Universidade Federal do Amazonas
ICET - INSTITUTO DE CIÊNCIAS
EXATAS E TECNOLOGIA
Engenharia de Software

Felipe Rangel Silvestre
Nome do Integrante 2
Nome do Integrante 3
Nome do Integrante 4
Nome do Integrante 5

Sistema de Trânsito Inteligente

Trabalho Prático de Algoritmos e Estruturas de Dados II

ITI275 - Prof. Alternei Brito

Itacoatiara – AM, 25 de Novembro de 2025

Felipe Rangel Silvestre
Nome do Integrante 2
Nome do Integrante 3
Nome do Integrante 4
Nome do Integrante 5

Sistema de Trânsito Inteligente

Universidade Federal do Amazonas

Itacoatiara – AM
25 de Novembro de 2025

Felipe Rangel Silvestre
Nome do Integrante 2
Nome do Integrante 3
Nome do Integrante 4
Nome do Integrante 5

Sistema de Trânsito Inteligente

Universidade Federal do Amazonas

Itacoatiara – AM
25 de Novembro de 2025

Resumo

Este relatório apresenta o desenvolvimento de um Sistema de Trânsito Inteligente que integra estruturas de dados avançadas para otimização de rotas urbanas. O sistema utiliza um **Grafo Ponderado** para representar a malha viária e uma **Árvore AVL** para gerenciar eventos dinâmicos (como acidentes e obras) que impactam o tráfego. As principais contribuições incluem a implementação de persistência de dados via arquivos JSON, cálculo de rotas alternativas (K-Shortest Paths) e uma interface gráfica interativa. Os resultados demonstram a eficiência da atualização dinâmica de pesos no grafo em tempo real.

Palavras-chave: Árvore AVL. Grafo Ponderado. Dijkstra. JSON. Python.

Sumário

1 Introdução

O gerenciamento de tráfego urbano exige soluções que se adaptem rapidamente a mudanças. Sistemas de GPS modernos não calculam apenas a distância física, mas consideram o tempo de deslocamento afetado por eventos dinâmicos.

Este trabalho propõe um simulador de trânsito que une a teoria de grafos com estruturas de busca balanceadas. O diferencial do projeto reside na capacidade de **salvar e carregar cenários** (persistência) e oferecer **rotas alternativas** ao usuário, superando a especificação básica proposta.

1.1 Objetivos

- Implementar Grafo Ponderado e Árvore AVL sem uso de bibliotecas externas de estruturas de dados.
- Integrar as estruturas: eventos na AVL devem alterar automaticamente os pesos no Grafo.
- Desenvolver funcionalidade de persistência de dados (Salvar/Carregar estado).
- Implementar algoritmo para sugerir rotas alternativas (K-melhores caminhos).
- Prover interface gráfica amigável para simulação.

2 Fundamentação e Decisões de Projeto

Para manter o foco na aplicação prática, resumimos abaixo as escolhas técnicas.

2.1 Grafo Ponderado (A Malha Viária)

Utilizamos um grafo onde os **Vértices** são as interseções e as **Arestas** são as ruas.

- **Representação**: Lista de Adjacência (Dicionário de Dicionários em Python).
- **Peso**: O peso da aresta é o *Tempo de Deslocamento* (minutos), não apenas a distância. Isso permite que uma via curta, mas engarrafada, tenha um peso alto.

2.2 Árvore AVL (Gestão de Eventos)

Para gerenciar acidentes e obras, utilizamos uma AVL. A escolha se deve à garantia de complexidade $O(\log n)$ para inserções, remoções e buscas, essencial para sistemas que exigem resposta rápida. Cada nó da árvore armazena o ID do evento, tipo e o impacto no trânsito.

2.3 Algoritmo de Dijkstra e K-Caminhos

O algoritmo de Dijkstra foi escolhido pela sua eficiência em grafos com pesos não-negativos. Além da rota ótima padrão, implementamos uma variação para encontrar os **K-Melhores Caminhos**, penalizando temporariamente as arestas da rota principal para descobrir alternativas viáveis.

3 Arquitetura e Implementação

O sistema segue uma arquitetura modular, facilitando a manutenção e testes.

3.1 Estrutura de Classes

- `GrafoPonderado`: Responsável pela topologia da rede.
- `ArvoreAVL`: Responsável pelo CRUD de eventos balanceado.
- `SistemaTransitoMelhorado`: O "cérebro"que conecta o Grafo e a AVL. Quando um evento é adicionado na AVL, esta classe calcula o novo peso e atualiza o Grafo.

3.2 Persistência de Dados (Novidade)

Para cumprir o requisito de persistência, implementamos a serialização dos objetos em formato JSON. Isso permite salvar a topologia da malha e os eventos ativos.

Listing 3.1 – Lógica de Persistência Simplificada

```
def salvar_dados(self, nome_arquivo):
    dados = {
        "vertices": list(self.grafo.vertices),
        "vias": [...],  # Lista de arestas e seus pesos
        "eventos": [...] # Lista de eventos da AVL
    }
    with open(nome_arquivo, 'w') as f:
        json.dump(dados, f, indent=4)
```

A função de carregamento (`carregar_dados`) limpa as estruturas atuais e reconstrói o grafo e a árvore a partir do arquivo, garantindo que o estado do sistema seja restaurado fielmente.

3.3 Rotas Alternativas

A função `calcular_k_melhores_rotas` executa o Dijkstra iterativamente. Após encontrar o melhor caminho, ela aumenta temporariamente o peso das arestas utilizadas e roda o Dijkstra novamente para encontrar um "segundo melhor"caminho que evite as vias principais sobrecarregadas.

4 Interface Gráfica e Uso

A interface foi desenvolvida em `Tkinter` e organiza as funcionalidades em abas.

4.1 Funcionalidades Principais

1. **Menu Arquivo**: Permite Salvar e Carregar o estado completo da simulação.
2. **Visualização da Malha**: Canvas interativo que desenha interseções e ruas. Cores indicam o status:
   - **Azul**: Rota Principal.
   - **Verde/Laranja**: Rotas Alternativas.
   - **Vermelho**: Vias com eventos (acidentes/obras).
3. **Painel de Controle**: Permite selecionar origem/destino e injetar eventos no sistema.

5 Testes e Resultados

Os testes validaram tanto a lógica algorítmica quanto a persistência.

5.1 Teste de Persistência

- **Cenário**: Sistema com 100 vértices e 3 eventos ativos.
- **Ação**: Dados salvos em `backup.json`. Sistema reiniciado e dados carregados.
- **Resultado**: Todos os nós, arestas e eventos foram restaurados. O cálculo de rota produziu o mesmo resultado antes e depois do carregamento.

5.2 Teste de Integração Dinâmica

- **Cenário**: Rota A $\rightarrow$ B leva 10 minutos.
- **Ação**: Inserção de "Acidente"na via principal (redução de velocidade para 10km/h).
- **Resultado**: O sistema detectou a mudança de peso e o Dijkstra recalculou automaticamente uma rota alternativa que, embora mais longa em distância, era mais rápida em tempo (evitando o acidente).

6 Conclusão

O projeto atingiu todos os objetivos técnicos e acadêmicos. A implementação manual das estruturas de dados reforçou o entendimento sobre complexidade e manipulação de memória.

Destaques finais:

- A **persistência em JSON** facilitou muito os testes e a demonstração, permitindo criar cenários complexos previamente.
- A funcionalidade de **K-Rotas** adicionou um nível de realismo importante para sistemas de GPS.
- A integração Grafo-AVL provou ser uma arquitetura robusta para problemas de otimização dinâmica.

O sistema encontra-se funcional, documentado e disponível no repositório do grupo.

Referências

CORMEN, T. H. et al. **Introduction to Algorithms**. 3. ed. MIT Press, 2009.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische Mathematik**, 1959.

PYTHON SOFTWARE FOUNDATION. **Python Language Reference**, version 3.10.