Trabalho Prático 1
O Plano de Campanha

Pedro Gomes Santiago Pires Beltrão - 2021039760

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

pedrosantiago@ufmg.br

1. Introdução

O problema trata de um deputado que tem propostas a serem implementadas, porém ele quer saber a aceitação dessas propostas pelos seus seguidores, para isso disponibiliza suas propostas, dessas, os seguidores citarão duas que acham necessárias a aprovação e duas que acham descartáveis. Com isso o deputado espera agradar todos os seus seguidores de forma que pelo menos uma das duas citadas para aprovação deve ser aprovada e pelo menos uma das duas citadas para o descarte deve ser descartada, isso para cada um de seus seguidores que opinou na pesquisa. Vale ressaltar que os seguidores podem não escolher nenhuma das propostas (o que reflete em zero na entrada), quando não é escolhida significa que a outra proposta deverá ser cumprida, já em caso de não escolher nenhuma das duas propostas que devem ser aprovadas, automaticamente essa pessoa já está satisfeita em relação a aprovação de projetos (o mesmo pode ser considerado para o descarte e para a comutação de ambos).

2. Modelagem

Em primeiro plano é preciso entender quais algoritmos e estruturas de dados foram escolhidas para a solução do problema, para depois compreender como foram adaptados e quais os complementos para funcionamento geral do algoritmo.

Estruturas de dados: para solucionar a satisfatibilidade do problema, é preciso primeiramente montar uma implicação lógica, dessa implicação tem-se 4 propostas: p1, p2, p3 e p4, dessas é preciso que p1 ou p2 seja resolvida ao mesmo tempo que p3 ou p4, ou seja (p1 or p2) and (p3 or p4). Para transformar isso em um grafo e facilitar a solução, pode-se transformar p1 or p2 em p1’ -> p2 e p2’ -> p1, isso será feito para todos as propostas de cada seguidor, dessa forma formando um grafo direcionado. A estrutura utilizada foi a lista adjacência, então para cada implicação é realizado um *push_back*, armazenando cada vizinho de um determinado vértice. No fim tem-se uma classe grafo, armazenada como vetor e para cada um de seus elementos (os vértices) possui-se outro vetor com seus vizinhos. Além disso, foi utilizado uma stack para armazenar a ordem topológica, que recebe push depois que realizar a chamada recursiva do DFS, dessa forma os elementos são empilhados na ordem do último que foi visitado

(em cima) para o primeiro (último da pilha). A lista adjancência foi escolhida por ser uma forma simples de representar o grafo e por facilitar os algoritmos que seriam usados. Já a stack é a forma mais simples de representar a ordem por precisar acessar apenas os vértices no topo da pilha.

Algoritmos: Desse grafo basta verificar se nenhum vértice e seu transposto se encontram dentro da mesma componente fortemente conexa. Para encontrar as componentes fortemente conexas do grafo, foi utilizado o algoritmo Kosaraju's, que por sua vez faz a utilização de DFS (depth first search). As DFS são utilizadas para visitar todo o grafo (no trabalho em específico foi feito de forma recursiva), para isso é preciso demarcar quais vértices já foram visitados. Então a principal ideia ao entrar na função DFS é marcar que o vértice já foi visitado, para então rodar um laço por toda lista adjacência e verificar se os vértices já foram visitados, caso não tenha sido visitado ainda a função DFS é chamada novamente para o vértice ainda não visitado. Também é importante notar que nessa primeira chamada da DFS é montada a ordem topológica do grafo em uma pilha. Seguindo a compreensão do Kosaraju's, primeiramente ele realiza uma DFS no grafo, obtendo a ordem topológica, para em sequência realizar um DFS no grafo transposto (em que são invertidas as direções de cada aresta), acessando pela ordem topológica, dessa forma, para cada iteração completa do DFS, são obtidas as componentes fortemente conexas e quais os seus vértices. Para facilitar a análise se certo vértice e seu transposto aparecem na mesma componente fortemente conexa, cada vértice foi marcado com um id para a componente que ele pertence. Foi escolhido Kosaraju's como algoritmo pelo fato de ele separar todas as componentes fortemente conexas, processo esse que facilita muito a solução do problema. Já as DFSs foram realizadas de forma recursiva por facilitar a implementação e serem essenciais para realização do Kosaraju's. É importante notar que a complexidade do Kosaraju's é de O(v+e), o número de vértices mais o número de arestas, portanto representa uma solução de complexidade de tempo linear, muito satisfatória para resolver o problema mesmo com grandes entradas.

É importante compreender alguns pontos importantes para conectar todos os conceitos de algoritmos e estruturas. Para tratar a entrada do programa, era preciso transformar as propostas em implicações, formando a lista, porém era preciso antes tratar os casos de zero, caso a pessoa não escolhesse uma proposta, seria o caso de 0 or p2, por exemplo, porém sabemos que 0 or p2 é o mesmo que p2, ou p2 or p2, portanto, para esses casos, o valor de p1 (antes zerado) recebe a atribuição de p2, formando a lógica p2 or p2. Outro tratamento era para o caso de dois zeros, nesses casos a opinião da pessoa já é automaticamente aceita pelo deputado, então não era preciso fazer apontamentos no grafo. Outra adaptação necessária, é dobrar os vértices que representam a transposição, já que as estruturas escolhidas não recebem índices negativos. Seguindo, na segunda DFS do código, é preciso atribuir os valores dos ids

dos componentes, para que no final da main pudessem ser identificados os elementos na componente fortemente conexa. Para essa identificação, apenas é realizado um loop que passa por metade dos vértices, conferindo se o seu transposto possui a mesma id de componente. Por fim, a última adaptação foi relacionada ao laço na função main que permite ler diferentes blocos de entrada até alcançar os valores 0 e 0 de propostas e seguidores, vale notar que é preciso resetar os valores essenciais para formação do grafo e das SCCs (strongly connected components) em cada iteração do laço.

3. Conclusão

Por meio da realização do trabalho prático, foi possível compreender a importância de se estudar grafos, visto que estes podem ser utilizados para facilitar a solução de diversos problemas de lógica enfrentados. Portanto, nota-se também a importância de compreender os algoritmos e estruturas que facilitam a manipulação, compreensão, pesquisa e caminhamento em grafos. No caso do trabalho, o Kosaraju's facilita a identificação das componentes fortemente que estão ligadas à satisfação mútua entre os seguidores e as DFSs permitem a pesquisa nos grafos e identificação da ordem topológica. As listas adjacência facilitam a implementação e representação do grafo que, além disso, utiliza de uma stack para representação da ordem topológica.

Belo Horizonte - 11 de outubro de 2022