Aluno: Samuel Rodrigues Viana de Faria

Trabalho de Grafos
Controle de Voos e Rotas Aéreas

1 – Resumo

Este trabalho prático teve como objetivo a implementação de estruturas de dados que representam grafos de voos e rotas aéreas do território americano, os mesmos deveriam compartilhar dos mesmos vértices (aeroportos). Tais estruturas foram populadas (preenchidas) através da leitura de um arquivo, para que o foco do trabalho tivesse uma concentração na implementação dos algoritmos para gerar os relatórios designados.

2 – Funcionamento

2.1 Bibliotecas

java.util.Scanner;
//Utilizada para ler entrada de dados do usuário.

java.io.BufferedReader;
java.io.FileReader;
java.io.IOException;
//Utilizadas para a leitura do arquivo.

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
//Utilizadas para realizar implementação das estruturas de dados dos grafos.

java.time.Duration;
java.time.LocalTime;
java.time.OffsetTime;
java.time.ZoneOffset;
java.time.format.DateTimeFormatter;
//Utilizadas para formatar e gerar duração dos voos.

2.2 Estruturas de Dados

Inicialmente para a leitura do arquivo que continham todas as informações sobre o tráfego aéreo americano, foi-se utilizada a criação de Listas para armazenar cada aeroporto, rota e voo, cada qual na forma de objeto contendo todas suas informações e mais adiante em algumas outras funções utilizei de arrayLists para armazenar outros dados da mesma forma, e até de forma diferente para implementar uma fila.

Já para a implementação dos grafos, tanto o de voos quanto o de rotas, utilizei da estrutura Map do tipo HashMap que possui um índice que representa o aeroporto e armazena uma lista de voos ou de rotas que partem de cada aeroporto especifico, entre outras implementações utilizando a mesma lógica da estrutura HashMap.

3 – Classes e suas funções

- **Classe Principal:**
  1. main: Função principal responsável pela mostragem do menu ao usuário, criação dos objetos que serão utilizados e chamada das funções para o progresso do programa.
  2. imprimeLista: Função básica que imprime qualquer tipo de ArrayList passado por parâmetro.
  3. lerArquivo: Função que recebe como parâmetro o caminho do arquivo a ser lido e as listas que serão preenchidas a partir das informações do mesmo. Nessa função todos as variáveis de cada objeto das listas são setados com os valores do arquivo, e até mesmo algumas delas são calculadas nessa função para facilitar o preenchimento de todas as informações do objeto.

- **Classe Aeroporto**

  Possui as variáveis nome, fuso (fuso horário), cordX (coordenada X), cordY (coordenada Y), que são preenchidas na criação de um objeto do tipo Aeroporto. Também possui algumas funções básicas de um objeto como o toString para a impressão de um Aeroporto em terminal.

- **Classe Rotas**

  Possui as variáveis origem, destino, distância, que são preenchidas na criação de um objeto do tipo Rota. Também possui algumas funções básicas de um objeto como o toString para a impressão de uma Rota em terminal.

- **Classe Voos**

  Possui as variáveis básicas de um voo, como origem, destino, hora de chegada e partida, etc. Todas elas, menos a duração, são preenchidas durante a leitura do arquivo, tal variável é calculada de forma separada utilizando a função gerarDuracaoVoo que recebe como parâmetro o aeroporto de destino e o de chegada do determinado voo, a partir disso é feita uma conversão dos horários do voo com base no fuso horário do aeroporto, a fim de setarmos a duração daquele voo.

- **Classe GrafoRotas**

  Nessa classe é utilizada a estrutura de dados HashMap responsável por guardar todas as rotas a partir de cada aeroporto específico. O grafo é preenchido a partir da função criaGrafo que recebe como parâmetro toda a lista de rotas gerada pela leitura do arquivo.

Nessa função toda a lista de rotas é percorrida, caso o aeroporto origem ou destino (sempre é adicionado a rota da origem e do destino, para que o grafo seja bidirecional) ainda não possuir um índice no Map, o mesmo é criado e as rotas daquele aeroporto são adicionadas naquele índice, mas caso já haja um índice do mesmo, só acontece uma verificação para que não haja duplicação de rotas.

Outra função dessa classe é o imprimeGrafo responsável por percorrer todo o grafo e imprimir cada aeroporto com suas seguintes rotas.

- **Classe GrafoVoos**

  Nessa classe é utilizada a estrutura de dados HashMap responsável por guardar todas os voos a partir de cada aeroporto específico. O grafo é preenchido a partir da função criaGrafo que recebe como parâmetro toda a lista de voos gerada pela leitura do arquivo.

  Nessa função toda a lista de voos é percorrida, caso o aeroporto origem ainda não possuir um índice no Map, o mesmo é criado e as voos daquele aeroporto são adicionados naquele índice. Outra função dessa classe é o imprimeGrafo responsável por percorrer todo o grafo e imprimir cada aeroporto com seus seguintes voos.

- **Classe Relatórios**

  Classe especialmente criada para a implementação dos algoritmos de cada relatório designado.

  1. **relatorio1**: Função responsável pelo primeiro relatório, onde o usuário passa o aeroporto de origem e o de destino, a partir desses dados e o grafo de rotas é gerado um caminho(mínimo) entre os aeroportos. Nessa função implementei o algoritmo de Dijkstra que percorre por todo grafo de rotas a partir do aeroporto origem e me retorna a menor distância entre o aeroporto origem e todos os demais aeroportos, além disso minha função também guarda uma estrutura que armazena os nós anteriores de cada aeroporto, fato que facilita a construção do menor caminho entre o aeroporto origem e o aeroporto destino. Montei o caminho inverso a partir do meu destino e com isso imprimi o caminho entre os aeroportos.

  2. **relatorio2**: Função responsável pelo segundo relatório, onde o usuário passa um aeroporto determinado, a partir disso é printado todos os voos que que partem daquele aeroporto que não possuem escalas. Nessa função apenas percorri a lista de voos presente na posição do aeroporto e fiz a verificação de quais não possuíam nenhuma parada.

  3. **relatorio3_1**: Primeira função responsável pelo relatório 3. Função responsável por determinar o menor caminho entre dois aeroportos a partir do grafo de Voos. Essa função também utiliza o algoritmo de Dijkstra e gera o menor caminho com base na distância entre os aeroportos, tal algoritmo segue a mesma lógica do usado no primeiro relatório.

4. **relatorio3_2**: Segunda função responsável pelo relatório 3. Função responsável por determinar o menor caminho entre dois aeroportos a partir do grafo de Voos. Essa função também utiliza o algoritmo de Dijkstra e gera o menor caminho com base na duração do voo entre os aeroportos, tal algoritmo segue a mesma lógica do usado no primeiro relatório. A única diferença entre essa função e a primeira é o tipo de peso utilizado no algoritmo, fato que pode fazer com que o caminho printado seja diferente.

5. **relatorio4**: Função responsável pelo quarto relatório, onde apenas é passado como parâmetro o grafo de rotas. A partir dessa função é possível determinar quais aeroportos são considerados “ponte”, essa atribuição indica que se algum desses aeroportos fossem retirados do grafo, faria com que alguns aeroportos perdessem uma ligação que permitia que o mesmo se conectasse com todos os aeroportos. Nessa função acontece uma busca por todos os aeroportos, cada um é passado por parâmetro para a função auxiliar (relatorio4_1) juntamente de uma lista de aeroportos percorridos. A função auxiliar utiliza da recursão para verificar quantos aeroportos pode se atingir a partir de cada aeroporto específico, caso o número de aeroportos atingidos for menor que o total de aeroportos indica que aquele aeroporto em específico é importante, porque através dele é possível a conexão entre todos os aeroportos.

6. **relatorio5**: Função responsável pelo quinto relatório, onde o usuário passa um aeroporto específico, e a partir disso juntamente do grafo de rotas é gerado um caminho que parte do aeroporto origem e liga todos aeroportos. Inicialmente implementei o algoritmo de PRIM que gera um grafo mínimo que conecta todos aeroportos. A partir desse grafo gerado foi criado um grafo de adjacência que mostra as rotas mínimas de cada aeroporto. Com base nesse grafo de adjacências e no grafoAGM é possível observar que é possível gerar uma rota que liga todos eles, mas com repetições, fato que indica que não é um circuito hamiltoniano. Por fim utiliza-se a função criaRota para realizar uma busca em profundidade pelo grafoAGM e printar todas as ligações que conectam todos os aeroportos.