U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Software Gestão de Rede de Tráfego Aéreo

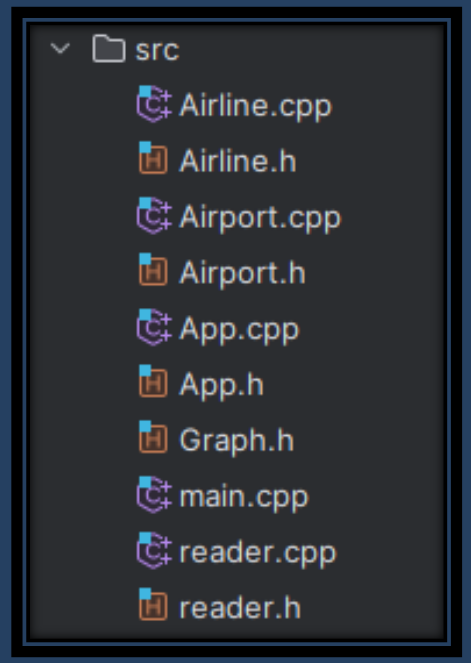
Trabalho realizado por :

Rafael Sousa Cunha 202208957

João Rebelo 202107209

Classes utilizadas :

- Airline
- Airport
- App
- Graph
- Reader



Leitura do Dataset :

- A classe Reader é responsável pela leitura de dados essenciais relacionados com aeroportos, companhias aéreas e voos, extraídos dos ficheiros CSV específicos localizados na pasta de recursos.

- Ao executar as funções readAndParseAirports(), readAndParseAirlines() e readAndParseFlights(), o sistema lê informação detalhada sobre cada um destes elementos que usa na construção de um grafo.

A importância das Hash Tables:

- Os unordered_maps (implementação de uma hash table em C++) desempenham um papel fundamental na eficiência do sistema, pois permitem o acesso em tempo constante O(1) a informações essenciais sobre a rede.
- O airlineMap possibilita uma pesquisa instantânea sobre detalhes de cada companhia aérea através do seu airline code.
- Já o airportMap oferece rápida identificação e acesso a dados de todos os vértices que representam aeroportos através do seu airport code.
- Por fim, o cityToAirportsMap associa eficientemente cidades a um vetor de vértices que representam os aeroportos correspondentes visto que cada cidade pode ter vários aeroportos.

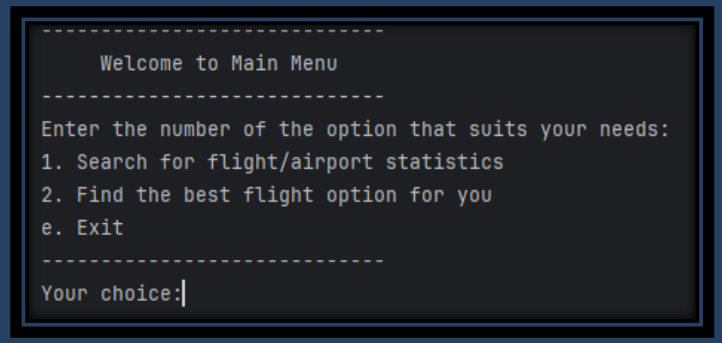
Representação do Dataset :

- Utilizamos uma estrutura de grafo para representar a rede de aeroportos e companhias aéreas. A classe Graph é composta por vértices (Vertex), cada um representando um aeroporto, e arestas (Edge), cada uma representando uma rota de voo entre dois aeroportos. Cada vértice contém informações sobre o aeroporto, uma lista de arestas adjacentes, e variáveis auxiliares que usamos nos algoritmos de grafos. Cada aresta contém um pointer para o vértice de destino e informações sobre a companhia aérea que opera o voo (decidimos alterar o peso de cada aresta para Airline type para facilitar a criação de certos algoritmos).

```
template <class T>
class Vertex {
    T info;
    vector<Edge<T> > adj;
    bool visited;
    bool processing;
    int num;
    int low;

    void addEdge(Vertex<T> *dest, Airline w);
    bool removeEdgeTo(Vertex<T> *d);
public:
    Vertex(T in);
    T getInfo() const;
    bool isVisited() const;
    void setVisited(bool v);
    const vector<Edge<T>> &getAdj() const;
    int getNum() const;
    void setNum(int num);
    int getLow() const;
    void setLow(int low);
    friend class Graph<T>;


};

template <class T>
class Edge {
    Vertex<T> * dest;
    Airline weight;
public:
    Edge(Vertex<T> *d, Airline w);
    Vertex<T> *getDest() const;
    Airline getWeight() const;
    friend class Graph<T>;
    friend class Vertex<T>;
};
```

Interface utilizador :

- O MainMenu inicia com uma operação de carregamento de dados essenciais, através da classe Reader. Após a leitura e análise desses dados, são construídas estruturas como o grafo de conexões entre aeroportos e mapas associativos para facilitar consultas futuras, este grafo é passado como argumento nos menus subsequentes.
- Conforme a escolha do utilizador (por exemplo, consultar estatísticas ou encontrar o melhor percurso de voo), o sistema redireciona para menus específicos.
- Para assegurar a integridade e segurança das operações, o sistema implementa validação de entrada (input). Caso o input não corresponda a uma opção válida, o sistema fornece feedback de forma a evitar comportamentos inesperados. Para este “Error Handling” utilizamos os unordered_maps para pesquisar em O(1) se estes dados realmente existem na nossa rede de tráfego aéreo.



Listagens Estatística :

- Esta parte do projeto concentrou-se essencialmente em fazer listagens de dados e pesquisa no grafo previamente populado. O objetivo principal foi compilar e apresentar informações relevantes relacionadas à rede de Tráfego Aéreo.

- Para isso utilizamos a class Graph e os seus algoritmos de pesquisa como o BFS, DFS e algoritmos baseados nestas duas implementações que facilitam a pesquisa.

Principais Algoritmos : Dijkstra para Encontrar Caminhos Mínimos

- Um dos algoritmos principais utilizados neste projeto é uma adaptação do algoritmo de Dijkstra, com o objetivo de determinar os caminhos mínimos entre um vértice de origem e um vértice de destino para um grafo.
- Durante a execução, ele inicializa distâncias e vértices predecessores, fazendo uso de uma fila de prioridade para explorar os vértices de forma eficiente. Ao processar as arestas, atualiza as distâncias e mantém registros dos predecessores até finalizar a determinação dos caminhos mínimos.
- A função “caminhosDjiskra” é responsável por retornar os caminhos mínimos encontrados, enquanto a função construirCaminhos é crucial para reconstruir os caminhos em ordem reversa, partindo do destino até a origem, utilizando recursão e mapas de predecessores para garantir a precisão dos caminhos identificados.
- A complexidade temporal deste algoritmo é $O((V+E)\log V)$

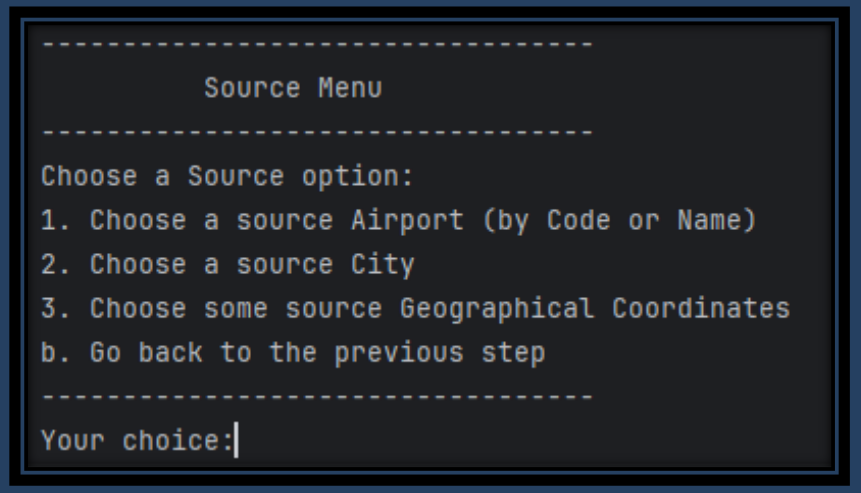
Adaptação do algoritmo anterior : Dijkstra Edges

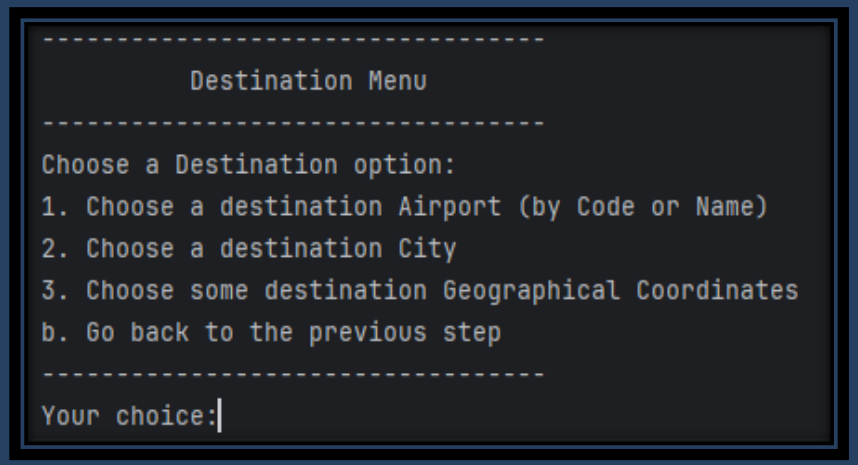
- O algoritmo em questão é uma variante adaptada para filtrar caminhos com base em companhias aéreas. Ao contrário do algoritmo anterior, este requer acesso aos vértices para determinar o peso (ou "Airline") das arestas utilizadas nos caminhos.
- Essencialmente, ele opera de forma semelhante ao algoritmo original, mas foi modificado para estabelecer uma associação direta entre as arestas e os vértices do grafo.
- Ao iniciar a busca a partir de um vértice de origem e avançar para os vértices adjacentes, o algoritmo calcula as distâncias mínimas até todos os outros vértices, considerando a "Airline" como critério.
- Além disso, para cada vértice visitado, ele mantém um registro dos predecessores e suas respectivas arestas, permitindo a reconstrução dos caminhos ótimos entre a origem e o destino.
- A complexidade temporal deste algoritmo é O((V+E)logV)

Menu Source e Destination :

- Cada um destes Menus está relacionado com a pesquisa da melhor opção de voo (ou conjunto de opções equivalentes) para uma origem e destino específicos, isto implica determinar o percurso com o menor número de escalas.

- O utilizador pode inserir os dados sobre a origem do voo e o seu destino de três formas distintas.





Menu Filtros :

- O menu apresentado oferece diversas opções de filtragem para personalizar a pesquisa de voos conforme as preferências do utilizador.

- A opção "No filters" mostra todos os voos disponíveis sem restrições
- "Filter by preferred airlines" permite escolher companhias aéreas específicas.
- A opção "Filter by least number of airline changes" prioriza voos com menos escalas.
- Por fim, "Filter to avoid certain Airlines" permite evitar Airlines que o utilizador deseje evitar no seu caminho.

Destaque de Funcionalidades :

- A implementação do algoritmo de Dijkstra que permitiu encontrar todos os caminhos ideais e a adaptação recursiva que permitiu criar apenas os caminhos para o par source-destination desejado.

- Implementação de filtros que não eliminam quaisquer possibilidades ao utilizador, mas simplesmente maximizam as sua preferências.

Problemas Encontrados :

- Inicialmente, o parsing do dataset não estava otimizado o suficiente, tivemos de encontrar uma solução para este problema.
- A leitura da longitude inicialmente não estava a ser feita de forma correta.
- Conseguir implementar um algoritmo que permitisse encontrar os melhores caminhos dado um par source-destination, e em seguida, adaptá-lo para conter as arestas do grafo.
- Conciliar a realização do projeto com o estudo para esta próxima época de exames.