Trabalho de Grafos

Controle de Voos e Rotas Aéreas

Aluno: Thomas Santos Pollarini

Resumo:

Este trabalho tem como objetivo a utilização de grafos para auxiliar o controle de voos e rotas aéreas de uma série de aeroportos americanos, para tal foi apresentando um arquivo de texto contendo diversas informações sobre os aeroportos, tais quais, rotas possíveis entre os aeroportos, dados dos voos de cada aeroporto, etc.

Após extrair as informações do arquivo, foram utilizadas diversas estruturas de dados para criação dos grafos, com os aeroportos representando os vértices e os voos/rotas representando as arestas.

Código Fonte:

1. Classes:

- **Aeroporto:** Classe utilizada para criação de objeto Aeroporto, o qual contém as seguintes informações:
  nome: Sigla do aeroporto
  fuso: Fuso horário da localização do aeroporto
  cordX: Coordenada vertical do aeroporto
  cordY: Coordenada horizontal do aeroporto

- **Rotas:** Classe utilizada para criação de objeto Rotas, o qual contém as seguintes informações:
  origem: Aeroporto de origem
  destino: Aeroporto de destino
  distancia: Distância entre aeroportos

- **Voos:** Classe utilizada para criação de objeto Rotas, o qual contém as seguintes informações:

  origem: Aeroporto de origem
  destino: Aeroporto de destino
  num: Número do voo
  hrPartida: Horário de partida do voo
  hrChegada: Horário de chegada do voo
  duracao: Tempo de duração do voo
  qtdParadas: Quantidades de escalas do voo
  distancia: Distância entre aeroportos

- **Principal:** Classe responsável pela leitura do arquivo, controle de dados, criação de variáveis auxiliares e criação dos grafos. Também apresenta menu de opções para os relatórios.

Funções:

1. **main:** Realiza controle do programa, criação de variáveis, chamada de funções, etc.
2. **lerArquivo:** Responsável pela leitura do arquivo, extrai informações linha por linha e atribui valores em listas de aeroportos, voos e rotas.
3. **calculaDistRotas:** Recebe uma lista de rotas, calcula e atribui distância entre aeroportos de cada rota na lista.
4. **calculaDistVoos:** Recebe uma lista de voos, calcula e atribui distância entre aeroportos de cada voo na lista.
5. **calculaDuracao:** Recebe uma lista de voos, calcula e atribui duração de cada voo na lista, levando em conta horário de partida e chegada, e o fuso horário do aeroportos.
6. **criaGrafoRotas:** Recebe uma lista de rotas, cria uma estrutura de dados que armazena listas de rotas separadas pelos aeroportos de origem.
7. **criaGrafoVoos:** Recebe uma lista de voos, cria uma estrutura de dados que armazena listas de voos separadas pelos aeroportos de origem.

8. **imprimeGrafoRotas:** Recebe grafo e imprime suas rotas.
9. **imprimeGrafosVoos:** Recebe grafo e imprime seus voos.

- **AGM:** Classe de objeto estático, usada para armazenar uma Árvore Geradora Mínima baseado no grafo de rotas. Utiliza o algoritmo de Prim para criação da AGM.

Funções:

1. **criaAGM:** Função utilizada para criar árvore, sua tarefa é dar início ao processo de criação da árvore e chamar a função que realiza o algoritmo de Prim. Seleciona um vértice inicial, adicionando-o em uma lista de percorridos e remove todas as ligações que levam a ele no grafo de rotas (grafo auxiliar).
2. **algPrim:** Implementação baseado no algoritmo de Prim, o processo conta com uma lista de percorridos e uma função auxiliar, a partir de um grafo verifica todos os vértices ligados aos nós percorridos e seleciona o com menor peso, adicionando-o na lista de percorridos e removendo as ligações até ele no grafo auxiliar. O algoritmo para após todos vértices serem percorridos.
3. **removeT:** Auxilia no algoritmo de Prim, função responsável pela remoção das ligações.
4. **imprimeAGM:** Imprime ligações da AGM.

- **Relatorio:** Classe responsável pela realização dos relatórios.

Funções:

1. **relatorio_1:** Pede dados necessários para realização do relatório e chama função auxiliar (buscaCaminho).
2. **buscaCaminho:** Recebe aeroporto de origem e de destino. Caminha do aeroporto de destino até o de origem, através de uma busca de profundidade pela AGM, quando encontra origem retorna 0 indicando fim da recursão, a cada retorno imprime a ligação entre os nós.

3. **relatorio_2:** Pede dados necessários para realização do relatório e chama função auxiliar (verificaVoos).

4. **verificaVoos:** Percorre todos os voos do aeroporto indicado e imprime aqueles com quantidade de paradas iguais a zero.

5. **relatorio_3:** Pede dados necessários para realização do relatório e chama função auxiliar (menorCaminho).

6. **menorCaminho:** Função que utiliza o algoritmo de Dijkstra para encontrar a menor rota entre a origem até o destino, utilizando como peso a distância e a duração, nos respectivos modos 1 e 2. Utiliza a função imprimirRota para mostrar o resultado.

7. **imprimirRota:** Imprime menor caminho entre dois aeroportos de acordo com o algoritmo de Dijkstra.

8. **relatorio_4:** Pede dados necessários para realização do relatório e chama função auxiliar para cada aeroporto do grafo (verificaVerticesPontes).

9. **verificaVerticesPontes:** Realiza uma busca em profundidade partindo do aeroporto indicado, não podendo retornar ao mesmo, adicionando todos os aeroportos percorridos em uma lista. Caso, ao final da busca, estejam faltando aeroportos na lista de percorridos conclui-se que o aeroporto inicial é um nó ponte, já que para percorrer por todos aeroportos é necessário passar por ele.

10. **relatorio_5:** Pede dados necessários para realização do relatório e chama função auxiliar (encontraRota).

11. **encontraRota:** Realiza busca em profundidade pela AGM, partindo do aeroporto indicado. A cada recursão imprime ligação utilizada, ao chegar em uma ponta retorna imprimindo o caminho de volta. A função acaba após ter percorrido por todas as rotas.

2.Bibliotecas:

- Bibliotecas relacionadas as estruturas de dados:
  java.util.List
  java.util.ArrayList
  java.util.Map
  java.util.HashMap
  java.util.Set
  java.util.HashSet

- Bibliotecas utilizadas para calcular duração dos voos:
  java.time.Duration
  java.time.LocalTime
  java.time.OffsetTime
  java.time.ZoneOffset
  java.time.format.DateTimeFormatter

- Bibliotecas utilizadas na leitura do arquivo:
  java.io.BufferedReader
  java.io.FileReader
  java.io.IOException

- Bibliotecas utilizadas para entrada de dados do usuário:
  java.util.Scanner