Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Teoria dos Grafos

Relatório Trabalho I

Alunos: Eduardo; Guilherme; Rogério; Mauricio

Professor: Stênio Sã Rosário Furtado Soares

Juiz de Fora
2017

1 Introdução

Grafos são uma forma de representação de informação formada por vértice e arestas. Eles servem para modelar diversas estruturas existentes no mundo real, e existem diversos algortimos capazes de resolver problemas referentes a essas estruturas.

Este documento detalha o desenvolvimento de um programa em linguagem de programação C++ que implementa uma estrutura básica de grafo e um conjunto de funcionalidades básicas de de acesso, manipulação, e armazenamento.

1.1 Instruções de uso

Após a compilação do programa, que pode ser realizada através de uma IDE configurada como o CodeBlocks ou através do MAKEFILE disponibilizado, basta acessar a pasta onde o executável foi gerado e executar o seguinte comando:

Listing 1: Comando a ser executado

```
GrafosGrupo5 arquivoEntrada arquivoSaida
```

O parâmetro arquivoEntrada se refere arquivo de texto com o grafo de entrada a ser processado pelo programa, e o parâmetro arquivoSaida se refere ao arquivo de texto onde um grafo de saída será gerado, de acordo com as opções acessadas pelo menu.

1.2 Formato do grafo

O arquivo de entrada a ser processado pelo programa deve possuir o seguinte formato:

Listing 2: Formato do arquivo de entrada

```
5
4 2
4 5
2 5
1 5
5 3
```

Onde a primeira linha se refere ao número de vértices do grafo, e cada linha após a primeira indica uma aresta formada entre o primeiro e o segundo número. Caso haja um terceiro número na linha, este indica o peso da aresta entre os vértices.

Para construir o grafo de maneira apropriada, o programa irá solicitar ao usuário informar se o grafo é direcionado ou não.

1.3 Listagem das funcionalidades

Ao executar o programa, o usuário poderá, através de um menu enumerado, escolher uma das seguintes funcionalidades:

- 1 - Criar vertice;
- 2 - Verificar se vertice existe;
- 3 - Deletar vertice;
- 4 - Deletar aresta;
- 5 - Criar aresta;
- 6 - Obter grau de um vertice;
- 7 - Obter grau do grafo;
- 8 - Verificar se o grafo é kregular;
- 9 - Consultar ordem do grafo;
- 10 - Consultar se o grafo é trivial;
- 11 - Consultar se o grafo é nulo;
- 12 - Calcular caminho minimo;
- 13 - Calcular fecho trasitivo direto de um no;
- 14 - Calcular fecho trasitivo indireto de um no;
- 15 - Imprimir sequencia de graus;
- 16 - Calcular subgrafo induzido;
- 17 - Imprimir o grafo complementar;
- 18 - Imprimir componentes fortemente conexas;
- 19 - Verificar se o Grafo é Euleriano;
- 20 - Apresentar nós de articulação;
- 21 - Apresentar arestas pontes;
- 22 - Apresentar o raio, diâmetro, centro e periferia do grafo;

- 23 - Apresentar o AGM ou florestas de custo mínimo;
- 0 - Salvar em arquivo e Sair.

2 Metodologia utilizada

Para o desenvolvimento deste projeto, duas componentes essenciais para o desenvolvimento do programa foram identificadas: a definição de uma estrutura de dados para a representação do grafo e a decisão de quais abordagens algorítimicas utilizar para alguma das funcionalidades não triviais oferecidas pelo programa. As decisões sobre essas duas componentes serão detalhas nas subseções seguintes.

2.1 Estruturas de dados utilizadas

Para a representação do grafo pelo programa, três classes foram utilizadas: Grafo, Vertice e Aresta. Objetos do tipo Grafo possuem uma lista de objetos do tipo Vertice. Por sua vez, objetos Vertice possuem uma lista de objetos Aresta. Para simplificar a implementação da estrutura, os objetos Aresta também podem ser utilizados para indicar arcos, no caso de grafos direcionados.

Inicialmente, havia-se optado pela implementação dos vértices e arestas como uma lista de listas. Entretanto, observando-se a legibilidade do código e falta de escalabilidade para outras aplicações com mais parâmetros envolvidos, optou-se pela refatoração do grafo com objetos tanto para vértice quanto para aresta.

2.2 Abordagens algoritmicas usadas na solução

Esta subseção irá detalhar algumas das decisões relacionadas às abordagens algorítimicas de alguma das funcionalidades:

- Cálculo de caminho mínimo

  O caminho mínimo pode ser calculado utilizando o algoritmo de Dijkstra ou de Floyd. O algoritmo de Dijkstra começa pelo vértice 1 do caminho desejado e termina assim que o vértice 2 entra na solução. Já o algoritmo de Floyd calcula o caminho de todos os pares de vértices, atualizando a matriz de distâncias seguindo a lógica normal do algoritmo.

- Cálculo de fecho trasitivo direto de um vértice

O fecho transitivo direto (ftd) de um vértice v é o conjunto de todos os vértices que podem ser atingidos por algum caminho iniciando em v. Para calcular, foi utilizado o algoritmo de floyd que calcula o caminho entre todos os pares de nós, e, ao analisar na linha da matriz correspondente ao nó requisitado, vê-se quais nós possuem caminho partindo dele.

- Cálculo fecho trasitivo indireto de um vértice

  O fecho transitivo inverso (fti) de um vértice v é o conjunto de todos os vértices a partir dos quais se pode atingir v por algum caminho. Foi utilizado novamente o algoritmo de floyd, que retorna uma matriz com todos os caminhos para todos pares de nós do grafo. Analisa-se a coluna correspondente ao nó solicitado e verifica-se quais possuem caminho até esse nó.

- Impressão de componentes fortemente conexas

  Utilizando o algoritmo de Floyd, que define os caminhos entre todos os pares de vértices, é possível identificar quais pares de vértices não possuem caminho entre eles, assim é possível identificar quais são as componetes fortemente conexas, já que é necessário um caminho de a para b e de b para a para todo par (a, b) pertencente ao conjunto de vértices.

- Verificação de Grafo Euleriano

  Para verificar se um determinado grafo é euleriano ou não, duas verificações são realizadas no grafo que garantem essa condição.

  Primeiramente, verifica-se se todos os vértices pertencentes ao grafo são de grau par. Para isto, basta percorrer cada vértice e verificar a quantidade de objetos Aresta de cada vértice.

  Caso a primeira condição seja verdadeira, verifica-se se o grafo é conexo. Para essa verificação, utiliza-se uma busca em profundidade. A busca retorna um vetor de inteiros, onde cada posição representa o ID do vértice, e o valor da posição a ID de uma componente conexa. Se todas as posições tiverem somente 1 ID, temos que o grafo possui somente uma componente conexa e portanto é conexo.

- Cálculo de vértices de articulação e de arestas ponte

  A definição de vértice de articulação e de aresta pote são bastante similares: a remoção de algum deles tornará o grafo desconexo. Portanto, inicialmente, a abordagem utilizada para essas duas funções foi a mesma.

  Para a verificação de vértices de articulação, primeiro calculamos a quantidade de componentes conexas do grafo inicial. Então, removemos o vértice escolhido do grafo, removendo também todas as arestas associadas a ele. Caso o grafo resultante

tenha o mesmo número de componentes conexas do grafo inicial, temos que este não era um vértice de articulação.

Entretanto, após testes com grafos com mais de 100 nós, observou-se que o desempenho do algoritmo não era satisfatório. Buscando melhorar esse desempenho, o algoritmo de Tarjan foi implementado para encontrar estas arestas pontes através de uma busca em profundidade. Esta busca procura as arestas que não são abraçadas por nenhuma outra aresta do grafo e as armazena em uma lista. Ao final da busca, estas arestas são impressas. O desempenho do algoritmo passou a ser satisfatório mesmo para instâncias de grafos com 450 nós.

A abordagem para a verificação de arestas ponte é análoga à de vértices de articulação, onde a busca em profundidade, que gera uma arvore T.Cada nó da arvore recebe trez valores, sendo o primeiro(seq) sua ordem na sequencia de visitas da busca em profuncidade o terceiro a quantidade filhos que o no possui e o segundo(pow) valores que são atualizados de acordo com as duas funções:o menor valor seq para cada aresta (v,w) em G(grafo) que não aparece em T e o menor valor de pow para cada filho do no. Para ser um no articulção deve-se atender a dois requisitos.O primeiro diz respeito a raiz, sendo que ela so pode-ra ser considerada de articulação se possuir mais de um filho e para os demais os nos o pow do filho tem que ser maior ou igual a seq do no em questão.

- Cálculo de raio, diâmetro, centro e periferia do grafo

  Para a verificação do raio,diâmetro,centro e periferia do grafo foi utilizada uma abordagem que assemelha-se muito a abordagem utilizada para o calculo de caminho minimo entre todos os pares de vértices conhecida como algoritmo de Floyd.O que difere as duas é o fato de que o algoritmo de floyd soma na matriz os pesos enquanto a abordagem utilizada soma um a cada interação e guarda o caminho que passa pelo maior número de vértices,de modo que nas posições da matriz contenham o número de arestas do caminho de um vértice a outro.Após feito isso varremos a matriz linha a linha guardando o maior valor de cada índice em uma lista,posteriormente o maior valor dessa lista é o diâmetro,o menor o raio e a periferia e centro são os nós que tem o mesmo valor de excentricidade que o diâmetro e raio respectivamente,também tirados dessa lista.

- Cálculo AGM ou florestas de custo mínimo

  O algoritmo de Kruskall foi utilizado encontrar a AGM ou floresta de custo mínimo de um grafo. A opção de escolha do algoritmo de Kruskall em função do algoritmo de Prim foi a facilidade do mesmo ser desenvolvido na implementação atual. O funcionamento do algoritmo será explicado a seguir.

Primeiramente, fazemos uma cópia do grafo para garantir que o grafo original não será alterado. Então, começamos a copiar as arestas do grafo original para a cópia partindo da menor aresta encontrada. Esse processo é repetido iterativamente até que não temos mais nenhuma aresta a ser copiada (grafo já é era um AGM ou floresta de curso mínimo) ou se o número de arestas é igual ao número de vértices menos 1.

3 Experimentos computacionais

O projeto foi testado em um notebook Sansung com memória de 8GB e com um processador i5 terceira geração com 2.60GHz em um sistema operacional Ubuntu 14.04.1 versão Linux VM 3.13.0-44-generec. Foram compilados grafos não ponderados e direcionados de 450,595,945,1150,1400 e 1534 vertices.Abaixo serao citados os resultados encontrados paras as funções Calcular caminho minimo(metodo Floyd e Dijkstra),calcular fecho trasitivo direto de um no,calcular fecho trasitivo indireto de um no,calcular subgrafo induzido, imprimir o grafo complementar imprimir componentes fortemente conexas verificar se o Grafo é Euleriano,apresentar nós de articulação,apresentar arestas pontes ,apresentar o raio, diâmetro, centro e periferia do grafo,apresentar o AGM ou florestas de custo mínimo. Para as funções que necessitam de ids como paramtros foram usados 20 valores escolidos de maneira randomica e a velocidade de execução das funções foi a media entre esses valores

3.1 Teste para grafos de 450 vertices

O tempode compilação foi de 1 segundo

Funções	Tempo
Calcular caminho minimo metodo Floyd	488.15 ms
Calcular caminho minimo metodo Dijkstra	131.5 ms
Calcular fecho trasitivo direto de um no	601.85 ms
Calcular fecho trasitivo indireto de um no	515.75 ms
Calcular subgrafo induzido	7.6 ms
Imprimir o grafo complementar	71706 ms
Verificar se o Grafo é Euleriano	1 ms
Imprimir componentes fortemente conexas	846 ms
Apresentar nós de articulação	83 ms
Apresentar arestas pontes(nao orientado)	26 ms
Apresentar o raio, diâmetro, centro e periferia do grafo	83 ms
Apresentar o AGM ou florestas de custo mínimo(nao orientado)	80440 ms

3.2 Teste para grafos de 595 vertices

O tempo de compilação foi de 1 segundo

Funções	Tempo
Calcular caminho minimo metodo Floyd	1134.1 ms
Calcular caminho minimo metodo Dijkstra	432.65 ms
Calcular fecho trasitivo direto de um no	1305.35 ms
Calcular fecho trasitivo indireto de um no	1142.35ms
Calcular subgrafo induzido	25.33 ms
Imprimir o grafo complementar	100411 ms
Imprimir componentes fortemente conexas	1699ms
Verificar se o Grafo é Euleriano	1 ms
Apresentar nós de articulação	94 ms
Apresentar arestas pontes(nao orientado)	39 ms
Apresentar o raio, diâmetro, centro e periferia do grafo	ms
Apresentar o AGM ou florestas de custo mínimo(nao orientado)	200164 ms

3.3 Teste para grafos de 945 vertices

O tempode compilação foi de 3 segundos

Funções	Tempo
Calcular caminho minimo metodo Floyd	4703.8 ms
Calcular caminho minimo metodo Dijkstra	1429.15 ms
Calcular fecho trasitivo direto de um no	6037.95 ms
Calcular fecho trasitivo indireto de um no	4837,8 ms
Calcular subgrafo induzido	17.33 ms
Imprimir o grafo complementar	191893 ms
Imprimir componentes fortemente conexas	6673 ms
Verificar se o Grafo é Euleriano	1 ms
Apresentar nós de articulação	155 ms
Apresentar arestas pontes(nao orientado)	92 ms
Apresentar o raio, diâmetro, centro e periferia do grafo	ms
Apresentar o AGM ou florestas de custo mínimo(nao orientado)	$1.10303*10^6 ms$

3.4 Teste para grafos de 1150 vertices

O tempode compilação foi de 4 segundos

Funções	Tempo
Calcular caminho minimo metodo Floyd	8277.25 ms
Calcular caminho minimo metodo Dijkstra	2272,65 ms
Calcular fecho trasitivo direto de um no	9685.05 ms
Calcular fecho trasitivo indireto de um no	8181.65ms
Calcular subgrafo induzido	40 ms
Imprimir o grafo complementar	349978 ms
Imprimir componentes fortemente conexas	9226 ms
Verificar se o Grafo é Euleriano	1 ms
Apresentar nós de articulação	187 ms
Apresentar arestas pontes(nao orientado)	112 ms
Apresentar o raio, diâmetro, centro e periferia do grafo	177130 ms
Apresentar o AGM ou florestas de custo mínimo(nao orientado)	$1.9028*10^6 ms$

3.5 Teste para grafos de 1400 vertices

O tempode compilação foi de 8 segundo

Funções	Tempo
Calcular caminho minimo metodo Floyd	18079.55 ms
Calcular caminho minimo metodo Dijkstra	6142.25 ms
Calcular fecho trasitivo direto de um no	16720.3 ms
Calcular fecho trasitivo indireto de um no	12886.35 ms
Calcular subgrafo induzido	14.66 ms
Imprimir o grafo complementar	600876 ms
Imprimir componentes fortemente conexas	14415 ms
Verificar se o Grafo é Euleriano	1 ms
Apresentar nós de articulação	169 ms
Apresentar arestas pontes(nao orientado)	136 ms
Apresentar o raio, diâmetro, centro e periferia do grafo	136503 ms
Apresentar o AGM ou florestas de custo mínimo(nao orientado)	2.9192 * $10^6 ms$

3.6 Teste para grafos de 1534 vertices

O tempode compilação foi de 5 segundo

Funções	Tempo
Calcular caminho minimo metodo Floyd	16299.5 ms
Calcular caminho minimo metodo Dijkstra	6474.3 ms
Calcular fecho trasitivo direto de um no	21895.16 ms
Calcular fecho trasitivo indireto de um no	16477.33 ms
Calcular subgrafo induzido	9 ms
Imprimir o grafo complementar	456725 ms
Imprimir componentes fortemente conexas	18928 ms
Verificar se o Grafo é Euleriano	1 ms
Apresentar nós de articulação	221 ms
Apresentar arestas pontes(nao orientado)	1 ms
Apresentar o raio, diâmetro, centro e periferia do grafo	159822 ms
Apresentar o AGM ou florestas de custo mínimo(nao orientado)	4.568*10^6

4 Conclusões

Durante o desenvolvimento deste trabalho, diversas questões relacionadas a implementação de uma estrutura de grafos em código e a implementação de algoritmos foram observadas.

Primeiramente, a estrutura de grafos estava sendo desenvolvida como uma lista de pares, onde o primeiro elemento indicava o ID do vértice atual, e o segundo elemento indicava

uma lista de pares ID-peso referente a arestas. Embora esta se mostrasse inicialmente como uma abordagem viável, observou-se que a implementação ficava bastante restrita, não sendo possível adicionar e.g. atributos nos vértices. Optou-se então por uma implementação com lista de objetos Vértice, que possuíam uma lista de objetos Aresta. Desta forma, a aplicação se tornou mais escalável para a adição de novas funcionalidades e a legibilidade do código foi melhorada.

Durante a implementação de algumas das funcionalidades, também observou-se que a escolha da abordagem algorítmica é de fundamental importância para a viabilidade da aplicação. Algumas formas de implementação de funções (usualmente, as mais triviais) se mostraram inviáveis em questão de tempo de processamento. Este fato motivou a busca de outras abordagens, como o algoritmo de Kruskal para a obtenção de AGMs e o algoritmo de Tarjan para a obtenção de pontes.

Finalmente, o desnevolvimento deste projeto permitiu um entendimento prático inicial e sobre a implementação e usos oferecidos pelo uso de grafos em computação. Este trabalho abre novos caminhos para a implementação de mais funcionalidades interessantes ou a integração a algum problema do mundo real.