Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

DCC005 - ALGORITMOS E ESTRUTURAS DE DADOS III - TURMA TE

Trabalho Prático 2 – Surpresa na Prova

Guilherme Saulo Alves

Professores - Jussara Marques, Olga Nikolaevna, Marcos Augusto, Wagner Meira

1. Introdução

Um professor aplicou uma prova para A alunos com uma questão em que tinha N alternativas de múltiplas escolhas. O proposito deste trabalho consiste em descobrir o maior conjunto de alunos que não trocaram respostas entre si. Para isso precisamos mapear esse estudo em um grafo para avaliar a quantidade de alunos que não tiveram resposta em comum.

Para solucionar o problema acima, aplicamos a ideia do conjunto independente de vértices. Em teoria de grafos, um conjunto independente de um grafo G (V, A) é um conjunto S de vértices de G tal que não existe dois vértices adjacentes contidos em S. Por exemplo, dados 2 alunos x e y em um conjunto independente S, então não há resposta comum entre x e y.

Um grafo pode ter vários conjuntos independentes, porém para esse trabalho devemos achar a quantidade de alunos que estão contidos no maior conjunto independente do grafo.

2. Implementação

Uma das primeiras implementações foi criar uma estrutura para representar um grafo por matriz, pois o TAD de matrizes adjacentes facilita a implementação do algoritmo ótimo. O algoritmo que implementei é capaz de obter a solução ótima baseado na técnica de enumeração.

A partir da enumeração de todas as possibilidades que existem, verifica-se para cada uma delas se ela é possível e se a mesma é maior que a maior solução encontrada até aquele momento. Essa técnica referece a força bruta, pois avalia todas as possibilidades existentes. O algoritmo será explicado detalhadamente ao longo da documentação.

2.1. Considerações Iniciais

O ambiente de desenvolvimento do código foi no CodeBlocks IDE 10.05 no Sistema Operacional Ubuntu Linux 13.04. Também foi usado o compilador GNU GCC Compiler via linha de comando e a ferramenta Makefile (comando make). Não foram feitos testes do programa em ambiente Windows. Segue os comandos para compilar o código:

Linha de comando:

```
cd Documentos
gcc -c main.c grafo.c solucaootima.c readline.c
gcc main.o grafo.o solucaootima.o readline.o -o tp2e
./tp2e input.txt output.txt
```

O código está dividido em sete arquivos principais:

- main.c: programa principal com leitura dos arquivos;
- grafo.c: funções do TAD grafo por matriz de adjacências;
- grafo.h: declarações da estrutura do TAD Grafo e das funções;
- solucaootima.c: algoritmo ótimo para o problema;
- solucaootima.h: declarações das funções do algoritmo ótimo;
- readline.c: algoritmo para ler cada linha do arquivo;
- readline.h: declaração das funções utilizadas;

2.2. Funções e Algoritmos

As soluções dos problemas serão apresentadas a seguir, através de pseudocódigos e esquemas ilustrativos.

2.3. Solução Ótima (Força Bruta)

Como os vértices são representados por números entre 1 o número de vértices A (Alunos), geramos todas as possibilidades existentes de combinação entre esses vértices. Para cada uma dessas possibilidades, verificamos se a mesma é uma solução válida para o problema de vértices independentes ou não.

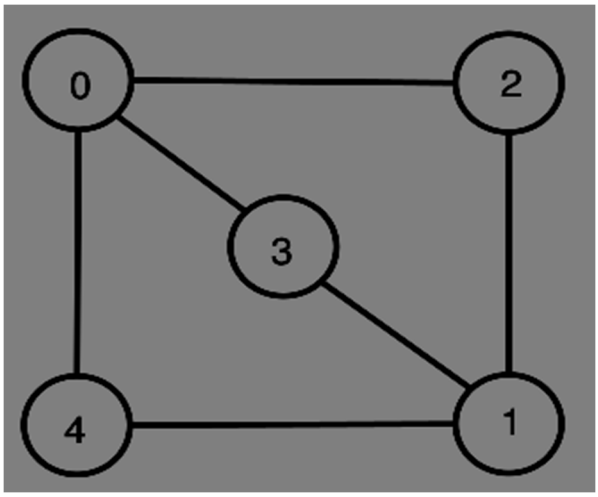
Essa verificação é feita através de uma função que, para cada par de vértices contido na combinação em questão, se existe uma aresta que os intercepta ou não. Caso a função de verificação não encontre nenhuma aresta que conecta nenhum par de vértices em questão, temos uma possível solução.

Como o objetivo do programa é encontrar o maior conjunto de vértices independentes, verifica-se então se o tamanho da solução encontrada é a maior que a solução encontrada até o momento. Caso a resposta seja afirmativa, a maior solução passa a ser a solução que acabamos de encontrar. Se não, descartamos a solução que acabamos de encontrar.

Isso é feito para todas as possíveis combinações e no final da execução temos a solução ótima do problema. O pseud. código para o problema é apresentado a seguir:

```
Seja G = (V, A)
ForcaBruta (G){
MaxConjuntoIndependente (G) ← ∅;
Para todo subconjunto S de vértices V ∈ G faça{
        se S é independente então
                se |S|>|MaxConjuntoIndependente então
                        MaxConjuntoIndependente ←  S
                }
        }
}
return Tamanho(S)
}
```

Para mostrar o funcionamento, apresentamos abaixo um exemplo de um grafo.



O maior conjunto independente é {2,3,4} e seu tamanho é 3.

2.4. Programa Principal

Contém a chamada da função principal e faz uso de todas as estruturas e módulos citados acima. No arquivo de entrada, temos linhas com o número de alunos A que representam os vértices e o número de alternativas N que representam as arestas.

Primeiramente, é feita a leitura de cada linha do arquivo por meio da função ***fgetline*** que retorna um vetor de string com o conteúdo de cada linha.

Trabalhamos com este vetor e uma matriz de adjacência de tamanho [N]x[N] é criada. Nessa matriz armazenamos as arestas de cada vértice. A solução do problema é direcionada para o arquivo de saída.

3. Analise de Complexidade

A função ***TestaIndependencia*** que verifica se um subconjunto é ou não uma solução possível, gera todas as combinações possíveis entre os vértices do grafo em questão. Nesse caso, temos que o número de soluções a serem avaliadas é 2^n, uma vez que o conjunto potência de um conjunto é 2 elevado ao numero de elementos do conjunto. Assim, podemos dizer que a complexidade de tempo dessa solução implementada é **O(**2^n**).**

4.Testes

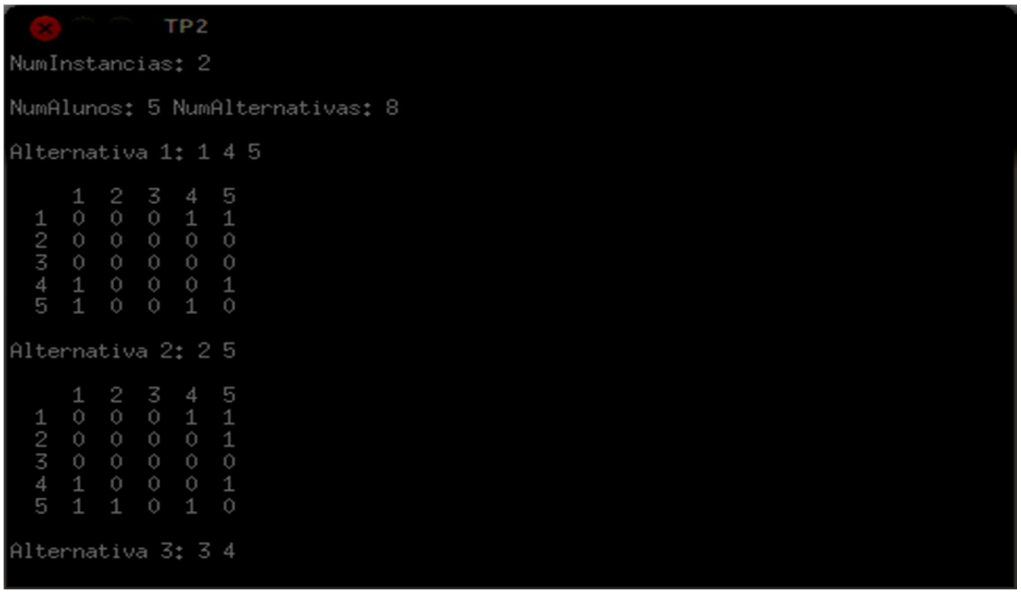
Vários testes foram realizados com o programa de forma a verificar o seu funcionamento. Os testes foram realizados em um Atom CPU D525, com 2 Gb de memória.

Criamos vários arquivos de entrada de grafos e executamos nosso programa variando o tamanho da entrada entre 10 vértices e 9 arestas até 26 vértices e 35 arestas. Medimos o tempo do usuário para cada uma das entradas. O resultado pode ser na tabela abaixo:

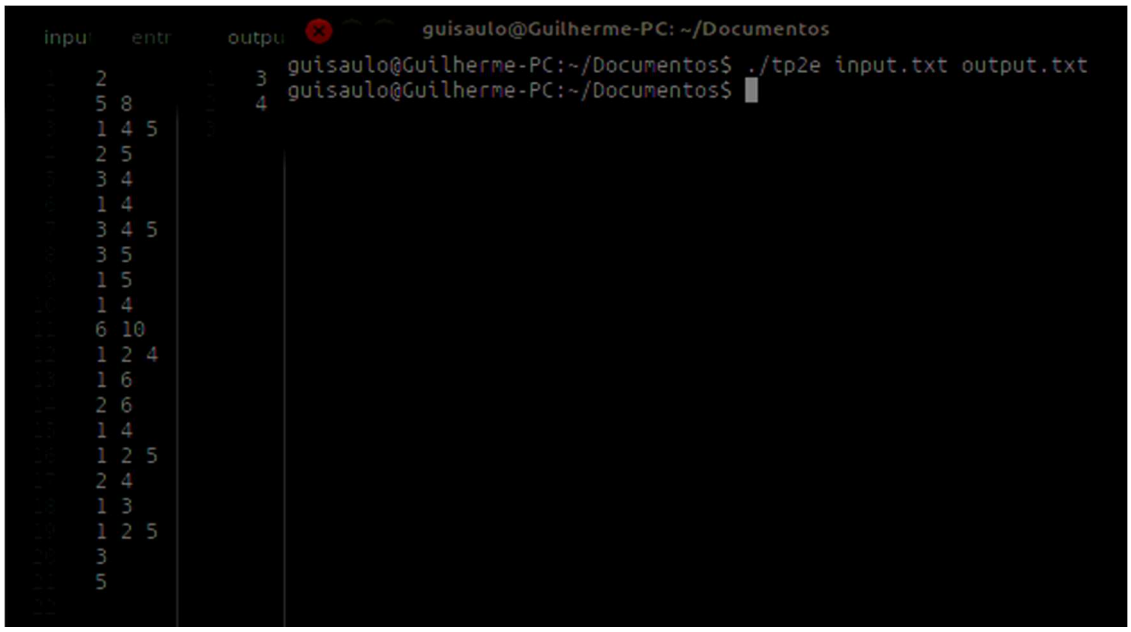
Quantidade de Vértices e Arestas	Tempo de execução _m_._seg
(V:10, A:09)	0m0.004s
(V:12, A:12)	0m0.020s
(V:14, A:15)	0m0.080s
(V:16, A:18)	0m0.440s
(V:18, A:21)	0m0.060s
(V:20, A:24)	0m10.0765s
(V:22, A:27)	0m48.712s
(V:24,A:30)	3m54.3965s
(V:26,A:35)	17m51.968s

Observando os resultados medidos da tabela, podemos perceber a natureza exponencial que o tempo de execução varia conforme a complexidade calculada.

Também foram realizados testes no CodeBlocks durante a elaboração do trabalho para verificar os valores de entrada e saída de algumas variáveis.



Para o teste da especificação do TP o resultado foi o esperado 3 e 4.



5.Conclusão

Com elaboração desse trabalho, estudamos o problema clássico da computação do Conjunto Máximo de Vértices Independentes, que é um problema NP-Completo. Para resolver esse problema foi necessário realizar um estudo mais profundo nas técnicas de programação dinâmica.

Mostramos que os possíveis algoritmos para solucionar este problema possui relação direta entre a qualidade da solução implementada e o tempo de execução.

Devido a semana corrida para realizar o trabalho, com muitas provas e trabalhos paralelos, não foi possível implementar uma heurística para aproximar o resultado para este problema.

Não foi detalhado o algoritmo para combinação de vértices e para ler cada linha do arquivo. Para mais detalhes desses algoritmos, favor analisar as referências [2], [3] e [4].

6.Referências

[1] Nívio Ziviani, Projeto de Algoritmos com implementação em Pascal e C - http://www2.dcc.ufmg.br/livros/algoritmos/ - Acessado em 29 de Março de 2014.

[2] Gerando Combinações Sem Repetição Pela Contagem Binária Em C - http://daemoniolabs.wordpress.com/2011/02/17/gerando-combinacoes-sem-repeticao-pela-contagem-binaria-em-c/ - Acessado em 10 de Abril de 2014.

[3] Gerando Permutações-r com Repetição em C- http://daemoniolabs.wordpress.com/2011/02/11/gerando-permutacoes-r-com-repeticao-em-c/ - Acessado em 10 de Abril de 2014.

[4] Ler linhas de um arquivo de texto de maneira portável - http://geekrepublic.wordpress.com/2007/09/04/ler-linhas-de-um-arquivo-de-texto-de-maneira-portavel/ - Acessado em 10 de Abril de 2014.