Projeto 1

Individual Route Planning Tool

Desenho de Algoritmos 2024/2025

Amanda Araujo Silva - up202411286
João Pedro Câmara Furtado - up202305812
Rodrigo

Objetivos

- Implementar de maneira realística algoritmos vistos em classe
- Abordagem de algoritmo guloso (*Greedy algorithmic approach*) para resolução de problemas de menores caminhos (*Shortest-path*)

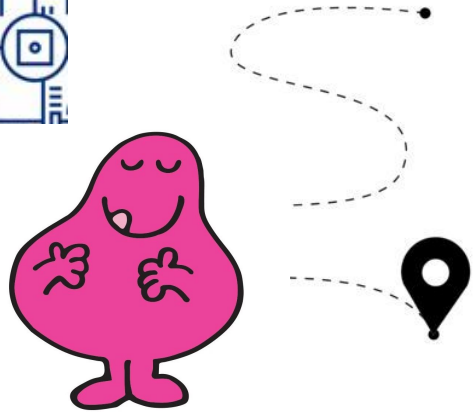


Diagrama de classes

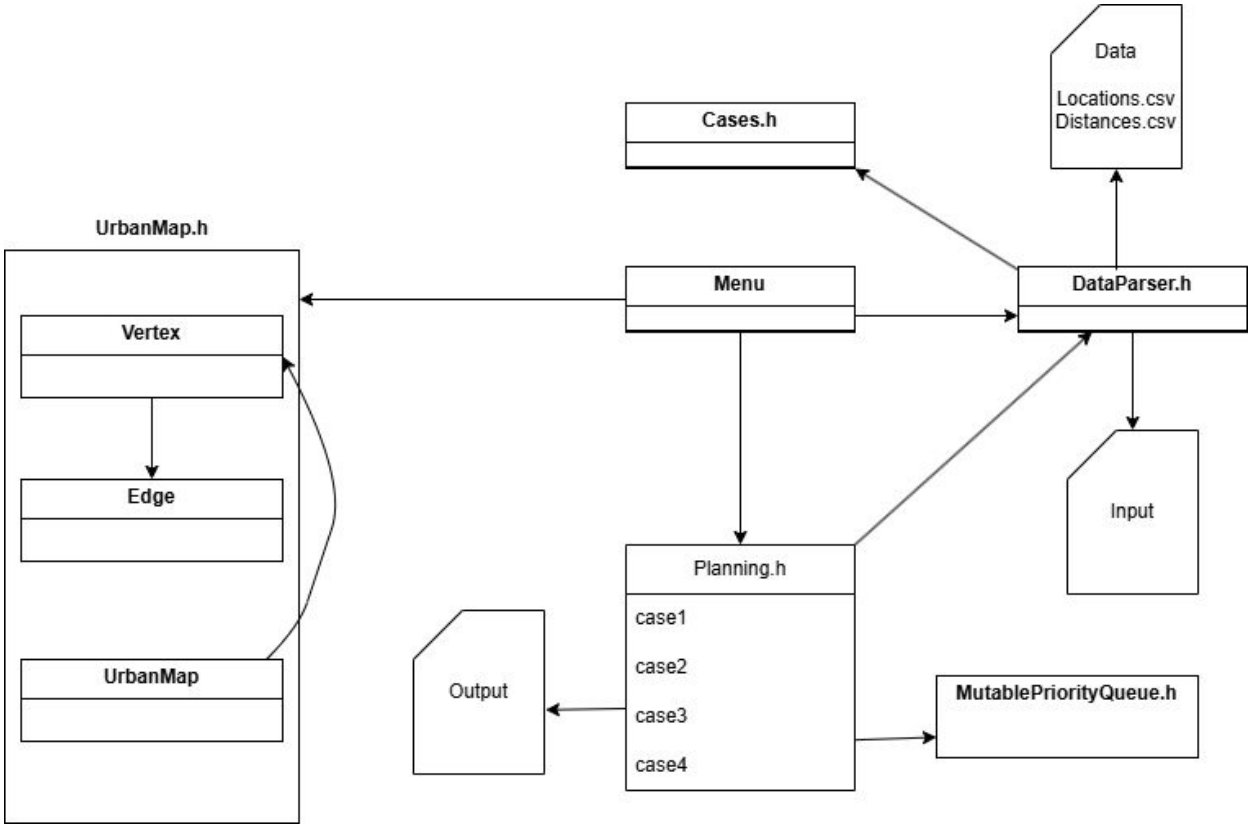


Diagrama de classes

- DataParser.h: contém as funções responsáveis por analisar os ficheiros *csv* e preencher o grafo com os Vértices e as Arestas, com os valores corretos. Para além disso, também têm funções que criam uma estrutura de dados para cada caso ao dar parse no *input.txt*
- UrbanMap.h: representa o mapa urbano como uma estrutura de grafo
  - Vertex
  - Edge
  - Urban Map
- Cases.h: estruturas para armazenar informações dos diferentes *inputs*
  - Case1Data
  - Case2Data
  - Case3Data

Leitura do *dataset*

Leitura dos arquivos de dados *Locations.csv* e *Distances.csv*

- createLocations(UrbanMap<T>* urban_map) >> Locations.csv
  urban_map->addLocation(Code, ID, parking)
- createRoads(UrbanMap<T>* urban_map) >> Distances.csv
  urban_map->addBidirectionalRoad(code1, code2, driving, walking)

1	Location	Id	Code	Parking
2	LIDADOR /HOSPITAL	1	LD3372	1
3	SRA.CAMPANHÃ	2	SR2852	0

Locations.csv

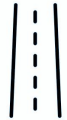
1	Location1	Location2	Driving	Walking
2	LD3372	QTI	3	17
3	LD3372	PR7649	4	24

Distances.csv

Representação em grafos

- UrbanMap: representa o mapa urbano como uma estrutura de grafo
  - Vértices ↔ Locais
  - Arestas ↔ Ruas (grafo bidirecional; duas arestas para cada)
  - Armazenar numa lista todos os nós com *parking*

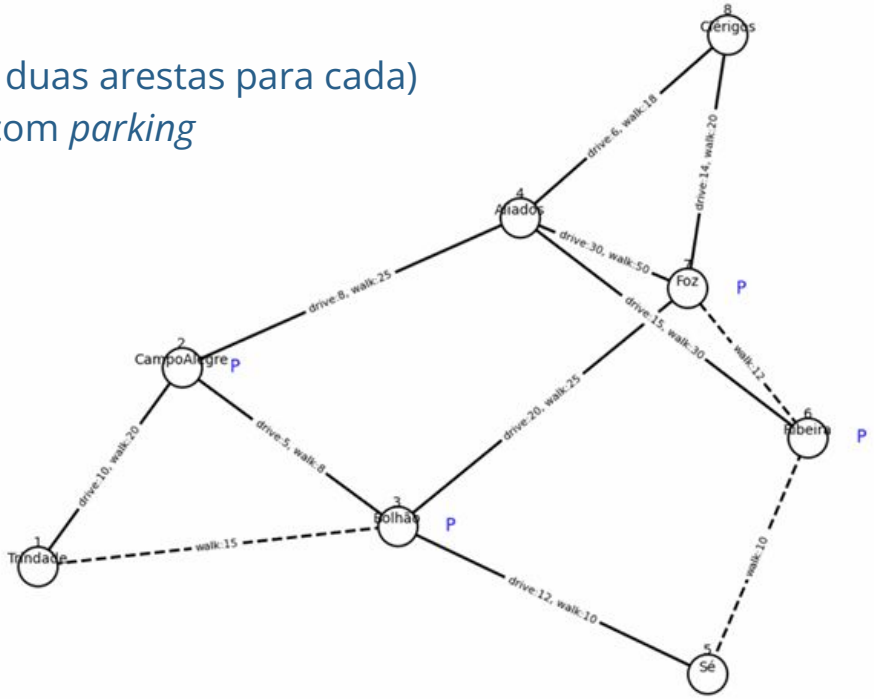
Locais



Ruas

- Code
- ID
- Parking

- Code1
- Code2
- Driving
- Walking



Funcionalidades e Algoritmos

- Identificação da rota independente **Caso 1**
  - Mais rápida
  - Segunda mais rápida
- Planejamento de rotas com restrições **Caso 2**
  - Exclusão de determinados vértices (locais proibidos)
  - Exclusão de determinados segmentos (ruas interditadas)
- Criação de uma rota *eco-friendly* combinando dirigir e caminhar com opções de estacionamento **Caso 3**

Dijkstra	Relax	GetPath	GetPath Economic

Dijkstra

```
68  void dijkstra(UrbanMap<std::string>* g, const int &origin) {
69      for (auto v : g->getLocationSet()) {
70          v->setDist(INT_MAX);
71          v->setPath(nullptr);
72      }
73      auto *s = g->getLocationSet()[origin-1];
74      s->setDist(0);
75      MutablePriorityQueue<Vertex<std::string>> pq;
76      for (auto v : g->getLocationSet()) {
77          pq.insert(v);
78      }
79
80      while (!pq.empty()) {
81          auto u = pq.extractMin();
82          if (u->isVisited())continue;    //if this vertex was visited in a past djkstra will not be included again
83          for (auto e : u->getAdj()) {
84              if (e->isSelected()) continue;  //avoid this segment if it was selected
85
86              if (relax(e,g->DrivingModeEnabled()))
87                  pq.decreaseKey(e->getDest());
88          }
89      }
90  }
```

Implementação do *algoritmo de Dijkstra* para caminhos mais curtos

→ Usa uma *Mutable Priority Queue*

→ É capaz de ignorar específicos nós que estão selecionados, tal como específicas *edges*

$$\mathcal{O}((V+E)log(V))$$

Relax

Função para *relaxação de arestas* para um algoritmo de grafos

→ Atualiza a distância do nó de destino da aresta caso seja encontrado um caminho mais curto

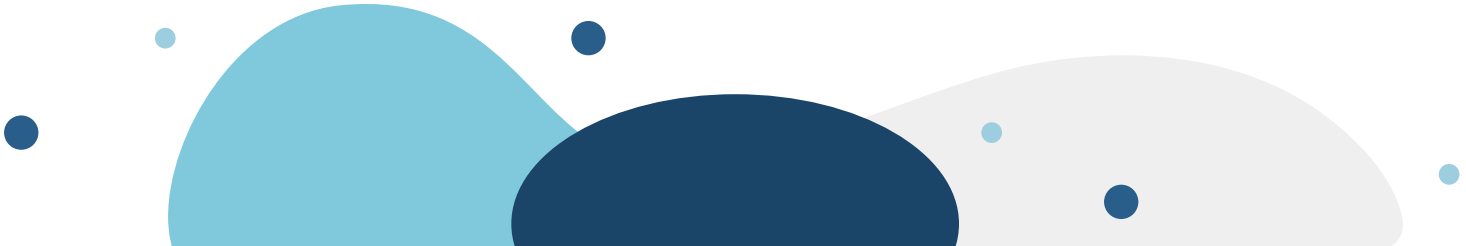
```
32  bool relax(Edge<std::string> *edge, bool Drivemode) {
33      if (Drivemode) {
34          if (edge->getDest()->getDist() > edge->getDriving() + edge->getOrig()->getDist()) {
35              edge->getDest()->setDist(edge->getDriving()+ edge->getOrig()->getDist());
36              edge->getDest()->setPath(edge);
37              return true;
38          }
39      }else {
40          if (edge->getDest()->getDist() > edge->getWalking() + edge->getOrig()->getDist()) {
41              edge->getDest()->setDist(edge->getWalking()+ edge->getOrig()->getDist());
42              edge->getDest()->setPath(edge);
43              return true;
44          }
45      }
46      return false;
47  }
```

Casos

Caso 1: Melhor rota e rota alternativa independente
Uso de Dijsktra duas vezes: o primeiro calcula melhor rota e o segundo executa após rotulação dos nós do primeiro, de modo a não os incluir no Dijkstra

Caso 2: Rota com restrições
Desativação de nós e arestas: booleano *isVisited* e booleano *isSelected*
Inclusão de nós: Dijkstra x 2; nó-fonte >> nó incluído >> nó-destino

Caso 3: Rota *eco-friendly* combinando dirigir e caminhar + estacionamento
Dijkstra nó-fonte até nó no vetor *parkingNodes* + Dijkstra nó de estacionamento até o destino usando distâncias de caminhada, para cada um dos nós de estacionamento escolha do melhor tempo geral

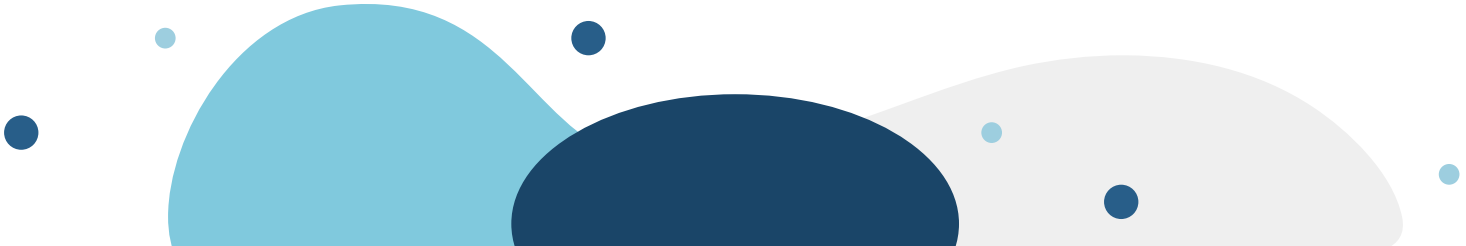


Estimation

Estimation ocorre quando não conseguimos encontrar nenhum caminho para o caso 3.

Em vez de recomputar aquilo que já fizemos mudando apenas o *maxWalkingTime*, o que resultaria numa complexidade absurda, optámos por armazenar aquilo que já tínhamos feito no caso 3.

Usando um vector de structs RouteOption que armazena o nó de *parking* e as respectivas rotas e tempos, para depois apenas dar *sort* do vector e pegar nos dois melhores caminhos e exibi-los.



Complexidade temporal

Caso 1

Rota independente

$$\mathcal{O}((V+E)log(V))$$

Caso 2

Rota restrições

$$\mathcal{O}((V+E)log(V))$$

Caso 3

Rota *eco-friendly*

$$\mathcal{O}(V(V+E)log(V))$$

Estimation

Alternativa Caso 3

$$\mathcal{O}(nlog(n))$$

Interface do usuário (UI)

O nosso UI revolve sobre perguntas ao utilizador e resposta deste mesmo, com a validação dessa resposta: *Menu*

→ É feita uma pergunta principal ao utilizador sobre que tipo de opção ele quer efetuar. Em caso de dúvida, este mesmo pode selecionar uma opção de ajuda, que explica o que cada caso faz.

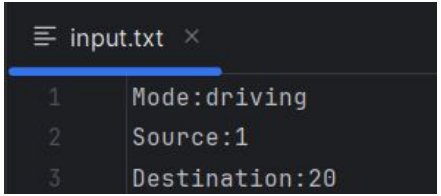
```
11  class Menu {
12      public:
13          Menu();
14          void init();
15          static void end();
16          void readOutputFile();
17          void readEstimationFile();
18          void help();
19
20      private:
21          void chooseOption();
22          void handleOption1(UrbanMap<std::string>* urban_map);
23          void handleOption2(UrbanMap<std::string>* urban_map);
24          void handleOption3(UrbanMap<std::string>* urban_map);
25
26  };
27
28  #endif //MENU_H
```

```
====================================
  Welcome to the Navigation System!
====================================
Loading menu....

 Wich type of route would you like to do today?

Choose an option:
1. Option 1
2. Option 2
3. Option 3
4. Help
0. Exit
Enter your choice:
```

Funcionalidade a destacar

Fizemos uma funcionalidade para demonstrar que é possível fazer pedidos ao utilizador, para assim preencher o ficheiro de *input* sem necessidade do utilizador copiar o ficheiro que já tinha em mente, e assim interagir mais com o programa.

Acabámos por decidir só fazer para o caso 1, porque entendemos que o projeto não requer esta funcionalidade, mas pelo menos é possível mostrar que funciona.




```
====================================
  Welcome to the Navigation System!
====================================
Loading menu....

 Wich type of route would you like to do today?

Choose an option:
1. Option 1
2. Option 2
3. Option 3
4. Help
0. Exit
Enter your choice: 1
Would you like to to set the input yourself
1. Input yourself
0. Ignore
1
Enter source ID: 1
Enter destination ID: 20
Enter the mode that you want the route to be
1.Driving
0.Walking
1
Data written to input.txt
```

Exemplo de uso

Interface *Navigation System - Help Menu* (opções)

```
====================================
  Welcome to the Navigation System!
====================================
Loading menu....

 Wich type of route would you like to do today?

Choose an option:
1. Option 1
2. Option 2
3. Option 3
4. Help
0. Exit
Enter your choice: 4
================ Help Menu =================
Available Commands:
  1. option1 - Get the best two routes from your source to the destination
  2. option2 - Select specific locations and roads that you would like to avoid, and also include a location you would like to pass through
  3. option3 - See the best option to reach your destination in a economic and environment-friendly way by parking your car and then walking to the destination.
  4. exit    - Exit the application
============================================
```

Exemplo de uso

Caso 1 - Melhor rota e rota alternativa independente *(Exemplo 1.)*

input.txt

```
1  Mode:driving
2  Source:3
3  Destination:8
```

Output esperado:

```
Source:3
Destination:8
BestDrivingRoute:3,2,4,8(19)
AlternativeDrivingRoute:3,7,8(34)
```

```
=====================================
  Welcome to the Navigation System!
=====================================
Loading menu....

 Wich type of route would you like to do today?

Choose an option:
1. Option 1
2. Option 2
3. Option 3
4. Help
0. Exit
Enter your choice: 1
Would you like to to set the input yourself
1. Input yourself
0. Ignore
0
Would you like to read the output file?:
1.Yes
0.No
1
Source:3
Destination:8
BestDrivingRoute:3,2,4,8(19)
AlternativeDrivingRoute:3,7,8(34)
```

Exemplo de uso

Caso 2 - Rota restrita excluindo nós e/ou arestas *(Exemplo 5.)*

input.txt

```
1  Mode:driving
2  Source:5
3  Destination:4
4  AvoidNodes:2
5  AvoidSegments:(4,7)
6  IncludeNode:
```

```
Source:5
Destination:4
RestrictedDrivingRoute:5,3,7,8,4(52)
```

```
====================================
  Welcome to the Navigation System!
====================================
Loading menu....

 Wich type of route would you like to do today?

Choose an option:
1. Option 1
2. Option 2
3. Option 3
4. Help
0. Exit
Enter your choice: 2
Please make sure the input file has the correct format
Would you like to read the output file?:
1.Yes
0.No
1
Source:5
Destination:4
RestrictedDrivingRoute:5,3,7,8,4(52)
```

Exemplo de uso

Caso 3 - Rota *eco-friendly* combinando dirigir e caminhar *(Exemplo 7.)*

input.txt

```
Mode:driving-walking
Source:8
Destination:5
MaxWalkTime:18
AvoidNodes:
AvoidSegments:
```

```
Source:8
Destination:5
DrivingRoute:8,4,2,3(19)
ParkingNode:3
WalkingRoute:3,5(10)
TotalTime:29
```

```
====================================
  Welcome to the Navigation System!
====================================
Loading menu....

 Wich type of route would you like to do today?

Choose an option:
1. Option 1
2. Option 2
3. Option 3
4. Help
0. Exit
Enter your choice: 3
Please make sure the input file has the correct format
Would you like to read the output file?:
1.Yes
0.No
1
Source: 8
Destination: 5
DrivingRoute:8,4,2,3(19)
ParkingNode:3
WalkingRoute:3,5(10)
TotalTime:29
```

Exemplo de uso

Caso 3 (Estimation) - Solução aproximada *(Exemplo 8. e 9.)*

```
Message:No possible route with max. walking time of 5 minutes.
```

input.txt

```
1  Mode:driving-walking
2  Source:8
3  Destination:5
4  MaxWalkTime:5
5  AvoidNodes:
6  AvoidSegments:
```

```
/////////////// Estimation File ////////////////////
Source:8
Destination:5
DrivingRoute1:8,4,2,3(19)
ParkingNode1:3
WalkingRoute1:3,5(10)
TotalTime1:29
DrivingRoute2:8,4,6(21)
ParkingNode2:6
WalkingRoute2:6,5(10)
TotalTime2:31
```

```
Source:8
Destination:5
DrivingRoute1:8,4,2,3(19)
ParkingNode1:3
WalkingRoute1:3,5(10)
TotalTime1:29
DrivingRoute2:8,4,6(21)
ParkingNode2:6
WalkingRoute2:6,5(10)
TotalTime2:31
```

Dificuldades e Participação

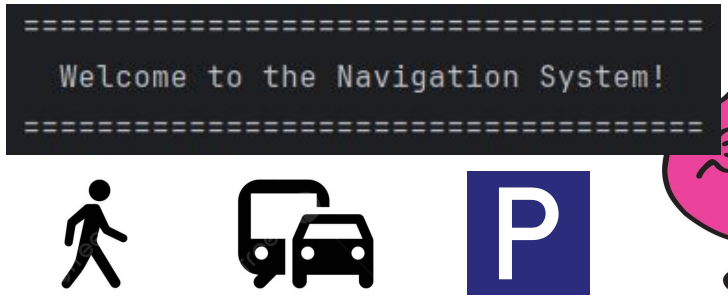
- Este projeto revelou-se ser bastante difícil no tratamento de *corner cases* e de *error cases*, já que tinha que se fazer várias condições para identificar qualquer anomalia presente.

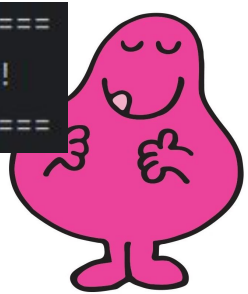
- Quanto à participação dos membros da equipa:
  up202305812 > responsável pela implementação dos algoritmos e do tratamento dos casos do projeto.

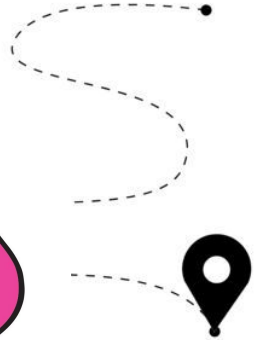
  up202411286 > responsável pelas estruturas para armazenar informações dos diferentes *inputs* e apresentação.

Conclusão

- Implementação bem sucedida de um Sistema de Navegação com interface usuário e documentação organizada pelo Doxygen
- Uso do algoritmo de Dijkstra e variações para resolução de problemas realísticos de menores caminhos







Obrigada!

Porto.