Larissa Gremelmaier Rosa (20100531)

Raquel Cristina Schaly Behrens (20100544)

Atividade A2 – Relatório

Grafos (INE5413)

26 de junho de 2020

Sumário

Exercício 1 – Componentes Fortemente Conexas

Para a execução do algoritmo de Componentes Fortemente Conexas (CFC) criou-se a classe CFC, em java, que implementa as funções: CFC, DFS, DFSAdaptado e DFSVisit. A classe *main* cria uma instância de GrafoDirigido, lê o arquivo de entrada, e chama a função CFC enviando o grafo dirigido.

A função CFC (implementação do algoritmo 15 das anotações da disciplina) chama a função DFS (implementação do algoritmo 16). Na função DFS, criou-se quatro estruturas de dados listas do tipo LinkedList para implementar as listas C, T, F e V do algoritmo original.

- Optou-se pela utilização de LinkedLists, porque muitos métodos da classe CFC utilizam a função *get* e *set*, que, para LinkedLists, essas funçõe possuem a complexidade O(n) no pior caso, e, para ArrayLists, possui complexidade O(1).

A função DFSVisit (chamada por DFS), chama o método *vizinhos* de GrafoDirigido, a qual retorna N+, como especificado no algoritmo 17 das anotações da disciplina.

Seguindo a execução da função CFC, é chamada a função DFSAdaptado, que é quase idêntica à função DFS, sendo a diferença a seguinte: o seu *for* é executado percorrendo a lista de vértices do grafo do final para o início (reversamente).

Exercício 2 – Ordenação topológica

Para a execução do algoritmo de Ordenação topológica, foi utilizado a classe GrafoDirigido, construído no exercício 1 e a classe OrdemTopologica, que contém as funções que implementam os algoritmos 18 e 19 das anotações da disciplina. Para resolver o problema, criou-se as funções:

- dfs_visit_ot → implementa o algoritmo 19 e é uma função recursiva que faz a busca em nível;
- dfs_to_topologic_order → chama a função acima para cada vértice ainda não visitado.

Basicamente, seguiu-se o pseudo algoritmo apresentado no livro, onde as listas foram feitas usando a estrutura HashMap, por ser mais fácil de manipular posições na lista, algo muito utilizado neste algoritmo e terem um tempo de get == 1.

Exercício 3 – Algoritmo de Kruskal

Para a execução do algoritmo de Kruskal foram utilizadas a classe Grafo, desenvolvida no trabalho 1, e a classe Kruskal. Para resolver tal exercício, foi necessário implementar uma nova função na classe Grafo:

- ordena_peso → retorna a lista de arestas ordenada por peso, de maneira decrescente.

Na criação da classe Kruskal, foi implementada apenas uma função que resolve o algoritmo conforme demonstrado no livro-texto. Nessa função, foram utilizados Hashsets, pois muitas das operações do algoritmo envolvem união e comparação entre conjuntos, o que é implementado nessas estruturas. Junto com Hashsets, foram utilizados Lists para as Arestas, pois ela é de fácil utilização e funciona bem para uma quantidade pequena de elementos, o que é sempre verdade no caso das Arestas (sempre 3 elementos, os vértices e o peso) e HashMaps para manipulação de listas de conjuntos, tanto pela manipulação facilitada de indexes como pela velocidade de busca dessa estrutura.