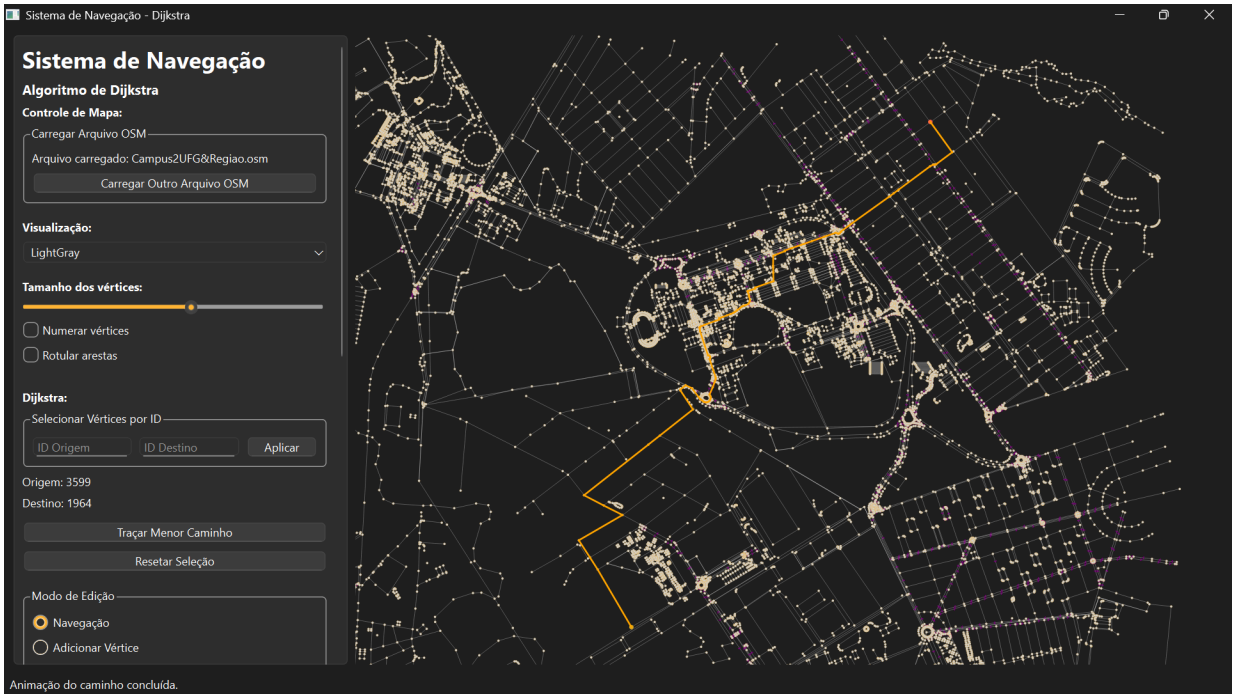
UNIVERSIDADE FEDERAL DE GOIÁS (UFG)
INSTITUTO DE INFORMÁTICA - SEMESTRE SELETIVO 2025/2
ALGORITMOS E ESTRUTURAS DE DADOS 2 - DOCENTE: ANDRÉ LUIZ MOURA

Grupo
ANA LUISA GONÇALVES
GABRIEL RODRIGUES DA SILVA
JÚLIA DE SOUZA NASCIMENTO
LUIS VITTOR FERREIRA NUNES

AnicunsMap - Sistema de Navegação Dijkstra



1. Visão Geral do Projeto

O **AnicunsMap** é um sistema de navegação robusto e intuitivo, desenvolvido para encontrar o menor caminho entre dois pontos em um grafo utilizando o eficiente algoritmo de Dijkstra. Este projeto adota uma arquitetura cliente-servidor simplificada, onde o processamento de dados geográficos e o cálculo de rotas são realizados por um backend em **C**, enquanto a interação do usuário e a visualização gráfica são gerenciadas por um frontend em **Python** com a biblioteca **PySide6**. A integração entre as duas camadas é feita de forma transparente através de ctypes, permitindo que o Python acesse e utilize as funcionalidades da biblioteca C de forma eficiente.

2. Requisitos (Funcionais e Não Funcionais)

Este sistema foi projetado para atender a um conjunto específico de requisitos, garantindo tanto a sua funcionalidade quanto a sua qualidade.

2.1 Requisitos Funcionais (RFs)

- **RF01 - Carregamento de Mapa:** O sistema é capaz de carregar arquivos OpenStreetMap (.osm) identificado dentro da pasta /data e convertê-los internamente para um formato POLY otimizado, construindo assim o grafo correspondente para processamento.
- **RF02 - Visualização do Grafo:** O usuário pode visualizar o grafo carregado na interface gráfica, observando a disposição dos vértices (pontos) e arestas (conexões) que representam as vias.
- **RF03 - Seleção de Origem e Destino:** A interface permite que o usuário selecione visualmente os vértices de origem e destino diretamente no mapa, facilitando a interação.
- **RF04 - Cálculo de Menor Caminho:** O coração do sistema, esta funcionalidade calcula

e exibe o menor caminho entre os pontos de origem e destino selecionados, utilizando o algoritmo de Dijkstra. O caminho encontrado é visualmente destacado no mapa.
- **RF05 - Edição do Grafo:** O sistema oferece modos de edição que permitem a manipulação do grafo, incluindo a adição e remoção de vértices e arestas, proporcionando flexibilidade na construção e modificação de mapas.
- **RF06 - Informações do Caminho:** Após o cálculo, o sistema exibe detalhes sobre o caminho encontrado, como a distância total percorrida, o número de vértices que compõem a rota e a sequência exata de vértices que formam o trajeto.
- **RF07 - Estatísticas de Execução:** Para análise de desempenho, o sistema apresenta estatísticas cruciais sobre o tempo de processamento do algoritmo de Dijkstra e o número de nós (vértices) explorados durante a busca pelo menor caminho.
- **RF08 - Exportação de Imagem:** O usuário tem a capacidade de copiar a imagem atual do mapa para a área de transferência, facilitando o compartilhamento ou a documentação visual.
- **RF09 - Controles de Visualização:** A interface oferece controles para personalizar a visualização do grafo, permitindo ajustar o zoom, selecionar a cor do grafo (cinza, cinza escuro, cinza claro), definir o tamanho dos pontos (vértices) e optar por numerar os vértices ou rotular as arestas para maior clareza.
- **RF10 - Interatividade da Tela:** A tela de visualização é altamente interativa, permitindo que o usuário arraste o mapa com o mouse para navegar por diferentes áreas e utilize o scroll para aplicar zoom in/out, proporcionando uma experiência de usuário fluida.
- **RF11 - Seleção de Vértices Interativa:** Os vértices podem ser selecionados diretamente com o mouse, simplificando a definição dos pontos de origem e destino, bem como a interação nos modos de edição.
- **RF12 - Modos de Edição Detalhados:**
    - **Modo de Navegação:** Permite que o usuário selecione apenas os vértices de origem e destino, e o programa automaticamente desenha o menor caminho.
    - **Adicionar Vértices:** Neste modo, o usuário pode inserir novos vértices no grafo em qualquer posição desejada no mapa.
    - **Adicionar Arestas:** Permite criar novas conexões (arestas) entre dois vértices existentes, ligando-os no grafo.
    - **Remover Vértice:** Oferece a funcionalidade de excluir vértices específicos do grafo, juntamente com todas as arestas a eles conectadas.
    - **Remover Arestas:** Permite remover arestas individuais entre dois vértices, desfazendo conexões específicas.
- **RF13 - Exibição de Origem e Destino:** O sistema exibe claramente os valores (IDs ou coordenadas) dos vértices selecionados como origem e destino, oferecendo feedback direto ao usuário.
- **RF14 - Reset de Seleção:** Um botão de "resetar a seleção" está disponível para limpar as arestas e/ou vértices selecionados, permitindo que o usuário reinicie a operação de busca de caminho ou edição.

2.2 Requisitos Não Funcionais (RNFs)

Os requisitos não funcionais descrevem como o sistema deve operar, focando na qualidade e nas restrições.

- **RNF01 - Usabilidade:** A interface do usuário é projetada para ser intuitiva e fácil de usar, minimizando a curva de aprendizado para novos usuários.

- **RNF02 - Desempenho:** O algoritmo de Dijkstra no backend C é otimizado para ser eficiente, com um tempo de resposta aceitável (ex: menos de 5 segundos para cálculo de rota em um mapa de 10.000 vértices), mesmo para mapas de tamanho médio.
- **RNF03 - Portabilidade:** O sistema é compatível e executável em sistemas operacionais Linux e Windows, garantindo uma ampla base de usuários.
- **RNF04 - Confiabilidade:** O sistema é robusto e capaz de lidar com arquivos de entrada inválidos ou corrompidos, exibindo mensagens de erro claras e informativas para o usuário.
- **RNF05 - Escalabilidade:** O backend em C é projetado para processar grafos maiores com um aumento proporcional, porém controlável, no tempo de processamento, permitindo a utilização com mapas mais extensos.
- **RNF06 - Segurança:** Para este projeto, o requisito de segurança não é aplicável, pois o sistema não manipula dados sensíveis nem oferece acesso remoto.
- **RNF07 - Manutenibilidade:** O código é estruturado de forma modular e bem documentado, facilitando futuras modificações, depurações e a compreensão por outros desenvolvedores.
- **RNF08 - Modularidade e Documentação:** O código é dividido em módulos lógicos, com documentação interna abrangente (comentários e docstrings), promovendo a clareza e a facilidade de manutenção.
- **RNF09 - Extensibilidade:** O design do sistema permite a adição de novos algoritmos de busca de caminho ou funcionalidades de visualização com impacto mínimo no código existente, tornando-o adaptável a futuras expansões.

3. Arquitetura do Sistema

O **AnicunsMap** segue uma arquitetura cliente-servidor simplificada, onde cada componente desempenha um papel específico. O frontend Python atua como o cliente, responsável pela interação com o usuário e pela exibição do mapa, enquanto o backend C funciona como o "servidor" de lógica, contendo os algoritmos de grafo e processamento de dados.

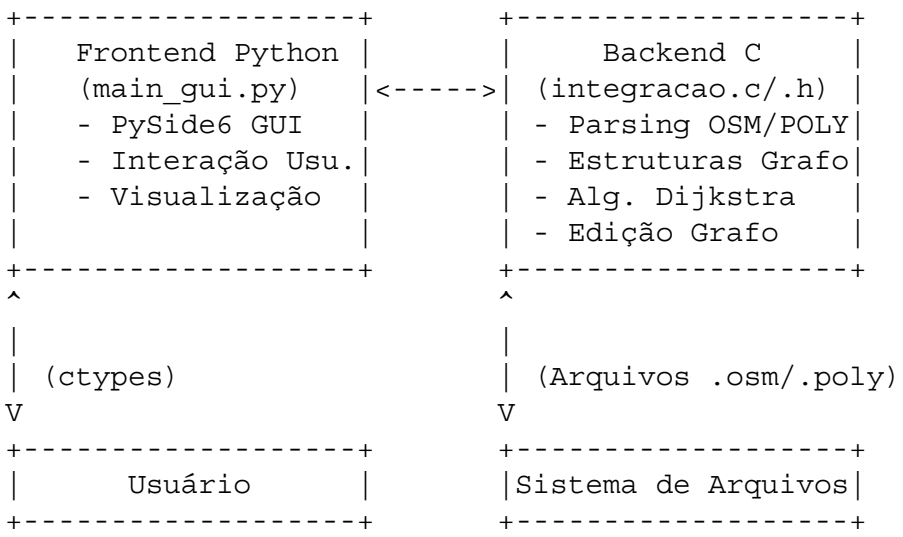
Componentes Principais:

- **Backend C (backend_c/):**
  - integracao.h: Contém as definições das estruturas de dados fundamentais (Nós, Vias, Vértices, Arestas, Heap, etc.) e os protótipos das funções que serão expostas ao frontend.
  - integracao.c: Implementa as funcionalidades principais, incluindo o parsing de arquivos OSM, o carregamento de dados do formato POLY, a construção eficiente do grafo, a implementação do algoritmo de Dijkstra para o menor caminho e as funções para edição do grafo (adição/remoção de vértices/arestas).
  - main_test.c: Um programa C opcional, mas altamente recomendado, para testar as funcionalidades do backend de forma independente da interface gráfica, agilizando o desenvolvimento e a depuração.
- **Frontend Python (frontend_python/):**
  - main_gui.py: É o arquivo principal que implementa a interface gráfica do usuário utilizando a biblioteca PySide6. Ele é responsável por capturar as interações do usuário, renderizar a visualização do mapa e, crucialmente, invocar as funções do backend C através da biblioteca ctypes.
  - assets/: Diretório que armazena recursos adicionais, como arquivos de exemplo ou

outros assets visuais.

- **Scripts de Build/Execução:**
  - build_and_run.sh: Um script shell projetado para sistemas Linux/macOS que automatiza o processo de compilação da biblioteca C e, em seguida, inicia a aplicação Python.
  - build_and_run.bat: O script equivalente para sistemas Windows, que executa as mesmas etapas de compilação e execução.

Diagrama de Componentes:



4. Tecnologias Utilizadas

O desenvolvimento do AnicunsMap emprega uma combinação de tecnologias de código aberto para garantir desempenho, portabilidade e uma interface de usuário responsiva.

- **Linguagens de Programação:**
  - **C:** Utilizada para o backend devido à sua eficiência e controle de baixo nível, ideal para o processamento de grafos e algoritmos de caminho.
  - **Python:** Empregada no frontend para o desenvolvimento rápido da interface gráfica e a facilidade de integração com o backend C via ctypes.
- **Framework GUI:**
  - **PySide6:** Um binding Python para a poderosa biblioteca Qt, que permite a criação de interfaces gráficas de usuário ricas e multiplataforma.
- **Ferramentas de Build:**
  - **GCC (GNU Compiler Collection):** O compilador padrão para o código C no backend.
  - venv (Python): Utilizado para criar ambientes virtuais Python isolados, garantindo que as dependências do projeto não interfiram com outras instalações Python no sistema.
- **Formatos de Dados:**
  - **OpenStreetMap (.osm):** O formato de entrada padrão para dados geográficos, permitindo o uso de mapas reais.

- **POLY:** Um formato proprietário (interno) utilizado para representar o grafo de forma otimizada após o parsing do arquivo OSM.

5. Estrutura de Pastas

A organização do projeto segue uma estrutura lógica para facilitar a navegação, o desenvolvimento e a manutenção.

```
TrabalhoFinalAED2/

├── backend_c/
│   ├── build/                  # Arquivos compilados para teste do backend
│   │   └── main_test
│   ├── lib/                    # Bibliotecas C compiladas para uso no Python
│   │   ├── integracao_lib.dll
│   │   └── integracao_lib.so
│   └── src/                    # Código-fonte do backend em C
│       ├── integracao.c
│       ├── integracao.h
│       └── main_test.c
├── data/                       # Arquivos de mapa .osm
│   ├── Campus2UFG&Regiao.osm
│   └── anicuns.osm
├── frontend_python/            # Código-fonte do frontend em Python
│   ├── assets/
│   ├── venv/                   # Ambiente virtual Python
│   ├── main_gui.py

│   ├── requirements.txt        # Dependências do projeto Python
│   └── style.pss

├── .gitattributes              # Arquivo de configuração de atributos do Git

├── README.md                   # Este arquivo de documentação
├── build_and_run.sh            # Script para compilar e executar em
Linux/macOS
├── compila_dll.bat             # Script para compilar a DLL do backend no
Windows
└── interface.bat               # Script principal para executar a aplicação
no Windows
```

6. Como Compilar e Executar

Graças aos scripts de automação, colocar o AnicunsMap para funcionar é um processo direto. Os scripts cuidam da criação de ambientes virtuais, instalação de dependências, compilação e execução.

6.1. Pré-requisitos

Garanta que os seguintes softwares estejam instalados em seu sistema.

- **Para todos os usuários (Windows, Linux, macOS):**

    - **Python 3.x:** A versão mais recente é recomendada. Baixe em python.org e certifique-se de adicioná-lo ao PATH do sistema durante a instalação.
- **Para usuários de Linux/macOS (ou desenvolvedores Windows que modificarão o C):**
    - **Compilador C (GCC):**
        - **Linux:** Instale o pacote build-essential (em sistemas Debian/Ubuntu) ou Development Tools (em sistemas Fedora/RHEL).
        - **macOS:** Instale as Xcode Command Line Tools com o comando xcode-select --install.
        - **Windows (Opcional):** Para modificar o backend, instale o MinGW-w64.

6.2. Executando o Programa

Com os pré-requisitos atendidos, basta executar um único script. Não é necessário criar ou ativar o ambiente virtual manualmente.

Para Linux/macOS
Execute o script build_and_run.sh a partir da pasta raiz do projeto (TrabalhoFinalAED2/).
`./build_and_run.sh`

Este script irá automaticamente:
1. Verificar se um ambiente virtual existe em frontend_python/venv e, se não, irá criá-lo.
2. Ativar o ambiente virtual.
3. Instalar as dependências listadas no requirements.txt.
4. Compilar o código C do backend para gerar a biblioteca integracao_lib.so.
5. Iniciar a aplicação gráfica.

Nota: Na primeira vez que executar, talvez seja necessário dar permissão de execução ao script com o comando: chmod +x build_and_run.sh.

Para Windows
Execute o arquivo interface.bat com um duplo-clique ou pelo terminal.
`interface.bat`

Este script irá automaticamente:
1. Verificar se um ambiente virtual existe e, se não, irá criá-lo.
2. Ativar o ambiente virtual.
3. Iniciar a aplicação gráfica utilizando a biblioteca integracao_lib.dll pré-compilada.

*Nota para Desenvolvedores Windows: O script principal não recompila o código C. Se você fizer qualquer alteração nos arquivos do backend (.c ou .h), você deve executar o script compila_dll.bat manualmente **antes** de rodar o interface.bat para que suas mudanças tenham efeito.*

7. Autores

Este projeto foi desenvolvido com a colaboração dos seguintes autores:
- Ana Luisa Gonçalves

- Gabriel Rodrigues da SIlva
- Julia de Souza Nascimento
- Luis Vittor Ferreira Nunes