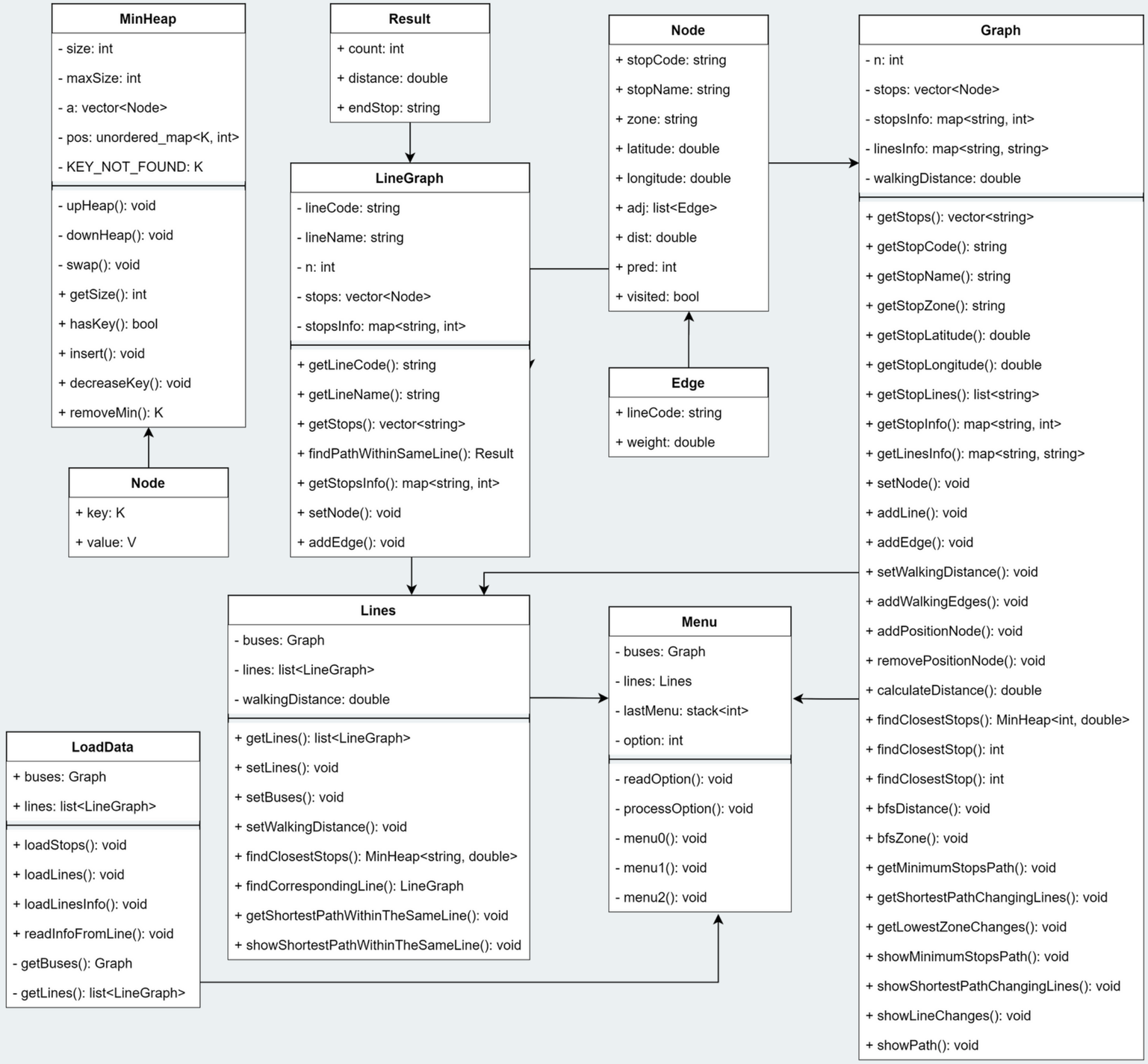
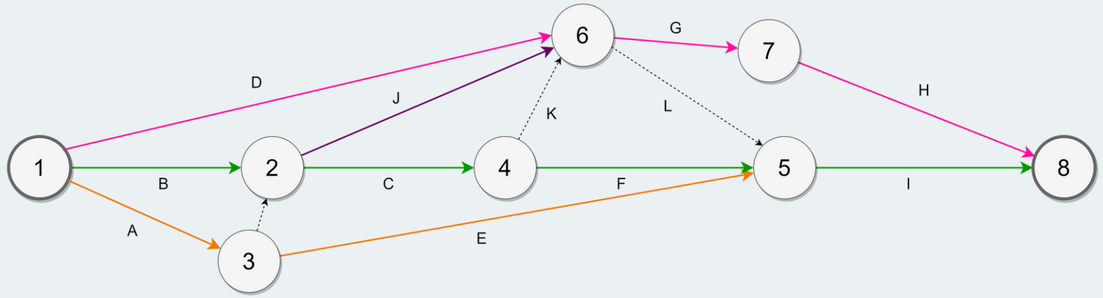
NAVEGAÇÃO NOS TRANSPORTES PÚBLICOS DO PORTO

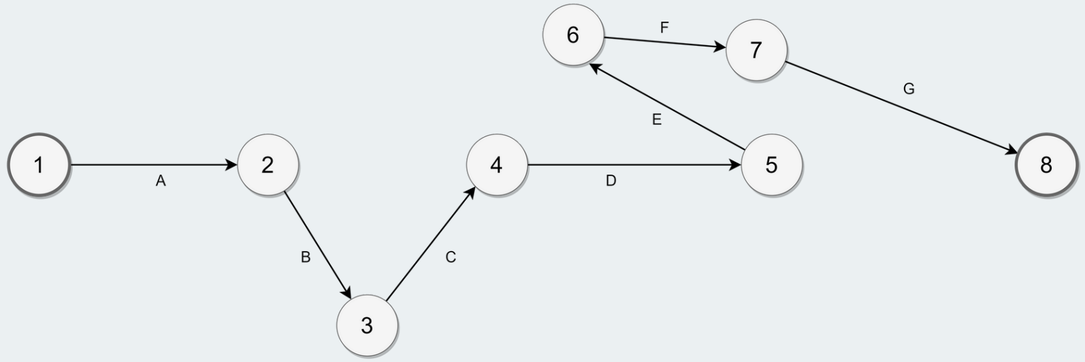
Turma 6 Grupo 60
Isabel Amaral - up202006677
Milena Gouveia - up202008862
Sofia Moura - up201907201

DIAGRAMA DE CLASSES



GRAFOS USADOS PARA REPRESENTAR O DATASET





Os nós representam as paragens de cada linha (cada número corresponde ao inteiro usado para representar internamente cada paragem da classe **Graph**), as letras correspondem às distâncias entre paragens (peso das arestas) e as arestas indicam o caminho que é percorrido.
As arestas a tracejado representam as distâncias percorridas a pé.

Além de termos um grafo (buses) que contém informação sobre todas as paragens e todas as linhas operadas pela STCP, usamos também uma lista de ojetos da classe **LineGraph** (grafos com estrutura semelhante ao primeiro mas que apenas contêm informação sobre as paragens e respetivas ligações de uma linha).
Isto facilitou a implementação de alguns algoritmos.

LEITURA DO DATASET

Para armazenar a informação de cada ficheiro, decidimos implementar uma classe cujo único objetivo é esse: a classe **LoadData.** Esta tem implementados métodos "load" que criam uma variável do tipo ifstream, onde será feita a abertura do ficheiro e lidas as linhas de texto do mesmo até atingir o seu fim.
Métodos:

- **loadStops**: leitura do ficheiro stops.csv e para cada linha desse ficheiro adiciona um nó com as informações lidas ao grafo buses.
- **loadLines**: leitura do ficehiro lines.csv e para cada linha adiciona um par (lineCode, lineName) a um map que guarda informação sobre as linhas no grafo.
- **loadLinesInfo**: leitura dos ficehiros do tipo line_[code]_[dir].csv um por um e criação das edges que ligam paragens consecutivas da mesma linha.

Como cada linha dos ficheiros contém informação relativa a vários parâmetros, criamos variáveis correspondentes a cada um, guardando assim os dados.

FUNCIONALIDADES IMPLEMENTADAS E ALGORITMOS

Origem/Destino:
Optámos por desenvolver os algoritmos de maneira a que o utilizador pudesse inserir uma localização qualquer, independentemente de corresponder ou não à localização de uma paragem. Para isto, utilizamos a fórmula de Haversine, que nos permitiu calcular a distância das coordenadas do input do utilizador para cada paragem que se encontrava a uma distância igual ou inferior à que o utilizador define que está disposto a caminhar (atributo walkingDistance).

Mudança de autocarro (paragens vizinhas):
Como forma de identificar que era necessário efetuar uma mudança de autocarro, após a leitura dos vários ficheiros usados na criação dos grafos, adicionamos algumas edges (com lineCode = "" dado não se tratar de um percurso realizado por um autocarro) referentes aos caminhos a serem percorridos a pé entre as paragens que estivessem a menos de uma certa distância (no caso, 50 metros) entre si.

FUNCIONALIDADES IMPLEMENTADAS E ALGORITMOS

Conceito de melhor caminho:

- caminho que percorre menor número de paragens
  Graph::getMinimumStopsPath()
- caminho mais curto sem mudar de linha
  Lines::getShortestPathWithinSameLine()
- caminho mais curto com mudanças de linha
  Graph::getShortestPathChangingLines()
- caminho mais barato, isto é, mudando de zonas o menor número de vezes
  Graph::getLowestZoneChanges()

FUNCIONALIDADES IMPLEMENTADAS E ALGORITMOS

No início de todos os algoritmos (exceto no getShortestPathWithinSameLine) adicionamos dois novos nós ao grafo (partida e destino) e algumas ligações (edges) com as paragens mais próximas dos mesmos. No final de cada algoritmo, estes novos dados eram sempre removidos.

getMinimumStopsPath()

- Algoritmo: Pesquisa em largura (BFS)
- Implementação: Usou-se o bfs para determinar a que distância (nº de nós) a que cada uma das paragens estava do ponto de partida.
- Complexidade temporal: O(N + E)

E: número total de edges existentes no grafo.
N: Tamanho/número de nós do grafo.

getShortestPathWithinSameLine()

- Algoritmo: Adaptação de pesquisa em largura (BFS)
- Implementação: O bfs foi aplicado a cada LineGraph correspondente a cada linha a ser analisada. Cada iteração do algoritmo corresponde a avançar de uma paragem para a imediatamente seguinte, dado que em cada paragem só há uma ligação a outra paragem na mesma linha
- Complexidade temporal: O(Ns) * O(L) * O(n)
  Ns: nº de paragens próximas à posição inicial
  L: nº de linhas operadas pela STCP
  n: nº de paragens na linha L

FUNCIONALIDADES IMPLEMENTADAS E ALGORITMOS

getShortestPathChangingLines()

- Algoritmo: Dijkstra
- Implementação: determinou-se as paragens mais próximas ao ponto de partida e ao ponto de chegada, acrescentando-se depois dois novos nós ao grafo correspondentes a esses pontos. Posteriormente utilizou-se o algoritmo de Dijkstra para encontrar o caminho mais curto e removeu-se os nós acrescentados ao grafo, assim como as arestas que os ligavam ao grafo.
- Complexidade temporal: O(E*log(N))
  E: número total de edges que o grafo possui.
  N: Tamanho/Número de nós do grafo.

getLowestZoneChanges()

- Algoritmo: Adaptação de pesquisa em largura (BFS)
- Implementação: Para determinar o número de zonas que são atravessadas desde a origem até ao destino, o atributo numZones de cada paragem é incrementado uma unidade sempre que a sua zona for diferente da do seu precedente (pred). Caso contrário, este atributo é igual ao do pred.
- Complexidade temporal: O(N + E)
  E: número total de edges que o grafo possui.
  N: Tamanho/número de nós do grafo.

INTERFACE COM O UTILIZADOR
(COM EXEMPLOS)

- Com o objetivo de expor as funcionalidades que implementamos, criamos uma nova classe Menu.
- Quando se faz *run* do projeto, é mostrado na consola um menu inicial com duas opções: uma para ver as paragens mais próximas às coordenadas inseridas pelo utilizador e outra para mostrar os melhores caminhos, tendo ambas em conta a distância que o utilizador está disposto a percorrer a pé.
- Quando é selecionada a segunda opção, é ainda mostrado um segundo menu, para que o utilizador possa selecionar o melhor caminho de acordo com os parâmetros que prefere.

```
BEM-VINDO AOS SERVICOS STCP
1. Visualizar paragens perto de si
2. Visualizar percursos
0. Sair.

ESCOLHA UMA OPCAO:1
Indique quantos metros esta disposto a andar ate a paragem:
50
Indique a latitude a que se encontra:
41.17883940
Indique a longitude a que se encontra:
-8.598663760
Encontra-se a:
0.00667379 metros de: FACULDADE DE ENGENHARIA (FEUP1)
29.3242 metros de: FACULDADE DE ENGENHARIA (FEUP2)
```

Paragens mais próximas

```
BEM-VINDO AOS SERVICOS STCP
1. Visualizar paragens perto de si
2. Visualizar percursos
0. Sair.

ESCOLHA UMA OPCAO:2
1. Percurso que passa por menos paragens
2. Percurso mais curto sem mudancas de linha
3. Percurso mais curto com mudancas de linha
4. Percurso mais barato (que passa por menos zonas)
0. voltar
ESCOLHA UMA OPCAO:1
 Indique a latitude a que se encontra:
41.17883940
Indique a longitude a que se encontra:
-8.598663760
Indique a latitude do local para onde se deseja dirigir:
41.15384430
Indique a longitude do local para onde se deseja dirigir:
-8.641293870
Indique quantos metros esta disposto a andar ate a paragem:
50
```

Melhor percurso com menos paragens

ESFORÇO E PRINCIPAIS DIFICULDADES

DIFICULDADES

- Planeamento dos algoritmos utilizados para cada opção de percurso e implementá-los com complexidades temporais que não fossem excessivas.

ESFORÇO DOS ELEMENTOS DO GRUPO

- Numa fase inicial, optámos por dividir os métodos da leitura do dataset por cada elemento

e posteriormente, dada a separação existente nas tarefas seguintes, dividimos os métodos que implementam os algoritmos para os diferentes percursos e respetivos métodos que fazem a demonstração destes algoritmos (métodos show) por todos os elemento.