Trabalho Prático 1 – Algoritmos 1

O Plano de Campanha

Aluno: Matheus Flávio Gonçalves Silva

Matrícula: 2020006850

1. Introdução

O problema proposto consiste na verificação de satisfabilidade das propostas eleitores de um candidato hipotético. Para que suas propostas atinjam o status de satisfazíveis, é realizada uma pesquisa com N seguidores a respeito de M propostas, em que cada eleitor vota em duas propostas que lhes agradam e em outras duas que os desagradam.

Segundo a história apresentada, é necessário que, para cada eleitor, ao menos uma proposta que agrade permaneça no plano de campanha e ao menos uma proposta que desagrade seja retirada do plano de. Existe um caso especial que é um voto "0". Nesse caso, o eleitor se abstém de votar em mais uma proposta, seja esse voto positivo ou negativo, e isso força com que a proposta que não seja votada nula obrigatoriamente seja acatada ou rejeitada.

O objetivo desse processo é verificar a satisfabilidade das propostas desenvolvidas pelo político em relação à opinião de seus seguidores. O trabalho é entendido como um problema 2SAT que envolve duas variáveis de aceitação que podem ou não ser aceitas ao mesmo tempo e duas variáveis de negação que podem ou não ser negadas ao mesmo tempo.

2. Compilação e execução do programa

A compilação do programa é feita por meio da execução do comando **"make"** em um terminal aberto na pasta raiz do projeto. A compilação cria compilações intermediárias **".o"** na pasta **"obj"** a partir dos cabeçalhos inclusos na pasta **"include"** e do código provindo da pasta **"src"** e gera o executável **"tp01"** na pasta raiz do projeto.

A execução do programa é feita por meio do comando **"./tp01 < entrada.txt"**, sendo "entrada.txt" o caminho até o .txt com os dados de entrada. Não necessariamente precisa ter esse nome.

3. Implementação

O programa, bibliotecas e cabeçalhos utilizados foram desenvolvidos na linguagem C++, sendo utilizado o compilador G++ (com padrão -std=c++11 ditado no arquivo Makefile). O desenvolvimento se deu em um notebook Asus i5 6200u com 8Gb de ram e ambiente Linux Mint 21, com testes diversos em um computador (Ryzen 5 1600 com 16Gb de ram), a plataforma de desenvolvimento online Repl.it e testado localmente em um computador Linux do DCC. Todas as compilações foram executadas com o Makefile incluso na pasta raiz do projeto, que foi criado a partir de uma adaptação de Makefiles anteriores da disciplina de Estrutura de Dados. As execuções testes do programa foram realizadas por meio dos terminais: zsh (dentro Visual Studio Code), Windows Terminal, linha de comando padrão do site Repl.it e o terminal padrão das máquinas Linux do DCC.

Como o trabalho é entendido como um problema 2SAT composto, a resolução do mesmo se dá mediante apresentado em aula: são rodadas duas DFS, em que a primeira é realizada sobre o grafo original e armazena a ordem de término para cada componente e a segunda é realizada sobre o grafo reverso de acordo com a ordem decrescente de térmico armazenada na primeira DFS. Para saber se o problema é satisfazível, é analisada a relação entre vértices e componentes conectados.

3.1. Estruturas de Dados

Para a implementação do trabalho foi utilizado um Tipo Abstrato de Dados nomeado **“Graph”** que possui um inteiro **“size”** que indica o tamanho do grafo utilizado para modelar o problema, duas listas de inteiros para as adjacências **“adjacency”** e **“adjacencyReverse”** que funcionam como listas de adjacências dos vértices para o grafo original e reverso, um vetor de inteiros **“firstDFSOrder”** que armazena a ordem de término da primeira DFS, um vetor de inteiros **“componentRelation”** que armazena a relação entre os vértices e os componentes fortemente conectados e um vector de booleanos **“alreadyVisited”** que armazena quais vértices já foram visitados durante a execução da primeira DFS.

3.2. Funções e Procedimentos

Arquivo graph.cpp:

Graph::Graph(int size): Cria e inicializa um grafo de acordo com a quantidade de propostas, usando 2*proposal como sendo o valor base;

int Graph::getSize(): Retorna o valor de size do grafo (2*proposal);

vector<bool> Graph::getAlreadyVisited(): Retorna o vetor alreadyVisited;

vector<int> Graph::getComponentRelation(): Retorna o vetor componentRelation;

vector<int> Graph::getFirstDFSOrder(): Retorna o vetor firstDFSOrder;

void Graph::addEgde(int vertice1, bool negation1, int vertice2, bool negation2): Cria arestas em ambos vetores de adjacência simultaneamente de acordo com as implicações ensinadas em sala de aula a respeito do 2SAT;

void Graph::dfs(int vertice): Primeira DFS rodada a fim de gerar o vetor de ordem de conclusão em firstDFSOrder.

void Graph::dfs(int vertice, int component): Segunda DFS rodada levando em conta a ordem de término da primeira DFS e é realizada sobre o grafo reverso. Utilizada para gerar as relações de cada vértice com seu devido componente conectado. É feita como um overflow da primeira DFS ao acrescentar mais um argumento na chamada da função.

Arquivo 2sat.cpp:

bool get2SATViability(Graph *graph): Função booleana responsável por retornar se o problema é satisfazível ou não. Faz uso de 3 laços for e de todas funções do grafo com exceção da função addEdge.

3.3. Programa Principal

O programa principal cria uma série de variáveis de auxílio para serem passadas como parâmetros às outras funções, faz a leitura dos dados passados por meio dos arquivos txt, cria todas as arestas do grafo por meio do método addEdge e imprime na tela “sim” caso a promessa de campanha seja satisfazível ou imprime “nao” caso não seja. Além disso, caso um seguidor vote duas vezes 0 para a mesma categoria, o voto dele é ignorado. É responsável por ler todo o arquivo e parar somente com entrada 0 0.

4. Conclusão

De um modo geral, a implementação do trabalho ocorreu sem grandes problemas após o entendimento do problema, o que, infelizmente, só ocorreu no dia máximo de entrega, motivo pelo qual atrasei o trabalho por ter que refazer tudo.

A principal dificuldade encontrada foi no que diz respeito ao entendimento, pois inicialmente entendi que, para todos os seguidores, as propostas aceitadas e rejeitadas tinham que ser simultaneamente as mesmas. Inicialmente havia entendido também que o voto 0 poderia ser entendido como voto nulo, não como “voto duplicado”;

Referências

TP1 – O Plano de Campanha. Disponível em: < https://virtual.ufmg.br/20222/pluginfile.php/409956/mod_assign/introattachment/0/TP1.pdf?forcedownload=1>. Acesso em: 11 out. 2022.

Grafos-DAGe2SAT. Disponível em: < https://virtual.ufmg.br/20222/pluginfile.php/376885/mod_resource/content/3/Grafos-DAGe2SAT.pdf >. Acesso em 13 out 2022.

ZIVIANI, Nívio. **Projeto de Algoritmos com Implementações em Java e C++**. Disponível em: < http://www2.dcc.ufmg.br/livros/algoritmos-java/implementacoes-07.php >. Acesso em: 13 out. 2022.