Trabalho Prático 2: Ligação entre aeroportos

Ricardo Avelar

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) Belo Horizonte – MG – Brazil
ricardodiasmode@hotmail.com

1. Introdução

Esta documentação lida com o problema de desenvolver um script com a funcionalidade de realizar a distribuição de rotas entre aeroportos de cidades diferentes, haja vista que durante a pandemia várias companhias aéreas proibiram certas rotas. O problema gira em torno de fazer essa distribuição de forma mínima, ou seja, com o mínimo de rotas possíveis, mas fazendo com que todas as cidades sejam interligadas.

Essa documentação tem como objetivo facilitar o entendimento do script construído com a finalidade de completar essa tarefa. Aqui serão explicados todos os conceitos sobre as funções do script e como foram tomadas as decisões para se resolver os problemas necessários.

Para resolver o problema citado, foi seguida uma abordagem de criação de rotas entre componentes fortemente conectados de um grafo, tratando cada aeroporto como um nó e cada rota como uma aresta.

A seção 2 trata de uma visão sobre a implementação deste problema, tratando as decisões para a resolução dos problemas. Já na seção 3 é apresentado um sucinto passo-a-passo de como se executar o programa. A seção 4 faz a análise de complexidade do script e a seção 5 é a conclusão e visão final do projeto.

2. Implementação

O código desenvolvido para solução desse problema está organizado em:

Main.cpp: Aqui ocorre a inicialização do programa. Em um ‘for’, são lidas as informações necessárias para seu funcionamento. Depois da leitura das informações criamos cada SCC existente no grafo e em seguida criamos as arestas que os ligam. Depois disso é impresso o resultado no terminal.

Graph.h: Essa é a classe que representa o grafo. Aqui temos variáveis que representam cada característica e funcionalidade do grafo, sendo válido descrever duas em especial, que são definidas como ‘int* InDegree;’ e ‘int* OutDegree;’. Essas variáveis nos darão a resposta, que pode ser resumida como max(|A|,|B|), onde A é o número de vértices com InDegree = 0 e B é o número de vértices com OutDegree = 0.

Graph.cpp: Contém todos os métodos da classe Graph, que são utilizados no algoritmo que nos dá a resposta. Abaixo há uma breve descrição daqueles métodos não tão intuitivos.

AdicionarSCC: Esse método recebe como parâmetro um node ‘newNode’ e o vetor de SCCs ‘SCCs’. Ele nada mais faz do que criar um novo SCC no vetor, com o node ‘newNode’ dentro.

AdicionarNodeNaSCC: Esse método verifica se é possível colocar o node ‘newNode’ dentro da última posição do vetor ‘SCCs’, e se for possível a operação é realizada. O método recebe ambos como parâmetros.

AdicionarArestaNaSCC: Esse método recebe como parâmetros duas variáveis, que representam duas SCC's. Primeiro verificamos se elas são diferentes, e se forem nós adicionamos um grau de entrada e um grau de saída nos vetores InDegree e OutDegree, respectivamente.

DefinirArestasDaSCC: Após verificar os parâmetros corretos e inicializar os vetores de grau, esse método chama o 'AdicionarArestaNaSCC' com os parâmetros verificados.

MaxDegree: Finalmente, esse é o método que nos retorna a resposta que foi resumida como max(|A|,|B|) acima.

Configuração utilizada para testar o programa:

- Sistema Operacional Windows 10;
- Linguagens C++;
- Compilador Mingw;
- Processador Ryzen 5 3600x e 16Gb RAM;

3. Instruções de compilação e execução

- Acesse o diretório do projeto (2019054960_RicardoDiasAvelar);
- Utilizando um terminal, execute o arquivo [Makefile] utilizando o seguinte comando: < make >;
- Com esse comando, devem ser criados os arquivos .o na pasta obj e o arquivo .exe na pasta bin;
- Proceda Utilizando o terminal para acessar o diretório bin, posicione os arquivos entrada.txt em qualquer pasta que queira utilizar e execute o arquivo [tp02.exe] utilizando o seguinte comando: < "tp02 < caminho/para/entrada.txt"> ;
- Dessa forma será impresso no prompt o relatório do processo;

4. Análise de complexidade

No main.cpp temos um 'for' que enquanto houver linhas no arquivo a serem lidas ocorre o loop. Esse 'for' tem complexidade de tempo O(n), com n sendo o número de rotas. A complexidade de espaço no main.cpp é O(v+v^2+m), com 'v' sendo o número de vértices, e 'm' o número de SCCs, pois é o espaço necessário para alocar as variáveis do Grafo e o vetor de SCCs.

void DefinirNumeroDeSCCs - complexidade de tempo: essa função é baseada no algoritmo de Kosaraju, e tem a mesma complexidade de O(n*m), com 'n' sendo o número de vértices e 'm' o número de arestas.

void DefinirNumeroDeSCCs - complexidade de espaço: essa função realiza todas as operações alocando apenas um vetor 'VerticesVisitados' de tamanho O(n), com 'n' sendo o número de vértices.

void DefinirArestasDaSCC - complexidade de tempo: essa função realiza operações constantes, em tempo O(1). Além disso, há um 'for' que itera de 'k=0' a 'n', dentro de um 'for' que itera em todos os nós da SCC em questão, dentro de outro 'for' que itera em todas as SCC's. Esses três 'for' podem ser resumidos pela complexidade Θ(n^2), sendo 'n' o número de vértices, pois os dois externos passam do primeiro nó até o último.

void DefinirArestasDaSCC - complexidade de espaço: essa função realiza todas as operações considerando estruturas auxiliares unitárias O(1) e não aloca nenhum array local, portanto sua complexidade de espaço é O(1). Note que os vetores alocados dentro da função ficam dentro da classe, por isso não são contabilizados aqui, e sim na função 'main()'.

int MaxDegree - complexidade de tempo: essa função tem um 'for' que itera de 0 até o tamanho do vetor 'SCCs', tendo como complexidade O(n), com 'n' sendo o tamanho do vetor.

int MaxDegree - complexidade de espaço: essa função realiza todas as operações considerando estruturas auxiliares unitárias O(1) e não aloca nenhum array, portanto sua complexidade de espaço é O(1).

5. Conclusão

Este trabalho lidou com o problema de distribuição de rotas entre aeroportos, no qual a abordagem utilizada para resolução foi o desenvolvimento de um script que faz a leitura de um arquivo txt com os dados das rotas e aeroportos buscando fazer a ligação mínima entre aeroportos com o objetivo de que haja rota de todo aeroporto para qualquer outro.

Com a solução adotada, pode-se verificar que a tentativa utilizada foi a criação de uma classe chamada Graph, e essa classe desempenha um papel especial como vimos no desenvolvimento da documentação.

Por meio da resolução desse trabalho, foi possível praticar os conceitos relacionados a grafos, com a necessidade de criação de rotas entre aeroportos, e também a resolução de problemas relacionados à alocação de ponteiros. Pela estratégia de resolução escolhida, além do algoritmo de Kosaraju, pratiquei principalmente a lógica de alocação de ponteiros, vetores, listas e stacks e sua utilização em funções externas.

Referências

Slides virtuais da disciplina de Algoritmos 1. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.