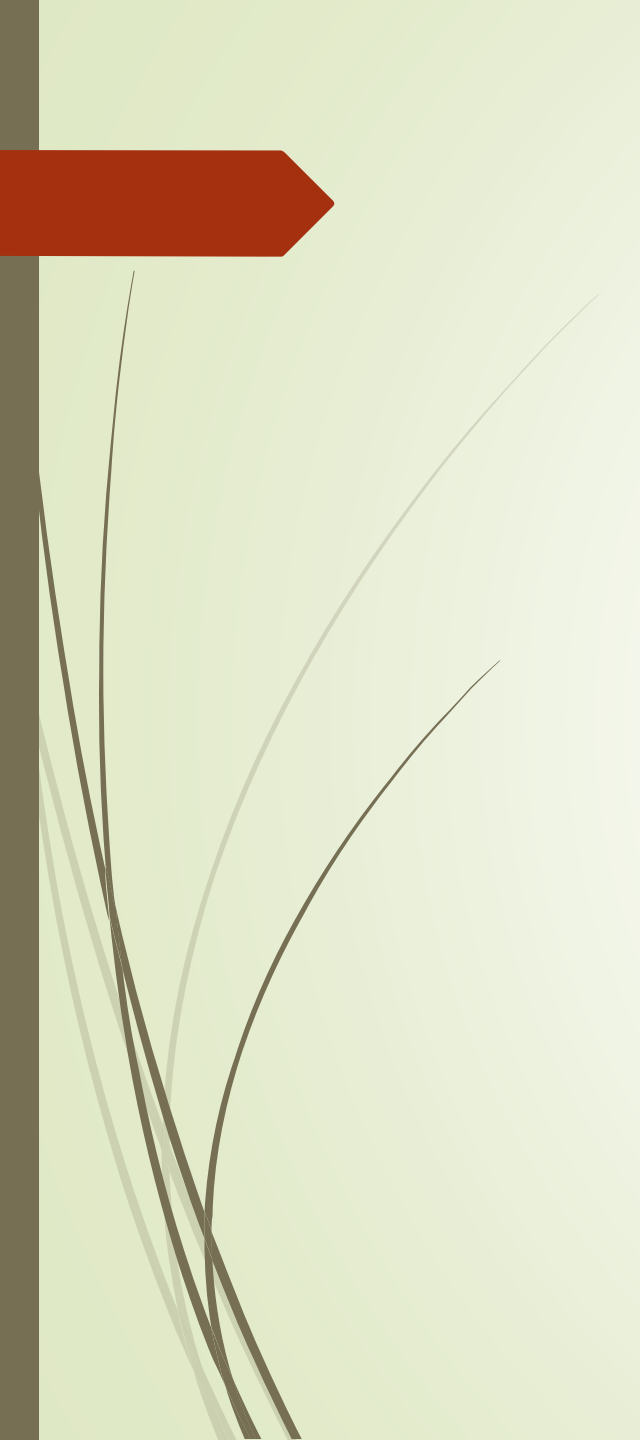


# Routing Algorithm for Ocean Shipping and Urban Deliveries

David Carvalho  
João Torres  
Leonardo Magalhães

up202208654  
up202205576  
up202208726



**“Ser programador é ser paciente e persistente na busca de novos conhecimentos.”**

*Igor Barros*

# Índice

- Introdução
- Diagrama de Classes
- Leitura do Dataset
- Estrutura do Grafo Utilizado
- Funcionalidades e Algoritmos Implementados
- Interface com o Utilizador
- TSP in the Real World
- Destaque de Funcionalidades
- Principais Desafios e Esforço do Grupo
- Conclusão

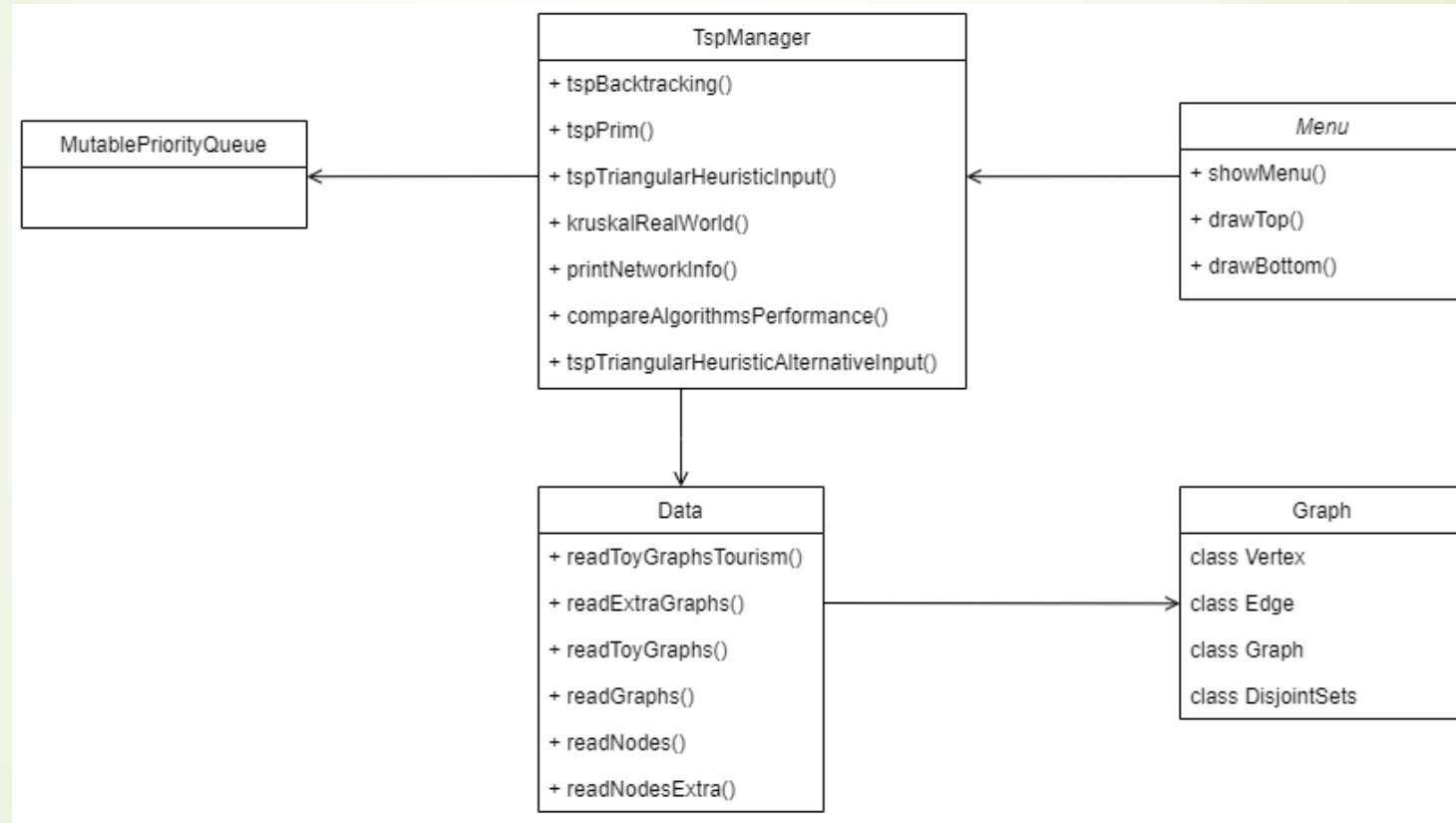


# Introdução



- No contexto da disciplina de Desenho de Algoritmos, empenhámo-nos na criação de um projeto que demonstra o nosso compromisso com a aplicação prática de conceitos avançados nesta área.
- Estamos confiantes de que o nosso projeto corresponderá e atingirá os objetivos propostos.

# Diagrama de Classes



# Leitura do Dataset

## ➤ **readToyGraphsTourism(const string &filename):**

- Lê os dados e cria arestas no grafo representando conexões turísticas entre locais.
- Insere vértices e arestas no grafo. Cada aresta representa a distância entre dois vértices (locais turísticos).
- Define rótulos para cada vértice no grafo, baseados nos nomes de origem e destino.

## ➤ **readExtraGraphs(const string &filename):**

- Lê os dados e cria arestas no grafo.
- Insere arestas no grafo. Cada aresta representa a distância entre dois vértices.

# Leitura do Dataset

## ➤ **readToyGraphs(const string &filename):**

- Lê os dados e cria arestas no grafo.
- Insere vértices e arestas no grafo. Cada aresta representa a distância entre dois vértices.

## ➤ **readGraphs(const string &filename):**

- Lê os dados e cria arestas no grafo.
- Insere arestas no grafo. Cada aresta representa a distância entre dois vértices.



# Leitura do Dataset

## ➤ **readNodes(const string &filename):**

- Lê os dados e cria vértices no grafo.
- Insere vértices no grafo com valores associados a cada vértice.
- Armazena a localização de cada nó em um mapa nodesloc com o ID do nó como chave

## ➤ **readNodesExtra(const string &filename, int limit):**

- Lê os dados e cria vértices no grafo, com um limite máximo de vértices a serem lidos.
- Insere vértices no grafo com valores associados a cada vértice. Armazena a localização de cada nó em um mapa nodesloc com o ID do nó como chave.
- Limita a leitura aos primeiros limit vértices do arquivo.



# Estrutura do Grafo Utilizado

- Vértices:

A propriedade associada a cada vértice é a info.

- Arestas :

As arestas conectam os vértices e têm um peso associado, representando a distância entre vértices. Esse peso pode ser usado para otimizar os Routing Algorithms.

- Este grafo fornece uma estrutura eficiente para modelar e analisar a conexão entre todos estes vértices, permitindo a implementação de algoritmos para Routing Algorithm for Ocean Shipping and Urban Deliveries.



# Funcionalidades e Algoritmos Implementados

```
/**
 * @brief Executes the backtracking algorithm for the TSP problem
 * @details Time complexity:  $O(n!)$ , where  $n$  is the number of vertices in the graph
 */
void tspBacktracking();

/**
 * @brief Executes the Prim's algorithm for the TSP problem
 * @details Time complexity:  $O(E \log V)$ , where  $E$  is the number of edges and  $V$  is the number of vertices in the graph
 * @param incompleteGraph Boolean indicating if the graph is incomplete
 */
void tspPrim(bool incompleteGraph);

/**
 * @brief Prints the network information
 * @details Time complexity:  $O(V)$ , where  $V$  is the number of vertices in the graph
 * @param system String indicating the system type
 */
void printNetworkInfo(const std::string &system);

/**
 * @brief Executes the triangular heuristic algorithm for the TSP problem with user input
 * @details Time complexity:  $O(V^2)$ , where  $V$  is the number of vertices in the graph
 */
void tspTriangularHeuristicInput();
```

# Funcionalidades e Algoritmos Implementados

```
/**
 * @brief Compares the performance of the backtracking, triangular heuristic and Prim's algorithms
 */
void compareAlgorithmsPerformance();

/**
 * @brief Executes the Kruskal's algorithm for the real world problem
 * @details Time complexity:  $O(E \log E)$ , where E is the number of edges in the graph
 */
void kruskalRealWorld();

/**
 * @brief Executes the triangular heuristic algorithm for the TSP problem with alternative user input
 * @details Time complexity:  $O(V^2)$ , where V is the number of vertices in the graph
 */
void tspTriangularHeuristicAlternativeInput();
```

# Interface com o Utilizador

Visualização do Menu de Seleção Inicial

```
Loading ...  
  
-----  
|===== Menu =====|  
|-----|  
| 1. Real World Graphs |  
| 2. Toy-Graphs        |  
| 3. Extra-Fully-Connected Graphs |  
| Q. Exit              |  
|-----|  
|=====|  
|-----|  
Choose an option:|
```

# Interface com o Utilizador

## Visualização dos Menus de Seleção de Datasets

```

-----
|===== Menu =====|
|-----|
| 1. Load shipping.csv (Not fully connected graph) |
| 2. Load stadiums.csv |
| 3. Load tourism.csv |
| Q. Exit |
|-----|
|=====|
|-----|
Choose an option:
  
```

```

-----
|===== Menu =====|
|-----|
| 1. Real-Graph 1 |
| 2. Real-Graph 2 |
| 3. Real-Graph 3 |
| Q. Exit |
|-----|
|=====|
|-----|
Choose an option:
  
```

# Interface com o Utilizador

## Visualização dos Menus de Seleção de Datasets

```
-----  
|===== Menu =====|  
|-----|  
| 1. Load Graph with 25 nodes |  
| 2. Load Graph with 50 nodes |  
| 3. Load Graph with 100 nodes |  
| 4. Load Graph with 200 nodes |  
| 5. Load Graph with 300 nodes |  
| 6. Load Graph with 400 nodes |  
| 7. Load Graph with 500 nodes |  
| 8. Load Graph with 600 nodes |  
| 9. Load Graph with 700 nodes |  
| A. Load Graph with 800 nodes |  
| B. Load Graph with 900 nodes |  
| Q. Exit |  
|-----|  
|=====|  
|-----|  
Choose an option:
```

# Interface com o Utilizador

## Visualização do Menu de Seleção de Algoritmo

```
Loading data...

-----
|===== Menu =====|
|-----|
| 1. Backtracking Algorithm |
| 2. Triangular Heuristic Approximation |
| 3. Prim's Algorithm |
| 4. TSP in the Real World |
| 5. Print Network Details |
| 6. Comparative Analysis |
| 7. Change Dataset |
| Q. Exit |
|-----|
|=====|
|-----|
Choose an option:
```



# TSP in the Real World

- Verificação de Conectividade:

Verificar se o grafo é conectado a partir do vértice inicial fornecido.

- Criar MST:

Construir uma MST do grafo a partir do vértice especificado.

Utilizar o algoritmo do Prim ou Kruskal para criar a MST.

- Percurso em Pré-Ordem da MST:

Realizar um percurso em pré-ordem da MST que começa do vértice dado.

Registrar a ordem na qual os vértices são visitados durante o percurso.

# TSP in the Real World

- Construir Ciclo Hamiltoniano:

Converter a ordem do percurso em pré-ordem num ciclo Hamiltoniano.

Assegurar que o percurso começa e termina no vértice inicial especificado.

- Devolver o Percurso:

Devolver o ciclo Hamiltoniano como a solução.

# Destaque de Funcionalidades

- Sentimos um grande orgulho na eficiente leitura e processamento dos dados, resultando na criação de um grafo dinâmico com informações cruciais.
- Neste projeto, orgulhamo-nos também de ter feito todos os parâmetros pedidos.
- Para além disso, ainda incluímos outras funções que melhoram a qualidade do nosso sistema.

# Principais Desafios e Esforço do Grupo

- Enfrentámos desafios iniciais na interpretação dos dados, que foram superados em conjunto. A implementação precisa do grafo dinâmico exigiu colaboração e a escolha cuidada de algoritmos, destacando a nossa capacidade de superar obstáculos no desenvolvimento deste sistema.
- Todos os elementos contribuíram para os muitos âmbitos deste projeto e o esforço de cada um foi equilibrado e essencial para o resultado desenvolvido e apresentado.

# Conclusão

- No âmbito do projeto proposto, dedicamo-nos totalmente à aplicação de conceitos avançados de desenho de algoritmos e estruturas de dados de forma eficaz e eficiente.
- Empenhámo-nos profundamente na análise de algoritmos, selecionando cuidadosamente as estruturas de dados mais apropriadas para assegurar o desempenho eficaz do projeto.



# Obrigado!