Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
DCC059 - Teoria dos Grafos Semestre 2015-2

Trabalho Prático de Teoria dos Grafos

Daniel André Carlos Cândido

Professor: Stênio Sã Rosário F. Soares

Relatório da primeira e segunda parte do trabalho de Teroria dos Grafos, parte integrante da avaliação da disciplina.

Juiz de Fora

Março de 2016

1 Introdução

O trabalho foi desenvolvido ao longo da disciplina de Teoria dos Grafos, que consiste em apresentar um conjunto de funcionalidades que permite manipular um grafo qualquer. Foi utilizado para representado do grafo uma listas de adjacências. Esse trabalho foi dividido em duas partes, que serão explicadas no topico "Estruturas de dados utilizadas" (é importante salienter que as funcionalidades implenetadas na primeira parte do trabalho são incorporada na segunda parte). E, os grafos analisados serão das instâncias fornecidas junto com a especificação do trabalho e uma instâncias desenvolvido pelo autor do trabalho.

2 Metodologia utilizada

Dentre as linguaguens de programação propostas, C++ foi a linguagem escolhida para implementar as funcionalidades do algoritmo solicitado. Pois, trata-se de uma abordagem flexível e multiparadigma permitindo ainda o uso de orientação a objetos, programação genérica e em alguns casos o uso de ambos em um mesmo código. Uma outra vantagem está nos problemas relacionados à memória, visto que nesses casos é possível adotar um estilo de mais baixo nível.

2.1 Estruturas de dados utilizadas

O algoritmo implementado possui as seguintes classes:

- Classe Grafo: Responsável pelas manipulações do grafo em si. O grafo é representado por listas de adjacências, e para cada vértice há uma lista encadeada contendo suas arestas com os demais vértices do grafo. As listas são compostas de item e são representações de grafos, logo grafos são listas, e essa relação pode portanto, ser definida como herança simples. Os vértices estão definidos como listas de arestas e itens do grafo, ou seja herança múltipla. As arestas, por sua vez, são apenas itens dos vértices o que nos faz apresentar outro exemplo de herança simples. A classe contém funções para: retornar o grau (dado um vértice como parâmetro), remover nós e arestas, adicionar nós e arestas, verificar a ordem do grafo, verificar se o grafo é k-regular, verificar a quantidade de arestas o grafo possui, verificar se um grafo é completo entre outras funcionalidades descritas na próxima subseção.
- Classe Vértice: Utilizada para manipulações referentes às arestas. Suas funções se referem basicamente a: verificar existência de arestas, encontrar arestas, e remover arestas (ambas apresentando o valor do id do vértice como parâmetro).

- Classe Item: Possui operações referentes à lista encadeada. Como por exemplo, pegar próximo, pegar anterior, pegar informação, entre outras.
- Classe Lista: Apresenta as operações referentes à lista de vértices. Como por exemplo, contar número total de itens, adicionar e deletar item, entre outras.
- Classe LeituraGravacao: Responsável pela leitura e gravação dos arquivos de texto. Sendo que os arquivos de leitura são as instâncias contendo os grafos.

2.2 Funções implementadas na primeira parte

Na primeira parte do desenvolvimento do trabalho, foi implementadas funcionalidades basicas para a manipulações do grafo e outras funcionalidades que serão citadas abaixo.

- Fucionalidades Basicas:
  - Leitura e escrita de aquivos.
  - Adição e remoção de nos.
  - Adição e remoção de arestas.
  - Retornar o grau do nó.
  - Verificar a k-regularidade do grafo, aonde o k é informado pelo usuário.
  - Informar a ordem do grafo.
  - Verificar se o grafo é trivial.
  - Verificar se o grafo é nulo.
  - Retornar o grau do grafo.
- Funcionalidades Intermediárias:
  - Mostra a vizinhaça aberta de um nó.
  - Mostra a vizinhaça fechada de um nó.
  - Verificar se o grafo é um multtgrafo.
  - Verificar se o grafo é completo.
  - Verificar se o grafo é bipartido.
  - Encontrar o menor caminho entre dois vértice(Utilizando o Dijkstra).

2.3 Funções implementadas na segunda parte

Na segunda parte do desenvolvimento do trabalho pratico, foram implementadas funções para manipular árvore geradora mínima e busca.

- Algoritmo de Kruskal e Algoritmo de Prim: Consistem em encontrar uma árvore geradora mínima em um grafo conexo.
- Busca em largura e pefundidade: Consiste em encontrar um dado vértice v no grafo.
- Algoritmo de Flody-Warshall: Consiste em encontrar o menor valor do caminho de um vértice v para todos os vétices u do grafo.

2.4 Abordagens utilizadas

A abordagem utilizada para o desenvolvimento do trabalho é a lista de adjacência e lista encadeada para manter as relações entre vértices e arestas. Em algumas funções, foi utilizada fila de prioridade, algoritmo de ordenação "sort" que utiliza o Quicksort para ordenação e a classe vector, onde todas as estruturas são da propria linguagem.

3 Experimentos computacionais

As funções implementadas foram desenvolvidas e testadas em computadores com sistema operacional Linux com as seguintes especificações.

- Processador: AMD FX 6300.
- Placa de Video: GeFoce GTX960.
- Menoria RAM: 12GB DDR3.

Como mencionado anteriormnete, as instâncias utilizadas formato .txt foram fornecidas junto com a especificação do trabalho e desenvolvida pelo autor do trabalho. Os resultados que serão mostrado a seguir, foi utilizado a instância "teste.txt" desenvolvida pelo autor do trabalho como mostrado na imagem abaixo.

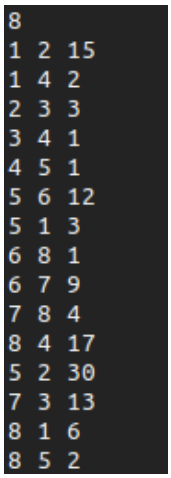

Figura 1: Instância: teste.txt

Resultados:

- Algoritmo Flody-Warshall:

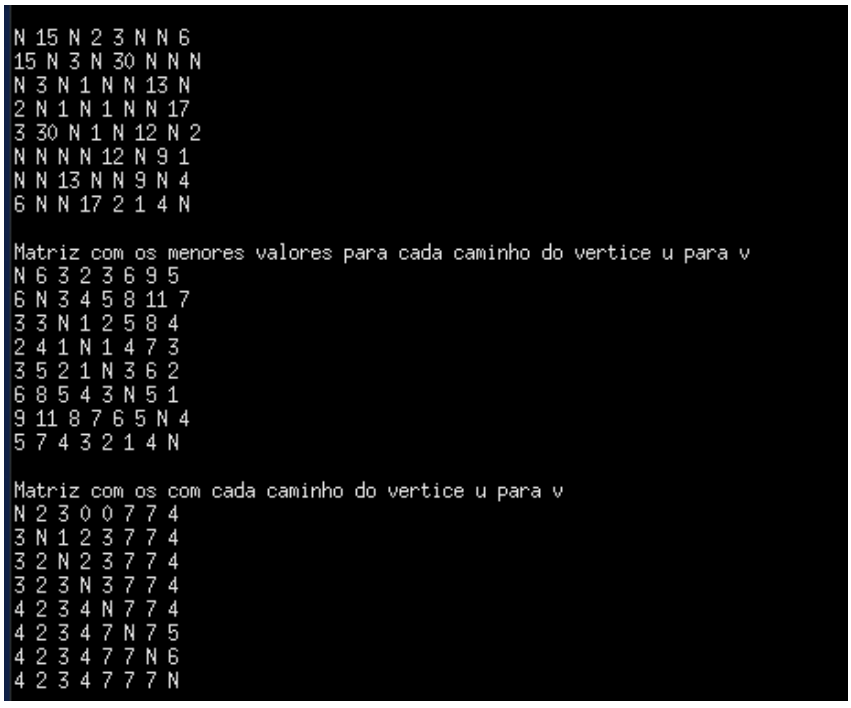

Figura 2: Algoritmo Flody-Warshall, a primeira matriz sem a execução do algoritmo, a segunda e resuldado do algoritmo e a terceira represeta o menor caminho do vértice v para cada vértice u

- Algoritmo Prim:

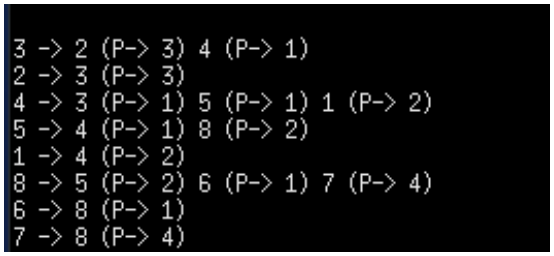

Figura 3: Arvore geradora mínima resultante do algoritmo Prim

- Algoritmo Kruskal:

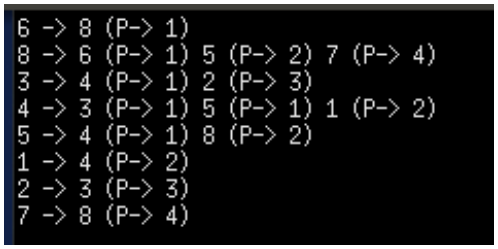

Figura 4: Arvore geradora mínima resultante do algoritmo Kruskal

- Algoritmo Dijkstra:

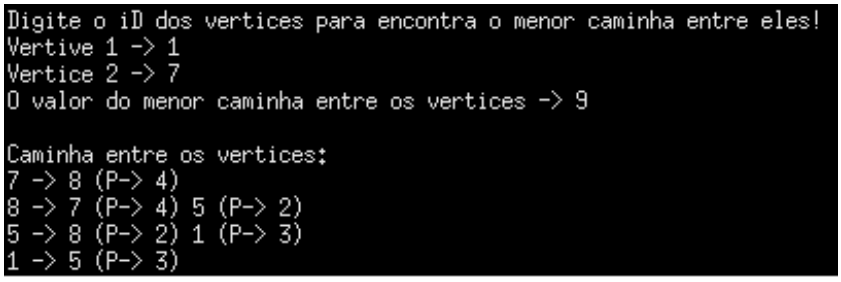

Figura 5: Menor valor e o caminho entre dois vértice utilizando o algoritmo Dijkstra

4 Conclusões

Nesse trabalho, como já foi mencionado, foi utilizada a lista de adjacência a lista encadeada. Apesar dessa estrutura não ser a mais rápida (a matriz de adjacência é mais rápida, porem consumo de memoria é excessivamente alto), ela tem um gasto de memoria baixo, se implementado de forma correta, por esse motivo que foi utilizada nesse trabalho. Na primeira parte da implementação do trabalho, não ouve dificuldade em desenvolver está parte, no entanto, na segunda parte, tive um certo trabalho para desenvolver os algoritmo de Prim, Dijkstra para encontrar o caminho entre os vértice e o algoritmo para verificar se o grafo é bipartido. E ainda, posso afirmar que a aplicação dessas funcionalidades intensificou no domínio perante os conceitos abordados em sala de aula.

5 Referências Bibliograficas

Análise da Complexidade do Algoritmo de Floyd-Warshall.

Algoritmo de Dijkstra Estudo e Implementação.

Kruskal.

Prim.

Algoritmos e teoria dos grafos.

Noções Básicas.