Design of Algorithms
Individual Route Planning Tool

Efficient navigation with customizable routing strategies

Group 2 - Class 15
João Pedro Pinto Lunet - 202207150
Pedro André Freitas Monteiro - 202307242

Project overview

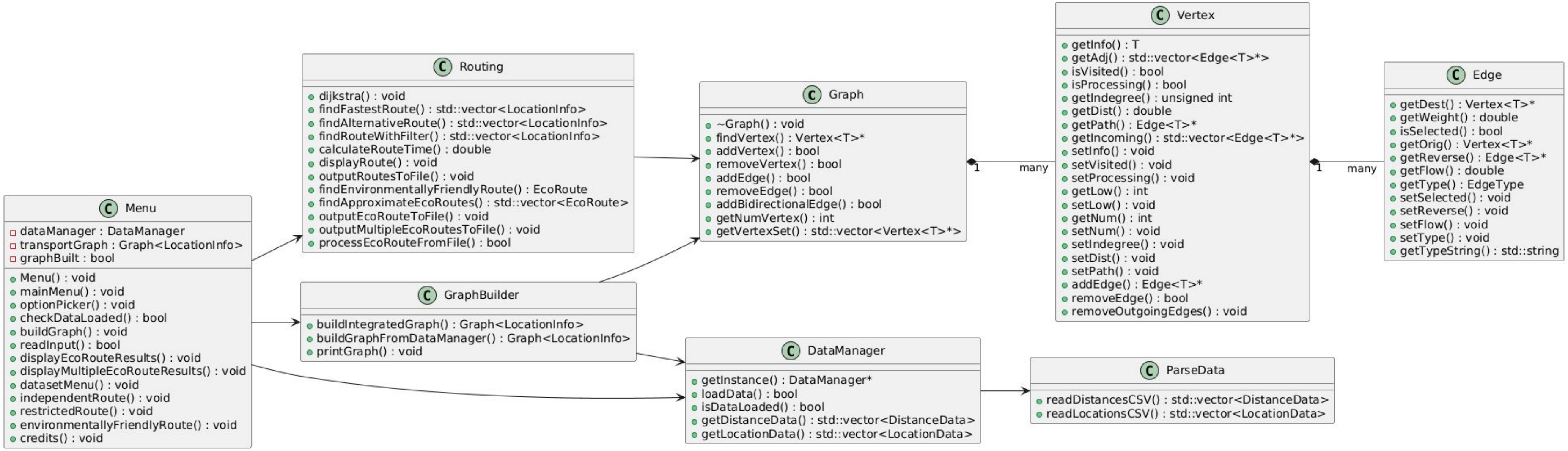
Purpose: Path-planning tool for urban navigation

Core Features:

1. Independent routing (fastest route)
2. Restricted routing (with constraints)
3. Environmentally-friendly routing (combined driving/walking)

We used graph-based greedy algorithms to find the shortest path between two given locations.

Class Diagram



Data Structure

ExampleLocations.csv

	Location	Id	Code	Parking
1	TRINDADE	1	TRND	0
2	CAMPO ALEGRE	2	CA	1
3	BOLHAO	3	BLH	1

ExampleDistances.csv

	Location1	Location2	Driving	Walking
1	TRND	CA	10	20
2	TRND	BLH	X	15
3	CA	BLH	5	8
4	CA	ALDS	8	25

```cpp
struct DistanceData {
    std::string location1;
    std::string location2;
    int driving;
    int walking;
};
```

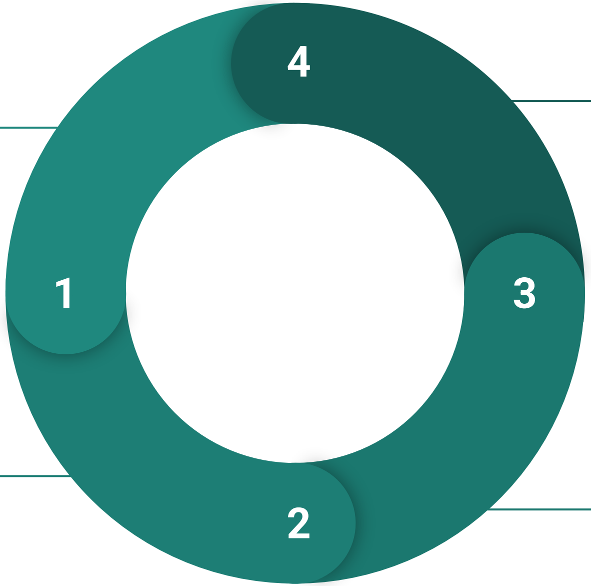
```cpp
struct LocationData {
    std::string location;
    int id;
    std::string code;
    int parking;
};
```

Load and Read

Load CSV files (Locations.csv and Distances.csv) from data directory and read them

Parse

Parse the data into LocationData and DistanceData objects



Convert

Convert data into graph structure via GraphBuild

Store

Store in DataManager singleton for global access

Graph Implementation

```cpp
struct LocationInfo {
      std::string name;        // Location name
      int id;                  // Numeric identifier
      std::string code;        // Text code (primary identifier)
      bool hasParking;         // Parking availability flag
  };
```

Key Modifications:

1. **Extended Edge Class:**
   - Added EdgeType enum: DRIVING, WALKING, DEFAULT
   - Edges now store type information along with weight
   - Example: Edge<LocationInfo> can represent either driving or walking connections
2. **LocationInfo Structure:**
   - Custom data type to store node metadata
   - Custom equality operator based on location code
3. **Enhanced Graph Operations:**
   - Modified Dijkstra's algorithm to filter edges by type
   - Added support for custom edge filtering functions
   - Example: EdgeFilter = std::function<bool(Edge<LocationInfo>*)>

Original Graph Structure:

- Template-based implementation with generic vertices and edges
- Simple weighted edges with no type differentiation
- Basic Vertex<T> and Edge<T> classes

Independent Route Demo

Dijkstra's Algorithm Implementation:

- **Core approach**: Find shortest path by iteratively selecting minimum-distance vertices
- **Key optimizations**:
  - Priority queue for efficient vertex selection (O(log V) per extraction)
  - Early termination when destination is reached
  - Transport mode filtering to only consider relevant edges

Restricted Route Demo

Restriction Types:

1. **Node Restrictions: Completely avoid specific locations**:
   - Example: Construction sites, unsafe areas
   - Implementation: Filter edges leading to avoided nodes
2. **Segment Restrictions: Avoid specific connections between locations**:
   - Example: Closed roads, traffic jams
   - Implementation: Filter edges connecting specific node pairs
3. Node Restrictions. Pass in specific locations:
   - Example: Drop off someone
   - Implementation: Filter edges leading to required nodes

Environmentally-Friendly Route Demo

Problem Definition:

- **Goal:** Find optimal combination of driving + walking to minimize total travel time
- **Constraints:** Maximum walking time limit (user-defined parameter)
- **Challenge:** Identifying the best parking location that balances:
  - Driving time to reach parking location
  - Walking time from parking to destination

Algorithm Approach:

```
For each potential parking node P:
  1. Find fastest driving route: Source → P
  2. Find fastest walking route: P → Destination
  3. Calculate total time = driving_time + walking_time
  4. If walking_time ≤ max_walking_time AND total_time < best_route_time:
     Update best route
```

Fallback Strategy:

- If no routes satisfy the maximum walking time constraint:
  - Offer approximate solutions the exceed the maximum walking time limit
  - Sort by total travel time
  - Present alternative options to user

User Interface

```
Design of Algorithms Project 1 - Spring 2025
Developed by Group 2 - Class 15

  0. Load dataset.

  1. Independent Route. Best (fastest) route between a source and destination.
  2. Restricted Route. Fastest route with specific routing restrictions.
  3. Environmentally-Friendly Route. Best (shortest overall) route for driving and walking.

  4. Exit.

Please select an option:
```

Terminal-based Menu System:

- Main menu with numeric options
- Data loading interface
- Route planning options with constraints

Input Methods:

- Manual (interactive) input
- File-based input (input.txt)

Output Formats:

- Console display with detailed routes
- File output (output.txt)

What made us proud!👏

Segment Avoidance in Restricted Routing:

- Critical feature for real-world applications
- Technical challenge: bidirectional edge handling
- Implementation insight: proper edge filtering

Eco-routing's parking node optimization:

- Balancing driving vs. walking time
- Real-world usefulness

Difficulties and Group Participation

Main Difficulties:

- We did not plan the development of this project well enough, leading to repeated logic that could be done once and reutilized in other places.

Group Participation:

- João Pedro Pinto Lunet - 50%
- Pedro André Freitas Monteiro - 50%