Trabalho Prático 1

DCC215 - Algoritmos 1

Igor Joaquim da Silva Costa

1. Introdução

O problema proposto foi verificar se é possível satisfazer todos seguidores de um político, a partir de uma série de propostas que podem ser aceitas ou não. Mais precisamente, são apresentados um número de seguidores de algum político, e quantas propostas o político possui, cada seguidor escolhe no máximo duas propostas para aceitar, e duas propostas para recusar. Um seguidor fica satisfeito se ao menos uma das suas propostas é aceita e ao menos uma das propostas é rejeitada, o problema é resolvido se existe uma combinação de aceitação/rejeição de propostas que satisfaça todos seguidores.

Para resolver o problema citado, cada proposta do político foi tratado como um literal booleano e toda requisição de usuário como uma cláusula de uma CNF, onde os literais são as propostas que o usuário quer promover, ou rejeitar, no caso da rejeição, são tomados os valores negativos do literais. Nesse sentido, o problema apresentado é reduzido à uma instância do problema 2SAT. Diante disso, é conhecido um algoritmo polinomial capaz de resolver uma instância de SAT, a partir de uma modelagem em grafo.

Diante do exposto, a documentação presente possui como objetivo detalhar como o sistema foi modelado (Seção 2), o quão eficiente ele pode ser (Seção 3) . Por fim, o projeto é sumarizado junto com os aprendizados gerados durante a produção do trabalho(Seção 4).

2. Modelagem

Esta seção tem como objetivo discutir as decisões que levaram à atual modelagem do programa.

2.1 2SAT

Como elucidado na seção 1, o problema apresentado pode ser reduzido ao problema 2SAT, onde cada par (x,y) de proposta que algum usuário quer que seja promovido, se torna a cláusula (X ou Y), e cada par (x,y) de propostas para rejeitar se torna a cláusula (~X ou ~Y).

Dessa forma, o problema pode ser todo escrito em uma única fórmula na forma normal conjuntiva de 2 literais, ou seja, uma instância de 2SAT.

2.2 Resolver 2SAT

A fim de resolver o problema 2SAT, foi utilizado uma outra redução, onde 2SAT pode ser resolvido por um problema de conectividade em grafos. Qualquer cláusula (X ou Y) pode ser reescrita pela forma (Se ~X então Y ou Se ~Y então X), nesse sentido, podemos modelar o problema usando grafos da seguinte forma: para cada cláusula (X ou Y) de 2SAT, crie uma aresta de (~X,Y) e (~Y,X) em um grafo G. Para representar um vértice ~X, sabendo que existem N literais, o index que representa ~X é X + N, logo , todo index maior que N representa um literal negativo.

Nesse grafo, qualquer caminho entre 2 vértices U e V representa que, se U for falso, então V deve ser verdadeiro. Logo, se existe caminho entre X e ~X e caminho entre ~X e X, para algum literal X, X deve ser verdadeiro e falso ao mesmo tempo, logo existe uma contradição e a fórmula não é satisfatível.

2.3 Encontrar Componente conexo

Como o problema foi reduzido a verificar se, para todo X, não existe caminho de X para ~X e vice-versa, basta utilizar um algoritmo que encontra todos os componentes fortemente conexos de um grafo, existe X e ~X no mesmo componente se e somente se a fórmula for satisfatível. Dessa forma, foi implementado o algoritmo de Kosaraju para encontrar componentes fortemente conexos, que consiste em rodar uma DFS para cada nó do grafo G, marcar qual a ordem que os vértices foram visitados e executar uma outra DFS no grafo reverso de G, na ordem decrescente em que os elementos foram visitados em G. Todos pares mutuamente alcançáveis estarão em um mesmo componente, a partir disso, basta verificar se existem X e ~X no mesmo componente, para algum literal X.

2.4 Estrutura de Dados

Para representar o grafo, foi tomada uma lista de adjacência para cada vértice presente, onde existe aresta entre (X,Y) se Y pertence a lista de adjacência de X. Essa decisão foi tomada pois, em grande parte dos casos, o grafo é pouco denso, fazendo a representação em

matriz de adjacência mais cara em espaço e tempo de execução. Nesse mesmo sentido, existe uma lista de adjacência para o grafo reverso.

Além disso, para executar o Kosaraju, foi usado um tipo vetor para simular uma pilha e armazenar a ordem que os elementos são descobertos na primeira DFS.

3. Análise de complexidade

3.1 Espaço

Seja N o número de usuários e M o número de propostas. Inicialmente, é criado 2*M vetores vazios para representar o grafo sem nenhuma aresta, no pior dos casos, cada vetor possuirá 2*M -1 elementos. Assim, a complexidade de espaço se torna $O(M^2)$ na quantidade de propostas.

3.2 Tempo

Para análise de tempo, considere N o número de usuários e M o número de propostas.

3.2.1 Kosaraju

Para se realizar a busca por componente conexos, são feitas 2 DFS, onde a complexidade de cada DFS é de $O(2M + (2M)^2)$. Logo, verificar se a fórmula é satisfatível cresce quadraticamente em função do número de propostas.

4. Conclusões

Com o intuito de saber se existe um conjunto de propostas que agrade todos os seguidores de algum político simultaneamente, foi implementado um programa que utiliza algoritmos de componente conexos para resolver o problema.

Durante o projeto do sistema foram levadas em consideração não só aspectos práticos da implementação de uma modelagem computacional, mas também como a linguagem de programação escolhida poderia ser uma ferramenta útil para chegar no objetivo esperado. Toda a questão de mapear um mini-mundo de interesse em um modelo computacional robusto se mostrou bastante produtiva, levando o aluno a pensar em formas criativas de se resolver e entender o problema, tendo como resultado um extenso aprendizado sobre como representar um grafo pode facilitar, ou atrapalhar, na implementação e na execução de determinados

algoritmos. Por fim, o tempo extra usado para projetar o sistema trouxe várias recompensas no sentido da implementação, sendo um aspecto a ser levado para trabalhos futuros.

Nesse sentido, todo o fluxo de trabalho foi essencial para a consolidação de conteúdos aprendidos em sala, além de apresentar, de forma prática, como softwares maiores, mais consistentes e robustos são projetados e implementados.

5. Instruções para compilação e execução:

5.1 Compilação

Existem partes do programa que são compatíveis apenas às versões mais recentes da linguagem c++, dito isso, deve-se seguir as seguintes configurações para a compilação:

Linguagem: C++
Compilador: Gnu g++
Flags de compilação: -std=c++11 -g
Versão da linguagem: standard C++11
Sistema operacional (preferência): distribuições baseadas no kernel Linux 5.15.

O comando para compilar o programa automaticamente está presente no arquivo **“Makefile”** e sua execução é chamada pelo comando **“make all”**. Deste modo, o executável “tp01” estará compilado e pronto para ser utilizado.

5.2 Execução

Seguem as instruções para a execução manual:

1. Certifique-se que o compilável foi gerado de maneira correta, se algum problema ocorrer, execute o comando “make all” presente no “Makefile”.

2. Uma vez que os passos anteriores foram cumpridos, execute o programa com o comando: ./tp01 < caso Teste01.txt)

3. A saída será impressa no terminal.