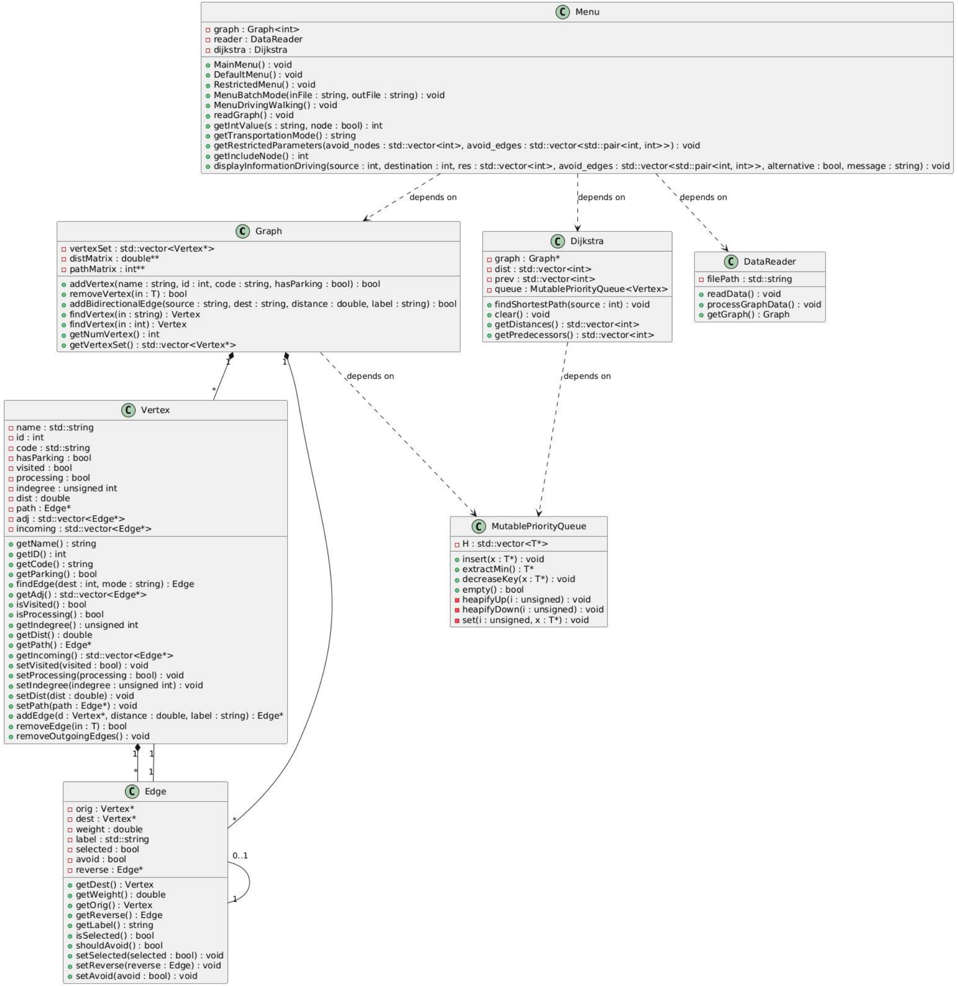
INDIVIDUAL ROUTE PLANNING

Diogo Campeão (up202307177)
Hugo Silva (up202307383)
Inês Francisco (up202304726)

TABLE OF CONTENTS

1. CLASS DIAGRAM



2. DATASET READING

readLocations

- Reads a file containing **location details** (name, ID, code, parking availability)
- Parses the data and **adds vertices**(nodes) to the graph
- Ensures error handling in case of missing or invalid files

readDistances

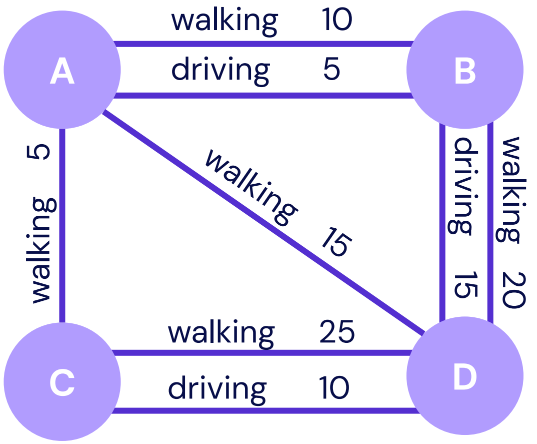
- Reads a file containing **distance data** between locations (nodes)
- Parses driving and walking distances, ensuring **bidirectional edges** in the graph
- Supports conditional parsing (skips missing values marked as “X”)

2. DATASET READING

readInputFile

- Reads a file defining **route planning parameters**:
    - Mode of transport
    - Start and Destination nodes
    - Restrictions (nodes/edges to avoid, maximum walking time, mandatory nodes)
- Ensures input validation and error handling for incorrect values

3. DATA REPRESENTATION

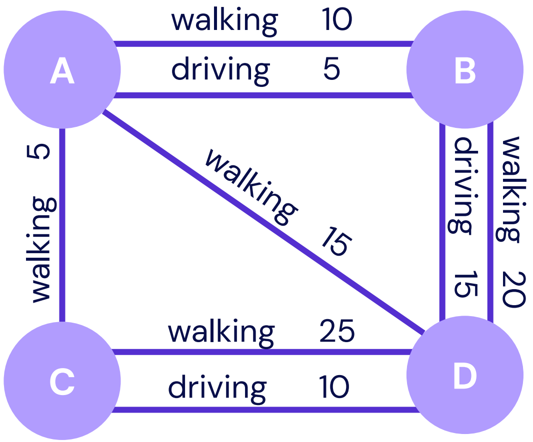


Multigraph Structure

Vertex: represent a place:
- ID, name, code
- bool for parking

Edge: represent connections between places (vertices):
- distance (weight)
- label (type of path – driving / walking
- flag for selection and avoidance

3. DATA REPRESENTATION



Multigraph Structure

Multigraph: undirected and weighted

- Allows for **multiple edges** between the same pair of vertices, representing the different transportation modes
- Easier to track down **driving** and **walking** paths
- Avoids the use of a single edge with two weights

This way, it is **simple** when we are trying to check only walking or only driving paths and **filter edges** by that or when paths are not accessible by both modes of transport

4. IMPLEMENTED ALGORITHMS

Dijkstra’s Algorithm for Shortest Path

Explanation:

- Greedy algorithm that finds the shortest path between the start node and all other nodes
- Maintains a priority queue (min-heap) where nodes with shortest distance are first
- Each node’s distance is updated through RELAXATION when shorter path is found

Steps:

1. **Initialize** all node **distances** to ∞, except the start node (set to 0)
2. **Push** the **start node** into the priority queue
3. **Extract-Min**: Remove the node with the smallest distance
4. **Relax its neighbors**: Update their distances if a shorter path is found
5. Reinsert or **update** affected nodes in **queue**
6. **Repeat** until every node is processed

Time Complexity:

- **Best Case**: O (V logV)
  If there are no edges, priority queue only processes V nodes
- **Average and Worst Case**: O ((V+E) logV)
  Every edge is processed:
  - Most graphs have E ≈ V
  - Fully connected graphs have E = V^2

4. IMPLEMENTED ALGORITHMS

Alternative Routes

Explanation:
The algorithm **marks** the nodes/edges to avoid **as visited** before running Dijkstra's algorithm

Time Complexity:
- **Avoiding Nodes:** O(V)
- **Avoiding Edges:** O(E)
- **Both:** O(V+E)

Since Dijkstra's algorithm execution dominates, the **overall** time complexity is: O((V+E) logV)

Path Reconstruction

Explanation:
Once Dijkstra's Algorithm has run, the shortest path is reconstructed by backtracking from the destination node using the stored edge paths

Steps:
1. Start at the destination node
2. Follow stored paths backward until reaching the source node
3. Reverse the path if needed

Time Complexity:
- **Best Case**: O (1)
  If the destination node is unreachable
- **Average and Worst Case**: O (V)
  The graph is a single long path, so the function traces V steps

4. IMPLEMENTED ALGORITHMS

Driving + Walking Pathfinding

Explanation:

- Finds the best driving path to possible parking locations using Dijkstra's algorithm
- Then finds the best walking path from those to the final destination
- Selects the optimal combination of both paths

Steps:

1. Run Dijkstra with "driving" edges
2. Store all reachable parking nodes and their travel time
3. Run Dijkstra with "walking" edges
4. Find the best parking spot where driving+walking time is minimized
5. If no valid parking spots, retry with no walking time constraint

Time Complexity:

- **Best Case**: O((V+E) logV)
  If there are no parking nodes or parking spot is found early
- **Average Case:** O((V+E) logV + PV)
  Where P is number of parking nodes considered
- **Worst Case:** O((V+E) logV + V^2)
  If every node has parking, the function iterated O(V) times over O(V) nodes

4. IMPLEMENTED ALGORITHMS

Input Handling

Explanation: the system receives input data, which may include:

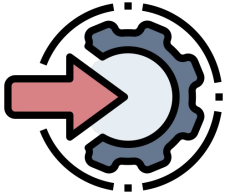
- Start and destination nodes
- Desired mode of transportation (driving / walking)
- Nodes and edges to avoid
- Maximum walking time allowed

Implementation:

- Input is handled by functions that read from files or receive parameters directly
- The graph is loaded into memory and represented using adjacency lists
- User-defined constraints are applied before running the Dijkstra's algorithm

Time Complexity:

- **Graph Reading:** O(V+E)
- **Applying constraints:**
  O(N) for nodes
  O(E) for edges



4. IMPLEMENTED ALGORITHMS

Output Handling

Explanation: after running Dijkstra's algorithm, the system one of following:

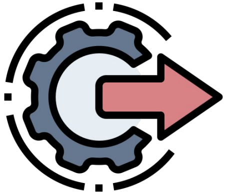
- The shortest path between the requested nodes
- An alternative route if available
- Error messages if no valid path is found
- In the case of the ***bestPathDriveWalk*** function, two paths: one for driving and another for walking

Implementation:

- Paths are reconstructed from the predecessor pointers stored during the algorithms execution
- The results are displayed in the terminal or saved into output files

Time Complexity:

- **Path Reconstruction:** O(V)
- **Printing Output:** O(P), where P is the number of nodes in returned path



5. USER INTERFACE

The interface is designed for clarity and efficiency, ensuring smooth navigation for all users

Purpose: provides users an intuitive way to find the shortest and most efficient paths taking into account their selected options

```
Individual Route Planning Tool
Desenho de Algoritmos 2025
[1] Best Route Mode
[2] Restricted Route Mode
[3] Driving-Walking Mode
[4] Batch Mode
[5] Leave application
```

Key Features:

- Allows users to **choose** which **path** they wanna **plan**:
  - with and without restrictions
  - with both driving and walking parts

- Allows users to further choose all of the **path's specifications**:
  - mode of transportation (in options 1 and 2)
  - source and destination locations
  - which nodes/edges to avoid or nodes to include
  - maximum walking time (in option 3)

5. USER INTERFACE

The interface is designed for clarity and efficiency, ensuring smooth navigation for all users

Purpose: provides users an intuitive way to find the shortest and most efficient paths taking into account their selected options

Use examples:

```
1
Enter mode: driving
Enter Source: 1
Enter Destination: 7
Source:1
Destination:7
BestDrivingRoute:1,2,3,7(35)
AlternativeDrivingRoute:none
```

Option 1

```
2
Enter mode: walking
Enter Source: 1
Enter Destination: 8
Enter Avoiding Nodes:
Enter Avoiding Edges:
Enter Include Node: 5
RestrictedDrivingRoute:1,3,5,6,7,8(67)
```

Option 2

```
3
Enter Source: 1
Enter Destination: 8
Enter Max Walking Time: 20
Enter Avoiding Nodes:
Enter Avoiding Edges:
Source:1
Destination:8
DrivingRoute:1,2,3,7(35)
ParkingNode:7
WalkingRoute:7,8(20)
TotalTime:55
```

Option 3

6. HIGHLIGHTED FUNCTIONALITY

Intelligent Route Calculation

An important functionality of our project is that every algorithm we implemented is not specific to certain user cases which allows it to be **efficient** and **flexible**.

Key Highlights:

- Uses Dijkstra’s algorithm to compute the shortest paths, **supporting both** driving and/or walking path calculation **all in one**
- Allows us to choose to **apply** or not other **constraints** to the path, **without** needing to run more **specific functions**(algorithms)
- In general, everything works independently of the prompt given without needing to implement different functions for each small case

7. PARTICIPATION & MAIN DIFFICULTIES

Diogo Campeão

CLI Menu and Batch Menu
Data Handling
Restricted, Driving-Walking Approximate

Main Difficulty:
Finding solutions to not repeat code

Hugo Silva

Dijkstra implementation
Driving, Restricted, Driving-Walking

Main Difficulty:
Ensuring good time complexity

Inês Francisco

Doxygen documentation, html generation
Class Diagram, code cleaning
Powerpoint presentation

Main Difficulty:
Documenting the code using doxygen

THANKS!

Diogo Campeão (up202307177)
Hugo Silva (up202307383)
Inês Francisco (up202304726)