


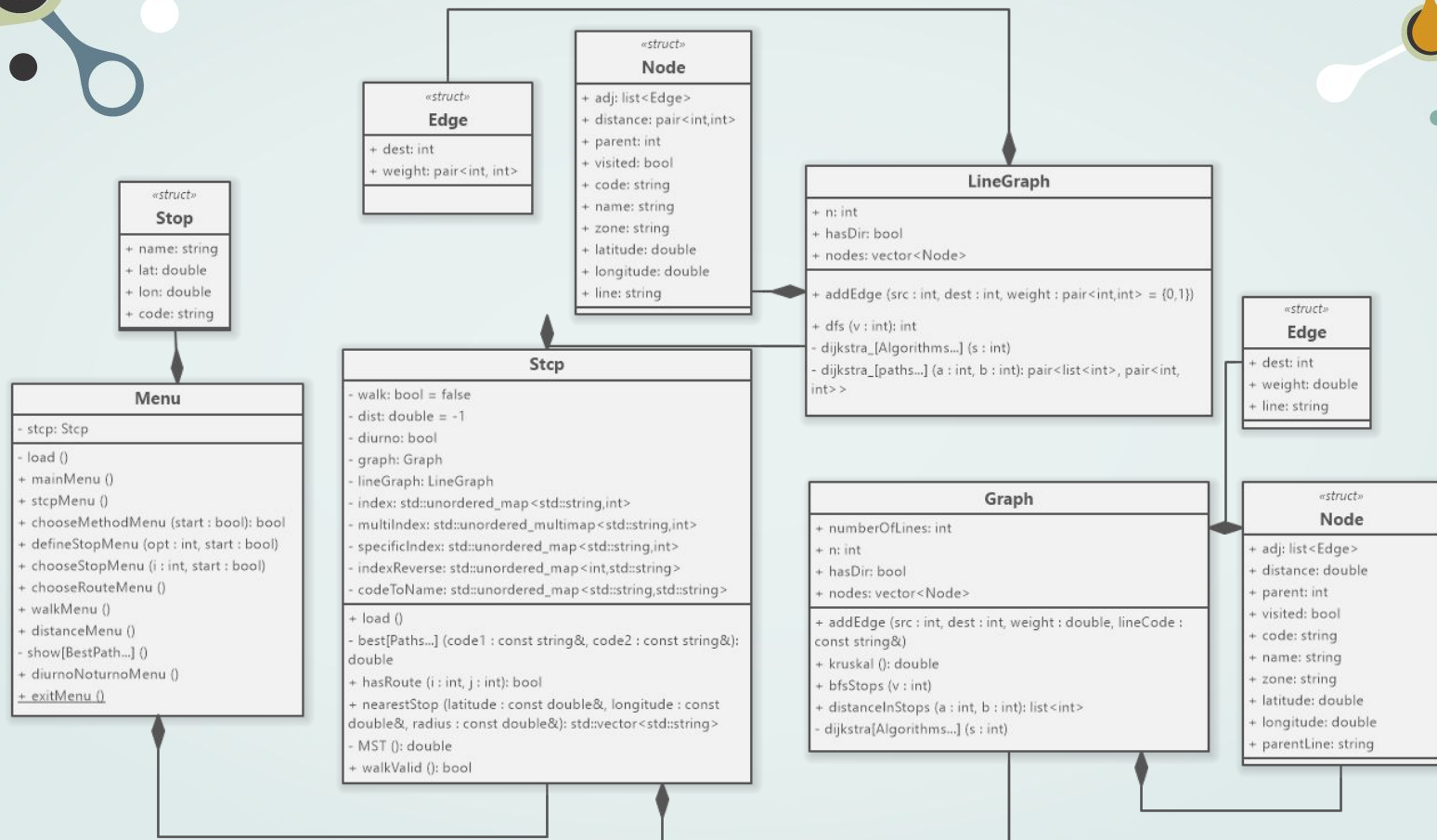
# STCP - Rede de Transportes

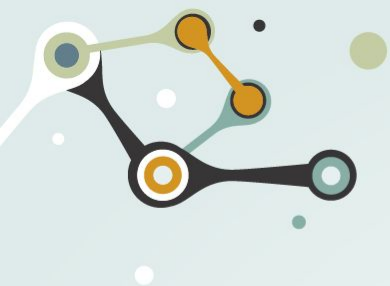
Qual o melhor percurso de autocarro para as necessidades dos passageiros?

## Grupo 84

- Gustavo Costa, turma 9, [up202004187@edu.fe.up.pt](mailto:up202004187@edu.fe.up.pt)
  - João Matos, turma 9, [up202006280@edu.fe.up.pt](mailto:up202006280@edu.fe.up.pt)
  - Ricardo Cavalheiro, turma 16, [up202005103@edu.fe.up.pt](mailto:up202005103@edu.fe.up.pt)
- 

# Diagrama de Classes





# Leitura do Dataset

01

*stops.csv*

As paragens foram lidas e criados nós num grafo para as representar

02

*lines.csv*

Para cada linha "XX" foram lidos os ficheiros *line\_XX.\_0.csv* e *line\_XX.\_1.csv*

03

*line\_XX\_#.csv*

Foram criadas arestas no grafo contendo a linha e a distância entre as consecutivas paragens da linha

Não foram criados ficheiros adicionais



# Grafos Utilizados



## Graph

O grafo principal que guarda sobre as paragens: código, nome, zona e coordenadas. Nos nós podem ser guardados: distância, linha e nó predecessores e se já foi visitado, o que é útil para os algoritmos utilizados.

Nas arestas guarda-se: o nó de destino, o peso que pode ser a distância entre paragens e de linha faz parte a ligação.

```
class Graph {  
    friend class Stop;  
    struct Edge {  
        int dest; // Destination node  
        double weight; // An integer weight  
        string line;  
    };  
  
    struct Node {  
        list<Edge> adj; // The list of outgoing edges (to adjacent nodes)  
        double distance;  
        int parent;  
        bool visited;  
        string code;  
        string name;  
        string zone;  
        double latitude;  
        double longitude;  
        string parentLine;  
    };  
  
    int n; // Graph size (vertices are numbered from 1 to n)  
    bool hasDir; // false: undirect; true: directed  
    vector<Node> nodes; // The list of nodes being represented
```

# Grafos Utilizados

```
class LineGraph {
    friend class Stop;
    struct Edge {
        int dest;    // Destination node
        pair<int, int> weight; // Pair<line change, stop change>
    };

    struct Node {
        list<Edge> adj; // The list of outgoing edges (to adjacent nodes)
        pair<int,int> distance;
        int parent;
        bool visited;

        string code;
        string name;
        string zone;
        double latitude;
        double longitude;
        string line;
    };

    int n;          // Graph size (vertices are numbered from 1 to n)
    bool hasDir;     // false: undirect; true: directed
    vector<Node> nodes; // The list of nodes being represented
};
```

## LineGraph

Um grafo derivado de *Graph* que guarda a mesma informação sobre as paragens, acrescentando a linha à qual pertence, isto é, existe um nó por cada conjunto paragem-linha. A distância passa a ser um par que indica o número de mudanças de linha-mudanças de paragem.

As arestas contêm o nó de destino e uma distância no formato supramencionado que podem ser (0,1) entre duas paragens da mesma linha ou (1,0) entre a mesma paragem em duas linhas diferentes.






# Funcionalidades

- Caminho mais curto -  $O(|E|\log|V|)$

- É mostrado ao utilizador o percurso que percorre menor distância entre as paragens inicial e final.
- A escolha das paragens pode depender da localização escolhida pelo utilizador.

Classes que implementam esta solução:

- `Stcp :: bestDistance`
  - `Graph :: dijkstra_distance2`
  - `Graph :: dijkstra_path`
- 

# Funcionalidades

- Menos paragens percorridas -  $O(|E|\log|V|)$ , hipótese de desempate por menos mudanças de linha -  $O(|E|\log|V|)$

- É mostrado ao utilizador o percurso que percorre menos paragens entre as paragens inicial e final.
- A escolha das paragens pode depender da localização escolhida pelo utilizador.
- Há hipótese de usar o menor número de mudanças de linha como desempate.

Classes que implementam esta solução:

- Stcp :: leastStops
- Graph :: distanceInStops
- Graph :: bfsStops

Classes que implementam esta solução:

- Stcp :: leastStopsWithLeastLines
- LineGraph :: dijkstra\_pathStops
- LineGraph :: dijkstra\_stopsFirst

# Funcionalidades

- Menos mudanças de linha, desempate por menos mudanças de paragem  
-  $O(|E|\log|V|)$

- É mostrado ao utilizador o percurso que muda menos vezes de linha. O desempate é feito pelo menor número de paragens diferentes percorridas.
- A escolha das paragens pode depender da localização escolhida pelo utilizador.

Classes que implementam esta solução:

- `Stcp :: leastLineChanges`
- `LineGraph :: dijkstra_pathLines`
- `LineGraph :: dijkstra_LinesFirst`






# Funcionalidades

- Menor distância até à zona inicial -  $O(|E|\log|V|)$

- É mostrado ao utilizador o percurso mais barato ao utilizador, usando a menor distância total como desempate.
- Tal como com um bilhete Z2 é possível percorrer várias zonas desde que sejam todas adjacentes à zona inicial, também é assim que funciona esta implementação.


Classes que implementam esta solução:

- `Stcp :: leastZoneChanges`
  - `Graph :: dijkstraInZones`
  - `Graph :: dijkstraZones`
- 




# Funcionalidades

- **Início e fim de percurso nas proximidades da localização do utilizador**

- Ao início, é perguntado ao utilizador se quer iniciar o percurso numa paragem em específico ou perto de uma localização em coordenadas. Daqui é possível escolher em que paragem iniciar o percurso.
  - Para escolher a paragem de chegada, o processo é o mesmo.
- 



# Funcionalidades

- Mudar de linha a pé, distância máxima definida pelo utilizador
  - É dada a hipótese ao utilizador de poder viajar a pé entre paragens de autocarro.
  - O utilizador escolhe a distância máxima que pretende andar entre paragens.
- 

# Interface

```
-----  
| Transportes publicos do Porto |  
-----
```

```
1.Planear Viagem  
2.MST da Rede STCP  
0.Sair  
Selecione o pretendido:
```

Menu inicial

```
1.Diurno  
2.Noturno  
0.Sair  
Selecione o tipo de horario no qual pretende viajar|
```

Menu de escolha do horário a usar nas viagens

```
Deseja ver a Minimum spanning tree da rede STCP...  
1.Global  
2.Diurna  
3.Noturna  
4.Back  
0.Sair  
Selecione uma opcao das anteriores:
```

Menu de escolha da MST

# Interface

- 1.Introduzir coordenadas
- 2.Introduzir nome
- 3.Introduzir código
- 4.Voltar
- 0.Sair

Escolha a maneira que pretende definir a paragem de partida:

## Escolha de paragens

Existem varias opcoes de calculo de percurso:

- 1.Menor numero de paragens durante a viagem
- 2.Menor distancia percorrida
- 3.Menor mudanca de autocarros
- 4.Menor preco
- 5.Menor numero de paragens e menor numero de troca de autocarros
- 6.Voltar
- 0.Sair

Selecione a que melhor se adequa a si:

## Maneiras de cálculo do percurso

Pretende que um percurso a pé seja tido em consideracao no calculo do percurso?

- 1.Sim
- 2.Nao
- 0.Sair

ATENCAO - E possivel que caminhar um pouco ate uma dada paragem seja benefico  
Indique uma das opcoes:|

## Escolha se caminhos a pé são válidos e a distância máxima respetivo

A distancia deve ser inferior a 0.400 Km por motivos de complexidade temporal

0.Exit

Introduza a distancia maxima que tolera ser percorrida a pé (recomendado:0.150 km):0.150

Percurso:

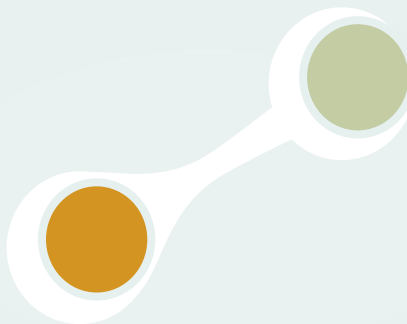
Paragem Inicial:FACULDADE DE ENGENHARIA (FEUP2) Zona:PRT3  
Deve seguir pela linha 204\_0 para a paragem FACULDADE DE ECONOMIA (FEP1) Zona:PRT3  
Deve seguir pela linha 204\_0 para a paragem MANUEL LARANJEIRA (MLAR2) Zona:PRT3  
Deve seguir pela linha 204\_0 para a paragem OUTEIRO (OUTE4) Zona:PRT1  
Deve seguir pela linha 300\_0 para a paragem SALGUEIROS (SAL4) Zona:PRT1  
Deve seguir pela linha 300\_0 para a paragem VITORINO DAMASIO (VDAM) Zona:PRT1  
Deve seguir pela linha 300\_0 para a paragem COSTA CABRAL (CCBR) Zona:PRT1  
Deve seguir pela linha 300\_0 para a paragem SILVA TAPADA (SVT2) Zona:PRT1  
Deve seguir pela linha 300\_0 para a paragem DIOGO CAO (DC1) Zona:PRT1  
Deve seguir pela linha 300\_0 para a paragem DIOGO CAO (DC4) Zona:PRT1  
Deve seguir pela linha 300\_0 para a paragem MONTE AVENTINO (MAV2) Zona:PRT1  
Zonas percorridas: 2

## Apresentação do percurso

# Funcionalidades Destacadas

## Menor mudança de linhas

É usado um grafo em que as arestas contêm como peso um par<mudanças de linha, mudanças de paragem>, facilitando o uso do algoritmo de Dijkstra. A criação do novo grafo a partir do grafo inicial foi a parte mais elaborada porque, entre outros, é preciso ter em atenção os nós terminais das linhas. Como existem vários nós para a mesma paragem (o índice é guardado num *multiIndex* (várias paragens por código) e num *specificIndex* (uma paragem por código + linha)), é necessário calcular todas as hipóteses e apresentar ao utilizador a mais eficiente.



## Escolha de paragens

O utilizador pode escolher paragem através do seu código, nome ou coordenadas, tanto para a paragem inicial como final.

São usados *unordered\_maps* para guardar eficientemente, os atributos das paragens requisitados de modo aceder ao índice usado pelo grafo.

# Principais Dificuldades e Observações

- Organizar o tempo, escolhendo que funcionalidades era possível implementar;
  - Inicialmente, conceptualizar uma forma de minimizar as mudanças de linha (as primeiras tentativas sem criar um segundo grafo foram pouco produtivas);
  - Testar se os caminhos gerados são efetivamente os melhores, tendo em conta os filtros pretendidos.
- 
- ★ O trabalho foi dividido igualmente pelos membros do grupo.
  - ★ Por motivos de compatibilidade, foram retirados os caracteres acentuados e especiais do dataset.

# Tarefas de Valorização

01

Escolher entre Rede  
Diurna ou Noturna

02

## Árvore de Expansão Mínima\*

Permite contar a distância mínima entre todas as paragens e saber um bom percurso se tiver que fazer alterações a todas as paragens. É possível visualizar a árvore de expansão mínima da rede toda ou apenas de um dos horários diurno ou noturno. \*Trata o grafo como se não fosse dirigido.



# OBRIGADO

Algoritmos e Estruturas de Dados  
2021/22

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**

