PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Bacharelado em Engenharia de Software

Breno Rosa Almeida
Gabriel Victor Couto Martins de Paula
Luís Antônio de Souza e Sousa
Rúbia Coelho de Matos

Bridge finding

Belo Horizonte

2022

Breno Rosa Almeida
Gabriel Victor Couto Martins de Paula
Luís Antônio de Souza e Sousa
Rúbia Coelho de Matos

Bridge finding

Projeto de Bridge finding apresentado na disciplina Teoria dos Grafos e Computabilidade do curso de Engenharia de Software da Pontifícia Universidade Católica de Minas Gerais.

Belo Horizonte

2022

RESUMO

Palavras-chave: Matriz de Adjacência, Lista de Adjacência, Caminho Euleriano, Ponte, Conectividade, Biblioteca de Manipulação de Grafos.

SUMÁRIO

1 INTRODUÇÃO

1.1 Objetivos

Trabalho tem como objetivo fazer.

1.1.1 Objetivos específicos

2 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta “asdasd lkjlaskjd alskdjas asd sdoiosfugoigodf sps uh psdh sdip hdsoiph osdif. osidf jsodfij woerui jdsf hlieguh ioduhg eirh dlfigh epogri çsaodfji açefj sçofi jaçefoij açewfoi jwoçpefi jweaoifj”.

> sodisodfi osigu woriu çdoig çsoigj eçorig oeçrgji oier. psdh sdip hdsoiph osdasda qwre qwed asd qe asd q asd q asd qw asd q asd qe asd qwe asd qwe asd if. osidf jsodfij woerui jdsf hlieguh ioduhg eirh dlfigh epogri çsaodfji açefj sçofi jaçefoij açewfoi jwoçpefi jweaoifj. (AUTOR, ano).

2.1 xxxx

2.2 xxxx

3 DESENVOLVIMENTO

3.1 Biblioteca de manipulação de grafos

Como primeira etapa foi desenvolvida uma biblioteca de manipulação de grafos que será utilizada futuramente nas próximas etapas.

A biblioteca de manipulação é composta por um conjunto de funções e métodos que manipulam tanto um grafo representado em matriz, quanto um grafo representado por lista de adjacência.

Para um bom design de solução, que fosse reutilizável e compreensível, foi utilizada a linguagem Java como principal ferramenta. A arquitetura desenvolvida para a biblioteca, nos dá a liberdade para escolher qual estrutura de dados manipular e utilizar de módulos de geração, leitura e salvamento de grafos de maneira separada.

A Biblioteca como um todo é composta por duas classes principais que guardam as estruturas de dados de Grafos em matriz e em lista de adjacência. A primeira classe é chamada de GraphMatrix, responsável por manipular e armazenar um grafo não-direcionado em uma matriz de N linhas por N colunas tal que N é o número de vértices do grafo. A segunda classe se chama apenas Graph e ela herda todas as características de GraphMatrix. Graph, é responsável por manipular e armazenar tanto grafo em matriz, quanto o grafo em lista de adjacência.

Além das classes principais, existem outros dois módulos que auxiliam nos testes e desenvolvimento. Sendo o primeiro deles chamado de GraphIO, utilizado na leitura e salvamento de grafos em arquivos no formato Pajek NET, e o segundo módulo, chamado de GraphGenerator, responsável por gerar grafos com base na quantidade de vértices, número mínimo de grau por vértice e número máximo de grau por vértice.

Para garantir que todos os algoritmos estão funcionando corretamente e retornando resultados esperados, foram desenvolvidas baterias de teste para cada uma das funções do sistema. Assim, é possível ter certeza e confiar que tanto o processamento que ocorre nas matrizes, quanto os que ocorrem nas listas entregam os mesmos resultados de adição e remoção de arestas, ponderação de vértices e arestas, e para todas as outras funções. Os testes foram escritos utilizando a biblioteca Junit do próprio Java.

A Organização de todo o sistema pode ser representada pelo seguinte diagrama de classe:

Figura 1 – Diagrama de Classe

3.2 Ciclo Euleriano

Passa uma única vez por cada aresta no grafo, partindo e chegando a um mesmo vértice.

- Para haver um ciclo euleriano, todo vértice deve ter grau par, como já discutimos.

O algoritmo de Fleury, proposto em 1883, utiliza um grafo reduzido induzido pelas arestas ainda não marcadas

- Inicialmente todas as arestas estão não marcadas.
- As arestas vão sendo "marcadas", ou removidas do grafo, a medida em que vão sendo inseridas no ciclo

Regra da ponte: se uma aresta v, w é uma ponte no grafo reduzido, então v, w só deve ser escolhida caso não haja outra opção.

3.3 Algoritmo de Fleury

Passo a passo para um grafo não dirigido e não valorado

1. Verifique se o grafo apresenta as condições para ter um ciclo euleriano
2. Caso positivo, escolha um vértice V1 para começar.
3. Entre os vértices adjacentes a V1, faça
   (a) Se há apenas um vértice como opção, escolha este como V2.
   (b) Se há mais de um vértice possível, escolha um V2 apropriado dentre eles (ou seja, um que “não repita a ponte”).
4. Remova a aresta (V1, V2).
5. Se ainda houver arestas não percorridas, volte ao passo 3, partindo agora de V2 (V2 é o novo V1).
6. Caso contrário, imprima o caminho percorrido.

Exemplos do passo a passo do algoritmo funcionando:

Método de Fleury – Exemplo 1

1

Método de Fleury – Exemplo 1

1

Método de Fleury – Exemplo 1

1 / 2

Método de Fleury – Exemplo 1

1 / 2

Método de Fleury – Exemplo 1

1 / 2 / 3

Método de Fleury – Exemplo 1

1 / 2 / 3

Método de Fleury – Exemplo 1

1 / 2 / 3 / 4

Método de Fleury – Exemplo 1

1 / 2 / 3 / 4

Método de Fleury – Exemplo 1

1 / 2 / 3 / 4 / 2

Método de Fleury – Exemplo 1

1 / 2 / 3 / 4 / 2

Método de Fleury – Exemplo 1

1 / 2 / 3 / 4 / 2 / 5

Método de Fleury – Exemplo 1

1 / 2 / 3 / 4 / 2 / 5

Método de Fleury – Exemplo 1

1 / 2 / 3 / 4 / 2 / 5 / 6

Método de Fleury – Exemplo 1

1 / 2 / 3 / 4 / 2 / 5 / 6

Método de Fleury – Exemplo 1

1 / 2 / 3 / 4 / 2 / 5 / 6 / 4

Método de Fleury – Exemplo 1

1 / 2 / 3 / 4 / 2 / 5 / 6 / 4

Método de Fleury – Exemplo 1

1 / 2 / 3 / 4 / 2 / 5 / 6 / 4 / 5

Método de Fleury – Exemplo 1

1 / 2 / 3 / 4 / 2 / 5 / 6 / 4 / 5

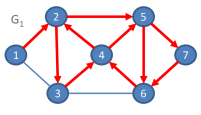
Método de Fleury – Exemplo 1

1 / 2 / 3 / 4 / 2 / 5 / 6 / 4 / 5 / 7

Método de Fleury – Exemplo 1

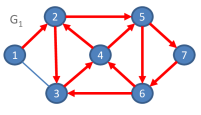
1 / 2 / 3 / 4 / 2 / 5 / 6 / 4 / 5 / 7

Método de Fleury – Exemplo 1



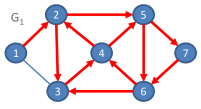
1 / 2 / 3 / 4 / 2 / 5 / 6 / 4 / 5 / 7 / 6

Método de Fleury – Exemplo 1



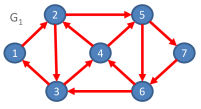
1 / 2 / 3 / 4 / 2 / 5 / 6 / 4 / 5 / 7 / 6

Método de Fleury – Exemplo 1



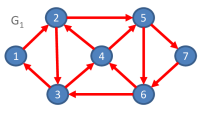
1 / 2 / 3 / 4 / 2 / 5 / 6 / 4 / 5 / 7 / 6 / 3

Método de Fleury – Exemplo 1



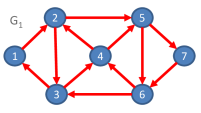
1 / 2 / 3 / 4 / 2 / 5 / 6 / 4 / 5 / 7 / 6 / 3

Método de Fleury – Exemplo 1



1 / 2 / 3 / 4 / 2 / 5 / 6 / 4 / 5 / 7 / 6 / 3 / 1

Método de Fleury – Exemplo 1



1 / 2 / 3 / 4 / 2 / 5 / 6 / 4 / 5 / 7 / 6 / 3 / 1 → **Ciclo euleriano**

4 PRIMEIRO CAPÍTULO DE EXEMPLO

4.1 Primeira seção

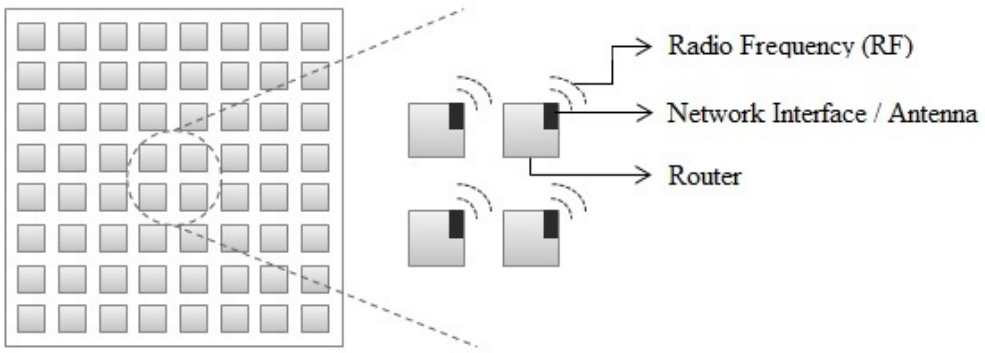
4.1.1 Primeira subseção

4.2 Segunda seção

5 SEGUNDO CAPÍTULO DE EXEMPLO

As figuras devem ser apresentadas pelos comandos abaixo. O parâmetro *width* determina o tamanho que a figura será exibida. No parâmetro *caption* o texto que aparece entre colchetes será o exibido no índice de figuras e o texto contido entre chaves será exibido na legenda da figura. Para citar a figura o comando ref deve ser usado juntamente com o label, como é mostrado nesse exemplo da Figura **??**.

Figura 15 – Principais componentes de WiNoCs



Os comandos abaixo são usados para apresentação de gráficos. A diferença está apenas na definição do tipo “grafico“ que permite a adição dos itens no índice de gráficos de forma automática. Os parâmetros são semelhantes aos usados para representação de figuras. O parâmetro *width* determina o tamanho do gráfico. O texto entre colchetes no *caption* será o exibido no índice de gráficos e o texto contido entre chaves será exibido na legenda.

Gráfico 1 – Percentual de pacotes enviados

Fonte: Dados da pesquisa

Um exemplo de criação de tabela é mostrado a seguir. As colunas são separadas por elementos & e as linhas por duas barras invertidas. Os comandos *hline* e | definem a criação de linhas e colunas para separar os conteúdos, respectivamente. A tabela pode ser referenciada usando o comando ref juntamente com o label, como na Tabela **??**.

Tabela 1 – Parâmetros definidos por classe

Benchmark	Parâmetro	Classe S	Classe W	Classe A	Classe B	Classe C	Classe D
BT	*Grid*	12^3	24^3	64^3	102^3	162^3	408^3
CG	Linhas	1400	7000	14000	75000	150000	1500000
EP	Pares	2^{24}	2^{25}	2^{28}	2^{30}	2^{32}	2^{36}
FT	*Grid*	64^3	$128^2 * 32$	$256^2 * 128$	$512 * 256^2$	512^3	$2048 * 1024^2$
IS	Chaves	2^{16}	2^{20}	2^{23}	2^{25}	2^{27}	2^{31}
LU	*Grid*	12^3	33^3	64^3	102^3	162^3	408^3
MG	*Grid*	32^3	128^3	256^3	256^3	512^3	1024^3
SP	*Grid*	12^3	36^3	64^3	102^3	162^3	408^3

6 OBSERVAÇÕES IMPORTANTES

Este documento foi compilado em ambiente linux (Ubuntu 10.04) usando o programa Kile - an Integrated LaTeX Environment - Version 2.0.85. Para correta formatação os seguintes arquivos do pacote *abntex* devem ser alterados.

a) Arquivo abnt.cls
   No Ubuntu o arquivo fica armazenado em */usr/share/texmf/tex/latex/abntex*. Comentar a linha 967: Linha comentada para reduzir o espaçamento entre o topo da página e o título. Alterar a linha 1143: Parâmetro alterado de 30pt para -30pt para reduzir o espaçamento entre o top da página e o título do apêndice. Alterar a linha 985: Parâmetro alterado de 0pt para -30pt para reduzir o espaçamento entre o top da página e o título. Alterar a linha 991: Parâmetro alterado de 45pt para 30pt para reduzir o espaçamento entre o texto e o título.
b) Arquivo acronym.sty
   No Ubuntu o arquivo fica armazenado em */usr/share/texmf-texlive/tex/latex/acronym*. Alterar a linha 225: Inserir o separador – entre acrônimo/descrição e remover o negrito com o *normalfont*.