INSTITUTO FEDERAL DE MINAS GERAIS

Campus Formiga
Curso Ciência da Computação - Estrutura de Dados II

Relatório Trabalho Voos e Rotas da malha aérea americana.

Higor Gabriel Lino Silva
Mat.: 0070308

Introdução

O presente trabalho propõe o desenvolvimento de uma aplicação que utiliza da estrutura de dados de grafos para representar e explorar as relações entre aeroportos, rotas e voos da malha aérea americana, através do uso dessas estruturas pode-se realizar diversos questionamentos e resolver certos problemas como por exemplo: mostrar caminhos, determinar caminhos mínimos verificar se a partir de um aeroporto posso chegar a todos os outros (verificar se o grafo é conexo), e entro outros, com intuito de aplicar o conhecimento sobre grafos e seus algoritmos aprendidos durante o semestre.

Detalhes de projeto

Todo o algoritmo foi feito utilizando a linguagem de programação Java, utilizando da orientação a objetos para facilitar a implementação. Também para facilitar a implementação, fiz uma modificação no arquivo transformando ele em json para poder facilitar a leitura do mesmo através de código.

Bibliotecas externas:

Foi utilizado a biblioteca externa “org.json” para ler os dados do arquivo json.

Estruturas adicionais:

Foram utilizadas durante a implementação dos algoritmos as estruturas de lista e de lista encadeada, tais estruturas já estão implementação por padrão no Java, porém como no trabalho o intuito era implementar todas essas estruturas, fiz a implementação das mesmas de forma que ficassem genéricas, aceitando todo tipo de dado.

Estrutura de grafos:

Foi desenvolvido no trabalho dois grafos diferentes, utilizando matriz de adjacência, sendo eles:

- Grafo de rotas:

  Este grafo é um grafo não direcionado, onde está armazenado as rotas diretas entre os aeroportos, suas arestas são valoradas conforme a distância calculada entre o aeroporto fonte e destino.

- Grafo de voos:

  Este grafo é um grafo direcionado, onde está armazenado todos os voos entre aqueles aeroportos, tem múltiplas arestas, sendo cada aresta um voo entre aqueles aeroportos, suas arestas recebem um objeto “schedule” onde está armazenado todos os dados referentes ao voo que foi lido do arquivo json, além de dados adicionais calculados no momento de leitura e de criação da aresta.

Classes:

O trabalho foi separado nas seguintes classes:

- Airport

  Essa classe define o objeto airport, onde é lido do arquivo json os dados dos 23 aeroportos.

- Graph

  Essa classe é uma estrutura de grafo que é herdadada pelas classes “Routes” e “ScheduleGraph” onde implementam realmente cada um dos grafos.

- Routes

  Esta classe é responsável pelo grafo de rotas e algoritmos relacionados a ele, estende de Graph e faz suas próprias implementações.

- Schedule

  Esta classe é responsável por ler os dados do arquivo JSON e guardar os dados de cada voo.

- ScheduleGraph

  Esta classe é responsável pelo grafo de voos, ela precisou ser separada de “Schedule” pois as arestas do grafo guardam uma lista de objetos de Schedule. Ela também estende de Graph.

- Relatorys

  Esta é a classe responsável pelos relatórios que foram pedidos no enunciado do trabalho, é nela que estão implementadas todas as verificações necessárias.

Relatórios:

5.1. Para dois aeroportos pesquisados mostrar o caminho, como uma sequência de aeroportos, com base no grafo das rotas;

Para resolver este problema foi utilizado o algoritmo de Busca em Largura (BFS - Breadth-First Search) para encontrar o caminho mais curto entre dois vértices em um grafo. A função `BFS(int s, int d, Routes routes)` realiza a busca em largura começando pelo vértice de origem e procurando pelo vértice de destino. Se o vértice de destino for encontrado, a função `printPath(int[] parent, int s, int d, Routes routes)` é chamada para imprimir o caminho do vértice de origem ao vértice de destino.

5.2. Mostrar, a partir de um aeroporto definido, quais os voos diretos (sem escalas e/ou conexões) que partem dele e a lista desses destinos.:

Para resolver esse problema, fiz uma função `showDirectFlightsWithoutConnections(ScheduleGraph scheduleGraph, int s)` exibe todos os voos diretos (sem escalas e/ou conexões) que partem de um aeroporto específico. Ela faz isso percorrendo a matriz de adjacência scheduleGraph.adjacencyMatrix que representa o grafo de horários dos voos. Para cada voo que parte do aeroporto especificado e tem duração diferente de zero e sem paradas durante o voo, a função imprime o aeroporto de destino e a duração do voo.

5.3. Dados uma origem e um destino, desenvolver um algoritmo para determinar a viagem com menor custo em termos de: distância total a percorrer e tempo de voo.:

Para resolver esse problema, foi implementado o algoritmo de Dijkstra(dijkstra(ScheduleGraph graph, int source, int destination) e dijkstraKM(ScheduleGraph graph, int source, int destination)) para encontrar o caminho mais curto em um grafo de voos representado pela classe ScheduleGraph. A ideia é calcular o caminho mais curto entre um aeroporto de origem e um destino, levando em consideração tanto a duração do voo quanto a distância em quilômetros, dependendo da função escolhida.

Começo inicializando algumas variáveis para armazenar informações sobre os aeroportos. Em seguida, configuro as distâncias iniciais para todos os aeroportos, exceto o de origem, como infinito. As distâncias e durações para o aeroporto de origem são configuradas como zero.

O algoritmo então realiza iterações para visitar todos os aeroportos. A cada iteração, escolhe-se o aeroporto com a menor distância (ou distância em quilômetros) entre os não visitados, marcando-o como visitado. As distâncias e durações dos aeroportos vizinhos são atualizadas se um caminho mais curto for encontrado.

Ao final, utilizo as informações das menores distâncias e os vértices anteriores no caminho mais curto para imprimir informações sobre o menor tempo de voo ou menor distância em quilômetros, bem como o caminho percorrido. A função auxiliar minDistance(int[] distances, boolean[] visited) ajuda a encontrar o índice do aeroporto com a menor distância não visitada.

5.4. Desenvolver um algoritmo para determinar se é possível, a partir de um aeroporto qualquer atingir qualquer outro (ou se será necessário em alguns casos fazer troca de aeroporto). Se for possível, quais os aeroportos que, se ficassem fora de serviço (apenas um de cada vez), impediriam essa situação.:

Para resolver esse problema, foi implementado 4 funções para tal, uma função isConnected(Routes routes, int startAirportIndex) que verifica se o grafo é conexo, se todos os aeroportos estão conectados, ela utiliza uma função dfs auxiliar para isso, além dela, também foi implementada a função getCriticalAirports(Routes routes, int startAirportIndex, que verifica os aeroportos críticos, ou seja, que se deixarem de funcionar irá tornar o grafo desconectado, a função faz isso percorrendo todos os aeroportos e realizando uma busca em profundidade (DFS) auxiliar que ignora o aeroporto atual. Se algum aeroporto não for visitado durante a busca, o aeroporto atual é adicionado à lista de aeroportos críticos.

5.5. Partindo de um aeroporto selecionado definir uma rota que consiga passar por todos os aeroportos e retornar até ele. Essa rota pode ser classificada como um circuito Hamiltoniano?:

Para resolver este problema, foi implementada uma função findRoute(Routes routes, String startAirport) que encontra um caminho que a partir de um aeroporto passa por TODOS os outros aeroportos e volta para ele mesmo, foi feito isso utilizando uma função DFS no grafo de rotas, utilizando da recursão da função DFS foi imprimido na tela o caminho completo, logo após isso, ela verifica se o caminho encontrado é um caminho hamiltoniano (que passa por todos os vértices sem

repetir nenhum), essa verificação acontece da seguinte forma: durante o DFS é guardado o caminho em um vetor de visitados, após o fim do DFS, é criado um vetor de tamanho 23 (quantidade de aeroportos) e em um loop, o algoritmo percorre o vetor de visitados gerado pelo DFS e vai somando em 1 no vetor de aeroportos, por exemplo, quando o aeroporto “ABQ” aparecer em visitados, ele no vetor de aeroportos irá ser somado em 1, caso no fim em algum aeroporto estiver um valor maior que 1 então o aeroporto repetiu, logo, não é um caminho hamiltoniano.

Conclusão

Em conclusão, a criação de dois grafos distintos, um para representar as rotas entre os aeroportos e outro para os voos, utilizando matrizes de adjacência, permitiu a aplicação de algoritmos específicos para resolver diferentes problemas. A análise de conectividade, determinação de caminhos mínimos e identificação de aeroportos críticos demonstraram a versatilidade da aplicação no tratamento de diversas questões relacionadas à malha aérea. Além disso, os relatórios gerados, abordando desde a visualização de caminhos entre dois aeroportos até a determinação de rotas que passam por todos os aeroportos e a verificação de circuitos Hamiltonianos, atendem às demandas propostas no enunciado do trabalho.

Dessa forma, o trabalho não apenas consolidou o entendimento teórico sobre grafos e algoritmos, mas também promoveu a aplicação prática desses conhecimentos em um contexto real, no qual a resolução de problemas relacionados à malha aérea americana foi abordada de maneira sistemática e eficaz. O desenvolvimento da aplicação representa um passo significativo na integração entre teoria e prática, proporcionando uma experiência enriquecedora para o aprendizado.

Referências

Feofiloff, Paulo. Algoritmos para Grafos - DFS. Disponível em: https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/dfs.html

Feofiloff, Paulo. Algoritmos para Grafos - BFS. Disponível em: https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/bfs.html

StudyTonight. Dijkstra's Algorithm in Java. Disponível em: https://www.studytonight.com/java-examples/dijkstras-algorithm-in-java

Wayan. How do I read JSON file using JSON-java (org.json) library? Disponível em: https://kodejava.org/how-do-i-read-json-file-using-json-java-org-json-library/