TRABALHO PRÁTICO 1
O PLANO DE CAMPANHA

Tarcizio Augusto Santos Lafaiete

Universidade Federal de Minas Gerais(UFMG)
Belo Horizonte - MG - Brasil

tarcizio-augusto@hotmail.com

1.Introdução

O problema proposto para implementação do trabalho prático foi modelar um algoritmo para o candidato Alan Turing, no qual através do voto de seus seguidores nas redes sociais ele determina se há um conjunto de propostas que satisfaça todos os seus seguidores. O programa recebe como entrada 4 votos de cada seguidor sendo os dois primeiros de propostas que ele quer que sejam mantidas e as duas últimas são propostas que ele deseja que sejam retiradas, com isso o algoritmo deve retornar “sim” caso os seguidores possam ficar satisfeitos e “não” se caso o contrário. Importante ressaltar que se entende como satisfeito caso pelo menos uma propostas para ser mantida e uma para ser retirada sejam atendidas.

2.Interpretando o problema

Com a base do problema estabelecida, pode-se partir para a interpretação deste problema em requisitos computacionais e algoritmicos. Analisando a entrada pode-se convertê-la para um expressão booleana de modo a ficar mais próxima de modelos e problemas computacionais já conhecidos. Primeiramente podemos pensar nos votos dos seguidores como os literais da expressão, sendo o voto para manter a proposta como um literal verdadeiro e o voto para remoção como o literal negado. A partir desta premissa e do fato de que para ser satisfeito uma das duas propostas tem que ser mantida e uma das outras duas deve ser retirada, pode-se montar cláusulas de 2 elementos cada, sendo que cada seguidor irá gerar duas cláusulas a primeira com os votos para manter e outra com os votos para retirada. Por fim pode-se juntar todas as cláusulas estabelecidas e formar uma CNF (Forma Normal Conjuntiva, do inglês Conjunctive normal form), assim temos a nossa expressão booleana formalizada.

Este tipo de expressão booleana é uma expressão conhecida e é denominada como o problema da Satisfatibilidade, na qual a ideia é através de alguma computação conseguir descobrir se há um conjunto de valores para os literais que torne a expressão verdadeira. Neste caso, temos um caso específico deste problema que é o denominado *2-Sat Problem,* que é quando as cláusulas da CNF tem exatamente 2 literais e para este problema tem-se uma resolução conhecida.

3. Modelagem do problema

Como dito no tópico anterior o problema proposto para este trabalho prático se encaixa na descrição de um *2-Sat Problem,* que possui uma solução conhecida. Inicialmente o pensamento mais óbvio para se resolver este problema seria testar todas as possibilidades de combinações de verdadeiro ou falso até se encontrar uma atribuição verdadeira ou até se esgotar as possibilidades, o que significaria que não é possível satisfazer a expressão booleana, contudo este método é ineficiente devido ao custo elevado de tempo necessário para se testar todas as combinações possíveis. Com isso em mente, cientistas da computação no passado pensaram em novas formas de representar o problema de forma a se tornar mais simples a sua execução foi assim que chegaram na ideia de representar a CNF através de grafos.

Os grafos, no geral, são um instrumento matemático com características e propriedades extremamente úteis para realizar o processamento de problemas de forma computacional. Neste caso, utiliza-se da conversão das cláusulas da CNF para sua forma implicativa que seria a conversão mostrada abaixo:

$$X_1 \vee X_2 = (\neg X_1 \rightarrow X_2) \wedge (\neg X_2 \rightarrow X_1)$$

A partir desta conversão pode-se criar um grafo direcionado, no qual as arestas serão a representação das implicações, para este tipo de grafo dá-se o nome de grafo de implicação.

Em 1967 o matemático Krom descreveu um procedimento para resolver o problema do 2-Sat, neste procedimento Krom foca sua atenção para a regra de transitividade na qual se temos duas cláusulas como, $(x \vee \neg z)$ e $(y \vee \neg z)$ pode-se gerar a cláusula $(x \vee y)$. Disto Krom afirma que caso sejam realizadas repetidas inferências dessa regra na expressão booleana de forma que para uma variável x qualquer seja gerada ambas as cláusulas $(x \vee x)$ e $(\neg x \vee \neg x)$, a expressão não é consistente, logo não é satisfazível.

Com isso em mente o professor Sambasiva Rao Kosaraju elaborou um algoritmo utilizando caminhamento em grafos para resolver o *2-Sat Problem* em tempo linear, este que seria o algoritmo de Kosaraju. Utilizando-se das duas noções anteriormente explicadas, Kosaraju primeiro transforma uma CNF em um grafo de implicação em seguida no grafo tenta-se descobrir se há alguma inconsistência lógica como a demonstrada por Krom e para isso Kosaraju utiliza-se do teorema de Aspvall na qual ele prova que se e somente se uma variável e sua negação pertencerem à mesma componente fortemente conexa do grafo(CFC) de implicações a regra de inferência de Krom poderá ser aplicada de modo a gerar uma inconsistência.

Sendo assim, o algoritmo de Kosaraju se foca em encontrar as componentes fortemente conexas do grafo e verificar se em alguma delas há esta contradição.

4. O algoritmo

O algoritmo de Kosaraju, como dito antes, tem como intuito descobrir as *CFS's* do grafo de implicação, para isso Kosaraju realiza dois caminhamentos no

grafo por profundidade, a chamada DFS (Deep First Search). Na primeira DFS, à medida que vai-se visitando os vértices do grafo, estes vértices vão sendo marcados como visitados e vão sendo empilhados em uma stack até que todos os vértices do grafo direcionado tenham sido visitados pela Primeira DFS. Em seguida inverte-se as arestas do grafo direcionado de forma a gerar um grafo inverso e então até que a pilha esteja vazia todos os elementos vão sendo retirados dela e são usados de parâmetros para a segunda DFS que é rodada no grafo invertido, com isso cada vértice que passa pela DFS segunda é marcado como visitado assim a DFS ao retornar do seu processo recursivo vai atribuindo um mesmo valor de CFC para os vértices visitados, e isso se repete até não houver mais vértices na pilha e sem serem visitados, assim sabe-se exatamente a CFC de cada vértice então pode-se verificar se há alguma CFC com uma contradição.

Este algoritmo ele funciona para encontrar *CFC's* pois quando passa-se a primeira DFS e empilha os vértices ordem de acesso tem-se a sequência de vértices visitados em ordem do último acesso até o primeiro, ou seja o caminho tomado pela DFS para encontrar os vértices. Com isso, ao passar como parâmetro os elementos desta pilha para a DFS Segunda que lida com o grafo invertido pode-se observar que como as arestas que conectam as *CFC's* estão invertidas os vértices apenas encontraram os elementos pertencentes a sua componente fortemente conexa.

5. Implementação

A implementação completa deste problema foi feita em um arquivo .cpp e para o desenvolvimento do algoritmo as estruturas de dados padrão do c + + utilizadas foram: stack, list, map e vector.

O primeiro passo do programa é analisar os votos do seguidores e armazená-los em um vector de inteiros no qual as propostas que devem ser mantidas são positivas e as que devem ser rejeitadas são negativas, caso de alguma entrada zero a proposta não zero do seguidor é duplicada e em caso do seguidor selecionar dois zeros em um dos pares de votos eles são ignorados. Em seguida com o vector das propostas montado passamos isso para um grafo através da conversão para a forma implicativa de modo que para cada seguidor montamos uma cláusula com os dois votos para manutenção e uma para os dois votos de retirada e de forma semelhante é criado o grafo invertido. Para representar estes grafos foram usados dois map com o par sendo um inteiro representando o vértice e o outro sendo a lista de adjacência.

Após isso, executa-se o processo da DFS Primeira no grafo de implicações dentro de um loop for e vai-se marcando em um vector de booleanos os vértices que já haviam sido visitados e os empilhando em uma stack. Com a stack totalmente empilhada de vértices inicia-se um loop até a pilha ficar vazia, no qual é executado a DFS Segunda com os itens desempilhados e em outro vector de booleanos é marco os vértices que já haviam sido visitados e a cada fim de recursão os elementos que não haviam sido visitados tem valor atribuído por um contador a um vector de inteiros na posição do vértice que identifica a qual CFC ele

pertence. Por fim, para verificar a satisfabilidade verifica-se se no vector de CFC se há algum componente e sua negação presentes no mesmo CFC e imprime na saída sim em caso de não encontrar e não caso contrário.

6. Conclusão

Por fim é importante ressaltar que o algoritmo de Kosaraju e o problema 2Sat no geral tem solução $O(n + m)$ no qual, n são os vértices e m as aresta, esta complexidade no geral é devido as duas DFS passadas pelos grafos que possuem esta mesma complexidade. Contudo o algoritmo implementado pelo aluno possui uma complexidade temporal pior que a do algoritmo original pois em cada DFS executada utiliza-se o método find do map que tem complexidade $O(log\ n)$ com isso tem-se que o algoritmo implementado tem complexidade de $O(n\ log\ n + m\ log\ n)$.