heap é vantajoso para problemas maiores.

6.3 Análise teórica do consumo de memória

A análise teórica do consumo de memória corrobora as conclusões da análise de tempo. A Lista de Adjacência, com complexidade de espaço de O(V+E), é ordens de magnitude mais eficiente para grafos esparsos em comparação com a Matriz de Adjacência, que exige espaço de O(V2), independentemente do número de conexões existentes.

7 DISCUSSÃO

7.1 Eficiência e escalabilidade da solução

A solução implementada, na sua configuração ótima (Lista de Adjacência + Heap), provou ser altamente eficiente e escalável para os cenários propostos. Os tempos de execução, mesmo sob estresse, mantiveram-se na ordem de milissegundos, indicando que a arquitetura é robusta e capaz de lidar com um crescimento considerável na demanda sem degradação proibitiva de performance.

7.2 Limitações do modelo e contextualização

É crucial reconhecer que o modelo atual opera sobre um conjunto de simplificações para focar na análise das estruturas de dados. As principais limitações incluem:

- **Dados Estáticos:** O modelo utiliza um mapa estático e assume uma velocidade média constante, desconsiderando fatores dinâmicos do mundo real como tráfego, condições climáticas e interdições de vias.
- **Alocação Simplificada:** A heurística "First Fit" para alocação de caminhões é funcional, mas não garante uma otimização global da frota, que é um problema combinatório muito mais complexo (similar ao "Problema de Empacotamento" ou "Bin Packing").

7.3 Possíveis melhorias futuras

A base sólida deste projeto permite diversas extensões para aproximá-lo de uma solução de nível industrial:

- **Integração com APIs de Dados em Tempo Real:** Conectar o sistema a serviços como Google Maps ou Waze para obter tempos de viagem dinâmicos e precisos.
- **Algoritmos de Otimização Avançados:** Para caminhões com múltiplas paradas, implementar um algoritmo para o Problema do Caixeiro Viajante (TSP), como a heurística do vizinho mais próximo ou algoritmos mais sofisticados, para otimizar a sequência de entregas.
- **Modelagem de Custos:** Expandir o modelo para incluir variáveis de custo monetário, como combustível, pedágios e custo por hora do motorista, permitindo uma otimização financeira direta.

8 CONCLUSÃO

Este projeto demonstrou com sucesso a aplicação de conceitos fundamentais de estruturas de dados e análise de algoritmos para resolver um problema de otimização logística do mundo real. Através da construção de um framework de simulação e da condução de uma análise de desempenho empírica, foi possível validar a teoria e extrair conclusões práticas. A análise de múltiplos cenários de teste, especialmente o de alto volume, confirmou de forma inequívoca que a **utilização de uma Lista de Adjacência para representar o grafo e uma Fila de Prioridade (Heap) para otimizar o algoritmo de Dijkstra constitui a arquitetura mais eficiente e escalável** para o problema proposto. O trabalho não apenas atinge os objetivos da avaliação, mas também evidencia a importância da experimentação e da análise crítica de benchmarks para a engenharia de software de alto desempenho.

9 REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6023: Informação e documentação — Referências — Elaboração**. Rio de Janeiro, 2018.

ROBSON ALMEIDA, Bússola da Informática. **Como Fazer Normas ABNT no word Atualizado I Normas ABNT no Word Passo a Passo ATUALIZADO 2025**. Disponível em: https://www.youtube.com/watch?v=KYJFGHBapwc.

PAPA THALYSON, **Algoritmo Dijkstra em 5 minutos – Exemplo**. YOUTUBE, 16 de out. de 2021. Disponível em: https://www.youtube.com/watch?v=BK-KykxuYfg.

CORMEN, T. H. et al. **Introduction to Algorithms (Algoritmos: Teoria e Prática)**. 3ª ed. Campus, 2012.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische mathematik**, v. 1, n. 1, p. 269-271, 1959.

PYTHON SOFTWARE FOUNDATION. **heapq — Heap queue algorithm**. Disponível em: https://docs.python.org/3/library/heapq.html.

INDUSTRIAL 21. **Algoritmo de Dijkstra (Exemplo Prático)**. Disponível em: https://www.youtube.com/watch?v=IIZOWRwKa_Q&t=5s.

ANEXOS

Repositório do código no GitHub: https://github.com/MatthSMLK/entrega-mais

Vídeo do Pitch de apresentação no Youtube:
https://www.youtube.com/watch?v=8ScuTE8qdGI