UNIVERSIDADE FEDERAL DE ITAJUBÁ
Campus Itabira

Engenharia da Computação
ECO027 – Projeto e Análise de Algoritmos

# Projeto 3 C3PO

25037 – André Viana Sena de Souza

Prof. M.e Walter Aoiama Nagai
walternagai@unifei.edu.br

Monitor: Emmanuell Miqueleti
emmanuellnogmiq@hotmail.com

Itabira – MG

10 de novembro de 2014

# Sumário

# 1 Introdução

O Projeto 3 é bastante extensivo, com diversos requerimentos e tarefas para serem cumpridos dentro do escopo de grafos. A partir de um arquivo, o programa deve ler vértices e suas arestas, e com base neles criar as demais estruturas necessárias para o solucionamento dos itens especificados. Neste relatório serão explicadas as implementações dos seguintes itens:

- Grau de vértices;
- Ciclos no grafo;
- Caminhos mais curtos entre vértices;
- Determinar se vértices estão fortemente conectados;
- Calcular Planet Rank.

Durante sua implementação, o projeto foi dividido em três partes. O cabeçalho grafo.h define a estrutura base, métodos de percurso, e uma função de leitura a partir do arquivo. O escopo da base dispõe de um grafo estático, mas não só engloba todas as definições de métodos auxiliares no acesso aos campos do grafo interno, como também soluções aos problemas endereçados ao programa. E, por fim, main.cpp contem todos os métodos relacionados à interface do projeto. Modularizando cada campo, para melhor leitura e entendimento do código.

Apesar da literatura utilizada nas aulas dispor de exemplos e implementação dos grafos em Java, este projeto foi implementado em C++. Essa linguagem foi escolhida por sua maior facilidade de manutenção do código e otimização. A IDE do projeto utilizada foi o Code::Blocks, e sua documentação foi feita a partir de anotações do Doxygen.

# 2 Algoritmos e estruturas de dados

## 2.1 Grafo

O objeto grafo foi implementado a partir de um struct, hospedando todos seus componentes. O componente de Aresta é bastante simples, contando apenas com um overload do operador ==. Em contraste, o componente Vertice mantem variáveis de informação (vértice antecessor, tempo de descoberta/término, cor), e também um ponteiro para um std::vector que armazena todos os seus vértices adjacentes. Para fácil manipulação dessa estrutura de dados dinâmica,

foram criados dois métodos auxiliares: AdicionarVerticeAdj(int), adiciona vértices à lista, instanciando o ponteiro quando é necessário, e ProximoVerticeAdj(int*) itera através da lista com um int iterador estático da função, retornando false quando a lista chega ao fim.

O construtor do objeto grafo precisa do número de vértices e um vetor de arestas. As matrizes de adjacências e incidência do grafo são instanciadas durante o mesmo, juntamente com o vetor de vértices, que é utilizado primariamente nos métodos de percurso. Ambos métodos de percurso de largura e profundidade foram implementados.

O método de CarregarArquivo é bastante simples, pois retorna um ponteiro instanciado de Grafo caso obtenha sucesso, ou um ponteiro nulo quando o arquivo não é legível.

## 2.2 Base

O segmento de base implementa os métodos mais importantes do C3PO. Dentro dele é definida a estrutura auxiliar MatrizPlanetRank, que serve como um contêiner para todas as matrizes utilizadas durante o calculo do Planet Rank. Em base.cpp, são instanciadas as variáveis estáticas _grafo, _rank, e _percursoAtual, que servem de base para o funcionamento dos demais métodos. As mesmas devem ser desalocadas ao final do programa através do método DescarregarGrafo().

Na resolução de cada problema foram utilizados estes métodos:

- Grau de vértices – Utiliza os métodos Grafo_GrauEntrada(int), Grafo_GrauSaida(int), que buscam suas informações através da matriz de adjacências e Grafo_GrauIncidencia(int) que usa a matriz de incidência.

- Ciclos no grafo – Faz uma busca por profundidade, sendo que a mesma, durante o processo de recursão, também busca por arestas de retorno.

- Caminhos mais curtos entre vértices – Após um percurso por largura com origem no vértice inicial do caminho, imprime recursivamente a partir do vértice final até ao inicial, se possível. Os métodos utilizados são Grafo_DeterminarCaminho(int,int) e Grafo_ImprimirCaminho(int, int).

- Determinar se vértices estão fortemente conectados – Uma implementação à risca do algoritmo do Nívio Ziviani no método Grafo_ComponentesFortementeConectados(int*), que após subsequentes chamadas à Grafo->_Profundidade_R(int), retorna um vetor com os vértices de todos os componentes encontrados na busca por profundidade.

- Calcular Planet Rank – Após uma chamada de CalcularPlanetRank(), as matrizes internas são inicializadas e PlanetRank(int) retorna o valor correspondente ao planeta em seu parâmetro.

### 2.3 Interface

A implementação da interface se encontra no escopo main.cpp. Nele estão diversos métodos e uma enumeração ComandoMenu, que serve como guia para as possíveis entradas diferentes dos menus. Cada opção selecionada tem um método correspondente, facilitando o processo de manutenção e leitura do código. O programa começa em um loop infinito, e uma vez que o grafo é carregado, o usuário é imediatamente levado ao menu de opções do grafo. Lá existem várias opções diferentes, cada uma levando diretamente ao seu método ResolverX correspondente, ou ao submenu, em que uma segunda opção será avaliada e repassada para a função ResolverX que lhe é necessária. Quando a opção Sair é selecionada, os recursos da base são desalocados e o programa é encerrado.

## 3 Apresentação e discussão dos resultados

Devido à absência dos componentes do grupo em algumas aulas importantes da matéria, ainda existem certas dúvidas com relação aos resultados que o programa retorna. Com isso em mente, foi tomado cuidado redobrado com relação à estrutura base do programa. Caso existam erros, eles podem ser corrigidos em pouquíssimo tempo e sem muitas alterações a serem feitas. Ainda existem outros possíveis pontos de otimização, como anotações de rotinas complicadas e/ou explicações mais detalhadas na documentação do Doxygen, mas o resultado final do projeto pode ser considerado satisfatório.