

**Instituto Tecnológico de Aeronáutica - ITA**

CTC-20 Estruturas Discretas para Computação

Prof. Carlos H. Q. Forster

Turma de 2016

**Primeira atividade de programação**

**Alunos: Lucas Soares Ferreira/Lucas França de Oliveira**

**Email: lucass@aluno.ita.br/**

**Objetivos**

a) Implementar um algoritmo para encontrar todos os subgrupos dada a tabela de um grupo.

b) Montar o diagrama de Hasse do reticulado formado pelos subgrupos (com a relação "ser subgrupo de").

**Introdução**

Implementou-se o projeto utilizando-se a linguagem C++. O código fonte completo pode ser encontrado no link: https://github.com/MexicanoT18/CTC-20-2016-Projeto-1.git.

Foram criadas três classes principais responsáveis por armazenar e manipular os elementos de um dado par (G, \*):

Table.h: Lê o arquivo de entrada, guarda um vetor contendo os elementos e mapeia o resultado da operação \* entre dois elementos.

Group.h: Verifica se (G, \*) corresponde de fato a um grupo; contém um vetor com os elementos do conjunto G.

Element.h: Guarda as propriedades de um elemento: valor (char) e se é válido (bool). Possui uma variável que mapeia o resultado da operação \* entre dois elementos.

HasseDiagram.h: Recebe a lista com todos os subgrupos a faz a montagem do diagrama de Hasse, assim como gera o arquivo em um formato compatível com o GraphViz.

**Leitura do arquivo de entrada**

A leitura é feita no método “readFile(const char \*path)” da classe Table.cpp. Seja G = {}, onde corresponde ao elemento neutro. Como hipótese para a matriz do arquivo de entrada, considerou-se que a propriedade do elemento neutro é válida e o j-ésimo caractere da i-ésima linha na matriz corresponde ao resultado da operação . Assim, como , os elementos de G podem ser obtidos lendo-se a primeira linha da matriz. Estes são então armazenados no vetor “vector<char> elements”.

Os valores de todas as possíveis operações , foram armazenados na variável “map<char, map<char, char> > table”. Deste modo o resultado de pode ser obtido em fazendo-se table.

Caso o formato da matriz de entrada seja inválido (não há um total de n linhas ou uma linha não possui n caracteres) será retornado um erro.

Exemplo:

Seja abaixo o conteúdo de um arquivo de entrada:

1abc

abc1

bd1a

c1ab

Então G = {‘1’, ’a’, ’b’, ’c’}, ou seja, os elementos da 1ª linha. As operações serão mapeadas na variável “table” da seguinte forma:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ai\aj | ‘1’ | ‘a’ | ‘b’ | ‘c’ |
| ‘1’ | table[ai][aj] = ‘1’ | ‘a’ | ‘b’ | ‘c’ |
| ‘a’ | ‘a’ | ‘b’ | ‘c’ | ‘1’ |
| ‘b’ | ‘b’ | ‘d’ | ‘1’ | ‘a’ |
| ‘c’ | ‘c’ | ‘1’ | ‘a’ | ‘b’ |

Como a matriz de entrada possui dimensionamento correto (4x4), “Table.cpp” não retorna erro.

Nota-se, entretanto, que (G, \*) não forma grupo. Isto será verificado posteriormente pela classe “Group.h”.

**Verificação das propriedades de grupo**

Após a leitura do arquivo, a classe “Group.cpp” analisará se os elementos de G (armazenados no vetor “vector<Element> elements”) formam um grupo.

O método “bool isClosed()” verifica a propriedade de fechamento. Se então a propriedade é válida e o método retorna “true”. Caso contrário retorna “false”.

O método “bool inverse()” verifica a propriedade do elemento inverso. então verifica-se a propriedade retornando-se “true”. Senão, retorna-se “false”.

O método “bool verifyAssociativity()” verifica a propriedade de associatividade. Se então a propriedade é válida retornando-se “true”. O valor retornado é “false” caso contrário.

(G, \*) formará um grupo somente se todos os métodos acima retornarem “true”.

Dos arquivos de testes disponíveis dois deles não formam grupo: “table 3.txt” não satisfaz a propriedade de associatividade; “table 4.txt” não satisfaz a propriedade de fechamento.

**Geração dos subgrupos**

Após a leitura e verificação da validade do grupo, executa-se um algoritmo força bruta de backtrack recursivo para gerar todos os grupos possíveis.

Os métodos “list<Group> generateSubgroups()” e “void backTrack(list<Group> & ans, int n)” implementam o algoritmo. É realizada uma verificação de fechamento do grupo a cada passo para acelerar o processo. A complexidade é de aproximadamente O(m), onde m é o número de subgrupos.

Na classe “HasseDiagram.h”, o diagrama de Hasse é construído no método “bool buildDiagram()” e escrito no arquivo pelo método “bool writeFile(char \* name)”.

O arquivo escrito pode então ser aberto no GraphViz.