Trabalho prático 2

Algoritmos 1

Gabriela da Silva Ciríaco (2019054595)

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

1 - Introdução

Neste trabalho, foi requisitado que se desenvolvesse uma solução para o problema de encontrar o número mínimo de arestas necessárias para conectar os aeroportos de forma que, possamos chegar em qualquer aeroporto da companhia a partir de outro adicionando o mínimo de rotas possível.

2 - Implementação

O programa foi desenvolvido em C++ em um ambiente Linux e compilado utilizando o G++.

2.1 - Modelagem computacional

Para resolver o problema proposto, foi utilizado um objeto Grafo contendo os métodos necessários para a manipulação dos dados. Como era necessário fazer com que, a partir de cada vértice do grafo seja alcançado qualquer outro, foi preciso utilizar o algoritmo de Tarjan que encontra as componentes fortemente conectadas do Grafo para que a partir delas um novo grafo seja montado, onde nele os vértices são cada componente fortemente conectado do grafo original. Para encontrar o número de arestas faltantes era necessário contabilizar o número de arestas de entrada e de saída que cada um dos vértices desse novo grafo precisava e então o máximo desses dois valores (arestas de entradas faltantes e arestas de saída faltantes) definia o número de arestas faltantes no total.

2.2 - Estrutura de Dados

Para representar o grafo, foi criada uma classe Grafo, que contém atributos para representar os componentes fortemente conectados, o número de vértices e uma lista de adjacências. Além disso, essa classe possui métodos para, adicionar uma aresta no Grafo, encontrar as componentes fortemente conectadas, verificar se um

componente possui aresta de entrada/saída e obter o número mínimo de arestas a se adicionar ao grafo para torná-lo fortemente conectado.

2.3 - Algoritmos

Para resolver esse problema, foi utilizado o algoritmo de Tarjan, que realiza um caminhamento com Depth First Search para encontrar os componentes fortemente conectados do grafo. Então, monta-se um novo grafo utilizando cada componente encontrado como sendo um vértice e itera-se sobre eles para contar quantos deles não possuem arestas de entrada e quantos não possuem arestas de saída e o maior valor entre esses dois valores é a resposta procurada, ou seja, o mínimo de arestas que se precisa adicionar ao grafo original para que ele se torne fortemente conectado.

```
function tarjan(V)
    tempoAtual := 0
    S := Pilha()
    componentesFC := Lista()

    for v in V do
        if v.tempoDeDescoberta = -1
            encontraComponente(v)
        end if
    end for

    function encontraComponente(v)
        v.tempoDeDescoberta := tempoAtual
        v.tempoAncestralProximo := tempoAtual
        tempoAtual++
        S.push(v)
        v.estaNaPilha := true

        for each (v, w) in E do
            if w.tempoDeDescoberta = -1
                encontraComponente(w)
                v.tempoAncestralProximo := min(v.tempoAncestralProximo,
w.tempoAncestralProximo)
            else if w.estaNaPilha
                v.tempoAncestralProximo := min(v.tempoAncestralProximo,
w.tempoDeDescoberta)
            end if
        end for

        if v.tempoAncestralProximo = v.tempoDeDescoberta
            componenteFC := Lista()
            do
                w := S.pop()
                w.estaNaPilha := false
                componenteFC.push(w)
```

```
            while w ≠ v
            componentesFC.push(componenteFC)
        end if
    end function

    return componentesFC
end function

function getArestasMinimas(V)
    arestasDeEntrada := 0
    arestasDeSaida := 0
    componentesFC := tarjan(V)

    for componenteFC in componentesFC
        if componenteFC não tem aresta de entrada
            arestasDeEntrada++
        end if

        if componenteFC não tem aresta de saída
            arestasDeSaida++
        end if
    end for

    return max(arestasDeEntrada, arestasDeSaida)
end function
```

3 - Análise de Complexidade Assintótica de Tempo

A operação mais custosa no programa é a de descobrir se cada componente possui vértices de entrada e de saída, pois para cada componente é preciso iterar sobre todos os vértices do grafo, possuindo assim uma complexidade de tempo O(n*m), sendo *n* o número de vértices e *m* o número de componentes fortemente conectados no grafo original.

4 - Conclusão

O trabalho apresentou um enorme desafio, foi de extrema dificuldade a elaboração da solução e a busca de referências que pudessem trazer um maior direcionamento para o desenvolvimento da solução. Foi um desafio interessante, no entanto com um nível muito alto de dificuldade. De fato, posso dizer com clareza que o trabalho foi muito desafiador, mas foi interessante aplicar na prática algoritmos vistos em sala.