Notas em Matemática Aplicada e-ISSN 2236-5915

Volume 24, 2012

Editado por

Eliana X.L. de Andrade
Universidade Estadual Paulista - UNESP
São José do Rio Preto, SP, Brasil

Rubens Sampaio
Pontifícia Universidade Católica do Rio de Janeiro -
Rio de Janeiro, RJ, Brasil

Geraldo N. Silva
Universidade Estadual Paulista - UNESP
São José do Rio Preto, SP, Brasil

Sociedade Brasileira de Matemática Aplicada e Computacional

2012

A Sociedade Brasileira de Matemática Aplicada e Computacional - SBMAC publica, desde as primeiras edições do evento, monografias dos cursos que são ministrados nos CNMAC.

Para a comemoração dos 25 anos da SBMAC, que ocorreu durante o XXVI CNMAC em 2003, foi criada a série **Notas em Matemática Aplicada** para publicar as monografias dos minicursos ministrados nos CNMAC, o que permaneceu até o XXXIII CNMAC em 2010.

A partir de 2011, a série passa a publicar, também, livros nas áreas de interesse da SBMAC. Os autores que submeterem textos à série Notas em Matemática Aplicada devem estar cientes de que poderão ser convidados a ministrarem minicursos nos eventos patrocinados pela SBMAC, em especial nos CNMAC, sobre assunto a que se refere o texto.

O livro deve ser preparado em **Latex (compatível com o Miktex versão 2.7), as figuras em eps** e deve ter entre **80 e 150 páginas**. O texto deve ser redigido de forma clara, acompanhado de uma excelente revisão bibliográfica e de **exercícios de verificação de aprendizagem** ao final de cada capítulo.

Veja todos os títulos publicados nesta série na página
http://www.sbmac.org.br/notas.php

SBMAC **Sociedade Brasileira de Matemática Aplicada e Computacional**

2012

REPRESENTAÇÕES COMPUTACIONAIS DE GRAFOS

Lilian Markenzon
markenzon@nce.ufrj.br
Oswaldo Vernet
oswaldo@nce.ufrj.br

Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro

Sociedade Brasileira de Matemática Aplicada e Computacional

São Carlos - SP, Brasil
2012

Coordenação Editorial: Sandra Augusta Santos

Coordenação Editorial da Série: Eliana Xavier Linhares de Andrade

Editora: SBMAC

Capa: Matheus Botossi Trindade

Patrocínio: SBMAC

Catalogação elaborada pela Biblioteca do IBILCE/UNESP
Bibliotecária: Maria Luiza Fernandes Jardim Froner

Markenzon, Lilian
Representações Computacionais de Grafos
- São Carlos, SP: SBMAC, 2012, 78 p.; 20,5cm.
- (Notas em Matemática Aplicada; v. 24)

e-ISBN 978-85-86883-89-7

1. Grafos. 2. Representações. 3. Códigos.
I. Markenzon, Lilian. II. Vernet, Oswaldo. III. Título. IV. Série.

CDD - 51

Esta é uma republicação em formato de e-book do livro original do mesmo título publicado em 2006 nesta mesma série pela SBMAC.

Conteúdo

Prefácio

Um dos aspectos mais importantes na implementação em computador de algoritmos em grafos é a escolha de uma representação adequada para os mesmos. A decisão sobre como armazenar o grafo na memória precede a implementação propriamente dita do algoritmo em uma linguagem de alto nível, uma vez que, dependendo da forma escolhida, as operações sobre o grafo terão implementações distintas. Conseqüentemente, uma escolha bem feita pode representar substancial melhoria na complexidade final do algoritmo, tanto do ponto de vista do espaço ocupado quanto do tempo de execução.

No entanto, os currículos regulares dos cursos de matemática, computação e engenharia, tanto de graduação quanto de pós-graduação, nem sempre contemplam, com os devidos detalhes, a teoria algorítmica de grafos nas disciplinas que oferecem. Por esta razão, julgamos de interesse a apresentação de um texto sobre representações computacionais de grafos.

No Capítulo 1, são introduzidos conceitos básicos em teoria de grafos, necessários ao entendimento do restante do texto. As representações tradicionais para grafos arbitrários são aqui revistas. Além delas, estudam-se representações mais compactas para famílias de grafos satisfazendo restrições.

No Capítulo 2, tratamos da codificação de Prüfer para árvores, que permite representá-las através de uma seqüência de vértices. A importância histórica desta representação reside no fato de ela haver sido definida em 1918 de forma puramente matemática, sem relação alguma com propósitos computacionais. O código obtido é utilizado na solução de problemas sobre geração e contagem para árvores irrestritas e com restrições.

No Capítulo 3, para os grafos cordais, apresentamos a representação por conjuntos de adjacência restritos, baseada no conceito de esquema de eliminação perfeita, que é fundamental na caracterização da família. Através desta representação, determinam-se eficientemente, para um grafo cordal, o tamanho da maior clique, o número cromático, uma coloração ótima, um conjunto independente máximo e uma cobertura mínima por cliques.

No Capítulo 4, a família dos grafos k-caminho, subfamília dos cordais, é apresentada e um código para a família é determinado. Este código é mais compacto do que a representação estudada no capítulo anterior.

Como pré-requisitos ao acompanhamento do presente texto, esperamos do leitor certa familiaridade com estruturas de dados básicas (listas em suas diversas moda-

lidades). Todos os algoritmos são apresentados em pseudo-código, de forma que a implementação em linguagens de programação de alto nível se dê com relativa facilidade. São igualmente requeridos conhecimentos acerca das complexidades de tempo e espaço de um algoritmo, expressas nas notações usuais O, Ω e Θ. No que concerne ao conhecimento matemático, a teoria básica de conjuntos e funções, bem como noções de combinatória elementar são suficientes.

Para finalizar, gostaríamos de prestar nossos sinceros agradecimentos à SBMAC, pelo espaço concedido para divulgação deste material, fruto de nossa experiência como professores da área. Este trabalho recebeu o apoio do CNPq através da bolsa 301068/2003-8 concedida ao primeiro autor.

Rio de Janeiro, 28 de junho de 2006.

Lilian Markenzon
Oswaldo Vernet

Capítulo 1

Introdução

Neste capítulo são introduzidos alguns conceitos básicos relativos à teoria de grafos e estudadas as principais representações.

1.1 Conceitos Básicos

Um *grafo não orientado*, ou simplesmente *grafo*, é um par $G = (V, E)$, onde V é um conjunto de *vértices* e E é um conjunto de *arestas*; cada aresta é um subconjunto de V com 1 ou 2 vértices. Dada uma aresta $\{v, w\} \in E$, os vértices v e w são denominados *extremidades* da aresta; dizemos também que a aresta *incide* sobre os vértices v e w. Uma aresta unitária, da forma $\{v\}$, é denominada *laço*.

Os grafos que consideramos neste texto são finitos (possuem um número finito de vértices) e não possuem laços. Usaremos sempre as convenções $n = |V|$ e $m = |E|$.

Geometricamente podemos visualizar um grafo através de um conjunto de pontos sobre uma superfície, representando os vértices; arestas correspondem a segmentos de curvas interligando os pontos que representam suas extremidades.

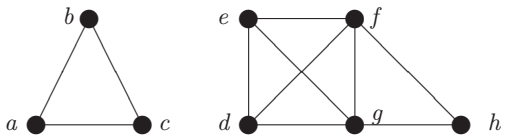


Figura 1.1: Representação geométrica do grafo $G = (V, E)$

Na Figura 1.1 vemos representado o grafo $G = (V, E)$, onde o conjunto de vértices $V = \{a, b, c, d, e, f, g, h\}$ e o conjunto de arestas $E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{d, e\}, \{d, f\}, \{d, g\}, \{e, f\}, \{e, g\}, \{f, g\}, \{f, h\}, \{g, h\}\}$.

Dois vértices são *adjacentes* ou *vizinhos* quando são extremidades de uma aresta. Os vizinhos de um determinado vértice $v \in V$ integram o *conjunto de adjacência*

$$Adj_G(v) = \{w \in V \mid \{v, w\} \in E\}.$$

A cardinalidade $|Adj_G(v)|$ é denominada *grau* de v, denotado por $d_G(v)$. O subscrito G será omitido sempre que o grafo estiver subentendido no contexto. No grafo da Figura 1.1, o vértice f tem como vizinhos $Adj(f) = \{d, e, g, h\}$ e seu grau é $d(f) = 4$.

Um *caminho* em $G = (V, E)$ é uma seqüência constituída por $k > 0$ vértices $[v_1, v_2, \ldots, v_k]$ tal que: ou $k = 1$ e o caminho é unitário ou $k > 1$ e $\{v_i, v_{i+1}\} \in E$, para $i = 1, \ldots, k-1$. Em ambos os casos, o *comprimento* do caminho é $k - 1$. Um caminho é dito *simples* se os vértices que o constituem são todos distintos. Um *ciclo* é um caminho de comprimento maior ou igual a 3 em que o primeiro e o último vértices coincidem. Um *ciclo simples* é um caminho de comprimento maior ou igual a 3 em que somente o primeiro e o último vértices coincidem. Um grafo que não possui ciclos é dito *acíclico*. Um *caminho* (*ciclo*) *hamiltoniano* é um caminho (ciclo) simples que contém todos os vértices do grafo. No grafo da Figura 1.1: $[f, h, g, d, f, e]$ é um caminho não simples (o vértice f se repete) de comprimento 5, $[f, h, g, d, e]$ é um caminho simples de comprimento 4 e $[d, f, h, g, e, d]$ é um ciclo simples de comprimento 5.

Um grafo $G' = (V', E')$ é *subgrafo* de $G = (V, E)$ quando $V' \subseteq V$ e $E' \subseteq E$. Dado um subconjunto de vértices $S \subseteq V$, denominamos

$$G[S] = (S, \{\{x, y\} \in E \mid x \in S \wedge y \in S\})$$

o *subgrafo de G induzido por S*. Em outras palavras, $G[S]$ é o subgrafo de G que tem S como conjunto de vértices e possui todas as arestas de E com extremidades pertencentes a S. Dizemos que S é uma *clique* quando $G[S]$ for um *grafo completo*, isto é, possuir todas as arestas possíveis. Uma t-clique, $t \geq 1$, é uma clique de cardinalidade t. Na figura 1.1, o subgrafo induzido por $S = \{d, f, g, h\}$ é

$$G[S] = (\{d, f, g, h\}, \{d, f\}, \{d, g\}, \{f, g\}, \{g, h\}),$$

que não é completo. Já $S = \{a, b, c\}$ é uma 3-clique, pois induz um subgrafo completo.

Um grafo é *conexo* quando existir pelo menos um caminho entre cada par de vértices; do contrário, é dito *desconexo*. O grafo da Figura 1.1 é desconexo, possuindo duas *componentes conexas*.

O Teorema 1.1 nos diz que o somatório das cardinalidades dos conjuntos de adjacência referentes a todos os vértices de um grafo é igual ao dobro do número de arestas.

Teorema 1.1. *Para qualquer grafo $G = (V, E)$, vale a igualdade:*

$$\sum_{v \in V} |Adj(v)| = 2m.$$

Demonstração: Basta ver que cada aresta contribui com 2 unidades para o somatório dos graus. □

1.2 Famílias de Grafos e o Problema do Reconhecimento

O conjunto de todos os grafos que satisfazem uma dada propriedade constitui uma *família* de grafos. Existem, na literatura, diversas famílias que vêm sendo, ao longo do tempo, exaustivamente estudadas: os grafos planares (aqueles que podem ser desenhados sobre uma superfície plana sem que as arestas se cruzem), os grafos hamiltonianos (aqueles que possuem um ciclo simples contendo todos os seus vértices), dentre outros.

Normalmente a definição de uma família consiste em mencionar a propriedade satisfeita pelos grafos que a constituem, exatamente como fizemos nesses dois exemplos. Em vários casos, esta propriedade pode ser acrescida de novas condições, originando propriedades mais específicas, que definem *subfamílias*. Tomemos, por exemplo, a família dos grafos conexos, que satisfazem a propriedade de haver pelo menos um caminho entre qualquer par de vértices. Se acrescentarmos a essa propriedade a condição de o grafo não possuir ciclos, definimos a família das *árvores*, uma subfamília da família dos grafos conexos constituída por grafos que, além de conexos, são acíclicos.

Quando uma nova família de grafos é definida, um importante problema computacional a ser resolvido é o do *reconhecimento*, que consiste em obter um algoritmo para verificar se um grafo qualquer, dado como entrada, pertence à família em questão. A saída de um algoritmo de reconhecimento é, portanto, de natureza booleana: sim ou não.

Em alguns casos, verificar diretamente se o grafo dado satisfaz ou não a propriedade que define a família conduz a algoritmos de reconhecimento ineficientes ou mesmo impraticáveis de serem implementados. Isto ocorre, por exemplo, com a família dos grafos planares (aqueles que podem ser desenhados em uma superfície plana sem cruzamentos de arestas). Utilizar esta propriedade certamente não conduz a um algoritmo de reconhecimento. Devem, portanto, ser pesquisadas *caracterizações adicionais* para a família, demonstrando-se novas propriedades equivalentes àquela utilizada na definição. No caso dos grafos planares, a primeira destas caracterizações adicionais que surgiu foi o famoso Teorema de Kuratowski.

1.3 Esquemas de Representação para Grafos

Vimos, na seção inicial deste capítulo, que um grafo é definido através de um par de conjuntos: ao primeiro pertencem os vértices e ao segundo, as arestas. Portanto, para explicitar um grafo segundo a definição, devem ser mencionados $n+2m$ símbolos, correspondentes aos n vértices e às m arestas (pares de vértices).

Um *esquema de representação* constitui-se de regras que conduzem a uma maneira alternativa de individualizar grafos, na qual se evita a menção explícita dos conjuntos de vértices e arestas. O resultado obtido pela aplicação de um esquema

a um grafo é chamado *representação* para o grafo. A partir dela, os conjuntos de vértices e arestas que definem o grafo sendo representado devem poder ser deduzidos sem ambigüidades. Existem esquemas de representação gerais, aplicáveis a grafos arbitrários (que não satisfazem nenhuma propriedade em especial) e esquemas específicos para determinadas famílias, que só podem ser utilizados com os grafos que as integram.

Nesta seção, analisamos três esquemas de representação bastante conhecidos para grafos arbitrários. A discussão sobre esquemas específicos será tema da Seção 1.4.

Esquema de Representação Geométrico

O esquema de representação mais usual para grafos arbitrários é o *geométrico*, segundo o qual uma representação para um grafo $G = (V, E)$ é obtida pela aplicação das seguintes regras:

- escolha uma superfície sobre a qual o grafo será representado;
- escolha n pontos distintos sobre esta superfície, nomeando-os de acordo com os vértices especificados em V;
- para cada aresta mencionada em E, una os pontos correspondentes às suas extremidades através de um segmento arbitrário de curva sobre a superfície escolhida.

A representação de um grafo através do esquema geométrico é, portanto, um desenho, a partir do qual vértices e arestas podem ser visualizados e os relacionamentos por eles expressos melhor compreendidos. Evidentemente, um mesmo grafo admite inúmeras representações segundo este esquema.

A área de pesquisa denominada *Traçado Automático de Grafos* tem por objetivo a concepção de algoritmos que produzem, para os grafos fornecidos como entrada, representações geométricas que obedecem a critérios diversos, alguns puramente geométricos, outros estéticos.

Esquema de Representação por Conjuntos de Adjacência

A representação de um grafo $G = (V, E)$ por conjuntos de adjacência consiste em mencionar, para todo vértice $v \in V$, o conjunto $Adj(v)$, construindo um conjunto de pares da forma $\{(v, Adj(v)) \mid v \in V\}$. Vale observar que esta representação é única para um dado grafo.

Em uma representação obtida por este esquema, devem ser mencionados, ao todo,

$$\sum_{v \in V} [1 + |Adj(v)|] = n + \sum_{v \in V} |Adj(v)| = n + 2m$$

símbolos, o que coincide exatamente com a quantidade mencionada na definição do grafo.

Esquema de Representação por Matriz de Adjacência

É também possível representar um grafo $G = (V, E)$ por uma *matriz de adjacência*, que é uma matriz simétrica 0–1 $\mathcal{A}_{n\times n}$ obtida da seguinte maneira:

- defina uma bijeção $\omega : \{1, \ldots, n\} \to V$, o que corresponde a numerar seqüencialmente os vértices a partir de 1;
- para $1 \leq i < j \leq n$, defina

$$\mathcal{A}(i,j) = \mathcal{A}(j,i) = \begin{cases} 1, & \text{se } \{\omega(i), \omega(j)\} \in E; \\ 0, & \text{se } \{\omega(i), \omega(j)\} \notin E. \end{cases}$$

Em outras palavras, recebem o valor 1 as entradas da matriz para as quais existe aresta entre o par de vértices correspondente; as demais recebem o valor 0. A representação obtida por este esquema exige a menção de n^2+n símbolos, correspondendo às entradas da matriz e à função ω.

Quando $V = \{1, \ldots, n\}$, a bijeção mais apropriada a considear é, evidentemente, a função identidade. Neste caso, ω não precisa ser representada, sendo economizados n símbolos.

1.4 Esquemas Específicos de Representação

Para algumas famílias particulares, é possível conceber esquemas que conduzem a representações mais compactas do que as obtidas através dos esquemas analisados para grafos arbitrários.

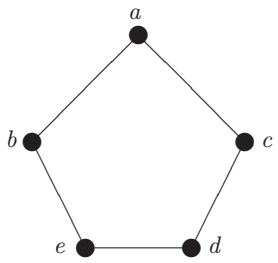
Consideremos, por exemplo, a subfamília dos grafos conexos cujos vértices possuem grau 2. Uma caracterização equivalente seria dizer que os grafos integrantes desta família são ciclos simples desprovidos de arestas interiores (cordas). Logo, grafos com n vértices pertencentes a esta família possuem exatamente n arestas. O esquema de representação por conjuntos de vizinhos exige, portanto, a menção de $n + 2n = 3n$ elementos. Um esquema alternativo mais compacto para representar tais grafos é utilizar uma seqüência de n vértices, dispostos na mesma ordem em que figuram no ciclo.

As representações oriundas dos quatro esquemas discutidos são ilustradas a seguir para um ciclo com 5 vértices a, b, c, d e e. Observe que a representação através de uma seqüência de n vértices não é única; o mesmo grafo poderia ser representado por outras seqüências, como $[a, b, e, d, c]$ ou $[c, d, e, b, a]$. Tampouco as representações geométrica ou por matriz de adjacência são únicas.

Definição do grafo:

$$G = (\{a, b, c, d, e\}, \{\{a, c\}, \{c, d\}, \{d, e\}, \{e, b\}, \{b, a\}\})$$

Representação Geométrica:



Representação por Conjuntos de Adjacência:

$$\{(a, \{b, c\}), (b, \{a, e\}), (c, \{a, d\}), (d, \{c, e\}), (e, \{b, d\})\}$$

Representação por Matriz de Adjacência:

$$\omega = [a, b, c, d, e] \quad \mathcal{A} = \left(\begin{array}{c|ccccc} & a & b & c & d & e \\ \hline a & 0 & 1 & 1 & 0 & 0 \\ b & 1 & 0 & 0 & 0 & 1 \\ c & 1 & 0 & 0 & 1 & 0 \\ d & 0 & 0 & 1 & 0 & 1 \\ e & 0 & 1 & 0 & 1 & 0 \end{array}\right)$$

Representação Específica:

$$[a, c, d, e, b]$$

É fundamental que, a partir de uma dada representação, a definição original do grafo possa ser restituída sem ambigüidades. Consideremos, no exemplo anterior, a representação específica através de uma seqüência. A partir dela, a definição original do grafo pode ser obtida da seguinte maneira:

- o conjunto de vértices é composto por todos os elementos da seqüência;
- o conjunto de arestas é constituído por $n - 1$ subconjuntos contendo elementos consecutivos da seqüência e um subconjunto constituído pelo primeiro e último elementos.

1.5 Armazenamento de Grafos em Memória Principal

A utilização computacional de grafos, quer como objeto principal a ser manipulado por um algoritmo, quer como estrutura de dados auxiliar, exige o armazenamento dos mesmos total ou parcial na memória principal do computador. Cabe ao implementador escolher, dentre as representações possíveis para os grafos com que ele

vai trabalhar, aquela que melhor se adequa às necessidades do algoritmo a ser implementado. É evidente que a *compacidade* é um fator importante, principalmente se o grafo a armazenar possui milhares de vértices ou reduzida densidade (relação m/n). Entretanto, dependendo das operações que o algoritmo necessite realizar sobre o grafo, a economia de memória pode ser sacrificada em prol de um melhor desempenho para algumas dessas operações.

Como exemplo, citamos os algoritmos que precisam decidir, em determinados pontos, se existe uma aresta interligando um dado par de vértices. Sendo o grafo representado por conjuntos de adjacência, esta operação será traduzida por um percurso sobre o conjunto de vizinhos de um dos vértices dados, à procura do outro vértice. A mesma operação traduz-se simplesmente na inspeção de uma entrada da matriz de adjacência, caso o grafo esteja assim armazenado. Embora a primeira representação requeira menos memória, a segunda pode ser preferida se tal verificação ocorrer com significativa freqüência durante a execução do algoritmo.

As duas formas de armazenamento que estudaremos derivam diretamente dos esquemas de representação apresentados na seção anterior.

Armazenamento por Listas de Adjacência

O armazenamento em memória principal por *listas de adjacência* decorre do esquema de representação por conjuntos de adjacência. Consiste em uma estrutura ortogonal em que os nós da lista principal (vertical) armazenam informações sobre os vértices do grafo; de cada um deles emerge uma lista secundária (horizontal), que possui um nó para cada vizinho. Na Figura 1.2 vemos uma ilustração do armazenamento por listas de adjacência do grafo da Figura 1.1.

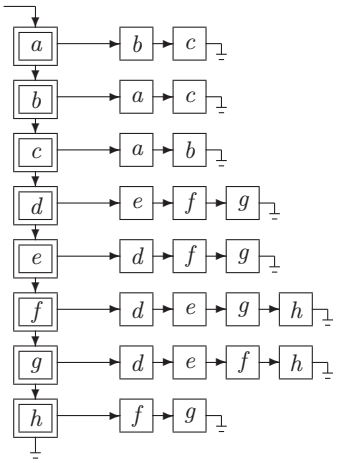


Figura 1.2. Armazenamento do grafo da Figura 1.1 por listas de adjacência

Em uma implementação real, é evidente que os nós das listas secundárias devem conter não os nomes dos vértices vizinhos, mas ponteiros para os nós da lista

principal que os representam. A lista principal poderá ser seqüencial ou encadeada, dependendo do tipo de problema a ser tratado: se o grafo sofre, por exemplo, acréscimo de vértices ao longo do processamento, pode ser interessante optar pela organização encadeada.

Se o grafo a ser armazenado possui n vértices e m arestas, haverá exatamente n nós na lista principal e $2m$ nós nas listas secundárias. O espaço consumido é, portanto, $O(n+m)$.

Armazenamento por Matriz de Adjacência

O armazenamento por *matriz de adjacência* consiste em manter na memória principal a matriz simétrica 0–1 $\mathcal{A}_{n\times n}$ obtida através do esquema de representação já estudado. Se $V = \{1, \ldots, n\}$ for o conjunto de vértices do grafo, não é necessário armazenar a correspondência entre os vértices e seus números; do contrário, esta informação deverá ser mantida em uma estrutura à parte.

A rigor, $\mathcal{A}$ deveria ser tratada como uma matriz de *bits*, já que os valores possíveis para suas entradas são apenas 0 e 1. Entretanto, como a maioria das linguagens de programação de alto nível não suporta confortavelmente a definição e o acesso a variáveis de tamanho inferior a um *byte*, na prática, $\mathcal{A}$ é implementada como uma matriz de n^2 *bytes*.

Por se tratar de uma matriz simétrica cuja diagonal principal contém zeros (pois estamos supondo que os grafos aqui tratados não possuem laços), é possível dispensar o armazenamento de $\frac{(n^2+n)}{2}$ elementos de $\mathcal{A}$, mantendo em um vetor na memória apenas os $\frac{(n^2-n)}{2}$ elementos subdiagonais, como ilustra a Figura 1.3.

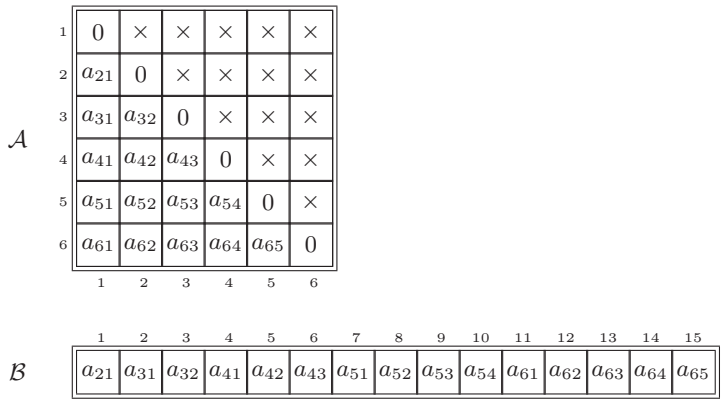


Figura 1.3. Armazenamento compacto da matriz de adjacência, quando $n = 6$

A idéia é armazenar consecutivamente em um vetor $\mathcal{B}$ de $\frac{(n^2-n)}{2}$ posições as porções subdiagonais das linhas de $\mathcal{A}$. Dados os valores de i e j, sendo $1 \leq j < i \leq n$,

relativos a um elemento de $\mathcal{A}$, a posição correspondente no vetor $\mathcal{B}$ é obtida pela expressão:

$$k = \frac{(i-1)(i-2)}{2} + j.$$

Reciprocamente, dado k, determinamos i e j da seguinte maneira:

$$i = \left\lfloor \frac{3 + \sqrt{8k-7}}{2} \right\rfloor \quad \text{e} \quad j = k - \frac{(i-1)(i-2)}{2}.$$

A quantidade de memória ocupada pela matriz de adjacência claramente independe do número de arestas do grafo, consumindo espaço $O(n^2)$.

1.6 Leitura de um Grafo para a Memória

Um problema freqüente enfrentado por programadores que implementam algoritmos em grafos é o da inicialização das estruturas de armazenamento para um grafo com informações oriundas de fonte externa, normalmente um arquivo ou de uma base de dados.

O ideal é que a leitura do grafo, acompanhada da respectiva inicialização das estruturas que o armazenam, possa ser implementada em tempo $O(n+m)$, proporcional ao tamanho do grafo. Quando se utilizam listas de adjacência, esta complexidade pode ser facilmente garantida se toda inserção nas listas secundárias (horizontais) for realizada em tempo $O(1)$, o que pode ser conseguido, por exemplo, inserindo-se novos nós sempre no início destas listas.

Entretanto, quando a representação escolhida é por matriz de adjacência, é necessário inicializar as n^2 posições, atribuindo o valor 1 àquelas que correspondem a vértices interligados e 0 às demais. Assim sendo, qualquer algoritmo que utilize esta representação possuirá um passo inicial de complexidade $\Theta(n^2)$, acarretando $\Omega(n^2)$ para a complexidade total, o que é indesejável quando se representam grafos esparsos.

Um engenhoso raciocínio, proposto por Aho, Hopcroft e Ullman, permite evitar a inicialização das posições que contêm zeros. A idéia é trabalhar com uma matriz simétrica $\mathcal{M}_{n\times n}$ de inteiros (em vez de *bytes*), juntamente com uma lista seqüencial $\mathcal{L}$, denominada *lista de certificados*, cujos elementos são subconjuntos de 2 vértices. A leitura do grafo, supondo $V = \{1, \ldots, n\}$, é realizada através do seguinte algoritmo:

Algoritmo 1.1. *Inicialização da Matriz de Adjacência (Lista de Certificados)*

$p \leftarrow 0$;
Para cada $\{i, j\} \in E$ **faça**
  $p \leftarrow p + 1$;
  $\mathcal{M}(i, j) \leftarrow \mathcal{M}(j, i) \leftarrow p$;
  $\mathcal{L}(p) \leftarrow \{i, j\}$;

Ao final da leitura, a lista $\mathcal{L}$ armazenará as m arestas do grafo. Observe que as posições de $\mathcal{M}$ correspondentes a arestas são preenchidas com índices para a lista $\mathcal{L}$; as demais não são inicializadas, permanecendo com algum valor inteiro irrelevante. O teste de existência da aresta $\{i, j\}$ resume-se, portanto, à verificação das seguintes condições:

$$(1 \leq \mathcal{M}(i,j) \leq m) \wedge (\mathcal{L}(\mathcal{M}(i,j)) = \{i,j\}).$$

Na primeira delas, é testado se $\mathcal{M}(i,j)$ armazena um índice válido para a lista $\mathcal{L}$; caso positivo, $\mathcal{L}(\mathcal{M}(i,j))$ é consultada, devendo conter o subconjunto $\{i,j\}$ para que a referida aresta seja considerada existente.

A complexidade de tempo do Algoritmo 1.1 é claramente $O(m)$, o que, dependendo da densidade do grafo (relação entre o número de arestas e o número de vértices), pode representar um ganho substancial em relação à inicialização integral da matriz de adjacências. Observe que a complexidade de espaço permanece $O(n^2 + m) = O(n^2)$.

1.7 Códigos

Se, ao utilizarmos um esquema, cada grafo da família possuir uma única representação, esta é denominada um *código*. Neste caso, a correspondência entre grafos e códigos é biunívoca. O processo de obtenção do código para um determinado grafo da família chama-se *codificação*. A *validação* de um código consiste em determinar se a ele corresponde um grafo da família, sem explicitamente exibir tal grafo. A construção explícita do grafo a partir de seu código chama-se *decodificação*.

Dentre as possibilidades examinadas neste capítulo, apenas a representação por conjuntos de adjacência constitui um código para grafos arbitrários, que redunda em uma representação por $n + 2m$ símbolos. Entretanto, para famílias específicas, é possível determinar códigos mais compactos, como estudaremos nos capítulos que se seguem.

1.8 Exercícios

1. Apresente, para o grafo da Figura 1.1, sua representação por conjuntos de adjacência e uma de suas possíveis representações por matriz de adjacência.

2. Esboce algoritmos para computar o grau de um dado vértice em um grafo armazenado em memória principal através de

   (a) listas de adjacência;

   (b) matriz de adjacência.

   Qual das representações acarreta a melhor complexidade de tempo para o cálculo do grau de um vértice quando o grafo armazenado é sabidamente esparso (quociente m/n baixo) ?

3. No máximo, quantas representações por matriz de adjacência admite um grafo $G = (V, E)$ com $|V| = n$ vértices ? Essas representações são necessariamente distintas ? Exemplifique.

4. O *grafo estrela* com $n+1$ vértices, $n > 0$, simbolizado por $K_{1,n}$, tem a seguinte definição:

$$K_{1,n} = (\{1, \ldots, n+1\}, \{\{1, i\} \mid i = 2, \ldots, n+1\}).$$

   (a) Obtenha uma representação geométrica qualquer para $K_{1,4}$.

   (b) Obtenha a representação por conjuntos de adjacência para $K_{1,n}$.

   (c) Que peculiaridade possui a matriz de adjacência que representa $K_{1,n}$?

   (d) Mostre que o número de vértices n constitui um código para $K_{1,n}$.

5. A definição de grafo estrela, dada no exercício anterior, pressupõe que o conjunto de vértices seja constituído pelos números $1, 2, \ldots, n+1$, sendo $n > 0$. Podemos generalizá-la para um conjunto qualquer de vértices da seguinte forma: Sejam $V \neq \emptyset$ um conjunto qualquer e $r \notin V$. O *grafo estrela com centro em r e borda V* é assim definido:

$$S_{r,V} = (\{r\} \cup V, \{\{r, v\} \mid v \in V\}).$$

   (a) Estabeleça uma caracterização para $S_{r,V}$ baseada apenas nos graus dos vértices.

   (b) Estabeleça um código para $S_{r,V}$ e compare com o código sugerido no último item do exercício anterior.

   Moral da história: na concepção de códigos, definir os grafos da família em questão utilizando como vértices números naturais consecutivos a partir de 1 dispensa a inclusão dos vértices no código, bastando mencionar a quantidade deles.

6. Sejam V_1 e V_2 conjuntos não-vazios e disjuntos. O *grafo bipartido completo* sobre V_1 e V_2 possui a seguinte definição:

$$K_{V_1,V_2} = (V_1 \cup V_2, \{\{v, w\} \mid v \in V_1 \wedge w \in V_2\}).$$

   (a) Mostre que o grafo estrela $S_{r,V}$ é um caso particular do grafo bipartido completo K_{V_1,V_2}.

   (b) Determine, em função de $|V_1|$ e $|V_2|$, o número de arestas de K_{V_1,V_2}.

   (c) Mostre que a seqüência $[V_1, V_2]$ é uma representação para K_{V_1,V_2}. Justifique por que ela não é um código.

   (d) Estabeleça um código para K_{V_1,V_2} em que seja mencionado o menor número possível de símbolos.

7. Que vantagens existem em considerar $V = \{1, \ldots, n\}$, $n > 0$, na definição de um grafo $G = (V, E)$, com respeito ao armazenamento do mesmo em memória principal por

   (a) listas de adjacência ?

   (b) matriz de adjacência ?

8. Um vértice de grau 0 em um grafo $G = (V, E)$ é denominado *vértice isolado*. Descreva as características das representações por conjuntos e matriz de adjacência de um grafo que possui tal tipo de vértice.

9. Que condição deve satisfazer um grafo qualquer $G = (V, E)$ para que a menção apenas do conjunto de arestas constitua uma representação para G ? Neste caso, como o conjunto de vértices V seria deduzido a partir de E ?

10. Considerando a representação sugerida no exercício anterior, esboce um algoritmo para armazenar um grafo na forma de listas de adjacência, supondo que o conjunto de arestas deva ser lido de um arquivo texto (um par de vértices por linha).

11. Deduza as expressões apresentadas ao final da Seção 1.5.

12. No armazenamento compacto sugerido para a matriz de adjacência na Seção 1.5, consideramos a porção subdiagonal da matriz, dispondo seqüencialmente as linhas em um vetor. O mesmo princípio poderia ser aplicado se considerássemos a porção sobrediagonal, conforme ilustra a Figura 1.4.

$\mathcal{A}$

	1	2	3	4	5	6
1	0	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}
2	×	0	a_{23}	a_{24}	a_{25}	a_{26}
3	×	×	0	a_{34}	a_{35}	a_{36}
4	×	×	×	0	a_{45}	a_{46}
5	×	×	×	×	0	a_{56}
6	×	×	×	×	×	0

$\mathcal{B}$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{23}	a_{24}	a_{25}	a_{26}	a_{34}	a_{35}	a_{36}	a_{45}	a_{46}	a_{56}

Figura 1.4. Uma variante para o armazenamento compacto, quando $n = 6$

Deduza, para esta variante, a expressão que permite obter a posição k no vetor correspondente a um elemento a_{ij} da matriz, sendo $1 \leq i < j \leq n$. Qual a vantagem em utilizar o esquema anteriormente sugerido no texto ?

1.9 Notas Bibliográficas

Neste capítulo, foram apresentados os conceitos básicos imprescindíveis à compreensão do texto. Os livros de Diestel [11] e Gross e Yellen [14] abrangem, ampla e sistematicamente, a teoria de grafos. Já os livros de McHugh [21], Tarjan [32] e Szwarcfiter [30] abordam, de maneira mais específica, aspectos referentes à área de algoritmos em grafos.

Conhecimentos básicos computacionais, principalmente em estruturas de dados e complexidade de algoritmos, são também de grande valia no estudo de representações de grafos. Para uma revisão, sugerimos o livro de Brassard e Bratley [5] e, como texto em português, Szwarcfiter e Markenzon [31].

O engenhoso armazenamento utilizando a matriz de adjacência acompanhada pela lista de certificados, mencionado na Seção 1.6, foi proposto no livro de Aho, Hopcroft e Ullman [1], como exercício 2.12 à página 71.

Fundamentos e algoritmos relativos à área de Traçado Automático de Grafos podem ser encontrados no excelente livro de Di Battista et *al.* [10].

Capítulo 2

Codificação de Árvores

Neste capítulo estudaremos o esquema de codificação de árvores mais antigo e mais conhecido: o código de Prüfer. Este esquema data de 1918, e sua aplicação imediata foi a contagem dos elementos da família. É interessante observar como o mesmo conceito é utilizado atualmente também para armazenar árvores na memória do computador de maneira a possibilitar a resolução eficiente de problemas algorítmicos, como o de geração aleatória.

2.1 Considerações Iniciais

Uma *árvore* é um grafo conexo e acíclico. O Teorema 2.1, encontrado na literatura, fornece quatro caracterizações adicionais para esta família.

Teorema 2.1 ([30])**.** *As cinco afirmativas seguintes são equivalentes:*

- *O grafo $T = (V, E)$ é uma árvore.*
- *T é conexo e $|E|$ é mínima.*
- *T é conexo e $|E| = |V| - 1$.*
- *T é acíclico e $|E| = |V| - 1$.*
- *T é acíclico e para todo $v, w \in V$, a adição de uma aresta $\{v, w\}$ produz um grafo contendo exatamente um ciclo.*

Pelo Teorema 2.1, toda árvore com n vértices possui exatamente $n - 1$ arestas. Em uma árvore com $n \geq 2$ vértices, todo vértice v tal que $d(v) = 1$ é chamado *folha* e os demais, *interiores*. No caso especial em que $n = 1$ (árvore trivial) o único vértice é considerado uma folha, embora possua grau 0. É sempre possível eleger um vértice da árvore como *raiz*, tornando-a *enraizada*.

O armazenamento de árvores através de listas de adjacência será assumido ao longo deste capítulo durante a elaboração de algoritmos. Neste caso particular,

como $m = n - 1$, o tamanho da representação é $O(n)$. Em se tratando de uma árvore enraizada, é usual que o primeiro nó da lista principal (vertical) corresponda ao vértice raiz.

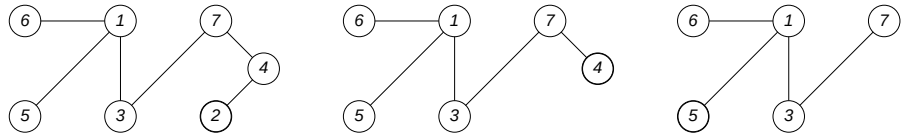
Sem perda de generalidade, vamos assumir que todas as árvores aqui tratadas possuem como conjunto de vértices $V = \{1, \ldots, n\}$, $n \geq 2$.

2.2 O Código de Prüfer

Seja $T = (V, E)$ uma árvore e considere o esquema de representação dado pelo seguinte processo iterativo:

- inicie com uma seqüência $\mathcal{S}$ de vértices vazia;
- a cada passo, remova a menor folha da árvore, acrescentando o único vértice a ela adjacente ao final de $\mathcal{S}$.
- o processo encerra-se quando restarem somente duas folhas.

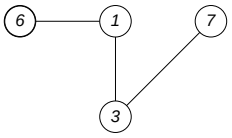
É imediato constatar que este esquema conduz a uma única representação para T, armazenada em $\mathcal{S}$, denominada *código de Prüfer*. Observe que, quando $n = 2$, $\mathcal{S}$ é a seqüência vazia. O código não está definido para árvores triviais ($n = 1$).



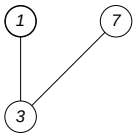
Menor folha: 2
Adjacente: 4
$\mathcal{S} = [4]$

Menor folha: 4
Adjacente: 7
$\mathcal{S} = [4, 7]$

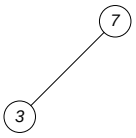
Menor folha: 5
Adjacente: 1
$\mathcal{S} = [4, 7, 1]$



Menor folha: 6
Adjacente: 1
$\mathcal{S} = [4, 7, 1, 1]$



Menor folha: 1
Adjacente: 3
$\mathcal{S} = [4, 7, 1, 1, 3]$



Restam 2 folhas

Figura 2.1. Um exemplo de codificação de Prüfer

A Figura 2.1 mostra um exemplo passo a passo de como é obtido o código $[4, 7, 1, 1, 3]$ para a árvore ilustrada.

Para que o esquema de Prüfer se aplique a árvores enraizadas, basta estabelecer que o vértice raiz nunca seja removido, ainda que se torne folha durante o processo. O código de Prüfer para árvores enraizadas é composto por $n - 1$ vértices, isto é,

um a mais do que na codificação das árvores sem raiz. Este vértice adicional é a raiz da árvore e ocupa a última posição do código.

Sejam duas árvores $T = (V, E)$ e $T' = (V, E)$, sendo a primeira sem raiz e a segunda enraizada no vértice n. O código de Prüfer da primeira é um prefixo do código de Prüfer da segunda. Isso ocorre porque, nos dois casos, o vértice n nunca é removido da árvore durante a codificação. Para árvores enraizadas no vértice n, este vértice não é removido por ser raiz e, para árvores sem raiz, sempre existe uma folha menor para ser removida. Por isso, para árvores sem raiz, diz-se que o código de Prüfer implicitamente assume o maior vértice como raiz. Por exemplo, a árvore $T = (V, E)$ (sem raiz) da Figura 2.1 tem código de Prüfer $[4, 7, 1, 1, 3]$ e a mesma árvore enraizada no vértice 7 tem código de Prüfer $[4, 7, 1, 1, 3, 7]$.

O Teorema 2.2 e o Corolário 2.1 relacionam a freqüência de ocorrência de um determinado vértice no código de Prüfer para uma árvore com o grau deste vértice.

Teorema 2.2. *Seja $T = (V, E)$ uma árvore. Todo vértice $v \in V$ ocorre exatamente $d_T(v) - 1$ vezes no código de Prüfer de T.*

Demonstração: No processo de codificação, toda vez que se remove uma folha, adiciona-se ao código o (único) vértice a ela adjacente. Como todos os vértices (exceto os dois últimos que restam) são removidos, v será inserido no código exatamente $d_T(v) - 1$ vezes. □

Corolário 2.1. *Se um vértice não aparece no código de Prüfer de uma árvore, este vértice é uma folha.*

2.3 Algoritmos para Codificação e Decodificação

O primeiro problema a ser observado ao se tratar de códigos é o do desenvolvimento de algoritmos eficientes capazes de obter o código correspondente a um determinado objeto (codificação) e, reciprocamente, reconstituir o objeto a partir de seu código (decodificação).

A seguir, apresentamos o algoritmo para obtenção do código de Prüfer relativo a uma árvore $T = (V, E)$. O método de construção já foi descrito na Seção 2.2. O código é armazenado na lista Cod, inicializada vazia. O conjunto $Folhas$ é inicializado com as folhas de T e destina-se a guardar os vértices que, ao longo do processo, venham a tornar-se folhas. A cada passo, a menor folha v é selecionada; por ser uma folha, só deve haver um vértice adjacente a ela. Seja u este vértice, que é adicionado ao fim da lista Cod. A folha v é removida da árvore e o grau de u é decrementado de uma unidade. Caso u tenha se tornado uma folha, deve ser inserido em $Folhas$. O processo se repete até que o conjunto V tenha apenas dois elementos. Ao final do algoritmo, a lista Cod contém uma seqüência com $n - 2$ vértices, que é o código de Prüfer para árvore $T = (V, E)$.

Algoritmo 2.1. *Obtenção do Código de Prüfer*

Entrada: Árvore $T = (V, E)$, com $V = \{1, \ldots, n\}, n \geq 2$;
Saída: Lista Cod com o código de Prüfer;
Início
  $Cod \leftarrow \emptyset$;
  $Folhas \leftarrow \{v \in V \mid d_T(v) = 1\}$;
  Enquanto $|V| > 2$ **faça**
    Encontre v, o menor elemento de $Folhas$; ..... (*)
    Seja u o único vértice adjacente a v;
    Remova v e $\{v, u\}$ de T; ..... (**)
    Adicione u ao fim da lista Cod;
    Se $d_T(u) = 1$ **então**
      $Folhas \leftarrow Folhas \cup \{u\}$;
Fim.

O próximo algoritmo tem por objetivo a decodificação, ou seja, construir a árvore correspondente a um determinado código de Prüfer. Seja t o tamanho da seqüência, armazenada em Cod. Como o código de Prüfer é composto por $n - 2$ vértices, conclui-se que $n = t+2$ e, portanto, $V = \{1, 2, ..., t+2\}$. Como, no processo de codificação, as folhas da árvore nunca são acrescentadas ao código sendo gerado, o conjunto $Folhas$ deve ser inicializado com todos os elementos de V que não pertencem a Cod. Inicia-se com o conjunto de arestas E vazio e constrói-se a árvore pela adição sucessiva de arestas $\{u, v\}$, determinadas da seguinte forma: Escolhe-se v como o menor elemento do conjunto $Folhas$; v está ligado ao primeiro elemento de Cod, o vértice u. Removem-se a folha v e o vértice u; caso este não apareça mais em Cod, isto indica que, para a seqüência que sobrou, este vértice é uma folha. O vértice u é então inserido em $Folhas$. Após todos os vértices da lista Cod terem sido analisados e removidos, insere-se a última aresta que é adjacente às duas folhas ainda não utilizadas.

Algoritmo 2.2. *Obtenção da Árvore a partir do Código de Prüfer*

Entrada: Lista Cod com o código de Prüfer;
Saída: Árvore $T = (V, E)$;
Início
  $n \leftarrow |Cod| + 2$; $V \leftarrow \{1, 2, ..., n\}$; $E \leftarrow \emptyset$;
  $Folhas \leftarrow \{v \in V \mid v \notin Cod\}$;
  Enquanto $Cod \neq \emptyset$ **faça**
    Retire o primeiro elemento u da lista Cod;
    Encontre v, o menor elemento de $Folhas$;
    $Folhas \leftarrow Folhas - \{v\}$; $E \leftarrow E \cup \{\{u, v\}\}$;
    Se $u \notin Cod$ **então**
      $Folhas \leftarrow Folhas \cup \{u\}$;
  // Sobram 2 vértices em $Folhas$
  $E \leftarrow E \cup \{Folhas\}$;
Fim.

2.4 Implementação e Complexidade dos Algoritmos

Vamos analisar, de início, alguns pontos acerca da implementação do algoritmo de codificação, de modo a garantir sua linearidade.

Primeiramente, a seleção do menor elemento do conjunto $Folhas$ (assinalada com (*) no Algoritmo 2.1) pode ser significativa na determinação da complexidade do algoritmo, pois pressupõe, a princípio, uma ordenação dos elementos do conjunto. Para que esta seja evitada, utilizamos um vetor contendo os graus correntes dos vértices. Assim, inicialmente, $grau[i] = |Adj(i)|$ para todo vértice i. A variável v armazena sempre a folha a ser removida. O vetor $grau$ é percorrido sempre com auxílio da variável i, sendo v o próximo vértice encontrado cujo grau seja 1. O vértice u, único adjacente a v ainda não removido, é determinado percorrendo-se $Adj(v)$ até encontrar um vértice que possua grau maior que 1. Se $u < i$, u é a próxima folha a ser processada; do contrário, torna-se a buscar o próximo vértice de grau 1, a partir da posição seguinte a i.

O segundo ponto a ser discutido é a remoção do vértice selecionado v e da aresta $\{v, u\}$ (assinalada com (**) no Algoritmo 2.1). Para evitar que esta remoção seja explicitamente realizada, o que implicaria o percurso de $Adj(u)$, os graus de v e u são atualizados, refletindo a remoção.

Algoritmo 2.3. *Implementação da Codificação*

Entrada: Árvore $T = (V, E)$, com $V = \{1, ..., n\}$, $n \geq 2$,
representada por listas de adjacência Adj dos vértices.
Saída: Vetor Cod com a codificação de Prüfer;
Início
  Para $i \leftarrow 1, \ldots, n$ **faça**
    $grau[i] \leftarrow |Adj(i)|$;
  $v \leftarrow i \leftarrow 0$;
  Para $j \leftarrow 1, \ldots, n-2$ **faça**
    Se $v = 0$ **então**
      Repita $v \leftarrow i \leftarrow i + 1$ **até** $grau[v] = 1$; ..... (*)
    Para $w \in Adj(v)$ **faça** ..... (**)
      Se $grau[w] > 1$ **então** $u \leftarrow w$;
    $Cod[j] \leftarrow u$;
    $grau[v] \leftarrow 0$; $grau[u] \leftarrow grau[u] - 1$;
    Se $grau[u] = 1$ **and** $u < i$ **então**
      $v \leftarrow u$;
    caso contrário
      $v \leftarrow 0$;
Fim.

Para concluir corretamente a linearidade do algoritmo, dois pontos devem ser observados:

- A malha assinalada com (*) é executada sempre que seja necessário localizar a

próxima folha. Observando que a variável i inicia o processamento com valor 0 e é sempre incrementada, concluímos que a malha será executada menos de n vezes;

- A malha assinalada com (**) é executada uma vez para cada folha v sendo removida do grafo, realizando um percurso na lista de adjacência de v. Ao total, haverá menos do que $\sum_{v \in V} |Adj(v)| = 2n - 2$ execuções desta malha, já que nem todos os vértices tornam-se folhas.

A implementação do algoritmo de decodificação é similar e está baseada no fato de que as folhas são justamente os vértices que não figuram no código. O vetor $freq$ é utilizado para armazenar o número de vezes que cada vértice aparece no código. Desta forma, pelo Corolário 2.1, as folhas são os vértices v que possuem $freq[v] = 0$. A complexidade do algoritmo de decodificação é também $O(n)$, seguindo uma análise similar àquela realizada para o algoritmo anterior.

Algoritmo 2.4. *Implementação da Decodificação*

Entrada: Vetor Cod, com o código de Prüfer;
Saída: Árvore $T = (V, E)$;
Início
  $n \leftarrow |Cod| + 2$; $V \leftarrow \{1, \ldots, n\}$; $E \leftarrow \emptyset$;
  $Cod[n-1] \leftarrow n$; // Arremata o código com o maior vértice
  Para $v \leftarrow 1, \ldots, n$ **faça** $freq[v] \leftarrow 0$;
  Para $j \leftarrow 1, \ldots, n-1$ **faça** $freq[Cod[j]] \leftarrow freq[Cod[j]] + 1$;
  $i \leftarrow v \leftarrow 0$;
  Para $j \leftarrow 1, \ldots, n-1$ **faça**
    Se $v = 0$ **então**
      Repita $v \leftarrow i \leftarrow i + 1$ **até** $freq[v] = 0$;
    $u \leftarrow Cod[j]$;
    $E \leftarrow E \cup \{\{u, v\}\}$;
    $freq[u] \leftarrow freq[u] - 1$;
    Se $freq[u] = 0$ **and** $u < i$ **então**
      $v \leftarrow u$;
    caso contrário
      $v \leftarrow 0$;
Fim.

2.5 Contagem e Geração de Árvores

No restante deste capítulo, analisaremos algumas aplicações relacionadas a contagem e geração de árvores, utilizando o código de Prüfer como base. Estes problemas serão abordados tanto para árvores irrestritas quanto para aquelas que satisfazem requisitos específicos, como possuírem um dado número de folhas.

Nos algoritmos seguintes, asssumimos a disponibilidade de um gerador aleatório de números inteiros uniformemente distribuídos entre a e b, capaz de produzir cada um dos números em tempo $O(1)$.

Árvores Irrestritas

Denotemos por $T(n)$ a quantidade de árvores distintas com vértices pertencentes ao conjunto $\{1, \ldots, n\}$, $n \geq 2$. A conhecida expressão $T(n) = n^{n-2}$ é devida a Cayley (1889), existindo na literatura várias demonstrações para ela. A mais simples parece ser a que utiliza o código de Prüfer, conforme o Teorema 2.3.

Teorema 2.3. $T(n) = n^{n-2}$.

Demonstração: Existe uma relação biunívoca entre o conjunto de todas as árvores cujos vértices são os inteiros $\{1, \ldots, n\}$ e o conjunto de todas as tuplas constituídas por $n-2$ desses inteiros. Como existem, ao todo, n^{n-2} tuplas distintas desta espécie, segue-se o resultado. □

Pelo Teorema 2.3, para gerar o código de Prüfer de uma árvore irrestrita com n vértices, basta gerar uma seqüência aleatória de $n-2$ inteiros pertencentes ao conjunto $\{1, \ldots, n\}$.

Algoritmo 2.5. *Geração do Código de Prüfer de uma Árvore Qualquer*

Entrada: n (número de vértices);
Saída: Vetor Cod com o código de Prüfer;
Início
  Para $i \leftarrow 1, \ldots, n-2$ **faça**
    $Cod[i] \leftarrow random(1, n)$;
Fim.

Uma vez gerado o código de Prüfer através do Algoritmo 2.5, podemos utilizar o Algoritmo 2.4 de decodificação para obter a árvore correspondente. É imediato constatar que este processo tem, em sua totalidade, complexidade $O(n)$.

Árvores com Seqüência de Graus Dada

A toda árvore $T = (V, E)$, onde $V = \{1, \ldots, n\}$, $n \geq 2$, podemos associar uma seqüência de inteiros em que cada elemento é o grau do vértice correspondente à sua posição: $[d(1), ..., d(n)]$. Esta seqüência é denominada *seqüência de graus* da árvore.

Para que uma seqüência de n inteiros $[d_1, \ldots, d_n]$ seja válida como seqüência de graus, i.e., para que seja possível construir uma árvore a ela correspondente, é necessário que $d_i \geq 1$, para $i = 1, \ldots, n$, e

$$\sum_{i=1}^{n} d_i = 2n - 2.$$

É interessante observar que diferentes árvores com a mesma quantidade de vértices podem ter a mesma seqüência de graus; entretanto, terão códigos de Prüfer diferentes. A Figura 2.2 mostra quatro árvores distintas com 7 vértices e seqüência de graus $[3, 2, 1, 2, 1, 2, 1]$.

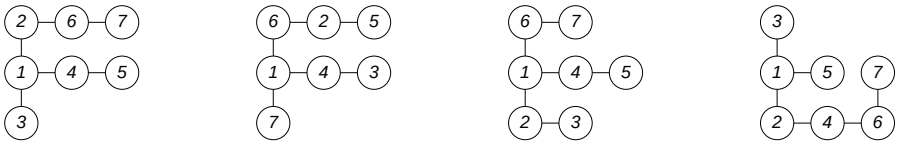


Figura 2.2. Quatro árvores com seqüência de graus $[3, 2, 1, 2, 1, 2, 1]$

No Teorema 2.4 é determinado o número de árvores distintas que correspondem a uma dada seqüência de graus.

Teorema 2.4. *A quantidade de árvores com $n \geq 2$ vértices pertencentes ao conjunto $\{1, \ldots, n\}$ e que possuem a seqüência de graus válida $[d_1, ..., d_n]$ é*

$$\frac{(n-2)!}{(d_1-1)!\ldots(d_n-1)!}.$$

Demonstração: Sendo a seqüência válida, temos que

$$\sum_{i=1}^{n}(d_i - 1) = \sum_{i=1}^{n} d_i - n = 2n - 2 - n = n - 2.$$

O número total de árvores com a seqüência de graus dada corresponde ao número de maneiras de preencher as $n-2$ posições do código com os n vértices, de modo que cada vértice i seja repetido $d_i - 1$ vezes. □

O Algoritmo 2.6 implementa a geração do código de Prüfer de uma árvore com n vértices, conhecida sua seqüência de graus $[d_1, ..., d_n]$. O processo resume-se a atribuir cada vértice i a exatamente $d_i - 1$ posições do código, aleatoriamente escolhidas.

A lista seqüencial $Disp$ é utilizada para evitar que a mesma posição de Cod seja escolhida mais de uma vez. $Disp$ contém, inicialmente, os índices $1, \ldots, n-2$ das posições disponíveis em Cod e a variável f armazena o índice da posição final desta lista ($f \leftarrow n-2$, no início). A cada iteração, uma posição k de $Disp$ é escolhida aleatoriamente entre 1 e f e o índice nela armazenado determina a posição em Cod que será preenchida. Ao final de cada iteração, $Disp[k]$ é substituído por $Disp[f]$ e f é decrementada, indicando uma possibilidade a menos de escolha.

Algoritmo 2.6. *Geração de Árvores com Seqüência de Graus Dada*

Entrada: n (número de vértices),
$[d_1, ..., d_n]$ (seqüência de graus válida);
Saída: Vetor Cod com o código de Prüfer;
Início
  $Disp \leftarrow [1, \ldots, n-2]$; $f \leftarrow n-2$;
  Para $i \leftarrow 1, \ldots, n$ **faça**
    Repita $d_i - 1$ **vezes**
      $k \leftarrow random(1, f)$;
      $Cod[Disp[k]] \leftarrow i$;
      $Disp[k] \leftarrow Disp[f]$;
      $f \leftarrow f - 1$;
Fim.

Sendo $[d_1, ..., d_n]$ uma seqüência válida, temos $\sum_{i=1}^{n}(d_i - 1) = n-2$ e os comandos da malha interna serão executados exatamente $n-2$ vezes. A complexidade total é, portanto, $O(n)$.

Árvores com Número de Folhas Pré-Fixado

Interessam-nos, agora, a contagem e geração de árvores cujo número de folhas é conhecido *a priori*. O Teorema 2.5 trata da contagem destas árvores, utilizando os *números de Stirling de segunda espécie* $S(a, b)$, que fornecem o número de maneiras distintas de particionar um conjunto de a elementos, $a \geq 1$, em exatamente b subconjuntos, sendo $1 \leq b \leq a$. Calculam-se os valores para $S(a, b)$ através da seguinte equação de recorrência:

$$S(a,b) = \begin{cases} 1, & \text{se } b = 1 \text{ ou } b = a; \\ bS(a-1,b) + S(a-1,b-1), & \text{se } 1 < b < a. \end{cases}$$

A tabela seguinte ilustra a partição do conjunto $\{1, 2, 3, 4\}$, com $a = 4$ elementos, em $b = 1, 2, 3$ e 4 subconjuntos.

b	$S(a,b)$	Partições	
1	1	$\{\{1,2,3,4\}\}$	
2	7	$\{\{1\},\{2,3,4\}\}$	$\{\{2\},\{1,3,4\}\}$
		$\{\{3\},\{1,2,4\}\}$	$\{\{4\},\{1,2,3\}\}$
		$\{\{1,2\},\{3,4\}\}$	$\{\{1,3\},\{2,4\}\}$
		$\{\{1,4\},\{2,3\}\}$	
3	6	$\{\{1\},\{2\},\{3,4\}\}$	$\{\{1\},\{3\},\{2,4\}\}$
		$\{\{1\},\{4\},\{2,3\}\}$	$\{\{1,2\},\{3\},\{4\}\}$
		$\{\{1,3\},\{2\},\{4\}\}$	$\{\{1,4\},\{2\},\{3\}\}$
4	1	$\{\{1\},\{2\},\{3\},\{4\}\}$	

Teorema 2.5. *Existem exatamente*

$$\frac{n!}{q!}S(n-2, n-q)$$

árvores distintas com vértices pertencentes ao conjunto $\{1, \ldots, n\}$*,* $n \geq 2$*, e* q *folhas,* $2 \leq q \leq n-1$*.*

Demonstração: Para uma árvore com n vértices e exatamente q folhas, o código de Prüfer possui exatamente $n-q$ vértices distintos, dentre os $n-2$ que o integram. Seja $[c_1, ..., c_{n-2}]$ o código de Prüfer de uma árvore contendo exatamente $n-q$ vértices distintos: $\ell_1, ..., \ell_{n-q}$. Para este código, definem-se os conjuntos $P_1, \ldots, P_{n-q}$ da seguinte maneira: $P_i = \{j \mid \ell_i = c_j\}$, para $i = 1, \ldots, n-q$. Ou seja, pertencem a P_i os índices das posições ocupadas pelo vértice ℓ_i no código de Prüfer. É fácil concluir que os conjuntos P_i satisfazem as seguintes propriedades:

- $P_i \neq \emptyset$, para $1 \leq i \leq n-q$;
- $P_1 \cup \ldots \cup P_{n-q} = \{1, \ldots, n-2\}$;
- $P_1 \cap \ldots \cap P_{n-q} = \emptyset$.

Em outras palavras, $\{P_1, \ldots, P_{n-1}\}$ é uma partição de $\{1, \ldots, n-2\}$.

Observando que cada código corresponde a uma partição diferente, existem, ao todo, $S(n-2, n-q)$ (1) códigos possíveis para uma dada escolha de vértices que aparecerão no código. Para escolher os q vértices que serão folhas e os $n-q$ interiores, existem um total de $\frac{n!}{q!}$ (2) possibilidades. De (1) e (2) prova-se o teorema. □

O processo de geração do código de Prüfer de uma árvore com $n \geq 2$ vértices e q folhas, $2 \leq q \leq n-1$, consiste em selecionar, dentre os n vértices, os $n-q$ que serão interiores (não-folhas), atribuindo cada um deles pelo menos uma vez às $n-2$ posições do código. A implementação compreende dois passos:

- Dentre os n vértices, escolhem-se aleatoriamente $n-q$ distintos, que serão os vértices interiores (não-folhas) da árvore. Para evitar escolhas repetidas do mesmo vértice, a lista seqüencial *Disp* armazena aqueles ainda não selecionados. Esta lista é inicializada com os n vértices disponíveis e a variável f armazena o índice da posição final desta lista (inicialmente, $f \leftarrow n$). Em cada uma das $n-q$ iterações, uma posição k da lista *Disp* é escolhida aleatoriamente entre 1 e f e o vértice nela armazenado é acrescentado ao conjunto de vértices interiores *Int* sendo construído. Ao final de cada iteração, o vértice da posição k é removido da lista *Disp*, sendo substituído pelo da posição f (o último).
- Cada um dos $n-q$ vértices interiores escolhidos no passo anterior é atribuído a uma posição no código, garantindo que cada um deles figure pelo menos uma vez. Às $n-2-(n-q) = q-2$ posições restantes, atribuem-se vértices interiores escolhidos aleatoriamente. A lista *Disp* é utilizada de forma análoga para evitar a repetição na escolha de posições do código.

A complexidade do Algoritmo 2.7 é visivelmente $O(n)$.

Algoritmo 2.7. *Geração com Número de Folhas Pré-Fixado*

```
Entrada: n (número de vértices), q (número de folhas);
Saída: Vetor Cod com o código de Prüfer;
Início
   Disp ← [1, ..., n]; f ← n;
   Para i ← 1, ..., n − q faça
      k ← random(1, f);
      Int[i] ← Disp[k];
      Disp[k] ← Disp[f];
      f ← f − 1;
   Disp ← [1, ..., n − 2]; f ← n − 2;
   Para i ← 1 ..., n − 2 faça
      k ← random(1, f);
      Se i ≤ n − q então
         Cod[Disp[k]] ← Int[i];
      caso contrário
         Cod[Disp[k]] ← Int[random(1, n − q)];
      Disp[k] ← Disp[f];
      f ← f − 1;
Fim.
```

2.6 Exercícios

1. Utilizando o Teorema 2.1, demonstre que o grafo estrela $K_{1,n}$, definido no Exercício 4 da Seção 1.8 é uma árvore.

2. Utilizando o esquema de Prüfer, obtenha o código correspondente a $K_{1,n}$.

3. Examinando as seqüências de graus possíveis para uma árvore com $n \geq 2$ vértices, demostre que ela possui, no mínimo, 2 e, no máximo, $n - 1$ folhas.

4. Uma árvore com $n \geq 2$ vértices é dita *degenerada* quando possui número mínimo de folhas. Que propriedade se pode deduzir para o código de Prüfer de uma árvore degenerada ?

5. Considere o Algoritmo 2.3, que implementa a obtenção do código de Prüfer referente a uma árvore armazenada em memória por listas de adjacência.

   (a) O que significam exatamente as igualdades $grau[v] = 0$ e $grau[v] = 1$, para um vértice v qualquer ?

   (b) Refaça o algoritmo, considerando a árvore armazenada por matriz de adjacência. Qual a complexidade de tempo desta nova versão ?

(c) Modifique o algoritmo de forma a produzir o código de Prüfer para árvores enraizadas, supondo que também seja fornecido como entrada o vértice raiz da árvore. Há alteração na complexidade de tempo deste novo algoritmo em relação à complexidade do algoritmo original ?

6. Vimos que qualquer seqüência de $n-2$ inteiros pertencentes ao conjunto $\{1,\ldots,n\}$ é um código de Prüfer válido, sendo possível construir uma única árvore não-enraizada a ele correspondente. O mesmo se pode afirmar para árvores enraizadas ? Em outras palavras, qualquer seqüência de $n-1$ inteiros pertencentes ao conjunto $\{1,\ldots,n\}$ é um código de Prüfer válido para uma árvore enraizada com n vértices ?

7. Considere o Algoritmo 2.4, que implementa a construção da árvore correspondente a um código de Prüfer dado como entrada, processo este que denominamos *decodificação*.

   (a) O comando de atribuição $Cod[n-1] \leftarrow n$ tem por finalidade acrescentar ao final do código fornecido o número do maior vértice. Refaça o algoritmo sem utilizar este artifício.

   (b) Releia o comentário sobre código de Prüfer para árvores enraizadas (final da página 26 e início da página 27).

   (c) Modifique o algoritmo de forma que ele receba como entrada o código de Prüfer de uma árvore enraizada (portanto, uma seqüência de $n-1$ inteiros).

8. Mostre que, em toda árvore $T=(V,E)$, escolhidos dois vértices distintos $v,w \in V$ quaisquer, existe um único caminho de v a w. (Dica: use redução ao absurdo).

9. Considere o seguinte esquema de representação para árvores $T=(V,E)$:

   - escolha um vértice $s \in V$ qualquer;
   - para cada um dos vértices $v \in V-\{s\}$, seja $\gamma_s(v)$ o vértice sucessor de v no único (veja exercício anterior) caminho existente de v a s.
   - o conjunto $\{(v,\gamma_s(v)) \mid v \in V-\{s\}\}$, cujos $|V|-1$ elementos são pares ordenados de vértices, é a representação desejada.

   Pede-se:

   (a) Utilizando este esquema, obtenha a representação para a árvore apresentada na Figura 2.1, considerando $s=1$.

   (b) Idem, considerando $s=3$.

   (c) Compare as representações obtidas nos dois itens anteriores. Que diferenças existem entre elas ?

(d) Mostre que o conjunto $\{(v, \gamma_s(v)) \mid v \in V - \{s\}\}$ é, de fato, uma representação para T, explicitando como, a partir dele, os conjuntos V e E da definição podem ser restabelecidos sem ambigüidades.

(e) Esboce um algoritmo linear em tempo e espaço para determinar os graus de todos os vértices da árvore a partir desta representação.

10. Considere a representação para árvores introduzida no exercício anterior. Observando que, com exceção de s, cada vértice ocorre exatamente uma vez como primeiro elemento de algum par na representação, podemos esboçar o seguinte algoritmo para obter o conjunto $\{(v, \gamma_s(v)) \mid v \in V - \{s\}\}$, dados s, V e E. O atributo *marca* indica se o vértice já apareceu como primeiro elemento de algum par (exceto para o vértice s, que possui $marca(s) = 1$ desde o início do algoritmo).

Entrada: uma árvore $T = (V, E)$ e um vértice $s \in V$;
Saída: o conjunto $\mathcal{P} = \{(v, \gamma_s(v)) \mid v \in V - \{s\}\}$;
Início
  Para $v \in V - \{s\}$ **faça** $marca(v) \leftarrow 0$;
  $marca(s) \leftarrow 1$;
  $\mathcal{P} \leftarrow \emptyset$;
  Enquanto $E \neq \emptyset$ **faça**
    Escolha $\{v, w\} \in E$ tal que $marca(v) + marca(w) > 0$;
    Se $marca(v) = 1$ **então**
      $\mathcal{P} \leftarrow \mathcal{P} \cup \{(w, v)\}$; $marca(w) \leftarrow 1$;
    caso contrário
      $\mathcal{P} \leftarrow \mathcal{P} \cup \{(v, w)\}$; $marca(v) \leftarrow 1$;
    $E \leftarrow E - \{\{v, w\}\}$;
Fim.

Suponha que:

- O conjunto E seja armazenado em memória através de uma lista circular duplamente encadeada, cujos nós contêm os vértices correspondentes a cada aresta;
- O conjunto $\mathcal{P}$ seja armazenado em memória através de uma lista simplesmente encadeada de pares de vértices, na qual todo novo elemento é inserido antes do primeiro.

Pede-se:

(a) Execute manualmente o algoritmo para a árvore da Figura 2.1, considerando $s = 1$ e a seguinte disposição das arestas do conjunto E na lista circular:

$$\{4, 2\}, \{3, 7\}, \{5, 1\}, \{1, 6\}, \{1, 3\}, \{4, 7\}.$$

(b) Repita o item anterior, mantendo $s = 1$ e dispondo as arestas na lista circular na seguinte ordem:

$$\{1,6\},\{1,5\},\{3,1\},\{3,7\},\{4,7\},\{4,2\}.$$

(c) Compare as execuções realizadas nos dois itens anteriores. Em qual delas o algoritmo apresenta desempenho superior em tempo ?

(d) Mostre que, durante a execução do algoritmo, para todo $\{v,w\} \in E$, tem-se $0 \leq marca(v) + marca(w) \leq 1$.

(e) Qual a complexidade de tempo do algoritmo se a árvore dada como entrada for um grafo estrela de centro s e borda $V - \{s\}$? (Veja definição no Exercício 5 da Seção 1.8).

(f) Que características deve possuir a árvore fornecida como entrada para que o algoritmo atinja o seu pior desempenho em tempo de execução ? Qual a complexidade nesta situação ?

11. O esquema utilizado nos dois exercícios anteriores corresponde meramente ao enraizamento da árvore no vértice s. Assim sendo, que interpretação podemos dar ao vértice $\gamma_s(v)$ em relação a v ?

12. É possível conceber algoritmos mais eficientes do que o sugerido no Exercício 10 para obter a representação em questão, utilizando *percursos* em árvores (pré-ordem, pós-ordem, por níveis). Supondo a árvore armazenada em memória por de listas de adjacência, qualquer percurso iniciado no vértice s é capaz de gerar o conjunto $\{(v,\gamma_s(v)) \mid v \in V - \{s\}\}$. Se você domina a implementação destes percursos, esboce um algoritmo de complexidade linear.

2.7 Notas Bibliográficas

O artigo pioneiro de Prüfer [27] sobre a codificação de árvores data de 1918. O tratamento algorítmico do assunto é bem mais recente; somente em 2000, Chen e Wang [7] propuseram um algoritmo linear de codificação. Logo após, Deo e Micikevicius [8] publicaram um *survey* sobre códigos tipo Prüfer, classificando diversos métodos de codificação, como os de Neville; estes autores propõem também um novo método em [9]. Caminiti *et al.* [6] abordaram o tema de maneira unificada, apresentando algoritmos lineares para diversos esquemas, sendo Prüfer um deles. O Algoritmo 2.1 é caso particular do proposto por Markenzon *et al.* [19] para a codificação de k-árvores de Rényi. Os problemas de geração e contagem de árvores, irrestritas ou satisfazendo requisitos específicos, foram estudados por Moon [23]. Uma versão preliminar deste capítulo foi apresentada no VII Encontro Regional de Matemática Aplicada e Computacional (ERMAC), em Vitória, ES, 2005 [20].

Capítulo 3

Representação de Grafos Cordais

3.1 Conceitos Básicos

Grafos cordais constituem uma classe bastante estudada, uma vez que possuem uma estrutura peculiar, baseada em cliques maximais. Esta estrutura favorece a solução de muitos problemas algorítmicos, alguns dos quais serão vistos neste capítulo. Justamente para a solução destes problemas, outras informações, além das habitualmente fornecidas por conjuntos de adjacências, são necessárias. Para apresentar alternativas, devemos primeiro conhecer alguns novos conceitos e propriedades desta família.

Sejam $G = (V, E)$ um grafo, com $n = |V|$, e $v \in V$ um vértice qualquer de G. Dizemos que v é *simplicial* quando $Adj(v)$ é uma clique em G (i.e., o subgrafo $G[Adj(v)]$ de G induzido por $Adj(v)$ é um grafo completo). No grafo da Figura 3.1, por exemplo, os vértices c, h e i são simpliciais.

Um *esquema de eliminação perfeita* (*EEP*) de G é uma função bijetora

$$\sigma : \{1, \ldots, n\} \rightarrow V$$

tal que $\sigma(i)$ é um vértice simplicial em $G[\{\sigma(j) \mid i \leq j \leq n\}]$, para $i = 1, \ldots, n$. Um *EEP* pode ser melhor visualizado se percebermos que sua definição induz a dispor os vértices em uma seqüência

$$\sigma(G) = [\sigma(1), \ldots, \sigma(n)],$$

de maneira que todo vértice seja simplicial no subgrafo de G induzido por ele e pelos que o seguem na seqüência σ. Assim, dada uma posição i na seqüência, $\sigma(i)$ denota o vértice que a ocupa e a inversa $\sigma^{-1}(v)$ corresponde à posição ocupada pelo vértice v.

Pela definição, é fácil determinar um *EEP* para um dado grafo. Inicia-se com uma seqüência vazia de vértices, que conterá um *EEP* ao final do processo. A cada passo, um vértice simplicial é escolhido, acrescentado à seqüência e excluído do grafo. Para o grafo da Figura 3.1, um possível *EEP* é a seqüência $[i, a, h, g, f, b, e, c, d]$: o vértice i é simplicial em $G[\{i, a, h, \ldots, c\}]$, o vértice a é simplicial em $G[\{a, h, g, \ldots, c\}]$ e assim sucessivamente. A seqüência $[h, c, i, f, g, e, d, a, b]$ é outro *EEP* para este grafo.

Observe, agora, o grafo da Figura 3.2. O vértice a é o único simplicial, sendo então o primeiro vértice de um *EEP*; a partir deste ponto é impossível prosseguir, pois não restam vértices simpliciais em $G[V - \{a\}]$.

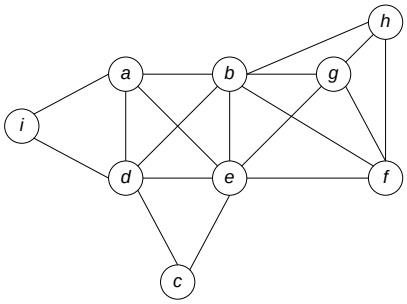


Figura 3.1: Um grafo cordal

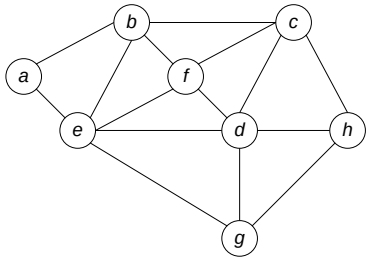


Figura 3.2: Um grafo não cordal

Um grafo $G = (V, E)$ é *cordal* quando todo ciclo simples de comprimento maior ou igual a 4 possui uma corda (i.e. uma aresta ligando dois vértices não consecutivos do ciclo). O grafo da Figura 3.1 é cordal. O da Figura 3.2 não o é; basta observar que o ciclo $[b, c, d, e, b]$, de comprimento 4, não tem corda.

A cordalidade equivale à existência de um *EEP*, conforme estabelecido no Teorema 3.1.

Teorema 3.1 ([13])**.** *Seja G um grafo. G é cordal se, e somente se, G tem um esquema de eliminação perfeita, que pode iniciar-se com qualquer vértice simplicial.*

Os algoritmos de reconhecimento de grafos cordais mais conhecidos e mais eficientes baseiam-se no Teorema 3.1 e constam de dois passos. Primeiramente, um percurso especial (por vizinhança máxima ou em largura lexicográfica, por exemplo) determina uma seqüência de vértices; em seguida, verifica-se se esta seqüência é um *EEP*. Esses dois passos podem ser implementados em algoritmos de complexidade linear em tempo e espaço. Se o grafo fornecido como entrada for sabidamente cordal, as seqüências de vértices produzidas por estes percursos constituem esquemas de eliminação perfeita para ele. Sob esta ótica, revisaremos, na Seção 3.3, o percurso por vizinhança máxima.

3.2 Esquema de Representação

Em última análise, um esquema de eliminação perfeita dispõe os vértices de um grafo cordal em uma seqüência, garantindo que cada um deles esteja ligado a uma clique formada por um subconjunto de seus sucessores. Esta estrutura sugere um processo de construção para grafos cordais, formalizado no Teorema 3.2.

Teorema 3.2. *Seja $G = (V, E)$ um grafo cordal, $v \notin V$ um novo vértice e $Q \subseteq V$ uma clique de G. O grafo $G' = (V \cup \{v\}, E \cup \{\{v, x\} | x \in Q\})$ é também cordal.*

Demonstração: Basta observar que v é um vértice simplicial de G'. Então, se $\overline{[v_1, v_2, \ldots, v_n]}$ é um *EEP* de G, $[v, v_1, v_2, \ldots, v_n]$ é um *EEP* de G' e, pelo Teorema 3.1, G' é cordal. □

A partir do Teorema 3.2, é possível conceber um processo indutivo de construção para grafos cordais: iniciando com um grafo trivial (composto por apenas um vértice), acrescenta-se, a cada passo, um novo vértice ligado a uma clique do grafo já existente. Em relação ao *EEP*, este novo vértice é acrescentado sempre como prefixo. A Figura 3.3 ilustra esta construção para o grafo da Figura 3.1, utilizando o *EEP* $[i, a, h, g, f, b, e, c, d]$.

Não é difícil constatar que um *EEP* é insuficiente para representar um grafo cordal, uma vez que, a partir dele, o conjunto de arestas não pode ser restabelecido. Para a reconstrução do grafo é necessário o conhecimento das cliques às quais cada vértice do *EEP* foi ligado durante o processo construtivo.

Sejam o grafo cordal $G = (V, E)$, σ um *EEP* de G e $v \in V$ um vértice qualquer. O conjunto

$$X_\sigma(v) = \{x \in Adj(v) \mid \sigma^{-1}(v) < \sigma^{-1}(x)\}$$

é denominado *conjunto de adjacência restrito* de v.

Observe que, para todo vértice $v \in V$, os elementos do conjunto $X_\sigma(v)$ são exatamente os elementos da clique à qual o vértice v é ligado no decorrer da construção segundo o *EEP* σ.

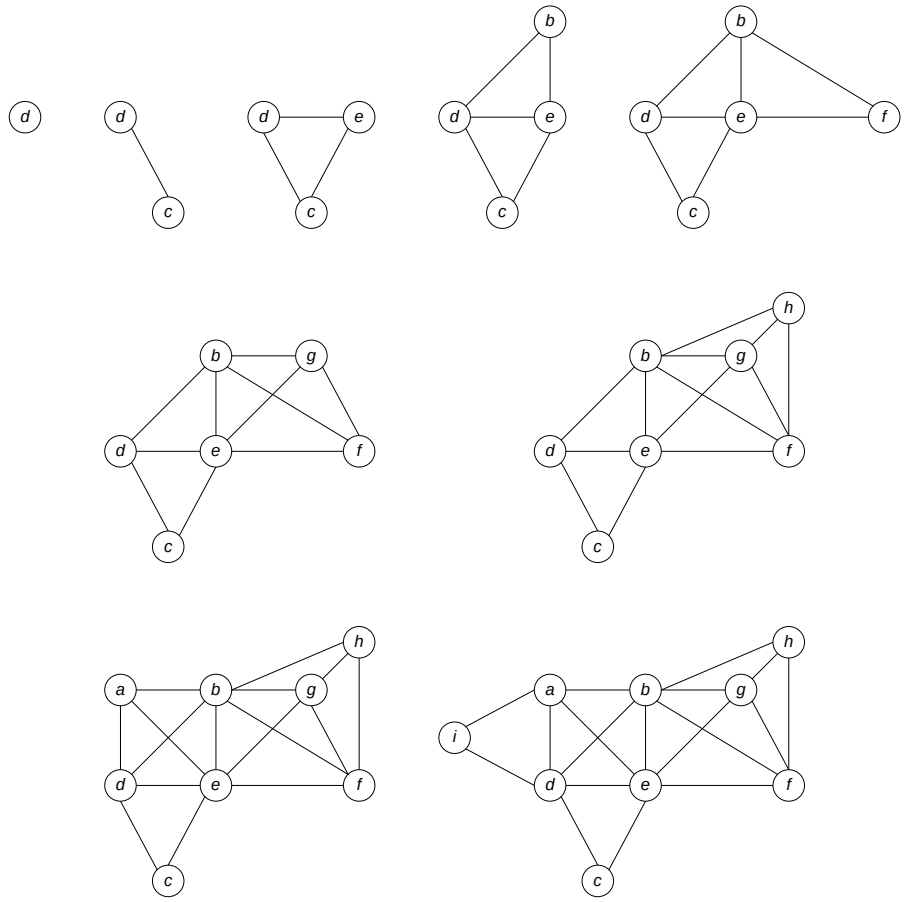


Figura 3.3: Construção indutiva de um grafo cordal

Dado um *EEP* σ, a representação de um grafo cordal $G = (V, E)$ por *conjuntos de adjacência restritos* consiste em uma seqüência de pares

$$\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n].$$

Vale observar que esta representação é única para um dado grafo e um *EEP*.

Considerando o *EEP* $[i, a, h, g, f, b, e, c, d]$, o grafo da Figura 3.1 tem a seguinte representação:

$$\begin{array}{llllll} [& (i, \{a, d\}), & (a, \{b, d, e\}), & (h, \{b, f, g\}), & (g, \{b, e, f\}) & \\ & (f, \{b, e\}), & (b, \{d, e\}), & (e, \{c, d\}), & (c, \{d\}), & (d, \emptyset) \quad]. \end{array}$$

Em uma representação obtida por este esquema devem ser mencionados, ao todo,

$$\sum_{v \in V} [1 + |X_\sigma(v)|] = n + m$$

símbolos. Pode-se notar que esta representação é mais econômica do que a representação por conjuntos de adjacência, vista na Seção 1.3, que demanda $n + 2m$ símbolos.

O armazenamento em memória desta representação consiste em uma estrutura ortogonal em que os nós da lista principal (vertical), obrigatoriamente seqüencial, armazenam informações sobre os vértices do grafo; a ordem em que os vértices aparecem nesta lista principal é a mesma do *EEP*. De cada um dos nós da lista vertical emerge uma lista secundária (horizontal), que possui um nó para cada elemento do conjunto de adjacência restrito.

A Figura 3.4 mostra o armazenamento do grafo da Figura 3.1 na memória, considerando o *EEP* $[i, a, h, g, f, b, e, c, d]$. Note que a última lista horizontal é sempre vazia.

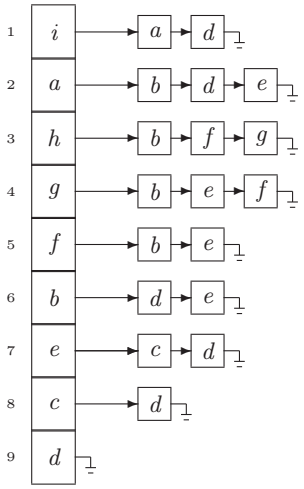


Figura 3.4. Armazenamento do grafo da Figura 3.1 por listas de adjacência restritas

Duas operações são fundamentais em algoritmos que utilizam esta representação: a determinação de $\sigma(i)$ e a determinação de $\sigma^{-1}(v)$. A primeira delas tem solução imediata (em $O(1)$) no armazenamento ora proposto. Para que a segunda operação seja também eficiente, é necessário utilizar uma estrutura de dados auxiliar que, para cada vértice do grafo, indique sua posição no *EEP*. Se o conjunto dos vértices for $V = \{1, \ldots, n\}$, $n \geq 1$, um vetor é suficiente para que a operação seja também executada em $O(1)$.

3.3 Obtendo a Representação

Obter a representação de um grafo cordal $G = (V, E)$ por conjuntos de adjacência restritos requer a determinação de um *EEP* de G. Para tal, utilizaremos o algoritmo

conhecido na literatura como *percurso por vizinhança máxima*.

Um *percurso* genérico sobre um grafo pode ser definido como um procedimento sistemático de varredura, que consiste em *visitar* exatamente uma vez cada vértice e aresta. O termo *visitar* deve ser aqui compreendido como a execução de qualquer ação sobre um elemento do grafo, exceto sua exclusão, possivelmente utilizando ou mesmo alterando a informação nele armazenada.

O *percurso por vizinhança máxima* é um caso particular do percurso genérico, no qual, a cada passo, é escolhido como vértice a visitar qualquer um dentre os que possuam o maior número de vizinhos já visitados. O percurso encerra-se quando todos os vértices houverem sido visitados.

Evidentemente, sobre um mesmo grafo, podem-se realizar diferentes percursos por vizinhança máxima, uma vez que, em alguns passos, pode haver mais de um vértice com a mesma quantidade máxima de vizinhos já visitados. Como produto final do percurso, fica determinada uma seqüência de vértices refletindo a ordem em que eles foram visitados ao longo do processo.

No Algoritmo 3.1, este percurso é utilizado com o intuito de obter uma representação de um grafo cordal por conjuntos de adjacência restritos. O conjunto V' armazena, a cada passo, os vértices ainda não visitados. Para cada vértice v do grafo, mantêm-se 3 rótulos:

- $visita(v)$ armazena o reverso da ordem em que v foi visitado, da seguinte maneira: o primeiro vértice visitado recebe o valor n para para este rótulo, o segundo, $n-1$ e assim por diante;
- $tam(v)$ armazena o número de vizinhos de v visitados até o momento;
- $X_\sigma(v)$ é um conjunto que armazena os vizinhos de v que já foram visitados quando v é visitado.

Durante a visita a um vértice v, escolhido dentre aqueles que possuem o maior valor de tam, a lista de adjacência de v é varrida e, para cada vértice w nela encontrado, duas situações podem ocorrer:

- se w ainda não foi visitado, w agora possui um vizinho a mais visitado (o vértice v); logo, $tam(w)$ é incrementado;
- se w já foi visitado, w é acrescentado ao conjunto $X_\sigma(v)$.

Sendo G cordal, pode-se demonstrar que, para todo vértice v sendo visitado, o conjunto $\{v\} \cup X_\sigma(v)$ corresponde a uma clique de G. Desta forma, o rótulo $visita$ armazenará, ao final do algoritmo, as posições dos vértices em um *EEP* e o rótulo $X_\sigma(v)$, o conjunto de adjacência restrito com respeito a este esquema.

Vamos acompanhar a execução do percurso sobre o grafo da Figura 3.1.

Algoritmo 3.1. *Obtenção da Representação*

Entrada: Um grafo cordal $G = (V, E)$, armazenado por listas de adjacência $Adj(v)$, $\forall v \in V$;
Saída: $\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$;
Início
  Para $v \in V$ **faça**
    $tam(v) \leftarrow visita(v) \leftarrow 0$; $X_\sigma(v) \leftarrow \emptyset$;
  $i \leftarrow n + 1$; $V' \leftarrow V$; $\mathcal{R} \leftarrow [\ ]$;
  Enquanto $V' \neq \emptyset$ **faça**
    Escolha $v \in V' \mid tam(v)$ seja máximo;
    $V' \leftarrow V' - \{v\}$;
    Para $w \in Adj(v)$ **faça**
      Se $visita(w) = 0$ **então**
        $tam(w) \leftarrow tam(w) + 1$;
      caso contrário
        $X_\sigma(v) \leftarrow X_\sigma(v) \cup \{w\}$;
    $i \leftarrow i - 1$;
    $visita(v) \leftarrow i$; $\sigma(i) \leftarrow v$;
    $\mathcal{R} \leftarrow [(v, X_\sigma(v))] \,||\, \mathcal{R}$;
Fim.

De início, nenhum dos vértices foi ainda visitado e qualquer um deles pode ser escolhido como aquele possuindo o maior número de vizinhos visitados.

Seja a o escolhido, que se torna *visitado*. Os vértices pertencentes a $Adj(a) = \{b, d, e, i\}$ têm o rótulo tam incrementado. A situação é apresentada a seguir.

$V' = \{b, c, d, e, f, g, h, i\}$, $\sigma = [a]$.

v	a	b	c	d	e	f	g	h	i
$visita(v)$	9	0	0	0	0	0	0	0	0
$tam(v)$	0	1	0	1	1	0	0	0	1
$X_\sigma(v)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Os vértices b, d, e e i têm, agora, o maior valor de tam e qualquer um deles pode ser escolhido. Seja i a nossa escolha. Os vértices adjacentes a i são a e d; como a já foi visitado, $tam(a)$ não se modifica e a é incluído em $X_\sigma(i)$.

$V' = \{b, c, d, e, f, g, h\}$, $\sigma = [i, a]$.

v	a	b	c	d	e	f	g	h	i
$visita(v)$	9	0	0	0	0	0	0	0	8
$tam(v)$	0	1	0	2	1	0	0	0	1
$X_\sigma(v)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{a\}$

Neste momento, somente d pode ser escolhido. $Adj(d) = \{a, b, c, e, i\}$; $tam(b)$, $tam(c)$ e $tam(e)$ são modificados; a e i são incluídos em $X_\sigma(d)$.

$V' = \{b, c, e, f, g, h\}$, $\sigma = [d, i, a]$.

v	a	b	c	d	e	f	g	h	i
$visita(v)$	9	0	0	7	0	0	0	0	8
$tam(v)$	0	2	1	2	2	0	0	0	1
$X_\sigma(v)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{a, i\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{a\}$

Agora, b ou e pode ser escolhidos; seja b. O resultado desta escolha é:

$V' = \{c, e, f, g, h\}$, $\sigma = [b, d, i, a]$.

v	a	b	c	d	e	f	g	h	i
$visita(v)$	9	6	0	7	0	0	0	0	8
$tam(v)$	0	2	1	2	3	1	1	1	1
$X_\sigma(v)$	$\emptyset$	$\{a, d\}$	$\emptyset$	$\{a, i\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{a\}$

A próxima escolha é única: o vértice e. Escolhendo, em seguida, sucessivamente os vértices c, f, g e h, tem-se:

$V' = \emptyset$, $\sigma = [h, g, f, c, e, b, d, i, a]$.

v	a	b	c	d	e	f	g	h	i
$visita(v)$	9	6	4	7	5	3	2	1	8
$tam(v)$	0	2	2	2	3	2	3	3	1
$X_\sigma(v)$	$\emptyset$	$\{a, d\}$	$\{d, e\}$	$\{a, i\}$	$\{a, b, d\}$	$\{b, e\}$	$\{b, e, f\}$	$\{b, g, f\}$	$\{a\}$

Neste exemplo, o *EEP* resultante é $\sigma = [h, g, f, c, e, b, d, i, a]$. Observe que, ao final do percurso, $tam(v) = |X_\sigma(v)|$, $\forall v \in V$.

É interessante observar que nem todo *EEP* pode ser obtido através de um percurso por vizinhança máxima, como o *EEP* $[i, a, h, g, f, b, e, c, d]$ para o grafo da Figura 3.1. Seguindo o esquema em ordem reversa, iniciamos o percurso pelo vértice d:

$V' = \{a, b, c, e, f, g, h, i\}$, $\sigma = [d]$.

v	a	b	c	d	e	f	g	h	i
$visita(v)$	0	0	0	9	0	0	0	0	0
$tam(v)$	1	1	1	0	1	0	0	0	1
$X_\sigma(v)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Seguindo o esquema, a próxima escolha recai sobre o vértice c:

$V' = \{a, b, e, f, g, h, i\}$, $\sigma = [c, d]$.

v	a	b	c	d	e	f	g	h	i
$visita(v)$	0	0	8	9	0	0	0	0	0
$tam(v)$	1	1	1	0	2	0	0	0	1
$X_\sigma(v)$	$\emptyset$	$\emptyset$	$\{d\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

A única escolha possível é a do vértice e, que coincide com o esquema:

$V' = \{a, b, f, g, h, i\}$, $\sigma = [e, c, d]$.

v	a	b	c	d	e	f	g	h	i
$visita(v)$	0	0	8	9	7	0	0	0	0
$tam(v)$	2	2	1	0	2	1	1	0	1
$X_\sigma(v)$	$\emptyset$	$\emptyset$	$\{d\}$	$\emptyset$	$\{c, d\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Agora, a ou b podem ser escolhidos; pelo *EEP*, a escolha é b:

$V' = \{a, f, g, h, i\}$, $\sigma = [b, e, c, d]$.

v	a	b	c	d	e	f	g	h	i
$visita(v)$	0	6	8	9	7	0	0	0	0
$tam(v)$	3	2	1	0	2	2	2	1	1
$X_\sigma(v)$	$\emptyset$	$\{d, e\}$	$\{d\}$	$\emptyset$	$\{c, d\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

A única escolha possível, neste ponto, é a do vértice a, que possui 3 vizinhos já visitados. O *EEP*, no entanto, indica-nos o vértice f, donde se conclui que este *EEP* não é produto de um percurso por vizinhança máxima.

3.4 Implementação do Percurso

Para garantir a linearidade do algoritmo de percurso por vizinhança máxima, é necessário agilizar a determinação do vértice com maior valor do rótulo tam, o que pode ser conseguido com o uso de uma estrutura de dados auxiliar.

O conjunto V' deve ser mantido em uma estrutura ortogonal, constituída por uma lista "vertical" duplamente encadeada com nó cabeça, cujos elementos representam valores distintos de tam, ordenados decrescentemente. De cada nó da lista vertical, emerge uma lista "horizontal" duplamente encadeada de vértices (com o mesmo valor de tam), cujos elementos possuem ponteiros de retorno para os nós da lista vertical. Apenas os nós correspondentes a valores de tam sendo utilizados estão presentes na lista vertical. Ou seja, não há listas horizontais vazias.

No Algoritmo 3.1, a manipulação desta estrutura dá-se em três momentos:

- Inicialmente, todos os vértices possuem valor 0 para o rótulo tam e devem ocupar uma única lista horizontal, correspondente a $tam = 0$. Este passo inicial tem complexidade $O(n)$.
- A seleção do vértice em V' com tam máximo resume-se a remover o primeiro nó da lista horizontal que emerge do nó seguinte ao cabeça na lista vertical. Este comando é, portanto, implementado em tempo $O(1)$.
- O incremento de $tam(w)$ consiste em remover w da lista horizontal correspondente a $tam(w)$ e inseri-lo na lista horizontal correspondente a $tam(w) + 1$,

que se inicia no nó imediatamente anterior da lista vertical. Se a lista horizontal onde w se situava tornar-se vazia, o nó correspondente da lista vertical deverá ser suprimido (o acesso a ele é provido pelo ponteiro de retorno à lista vertical); analogamente, a inserção em uma lista horizontal pode acarretar a criação do nó correspondente na lista vertical. Este comando é também implementado em tempo $O(1)$.

Desta maneira, o percurso por vizinhança máxima pode ser implementado por um algoritmo com complexidade de tempo $O(n + m)$.

3.5 Parâmetros Notáveis em Grafos Cordais

As definições e os parâmetros que se seguem, relacionados em geral a grafos arbitrários, são particularmente importantes quando tratamos de grafos cordais.

Coloração e Número Cromático

Uma *coloração* para um grafo $G = (V, E)$ é uma função

$$\tau : V \to C,$$

onde C é um conjunto de cores, de tal forma que,

$$\forall \{v, w\} \in E,\ \tau(v) \neq \tau(w).$$

Trata-se, portanto, de atribuir cores aos vértices do grafo de forma que extremidades de uma mesma aresta recebam cores distintas.

O parâmetro $\chi(G)$, denominado *número cromático* de G, é o menor número possível de cores para o qual existe uma coloração para G. Uma coloração é dita *ótima* quando utiliza exatamente $\chi(G)$ cores.

Cliques Maximais e Tamanho da Maior Clique

Uma *clique maximal* de G é uma clique Q tal que não exista outra clique Q' satisfazendo $Q \subset Q'$. O parâmetro $\omega(G)$ é a cardinalidade da clique máxima de G.

Conjunto Independente Máximo

Um *conjunto independente* de vértices é um subconjunto de V tal que seus elementos não são adjacentes dois a dois. O parâmetro $\alpha(G)$ é a cardinalidade do conjunto independente máximo de G.

Cobertura por Cliques

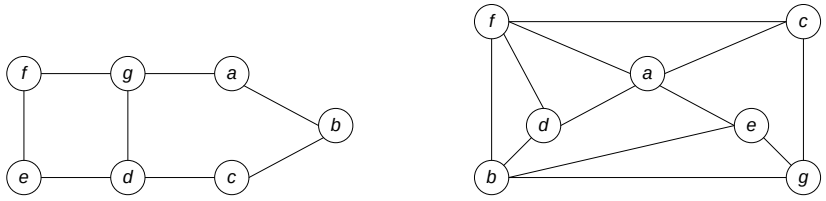
Uma *cobertura por cliques* de G é um conjunto $\{Q_1, \ldots, Q_k\}$ de cliques de V tal que

$$\bigcup_{i=1}^{k} Q_k = V.$$

O parâmetro $\kappa(G)$ é o número de cliques de uma cobertura mínima.

A definição de cobertura por cliques (em inglês *clique cover*) apresenta algumas variações na literatura. Uma delas exige que as cliques $Q_1, Q_2, \ldots, Q_k$ estabeleçam uma partição do conjunto de vértices V. Quando o problema for tratado na Seção 3.7, veremos que a solução para grafos cordais é praticamente a mesma.

A interseção de uma clique com um conjunto independente de um grafo qualquer G possui, no máximo, um vértice. Então, para qualquer grafo G, $\alpha(G) \leq \kappa(G)$. É igualmente imediato concluir que $\omega(G) \leq \chi(G)$.



G $\overline{G}$

Figura 3.5: Um grafo e seu complementar

Um resultado bastante conhecido da literatura relaciona os valores destes quatro parâmetros calculados para um grafo e seu complementar (recordemos que, dado o grafo $G = (V, E)$, o grafo complementar de G é definido como $\overline{G} = (V, \overline{E})$, onde $\overline{E} = \{\{x, y\} \mid x, y \in V \wedge \{x, y\} \notin E\}$):

Teorema 3.3. *Sejam G um grafo qualquer e $\overline{G}$ o seu complementar. Então*

$$\omega(G) = \alpha(\overline{G}) \quad e \quad \kappa(G) = \chi(\overline{G}).$$

No grafo G da Figura 3.5, um conjunto independente máximo tem cardinalidade 3 (por exemplo, $\{f, a, c\}$). Uma cobertura por cliques mínima possui quatro cliques (por exemplo, $\{\{g, f\}, \{a, b\}, \{c, d\}, \{e\}\}$). Então $\alpha(G) = 3 \leq \kappa(G) = 4$. É fácil ver que a cardinalidade da clique máxima é 2; entretanto, só é possível colorir este grafo usando três cores. Então, $\omega(G) = 2 \leq \chi(G) = 3$. Observe o grafo $\overline{G}$ da Figura 3.5, complementar de G. Então, $\omega(\overline{G}) = 3$, $\chi(G) = 4$, $\alpha(\overline{G}) = 2$ e $\kappa(\overline{G}) = 3$.

A determinação de cada um destes parâmetros para grafos quaisquer é um problema NP-difícil: não se conhecem algoritmos de complexidade polinomial capazes de obtê-los. Os problemas tornam-se mais simples quando tratamos de grafos cordais. Primeiramente, é resultado conhecido na literatura que, para grafos cordais, as expressões $\alpha(G) \leq \kappa(G)$ e $\omega(G) \leq \chi(G)$ tornam-se igualdades.

Teorema 3.4. *Seja G um grafo cordal. Então*

$$\chi(G) = \omega(G) \quad e \quad \alpha(G) = \kappa(G).$$

Voltando então ao grafo cordal G da Figura 3.1, constatamos que $\chi(G) = \omega(G) = 4$ e $\alpha(G) = \kappa(G) = 3$.

Nas seções seguintes deste capítulo, analisaremos como esses parâmetros podem ser obtidos para um grafo cordal a partir de sua representação por conjuntos de adjacência restritos.

3.6 Determinação de Cliques Maximais

O primeiro problema estudado, que utiliza a representação por conjunto de adjacência restritos em sua solução, é o da obtenção do conjunto de cliques maximais de um grafo $G = (V, E)$, o que nos permitirá deduzir o valor do tamanho da maior clique $\omega(G)$ e, conseqüentemente, do número cromático $\chi(G) = \omega(G)$. Veremos também como determinar uma coloração ótima para G, utilizando exatamente $\chi(G)$ cores.

Teorema 3.5. *Sejam $G = (V, E)$ um grafo cordal, Q uma clique maximal qualquer de G e $\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$ uma representação de G por conjuntos de adjacência restritos. Então, existe $v \in Q$ tal que $Q = \{v\} \cup X_\sigma(v)$.*

Demonstração: Tomemos $v \in Q$ tal que $\sigma^{-1}(v) = \min\{\sigma^{-1}(w) \mid w \in Q\}$. Todos os vértices de $Q - \{v\}$ são adjacentes a v, então $Q - \{v\} \subseteq X_\sigma(v)$. Suponha que exista $x \in X_\sigma(v) - Q$. Sabe-se, pela definição de *EEP*, que $\{v\} \cup X_\sigma(v)$ é uma clique; ora, esta clique conteria propriamente Q, o que contraria a hipótese de Q ser maximal. □

Por observação do grafo da Figura 3.1 e seu *EEP* $\sigma = [i, a, h, g, f, b, e, c, d]$, suas cliques maximais podem ser determinadas: $\{a, d, i\}$, $\{a, b, d, e\}$, $\{b, h, g, f\}$, $\{b, e, g, f\}$ e $\{c, d, e\}$. Para os vértices i, a, h, g e e, a expressão $\{v\} \cup X_\sigma(v)$ corresponde a uma clique maximal; para os demais vértices do grafo, não.

Considere a construção indutiva apresentada na Seção 3.2. Nesta construção, os vértices são acrescentados ao grafo cordal na ordem inversa à que aparecem no *EEP*. Ao acrescentarmos um novo vértice $\sigma(i)$, este é ligado à clique $X_\sigma(\sigma(i))$ de $G[\{\sigma(i+1), \ldots, \sigma(n)\}]$, de forma que $G[\{\sigma(i), \ldots, \sigma(n)\}]$ é um grafo cordal com um *EEP* $[\sigma(i), \ldots, \sigma(n)]$. Pelo Teorema 3.5, a clique $\{\sigma(i)\} \cup X_\sigma(\sigma(i))$ é maximal em $G[\{\sigma(i), \ldots, \sigma(n)\}]$, já que $\sigma(i)$ é o vértice desta clique de menor posição no *EEP*. Duas situações podem ocorrer:

- Se $X_\sigma(\sigma(i))$ é uma clique maximal em $G[\{\sigma(i+1), \ldots, \sigma(n)\}]$, então ela se transforma na clique maximal $\{\sigma(i)\} \cup X_\sigma(\sigma(i))$ em $G[\{\sigma(i), \ldots, \sigma(n)\}]$;
- Se $X_\sigma(\sigma(i))$ é subconjunto de uma clique maximal em $G[\{\sigma(i+1), \ldots, \sigma(n)\}]$, $X_\sigma(\sigma(i))$ e $\{\sigma(i)\} \cup X_\sigma(\sigma(i))$ são cliques maximais em $G[\{\sigma(i), \ldots, \sigma(n)\}]$.

O Algoritmo 3.2 determina as cliques maximais $Q_1, \ldots, Q_t$ de um grafo cordal G, representado por conjuntos de adjacência restritos. O *EEP* implícito na representação é percorrido em ordem reversa e, para cada vértice $\sigma(i)$, são executados os seguintes passos:

- Passo 1: determinar o vértice $v \in X_\sigma(\sigma(i))$ de menor posição no *EEP*;
- Passo 2: reconhecer se a clique $X_\sigma(\sigma(i))$ é maximal no grafo corrente. O rótulo $N(v)$ armazena o número da primeira clique maximal Q obtida durante o processo à qual v pertence. Como Q contém todas as subcliques possíveis que possuem v como menor vértice, $X_\sigma(\sigma(i))$ é maximal se, e somente se, $|X_\sigma(\sigma(i))| = |Q|$.

Algoritmo 3.2. *Determinação das cliques maximais e de $\omega(G)$*

Entrada: Um grafo cordal $G = (V, E)$, representado por
$\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$;
Saída: As cliques maximais $Q_1, \ldots, Q_t$;
O tamanho $\omega(G)$ da maior clique;
Início
$t \leftarrow 1$; $Q_1 \leftarrow \{\sigma(n)\}$; $N(\sigma(n)) \leftarrow 1$; $\omega(G) \leftarrow 0$;
Para $i \leftarrow n-1, \ldots, 1$ **faça**
$v \leftarrow \sigma(n)$;
Para $w \in X_\sigma(\sigma(i))$ **faça**
Se $\sigma^{-1}(w) < \sigma^{-1}(v)$ **então**
$v \leftarrow w$;
$k \leftarrow N(v)$;
Se $|X_\sigma(\sigma(i))| < |Q_k|$ **então**
$k \leftarrow t \leftarrow t + 1$; % Nova clique maximal
$Q_k \leftarrow \{\sigma(i)\} \cup X_\sigma(\sigma(i))$;
caso contrário
$Q_k \leftarrow \{\sigma(i)\} \cup Q_k$; % Expansão de uma clique antiga
$N(\sigma(i)) \leftarrow k$;
Se $|Q_k| > \omega(G)$ **então** $\omega(G) \leftarrow |Q_k|$;
Fim.

Com o uso de listas encadeadas para armazenar as cliques maximais, é possível implementar cada uma das operações sobre conjuntos mencionadas no Algoritmo 3.2 em tempo $O(1)$. Assim, a complexidade total de tempo é $O(m)$, considerando que, para cada vértice do *EEP*, seu conjunto de adjacência restrito é percorrido.

Sendo G cordal, sabemos que $\omega(G) = \chi(G)$ e um algoritmo para obter uma coloração ótima pode ser concebido a partir dessas mesmas idéias. Novamente a construção indutiva é utilizada. Seja v o vértice adicionado; logo, os vértices de $X_\sigma(v)$ já foram adicionados em passos anteriores; o vértice v recebe então uma cor distinta das cores empregadas nestes vértices. Observe que, desta forma, o número máximo de cores empregadas é $\omega(G)$, uma vez que o vértice v é simplicial no grafo em construção, e a cardinalidade da maior clique do mesmo é conhecida.

Para tornar o algoritmo mais eficiente, uma estrutura auxiliar é empregada: o vetor aux de dimensão $\omega(G)$, que armazena a cada iteração as cores já empregadas em $X_\sigma(v)$. Para evitar a inicialização do vetor a cada passo, em vez de um valor booleano, é utilizado o número da iteração corrente. A complexidade do Algoritmo 3.3 é também $O(m)$ graças a este vetor.

Aplicando-se o Algoritmo 3.3 ao exemplo da Figura 3.1, obtemos a seguinte coloração para seus vértices: $Cor(a) = 4$, $Cor(b) = 2$, $Cor(c) = 2$, $Cor(d) = 1$, $Cor(e) = 3$, $Cor(f) = 1$, $Cor(g) = 4$, $Cor(h) = 3$ e $Cor(i) = 2$.

Algoritmo 3.3. *Coloração Ótima de um Grafo Cordal*

Entrada: Um grafo cordal $G = (V, E)$, representado por
$\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$;
$\omega(G) = \chi(G)$;
Saída: $Cor : V \rightarrow \{1, \ldots, \chi(G)\}$, uma coloração ótima para G;
Início
  Para $i \leftarrow 1, \ldots, \omega(G)$ **faça**
    $aux[i] \leftarrow 0$;
  Para $v \in V$ **faça**
    $Cor(v) \leftarrow 0$;
  Para $i \leftarrow n, \ldots, 1$ **faça**
    $v \leftarrow \sigma(i)$;
    Para $w \in X_\sigma(v)$ **faça**
      $aux[Cor(w)] \leftarrow i$;
    $j \leftarrow 1$;
    Enquanto $Cor(v) = 0$ **faça**
      Se $aux[j] \neq i$ **então**
        $Cor(v) \leftarrow j$;
      caso contrário
        $j \leftarrow j + 1$;
Fim.

3.7 Conjunto Independente Máximo e Cobertura por Cliques Mínima

Nesta seção, dois importantes problemas são resolvidos para grafos cordais: a determinação de um conjunto independente máximo e de uma cobertura por cliques mínima.

Para tal, precisamos determinar uma nova seqüência, y_i, $i = 1, \ldots, t$, baseada na representação do grafo por conjuntos de adjacência restritos. Seja $y_1 = \sigma(1)$; y_i é o primeiro vértice em σ que segue y_{i-1} e que não pertence a $\bigcup_{j=1}^{i-1}\{X_\sigma(y_j)\}$.

Teorema 3.6. *Sejam G um grafo cordal e $\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$ uma representação de G por conjuntos de adjacência restritos. Então, o conjunto*

$\{y_1, y_2, \ldots, y_t\}$ é um conjunto independente máximo de G e a coleção de conjuntos $Y_i = \{y_i\} \cup X_\sigma(y_i)$, para $i = 1, 2, \ldots t$, forma uma cobertura por cliques de G, sendo $t = \alpha(G) = \kappa(G)$.

Demonstração: $\{y_1, y_2, \ldots, y_t\}$ é um conjunto independente, uma vez que se $\{y_j, y_i\} \in E$ para $j < i$, então $y_i \in X_\sigma(y_j)$, o que contraria a definição da seqüência. Logo,

$$\alpha(G) \geq t.$$

Por outro lado, cada um dos conjuntos $Y_i = \{y_i\} \cup X_\sigma(y_i)$ é uma clique e, juntos, formam uma cobertura de G (pela definição da seqüência). Assim

$$\kappa(G) \leq t.$$

Como G é cordal, sabe-se que $\alpha(G) = \kappa(G)$. Então

$$t \leq \alpha(G) = \kappa(G) \leq t. \quad \square$$

Em decorrência deste resultado, chamaremos a seqüência definida no início desta seção de *seqüência independente*.

O algoritmo para a determinação do conjunto independente máximo baseia-se em uma varredura da representação por conjuntos de adjacência restritos: a cada vértice v encontrado que pertencença à seqüência independente, marcam-se os vértices de $X_\sigma(v)$.

Algoritmo 3.4. *Conjunto Independente Máximo*

Entrada: Um grafo cordal $G = (V, E)$, representado por
$\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$;
Saída: Ind, um conjunto independente máximo de vértices de G;
Início
  Para $v \in V$ **faça**
    $marca(v) \leftarrow false$;
  $v \leftarrow \sigma(1)$;
  $Ind \leftarrow \{v\}$;
  Para $w \in X_\sigma(v)$ **faça**
    $marca(w) \leftarrow true$;
  Para $i \leftarrow 2, \ldots, n$ **faça**
    $v \leftarrow \sigma(i)$;
    Se not $marca(v)$ **então**
      $Ind \leftarrow Ind \cup \{v\}$;
      Para $w \in X_\sigma(v)$ **faça**
        $marca(w) \leftarrow true$;
Fim.

O Teorema 3.6 mostra como encontrar diretamente a cobertura por cliques mínima a partir do conjunto Ind determinado pelo Algoritmo 3.4: para cada vértice $v \in Ind$, determinar o subconjunto $\{v\} \cup X_\sigma(v)$.

No grafo da Figura 3.1, encontramos o conjunto independente máximo $\{i, h, e\}$ e a cobertura mínima $\{\{i, a, d\}, \{h, b, f, g\}, \{e, c, d\}\}$. Note que alguns subconjuntos se interceptam. A cobertura por cliques obtida é mínima pois tem número de subconjuntos igual à cardinalidade do conjunto independente máximo. Então, para determinar uma cobertura por cliques que estabeleça uma partição do conjunto V, basta evitar a utilização de um vértice mais de uma vez. O Algoritmo 3.5 mostra esta alternativa.

Algoritmo 3.5. *Cobertura por Cliques Mínima (partição)*

Entrada: Um grafo cordal $G = (V, E)$, representado por
$\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$;
O conjunto independente máximo Ind associado;
Saída: Uma cobertura $C_1, \ldots C_{\alpha(G)}$ por cliques de G;
Início
  Para $v \in V$ **faça**
    $marca(v) \leftarrow false$;
  $i \leftarrow 0$;
  Para $v \in Ind$ **faça**
    $marca(v) \leftarrow true$;
    $i \leftarrow i + 1$; $C_i \leftarrow \{v\}$;
    Para $w \in X_\sigma(v)$ **faça**
      Se not $marca(w)$ **então**
        $C_i \leftarrow C_i \cup \{w\}$;
Fim.

3.8 Grafos Periplanares Maximais

Nesta seção, apresentaremos um interessante caso particular de grafos cordais: a família dos grafos periplanares maximais. Vamos mostrar como a representação por conjuntos de adjacência restritos permite-nos resolver, com facilidade, o problema da obtenção do ciclo hamiltoniano de um grafo desta família.

Um grafo é dito *periplanar* se ele é *planar* (i.e., admite uma representação geométrica sobre uma superfície plana sem cruzamento de arestas) e todos os seus vértices podem situar-se em uma única face da representação. Na Figura 3.6, vemos um grafo planar (o grafo completo com quatro vértices) que não é periplanar.

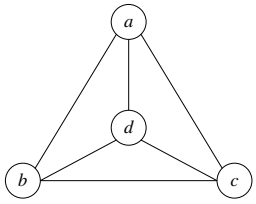


Figura 3.6: Um grafo planar que não é periplanar

Um grafo periplanar é *maximal* quando a adição de uma aresta entre dois vértices não adjacentes resulta em um grafo não periplanar. Grafos periplanares maximais (do inglês *maximal outerplanar graphs - mops*) são grafos cordais planares. Observe que nem todo periplanar é cordal; tal afirmativa pode ser feita apenas para os periplanares maximais. A Figura 3.7 mostra um grafo periplanar não cordal, um periplanar maximal e um periplanar cordal não maximal.

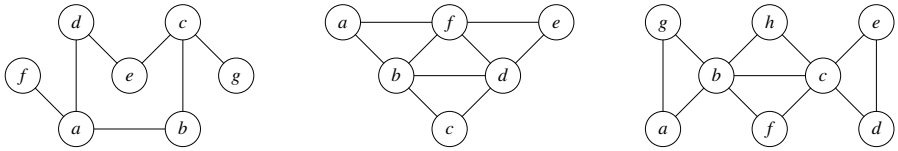


Figura 3.7: Três grafos periplanares: não cordal, maximal e cordal não maximal

Pode-se demonstrar que todo mop possui um ciclo hamiltoniano (i.e., um ciclo simples contendo todos os vértices do grafo). As arestas do mop que pertencem ao ciclo hamiltoniano são chamadas *externas*; as demais, *internas*. Em todo mop, existem n arestas externas e $n-3$ arestas internas, totalizando $2n-3$ arestas.

Um mop com $n \geq 3$ vértices pode ser indutivamente definido como se segue:

- Um grafo completo com 3 vértices é um mop.
- Se $G = (V, E)$ é um mop, $v \notin V$ e $\{s, t\} \in E$ é uma aresta externa, então $G' = (V \cup \{v\}, E \cup \{\{v, s\}, \{v, t\}\})$ é um mop.
- Nada mais são mops.

Um exemplo da construção indutiva de um mop é apresentado na Figura 3.8. Observe que todo vértice é ligado a uma clique de tamanho 2 (uma aresta externa); ao ser adicionado, a aresta à qual ele é ligado torna-se interna e surgem duas novas arestas externas, que passam a integrar o novo ciclo hamiltoniano.

Sendo um grafo cordal, um mop pode ser representado por conjuntos de adjacência restritos. Para o grafo da Figura 3.8 e o *EEP* $[i, h, g, f, e, d, c, b, a]$, fornecido por um percurso por vizinhança máxima, a representação é:

$$\begin{array}{llllll} [& (i, \{e, g\}), & (h, \{f, g\}), & (g, \{e, f\}), & (f, \{c, e\}), & \\ & (e, \{b, c\}), & (d, \{a, c\}), & (c, \{a, b\}), & (b, \{a\}), & (a, \emptyset) \quad]. \end{array}$$

Devido a particularidades estruturais dos mops, sua representação por conjuntos de adjacência restritos possui características especiais, como nos revela o Teorema 3.7.

Teorema 3.7. *Seja $G = (V, E)$ um grafo cordal, representado por $\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$. G é periplanar maximal se, e somente se,*

- $|X_\sigma(\sigma(i))| = 2$, *para* $i = 1, \ldots, n-2$;
- $X_\sigma(\sigma(i)) \neq X_\sigma(\sigma(j))$, $\forall i, j < n-2, i \neq j$.

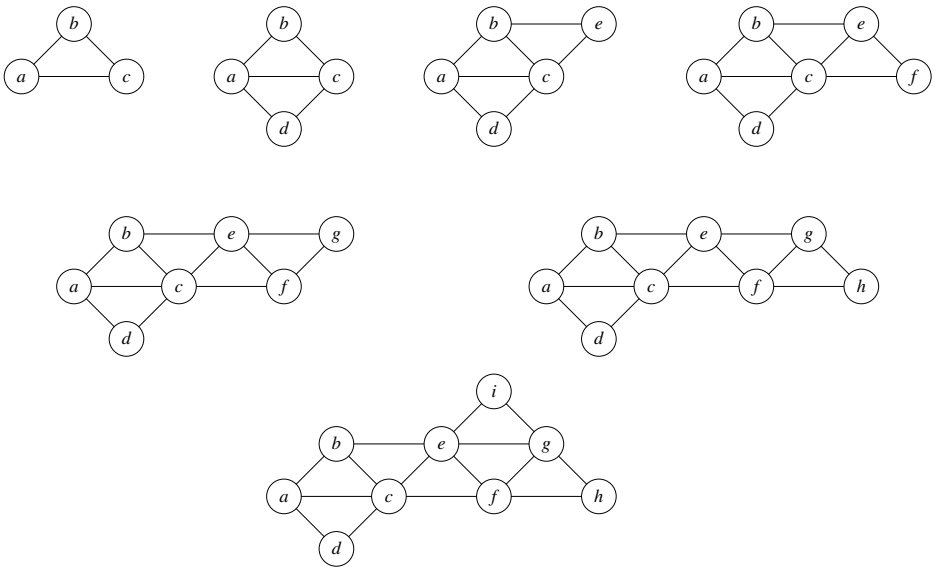


Figura 3.8: A construção de um mop

Observando o exemplo apresentado, podemos interpretar intuitivamente o que nos diz o teorema. De início, para qualquer grafo cordal conexo tem-se $|X_\sigma(\sigma(n))| = 0$ e $|X_\sigma(\sigma(n-1))| = 1$. Já sabemos ser possível construir um grafo cordal de acordo com a varredura de sua representação por conjuntos de adjacência restritos em ordem reversa. O primeiro item exige, portanto, que todo vértice novo, ao ser adicionado, deve ser ligado a uma clique de tamanho dois (uma aresta). O segundo item é também de fácil compreensão. Pela definição de mop, a construção deve ser iniciada a partir de uma clique de tamanho 3, formada, então, pelos vértices $\sigma(n)$, $\sigma(n-1)$ e $\sigma(n-2)$. Nas iterações seguintes, cada novo vértice deve ser ligado a uma aresta exterior, que se torna interna após o acréscimo do vértice. Daí conclui-se que cada aresta pode ser utilizada para ligar um novo vértice por uma única vez; além disso, as cliques $X_\sigma(\sigma(i))$, para $i = 1, \ldots, n-3$, são exatamente as arestas internas do ciclo hamiltoniano.

A representação por conjuntos de adjacência restritos conduz à determinação do ciclo hamiltoniano de um mop, como é mostrado no Algoritmo 3.6. Na implementação do algoritmo, *Lista* deve ser uma lista circular duplamente encadeada de vértices. Além disso, deve ser possível localizar um vértice nesta lista em $O(1)$, o que pode ser conseguido com o auxílio de um vetor de ponteiros, de cada vértice para o nó em *Lista* que lhe corresponde. Esta estrutura permite que as inserções na lista circular sejam feitas em $O(1)$ e que a complexidade total do algoritmo seja $O(n)$.

Algoritmo 3.6. *Ciclo hamiltoniano em um mop*

Entrada: Um mop $G = (V, E)$, representado por
$\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$;
Saída: $Lista$, seqüência hamiltoniana de vertices de G;
Início
  $Lista \leftarrow [\sigma(n), \sigma(n-1), \sigma(n-2)]$;
  Para $i \leftarrow n-3, \ldots, 1$ **faça**
    Seja $X_\sigma(\sigma(i)) = \{u, v\}$;
    Inserir $\sigma(i)$ entre u e v em $Lista$;
Fim.

Acompanhando passo a passo o algoritmo aplicado ao mop da Figura 3.8, podemos observar a formação do ciclo hamiltoniano. O último vértice aparece repetido apenas para enfatizar a circularidade da lista.

i	$\sigma(i)$	$X_\sigma(\sigma(i))$	$Lista$
			$[a, b, c, a]$
6	d	$\{a, c\}$	$[a, b, c, \underline{d}, a]$
5	e	$\{b, c\}$	$[a, b, \underline{e}, c, d, a]$
4	f	$\{c, e\}$	$[a, b, e, \underline{f}, c, d, a]$
3	g	$\{e, f\}$	$[a, b, e, \underline{g}, f, c, d, a]$
2	h	$\{g, f\}$	$[a, b, e, g, \underline{h}, f, c, d, a]$
1	i	$\{e, g\}$	$[a, b, e, \underline{i}, g, h, f, c, d, a]$

A representação por conjuntos de adjacência restritos e o ciclo hamiltoniano de um mop podem ser utilizados como entrada para um outro algoritmo, capaz de traçar o mop automaticamente, produzindo uma representação geométrica sem cruzamento de arestas e com todos os vértices ocupando uma única face. A idéia é dispor o ciclo hamiltoniano sobre uma circunferência, traçando, em seguida, as arestas internas. A Figura 3.9 mostra o traçado do mop construído na Figura 3.8.

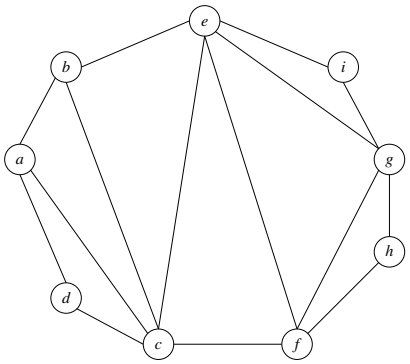


Figura 3.9: Traçado de um mop

3.9 Exercícios

1. Demonstre que a família das árvores é subconjunto próprio da família dos grafos cordais. Em outras palavras: toda árvore é um grafo cordal, mas não vale a recíproca.

2. Quem são os vértices simpliciais de uma árvore ?

3. Sejam $G = (V, E)$ um grafo cordal e $\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$ a representação de G por conjuntos de adjacência restritos segundo um *EEP* σ qualquer. Mostre que $\sum_{v \in V} |X_\sigma(v)| = m$.

4. Esboce um algoritmo com complexidade $O(m)$ para obter a representação por conjuntos de adjacência a partir da representação por conjuntos de adjacência restritos de um grafo cordal.

5. Sobre o Algoritmo 3.1:

   (a) Que relação existe entre os rótulos $visita$ e σ ?

   (b) O que significa a igualdade $visita(w) = 0$?

   (c) Se a escolha de um vértice de V' com máximo valor de tam for implementada através de uma varredura exaustiva do conjunto V', qual será a complexidade de tempo total do algoritmo ?

   (d) Refine o algoritmo de modo que a representação $\mathcal{R}_\sigma(G)$ obtida seja armazenada em memória conforme a estrutura sugerida na Figura 3.4.

6. Detalhe a implementação do Algoritmo 3.1 sugerida na Seção 3.4.

7. Demonstre ou forneça um contra-exemplo: o grafo complementar de uma árvore é conexo.

8. Determine os parâmetros $\omega(G)$, $\alpha(G)$, $\kappa(G)$ e $\chi(G)$, sendo G

   - $K_{1,n}$, o grafo estrela definido no Exercício 4 da Seção 1.8.
   - uma árvore degenerada com n vértices.

9. Uma *k-árvore*, sendo $k > 0$, pode ser indutivamente definida da seguinte maneira:

   - Um grafo completo com k vértices é uma k-árvore.
   - Se $G = (V, E)$ é uma k-árvore, $Q \subseteq V$ é uma clique de G com cardinalidade k e $v \notin V$, então o grafo

     $$G' = (V \cup \{v\}, E \cup \{\{v, w\} \mid w \in Q\})$$

     é uma k-árvore.
   - Nada mais são k-árvores.

Pede-se:

(a) Represente geometricamente uma 2-árvore com 8 vértices que não seja um grafo periplanar maximal (veja Seção 3.8).

(b) Quantos vértices simpliciais, no mínimo e no máximo, pode possuir uma k-árvore com:

- exatamente k vértices ?
- exatamente $k+1$ vértices ?
- n vértices, sendo $n > k+1$?

(c) Determine, em função de n e k, o número de arestas de uma k-árvore com n vértices, sendo $n \geq k$.

(d) Inspirando-se na construção indutiva dos grafos cordais, estudada na Seção 3.2, estabeleça para esta família uma definição indutiva, semelhante à apresentada para k-árvores.

(e) Compare as duas definições indutivas (a de grafos cordais obtida no item anterior e a de k-árvores apresentada no enunciado). Identifique a particularização ocorrida e conclua que a família das k-árvores é subconjunto próprio da família dos grafos cordais.

(f) Que relação se pode deduzir entre a família das k-árvores e a família das árvores ?

(g) Identifique os grafos que são simultaneamente k-árvores e $(k+1)$-árvores, para um dado $k > 0$ fixo.

(h) Que particularidade possui a representação de uma k-árvore por conjuntos de adjacência restritos ?

(i) Quanto vale $\omega(G)$, sendo G uma k-árvore ?

3.10 Notas Bibliográficas

A família dos grafos cordais é uma das mais extensamente estudadas em Teoria de Grafos, havendo vasta literatura disponível a seu respeito. Referências essencias são o livro de Golumbic [13] e o artigo de Peyton e Blair [4]. Os algoritmos de percurso mais utilizados no reconhecimento de grafos cordais são o percurso em largura lexicográfica, proposto por Rose, Tarjan e Lueker [28], e o percurso por vizinhança máxima, proposto por Tarjan e Yannakakis [33]. Métodos alternativos são apresentados por Panda [24]. A solução dos problemas tratados neste capítulo para grafos cordais pode ser encontrada em [12]. Syslo [29], Beyer *et al.* [3] e Mitchell [22] são referências introdutórias sobre grafos periplanares maximais. Um algoritmo eficiente de reconhecimento de um mop é encontrado em Markenzon e Justel [15]. Na área de traçado automático de mops, podemos citar Markenzon e Paciornik [18], que trata do traçado equilátero de um mop.

Capítulo 4

Codificação de Grafos k-Caminho

4.1 Conceitos Básicos

Na Seção 1.4, vimos que, para certas famílias específicas de grafos, é possível definir esquemas particulares de representação, que requerem a menção de menos símbolos do que os esquemas habitualmente empregados para representar grafos arbitrários (i.e., conjuntos e matriz de adjacência).

Neste capítulo, estudaremos a família dos grafos k-caminho, cuja disposição particular das cliques maximais permite conceber uma representação bastante econômica. Os grafos k-caminho são uma generalização natural do conceito de caminho simples, que corresponde ao caso $k = 1$.

A definição de caminho simples, mencionada na Seção 1.1, pode ser assim reescrita: um caminho simples de comprimento $p > 0$ é uma seqüência alternada de vértices e arestas, todos distintos:

$$[v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \ldots, v_{p-1}, \{v_{p-1}, v_p\}, v_p].$$

Por abuso de linguagem, podemos dizer que, em um caminho simples, alternam-se cliques de cardinalidades 1 (vértices) e 2 (arestas).

Um grafo que consiste apenas de um caminho simples é denominado *grafo caminho*. Em particular, $P_n = (\{1, \ldots, n\}, \{\{1, 2\}, \{2, 3\}, \ldots, \{n - 1, n\}\})$, $n > 1$. Observe que um grafo caminho pode ser construído a partir de outro já existente, mediante a adição de um novo vértice a uma das extremidades.

Dado um inteiro $k > 0$, um *k-caminho de comprimento $p > 0$* em um grafo arbitrário $G = (V, E)$ é uma seqüência

$$[B_0, C_1, B_1, C_2, B_2, \ldots, C_p, B_p],$$

onde:

- $B_i \subset V$, $0 \leq i \leq p$, são k-cliques distintas de G;
- $C_i \subseteq V$, $1 \leq i \leq p$, são $(k+1)$-cliques distintas de G;
- $B_{i-1} \subset C_i$, $B_i \subset C_i$ e nenhuma outra k-clique B_j, $0 \leq j \leq p$, $j \neq i-1$ e $j \neq i$, é um subconjunto de C_i, $1 \leq i \leq p$.

Por esta definição, fica claro que um caminho simples em um grafo G é exatamente um 1-caminho em G.

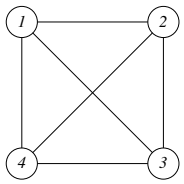


Figura 4.1: O grafo completo com quatro vértices

Considerando o grafo G da Figura 4.1, a seqüência

$$[\{1,2\},\{1,2,3\},\{2,3\},\{2,3,4\},\{3,4\},\{1,3,4\},\{1,4\}]$$

é um 2-caminho em G. Entretanto

$$[\{1,2\},\{1,2,3\},\{2,3\},\{2,3,4\},\{3,4\},\{1,3,4\},\{1,4\},\{1,2,4\},\{2,4\}]$$

não é um 2-caminho, porque $B_4 = \{2,4\} \subset C_2 = \{2,3,4\}$ e a terceira condição é violada.

Um *grafo k-caminho* pode ser indutivamente definido da seguinte maneira:

- Um grafo completo com $k+1$ vertices é um grafo k-caminho.
- Se $G = (V, E)$ é um grafo k-caminho, $Q \subset V$ é uma k-clique de G contendo ao menos um vértice simplicial e $v \notin V$, então o grafo

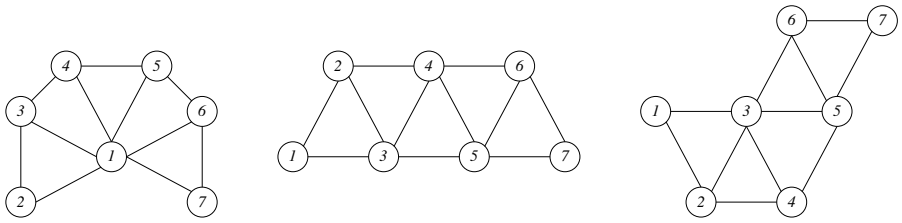
  $$G' = (V \cup \{v\}, E \cup \{\{v, w\} \mid w \in Q\})$$

  é um grafo k-caminho.
- Nada mais são grafos k-caminho.

Sem perda de generalidade, assumiremos que os grafos k-caminho tratados neste capítulo possuem como conjunto de vértices $V = \{1, 2, \ldots, n\}$, sendo $n > k \geq 1$.

É importante observar que um k-caminho maximal (i.e., aquele que não é subconjunto de nenhum outro k-caminho) em um grafo k-caminho contém obrigatoriamente todos os vértices do mesmo. A Figura 4.2 mostra três exemplos de grafos 2-caminho com sete vértices. No grafo G_1, podemos destacar o 2-caminho maximal

$[\{1,2\}, \{1,2,3\}, \{1,3\}, \{1,3,4\}, \{1,4\}, \{1,4,5\}, \{1,5\}, \{1,5,6\}, \{1,6\}, \{1,6,7\}, \{1,7\}]$; para o mesmo grafo, podemos também exibir o 2-caminho maximal $[\{2,3\}, \{1,2,3\}, \{1,3\}, \{1,3,4\}, \{1,4\}, \{1,4,5\}, \{1,5\}, \{1,5,6\}, \{1,6\}, \{1,6,7\}, \{1,7\}]$. É interessante notar que G_1, G_2 e G_3 são também grafos periplanares maximais (mops), estudados na Seção 3.8.



G_1 G_2 G_3

Figura 4.2: Três grafos 2-caminho

Pela definição, concluímos facilmente que grafos k-caminho são grafos cordais, visualizando aí seu processo indutivo de construção, durante o qual as cliques às quais os vértices são ligados apresentam a restrição de possuírem pelo menos um vértice simplicial. Esta restrição confere aos grafos desta família características estruturais bastante particulares.

A primeira delas é o fato de que todo grafo k-caminho com $n > k+1$ vértices possui exatamente dois vértices simpliciais. Logo, é sempre possível obter, para tais grafos, um *EEP* em que esses vértices ocupem as posições extremas: se iniciarmos o percurso por vizinhança máxima em um dos vértices simpliciais, ele ocupará a última posição no *EEP* e, como o primeiro vértice de um *EEP* é obrigatoriamente simplicial, o outro ocupará a primeira. A seguir, apresentamos a representação por conjuntos de adjacência restritos dos grafos da Figura 4.2; o *EEP* considerado, nos três casos, obedece a este princípio de construção.

$$\mathcal{R}_\sigma(G_1) = [(2,\{1,3\}),(3,\{1,4\}),(4,\{1,5\}),(5,\{1,6\}),(1,\{6,7\}),(6,\{7\}),(7,\emptyset)]$$

$$\mathcal{R}_\sigma(G_2) = [(1,\{2,3\}),(2,\{3,4\}),(3,\{4,5\}),(4,\{5,6\}),(5,\{6,7\}),(6,\{7\}),(7,\emptyset)]$$

$$\mathcal{R}_\sigma(G_3) = [(1,\{2,3\}),(2,\{3,4\}),(4,\{3,5\}),(3,\{5,6\}),(5,\{6,7\}),(6,\{7\}),(7,\emptyset)]$$

Durante o processo indutivo de construção, os $k+1$ últimos vértices do *EEP* formam uma $(k+1)$-clique. Uma vez formada esta clique, cada acréscimo subseqüente unirá um novo vértice a uma k-clique do grafo corrente, formando uma nova clique maximal de cardinalidade $k+1$. O número de arestas de um grafo k-caminho é então $m = kn - \frac{k(k+1)}{2}$.

Lema 4.1. *Seja $G = (V,E)$ um grafo k-caminho. Então existem $n-k$ cliques maximais de cardinalidade $k+1$ em G.*

Corolário 4.1. *Não existem cliques de cardinalidade maior do que $k+1$ em um grafo k-caminho.*

Corolário 4.2. *Os dois vértices simpliciais de um grafo k-caminho têm grau k.*

Ainda durante o processo indutivo de construção de um grafo k-caminho, cada novo vértice, ao ser inserido, é ligado a uma clique que possua pelo menos um vértice simplicial. Como o primeiro vértice inserido é simplicial no grafo final, este novo vértice é obrigatoriamente ligado ao vértice inserido imediatamente antes dele, isto é, aquele que o segue no *EEP*.

Lema 4.2. *Seja $G = (V, E)$ um grafo k-caminho. Todos os k-caminhos maximais de G possuem a mesma subseqüência de $(k+1)$-cliques $[C_1, C_2, \ldots, C_{n-k}]$, de modo que os dois vértices simpliciais pertencem às cliques extremas C_1 e C_{n-k}.*

É possível determinar o número de k-caminhos maximais distintos em um grafo k-caminho.

Lema 4.3. *Seja $G = (V, E)$ um grafo k-caminho. Então existem k^2 k-caminhos maximais em G.*

Finalmente, o Teorema 4.1 fornece uma caracterização de grafos k-caminho, que nos permite elaborar um algoritmo linear de reconhecimento para os membros da família.

Teorema 4.1. *Sejam $k > 0$, $G = (V, E)$ um grafo cordal com $n > k+1$ vértices e $\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$ uma representação de G por conjuntos de adjacência restritos. G é um grafo k-caminho se, e somente se,*

- $|X_\sigma(\sigma(i))| = \begin{cases} k, & \text{se } 1 \leq i \leq n-k \\ n-i, & \text{se } n-k+1 \leq i \leq n \end{cases}$;
- *G possui exatamente dois vértices simpliciais.*

4.2 Esquema de Representação

Seja $G = (V, E)$ um grafo k-caminho. Uma vez que existem k^2 k-caminhos maximais contendo todas as $n-k$ cliques de cardinalidade $(k+1)$ de G, o grafo não pode ser univocamente representado por um destes k-caminhos. Entretanto, dada a seqüência $[C_1, C_2, \ldots, C_{n-k}]$, é possível construir o grafo correspondente sem ambigüidades. Note que a seqüência reversa $[C_{n-k}, \ldots, C_2, C_1]$ é uma representação equivalente para o mesmo grafo. Para que a duplicidade não ocorra, fixamos que o menor vértice simplicial deve pertencer à clique C_1; neste caso a seqüência é única e é chamada *seqüência fundamental*, denotada $\mathcal{SF}(G)$.

Lema 4.4. *Sejam $k > 0$, $G = (V, E)$ um grafo k-caminho e uma representação de G por conjuntos de adjacência restritos $\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$ em que $\sigma(1)$ é o menor vértice simplicial de G e $\sigma(n)$, o maior. Então*

$$C_i = \{\sigma(i)\} \cup X_\sigma(\sigma(i)),$$

para $i = 1, \ldots, n-k$, são as cliques maximais da seqüência fundamental.

Demonstração: Sabemos, pelo Lema 4.1, que existem $n - k$ cliques maximais em G. Para determinar as cliques maximais, devemos percorrer o *EEP* dado em ordem inversa. Os últimos $k + 1$ vértices pertencem a uma única clique, de cardinalidade $k + 1$, que contém o vértice simplicial $\sigma(n)$. A cada iteração, a partir da $(k + 1)$-ésima, uma nova clique é determinada, uma vez que todos os conjuntos de adjacência restritos têm igual cardinalidade. Cada uma destas cliques tem k vértices em comum com outra já encontrada; e mais, um dos vértices desta clique é aquele processado na iteração anterior, uma vez que, pela definição de grafo k-caminho, o novo vértice deve ser ligado a uma clique que possua um vértice simplicial. Este vértice estabelece, então, uma nova clique maximal. □

A determinação da seqüência fundamental de um grafo k-caminho G compreende, portanto, dois passos:

- Determinar uma representação de G por conjuntos de adjacência restritos tal que seu primeiro vértice seja o simplicial de menor número de G e o último, o simplicial de maior número.
  - Encontrar os vértices simpliciais de G, que são os vértices de grau k.
  - Executar um percurso por vizinhança máxima, tendo como vértice inicial o simplicial de maior número.
- Determinar, para $1 \leq i \leq n - k$, $C_i = \{\sigma(i)\} \cup X_\sigma(\sigma(i))$.

Por ser única, a seqüência fundamental de um grafo k-caminho constitui um código para este grafo (Seção 1.7). No entanto, é possível obter um código que necessite de uma quantidade ainda menor de símbolos para representar um grafo k-caminho. As $(k + 1)$-cliques que compõem a seqüência fundamental exibem uma peculiar propriedade estrutural, provada no Lema 4.5, que suporta a definição de *seqüência reduzida*.

Lema 4.5. *Se $\mathcal{SF}(G) = [C_1, C_2, \ldots, C_{n-k}]$ é a seqüência fundamental de um grafo k-caminho com $n > k + 1$ vértices, então os conjuntos $C_i - C_{i+1}$ e $C_{i+1} - C_i$, $1 \leq i < n - k$, são unitários.*

Demonstração: Porque $|C_i| = |C_{i+1}| = k + 1$, $|C_i \cap C_{i+1}| = k$ e $C_i \neq C_{i+1}$. □

Com base no Lema 4.5, apresentamos a seguinte definição:

Sejam $G = (V, E)$ um grafo k-caminho e $\mathcal{SF}(G)$ sua seqüência fundamental. Denominemos

$$C_i - C_{i+1} = \{\ell_i\} \text{ e } C_{i+1} - C_i = \{r_i\}, 1 \leq i < n - k.$$

A seqüência

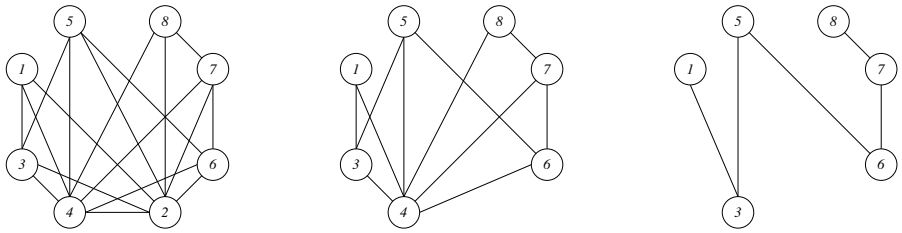
$$\mathcal{SR}(G) = [(\ell_1, r_1), (\ell_2, r_2), \ldots, (\ell_{n-k-1}, r_{n-k-1})]$$

é chamada *seqüência reduzida* de G. Quando $n = k + 1$, $\mathcal{SR}(G) = [\,]$ é vazia.

A seqüência reduzida $\mathcal{SR}(G)$ é constituída por $n - k - 1$ pares de vértices. Para os grafos 2-caminho da Figura 4.2, tem-se:

$$\begin{aligned}
\mathcal{SF}(G_1) &= [\{1,2,3\},\{1,3,4\},\{1,4,5\},\{1,5,6\},\{1,6,7\}] \\
\mathcal{SR}(G_1) &= [(2,4),(3,5),(4,6),(5,7)] \\
\mathcal{SF}(G_2) &= [\{1,2,3\},\{2,3,4\},\{3,4,5\},\{4,5,6\},\{5,6,7\}] \\
\mathcal{SR}(G_2) &= [(1,4),(2,5),(3,6),(4,7)] \\
\mathcal{SF}(G_3) &= [\{1,2,3\},\{2,3,4\},\{3,4,5\},\{3,5,6\},\{5,6,7\}] \\
\mathcal{SR}(G_3) &= [(1,4),(2,5),(4,6),(3,7)]
\end{aligned}$$

Observemos, agora, o grafo 3-caminho G_1 da Figura 4.3.



G_1 G_2 G_3

Figura 4.3: Remoção sucessiva de vértices universais

A seqüência reduzida de G_1 é $\mathcal{SR}(G_1) = [(1,5),(3,6),(5,7),(6,8)]$. Nota-se que os vértices 2 e 4 de G_1 não aparecem na seqüência reduzida. Estes vértices pertencem a todas as $(k+1)$-cliques do grafo, tendo grau $n-1$; são, por isso, denominados *universais*. Removendo um deles, encontramos um grafo $(k-1)$-caminho, que possui a mesma seqüência reduzida do grafo original. A Figura 4.3 ilustra esse processo.

Pelo fato de alguns vértices não figurarem na seqüência reduzida, para que ela possa ser utilizada como um código para um grafo k-caminho, é necessário mencionar, além dos pares de vértices que a compõem, o número de vértices n, possibilitando a recontrução do grafo sem ambigüidades. Denominamos *código compacto* de G o par $\mathcal{CC}(G) = (n, \mathcal{SR}(G))$.

Algumas considerações merecem destaque:

- $\mathcal{CC}(G)$ é um código para o grafo k-caminho G, isto é, uma representação única de G.

- O tamanho do código $\mathcal{CC}(G)$ depende apenas do número de vértices do grafo. Isto significa que, quanto maior for o valor de k, mais compacto é o código.

Para os grafos da Figura 4.3, temos então:

$$\mathcal{CC}(G_1) = (8, [(1,5),(3,6),(5,7),(6,8)])$$

$$\mathcal{CC}(G_2) = (7, [(1,5),(3,6),(5,7),(6,8)])$$

$$\mathcal{CC}(G_3) = (6, [(1,5),(3,6),(5,7),(6,8)])$$

4.3 Algoritmos de Codificação e Decodificação

Como vimos na seção anterior, o código compacto de um grafo k-caminho $G = (V, E)$ consta de um par $\mathcal{CC}(G) = (n, \mathcal{SR}(G))$. Nesta seção, serão apresentados algoritmos eficientes para a obtenção do código a partir do grafo dado e para a recuperação do grafo a partir do código.

Consideremos, de início, a codificação de um grafo k-caminho. Se $n = k + 1$, G é um grafo completo e $\mathcal{SR}(G) = [\,]$. Do contrário, é preciso determinar a seqüência reduzida de G. Aplicando a própria definição de $\mathcal{SR}(G)$, encontramos:

$$\{\ell_i\} = C_i - C_{i+1} \quad \text{e} \quad \{r_i\} = C_{i+1} - C_i,\ 1 \leq i < n - k.$$

O Lema 4.4 permite-nos determinar C_i, $1 \leq i \leq n-k$, a partir da representação por conjuntos de adjacência restritos. O algoritmo proposto utiliza um vetor auxiliar *conjunto*, de dimensão n, que armazena as diferenças entre conjuntos. Inicialmente o algoritmo determina as cliques C_1, ..., C_{n-k} e o vetor *conjunto* é inicializado com zeros, exceto para os elementos do conjunto C_1, inicializados com o valor 1. São executados $n - k - 1$ passos no algoritmo. No i-ésimo passo, o conjunto C_{i+1} é comparado ao conjunto C_i: o vértice que pertence somente a C_{i+1} é atribuído à variável r_i; o que pertence somente a C_i é atribuído à variável ℓ_i. O par (ℓ_i, r_i) fica, assim, bem determinado.

O Algoritmo 4.1 percorre as cliques maximais de G; sua complexidade é então $O(m) = O(k\,n)$.

Uma propriedade da seqüência reduzida diz respeito aos vértices simpliciais do grafo G codificado. Quando $n > k + 1$, G tem dois vértices simpliciais, e, pela definição de seqüência fundamental, eles pertencem às cliques C_1 e C_{n-k}. Como vértices simpliciais não podem pertencer a mais de uma clique maximal, conclui-se que estes correspondem, na seqüência reduzida, o menor, a ℓ_1 e o maior, a r_{n-k-1}, respectivamente.

O algoritmo de decodificação deve encontrar, a partir do código compacto $\mathcal{CC}(G)$, a representação de G por conjuntos de adjacência restritos. Primeiramente, a relação entre as representações por conjuntos de adjacência restritos e a seqüência reduzida precisa ser estabelecida.

Algoritmo 4.1. *Determinação do código compacto de um grafo k-caminho*

Entrada: Um grafo k-caminho $G = (V, E)$, representado por
$\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$;
Saída: $\mathcal{CC}(G) = (n, \mathcal{SR}(G))$, código compacto de G;
Início
  Para $i \leftarrow 1, \ldots, n-k$ **faça**
    $C_i \leftarrow \{\sigma(i)\} \cup X_\sigma(\sigma(i))$;
  Para $i \leftarrow 1, \ldots, n$ **faça**
    $conjunto[i] \leftarrow 0$;
  $\mathcal{SR}(G) \leftarrow [\ ]$;
  Para $v \in C_1$ **faça**
    $conjunto[v] \leftarrow 1$;
  Para $i \leftarrow 1, \ldots, n-k-1$ **faça**
    Para $v \in C_{i+1}$ **faça**
      Se $conjunto[v] \neq i$ **então**
        $r_i \leftarrow v$;
      $conjunto[v] \leftarrow i+1$;
    Para $v \in C_i$ **faça**
      Se $conjunto[v] \neq i+1$ **então**
        $\ell_i \leftarrow v$;
    $\mathcal{SR}(G) \leftarrow \mathcal{SR}(G)$ || $[(\ell_i, r_i)]$;
  $\mathcal{CC}(G) \leftarrow (n, \mathcal{SR}(G))$;
Fim.

Lema 4.6. *Seja G uma grafo k-caminho, $\mathcal{R}_\sigma(G)$ sua representação por conjuntos de adjacência restritos e $\mathcal{SR}(G)$ sua seqüência reduzida. Então $\sigma(i) = \ell_i$, para $i = 1, \ldots, n-k-1$.*

Demonstração: Pelo Lema 4.4 sabe-se que $C_{n-k} = \{\sigma(n-k)\} \cup X_\sigma(\sigma(n-k))$. Pelo processo indutivo de construção, o vértice $\sigma(n-k-1)$ é ligado a uma k-clique, subclique de C_{n-k}. Pela definição de seqüência reduzida, $\{\ell_{n-k-1}\} = C_{n-k-1} - C_{n-k}$. Então, $\ell_{n-k-1} = \sigma(n-k-1)$. Como a seqüência fundamental $\mathcal{SF}(G)$ é única, conclui-se a tese. □

Corolário 4.3. $C_{n-k} = V - \{\ell_1, \ell_2, \ldots, \ell_{n-k-1}\}$.

Juntamente com a definição da seqüência reduzida, este corolário permite-nos determinar as cliques maximais de um grafo k-caminho. Conhecendo-se a $(k+1)$-clique C_{n-k}, as demais podem ser assim expressas:

$$C_i = C_{i+1} - \{r_i\} \cup \{\ell_i\} \text{ para } i = 1, \ldots, n-k-1.$$

O algoritmo de decodificação consta de duas etapas distintas. Na primeira, a $(k+1)$-clique C_{n-k} é encontrada e as entradas correspondentes a estes vértices na representação são processadas. Na segunda etapa, a cada passo, a $(k+1)$-clique maximal corrente C_i é determinada. Como, em um grafo k-caminho, vale a igualdade $C_i = \{\sigma(i)\} \cup X_\sigma(\sigma(i))$, podemos estabelecer a entrada correspondente da representação. O vetor *clique*, de dimensão $k+1$, armazena, a cada passo, os elementos constituintes da clique C_i.

Algoritmo 4.2. *Decodificação de um grafo k-caminho*

Entrada: Um grafo k-caminho $G = (V, E)$, representado por
$\mathcal{CC}(G) = (n, \mathcal{SR}(G))$, código compacto de G;
Saída: $\mathcal{R}_\sigma(G) = [(\sigma(i), X_\sigma(\sigma(i)) \mid i = 1, \ldots, n]$;
Início
  $\mathcal{R}_\sigma(G) = [(r_{n-k-1}, \emptyset)]$;
  $V' \leftarrow V - \{\ell_1, \ell_2, \ldots, \ell_{n-k-1}\}$;
  $clique[1] \leftarrow r_{n-k-1}$; $i \leftarrow 1$;
  Para $v \in V' - \{r_{n-k-1}\}$ **faça**
    $X_\sigma(v) \leftarrow \{w \mid w = clique[j], j = 1, \ldots, i\}$;
    $\mathcal{R}_\sigma(G) \leftarrow [(v, X_\sigma(v))] \,||\, \mathcal{R}_\sigma(G)$;
    $i \leftarrow i + 1$; $clique[i] \leftarrow v$;
  Para $i \leftarrow n-k-1, \ldots, 1$ **faça**
    $\sigma(i) \leftarrow \ell_i$; $X_\sigma(\sigma(i)) \leftarrow \emptyset$;
    Para $j \leftarrow 1, \ldots, k+1$ **faça**
      Se $clique[j] = r_i$ **então**
        $clique[j] \leftarrow \ell_i$;
      caso contrário
        $X_\sigma(\sigma(i)) \leftarrow X_\sigma(\sigma(i)) \cup \{clique[j]\}$;
  $\mathcal{R}_\sigma(G) \leftarrow [(\sigma(i), X_\sigma(\sigma(i)))] \,||\, \mathcal{R}_\sigma(G)$;
Fim.

A complexidade deste algoritmo é também $O(k\,n)$.

4.4 Caminhos Hamiltonianos

Como foi visto na Seção 1.1, um caminho hamiltoniano é um caminho simples que contém todos os vértices do grafo. Encontrar estes caminhos é um importante problema algorítmico; nesta seção será mostrado como o código compacto pode fornecer uma solução quase imediata deste problema para os grafos k-caminho.

Teorema 4.2. *Seja $G = (V, E)$ um grafo k-caminho com n vértices e $\mathcal{SR}(G) = [(\ell_1, r_1), \ldots, (\ell_{n-k-1}, r_{n-k-1})]$ sua seqüência reduzida. Então G possui $k!$ caminhos hamiltonianos tendo a seqüência $[\ell_1, \ell_2, \ldots \ell_{n-k-1}]$ como prefixo e r_{n-k-1} como último vértice.*

Demonstração: Mostraremos, primeiramente, que a seqüência $[\ell_1, \ldots \ell_{n-k-1}]$ é um caminho simples em G. Pelo Lema 4.6, $\ell_i = \sigma(i)$, para $i = 1, 2, \ldots, n-k-1$. Então, $\ell_1 \neq \ell_2 \neq \ldots \neq \ell_{n-k-1}$.

Resta provar que $\{\ell_i, \ell_{i+1}\} \in E$. Seja $n > k+2$; para $n \leq k+2$, a afirmativa é óbvia. Pela definição de seqüência reduzida, sabe-se que:

$$\begin{array}{lll} \ell_i \in C_i; & \ell_i \notin C_{i+1}; & \ell_{i+1} \in C_{i+1}; \\ r_i \notin C_i; & r_i \in C_{i+1}; & \ell_{i+1} \notin C_{i+2}. \end{array}$$

Sejam então, sem perda de generalidade, $C_i = \{\ell_i, v_1, \ldots, v_k\}$ e $C_{i+1} = \{r_i, v_1, \ldots, v_k\}$. Logo, ℓ_i só não é adjacente ao vértice r_i em C_{i+1}. Suponhamos, por absurdo, que $r_i = \ell_{i+1}$. Então, como $\ell_{i+1} \notin C_{i+2}$, $C_i \cap C_{i+1} = C_{i+1} \cap C_{i+2}$, o que contraria a definição de grafo k-caminho. Logo $r_i \neq \ell_{i+1}$ e existe a aresta $\{\ell_i, \ell_{i+1}\}$.

Pelo Corolário 4.3, $C_{n-k} = V - \{\ell_1, \ldots, \ell_{n-k-1}\}$. O vértice r_{n-k-1} pertence a esta clique; ele é o último vértice dos caminhos hamiltonianos. Os outros k vértices podem ser considerados em $k!$ seqüências. O vértice ℓ_{n-k-1} está ligado a todos esses vértices. Então os $k!$ caminhos hamiltonianos são a simples concatenação de dois caminhos simples: um prefixo $\{\ell_1, \ldots, \ell_{n-k-1}\}$ e um sufixo contendo os vértices de C_{n-k} terminando em r_{n-k-1}. □

Observe o grafo G_3 da Figura 4.3. Seu código compacto é

$$\mathcal{CC}(G_1) = (8, [(1,5), (3,6), (5,7), (6,8)]).$$

A seqüência $[1, 3, 5, 6]$ constitui um caminho simples. A 4-clique C_5 pode ser determinada pelo corolário 4.3:

$$C_5 = V - \{1, 3, 5, 6\} = \{2, 4, 7, 8\}.$$

O maior vértice simplicial é 8 e, portanto, o último vértice dos caminhos hamiltonianos a serem obtidos. Aplicando o Teorema 4.2 encontramos os seguintes caminhos hamiltonianos em G_3:

$$[1, 3, 5, 6, 2, 4, 7, 8]$$
$$[1, 3, 5, 6, 2, 7, 4, 8]$$
$$[1, 3, 5, 6, 4, 2, 7, 8]$$
$$[1, 3, 5, 6, 4, 7, 2, 8]$$
$$[1, 3, 5, 6, 7, 2, 4, 8]$$
$$[1, 3, 5, 6, 7, 4, 2, 8].$$

Assim, se o grafo k-caminho está representado por seu código compacto, cada um dos caminhos acima pode ser determinado em complexidade de $O(n)$.

4.5 Exercícios

1. Utilizando a definição indutiva, construa um grafo 3-caminho G com 8 vértices.
   (a) Obtenha uma representação geométrica para G na qual não haja cruzamento de arestas.
   (b) Assinale os vértices simpliciais s_1 e s_2 de G.
   (c) Destaque três 3-caminhos maximais em G.
   (d) Apresente a seqüência fundamental de G.
   (e) Apresente a seqüência reduzida de G.
2. Sejam G um grafo 4-caminho e $\mathcal{CC}(G) = (10, [(2,8),(5,7),(1,6),(7,9),(4,10)])$ seu código compacto.
   (a) Observando apenas $\mathcal{CC}(G)$, diga quais são os vértices simpliciais de G.
   (b) Observando apenas $\mathcal{CC}(G)$, diga quais são os vértices universais de G.
   (c) Determine as cliques maximais de G.
3. Defina indutivamente a subfamília dos grafos k-caminho que possuem garantidamente pelo menos um vértice universal.
4. Sejam $G = (V, E)$ um grafo k-caminho, $v \in V$ e $q(v)$ o número de cliques maximais às quais v pertence. Prove que $d(v) = q(v) + k - 1$.

4.6 Notas Bibliográficas

Beineke e Pippert [2] introduziram o conceito de k-caminhos, generalizando caminhos simples. Em [16] e [17], Markenzon *et al.* definiram e caracterizaram grafos k-caminho. O código compacto foi apresentado em Pereira *et al.*[25]; novas propriedades da família e aplicações do código na resolução de problemas são mostradas em [26].

Bibliografia

[1] A.V. Aho, J.E. Hopcroft e J.D. Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley, Reading, Mass., 1974.

[2] L.W. Beineke e R.E. Pippert, Properties and characterizations of k-trees, *Mathematika*, **18** (1971), 141-151.

[3] T. Beyer, W. Jones e S. Mitchell, Linear algorithms for isomorphism of maximal outerplanar graphs, *J.of ACM*, **26** (1979), 603-610.

[4] J.R.S. Blair e B. Peyton, An Introduction to Chordal Graphs and Clique Trees, in "Graph Theory and Sparse Matrix Computation", IMA Vol. 56, pp. 1-29, 1993.

[5] G. Brassard e P. Bratley, "Fundamentals of Algorithms", Prentice Hall Inc., Englewood Cliffs, New Jersey, 1996.

[6] S. Caminiti, I. Finocchi e R. Petreschi, A unified approach to coding labelled trees, Proceedings of the 6^{th} LATIN'04, pp. 339-348, 2004.

[7] H.-C. Chen e Y.-L. Wang, An efficient algorithm for generating Prüfer codes from labelled trees, *Theory of Computing Systems*, **33** (2000), 97-105.

[8] N. Deo e P. Micikevicius, Prüfer-like codes for labeled trees, *Congressus Numerantium*, **151** (2001), 65-73.

[9] N. Deo e P. Micikevicius, A new encoding for labeled trees employing a stack and a queue, *Bulletin of the Institute of Combinatorics and its Applications*, **34** (2002), pp. 77-85.

[10] G. Di Battista, P. Eades, R. Tamassia e I. Tollis, "Graph Drawing: Algorithms for the Visualization of Graphs", Prentice-Hall, Upper Saddle River, 1999.

[11] R. Diestel, "Graph Theory", 2nd ed., Graduate Texts in Mathematics 173, Springer-Verlag New York, Inc., 2000.

[12] F. Gavril, Algorithms for minimum coloring, minimum clique, minimum covering by cliques, and maximum independent set of a chordal graph, *SIAM J. of Comput.*, **1** (1972), 180-187.

[13] M.C. Golumbic, "Algorithmic Graph Theory and Perfect Graphs", Academic Press, New York, 1980.

[14] J. Gross e J. Yellen, "Graph Theory and its Applications", the CRC Press series on Discrete Mathematics and its Applications, 1998.

[15] C. M. Justel e L. Markenzon, Lexicographic breadth first search and k-trees, Proceedings of JIM'2000 - Secondes Journées de l'Informatique Messine, pp. 23-28, 2000.

[16] L. Markenzon, C.M. Justel e N. Paciornik, K-paths and k-path graphs, Proceedings of Two Days on Combinatorial Optimization: a Bridge Between Rio and Niterói, pp.15, 2003.

[17] L. Markenzon, C.M. Justel e N. Paciornik, Subclasses of k-trees: characterization and recognition, *Discrete Applied Mathematics*, **154** (2006), 818-825.

[18] L. Markenzon e N. Paciornik, Equilateral drawing of 2-connected planar chordal graphs, Proceedings of the 6^{th} Twente Workshop on Graphs and Combinatorial Optimization, pp. 149-153, 1999.

[19] L. Markenzon, O. Vernet e P.R.C. Pereira, (L,U)-Bounded priority queues and the codification of Rényi k-trees, Rel Téc. NCE-UFRJ 05/05, 2005.

[20] L. Markenzon, O. Vernet, P.R.C. Pereira, "Codificação de Árvores: Conceitos e Algoritmos", minicurso do ERMAC 2005, Vitória, ES, 2005.

[21] J. A. McHugh, "Algorithmic Graph Theory", Prentice Hall Inc., Englewood Cliffs, New Jersey, 1990.

[22] S. Mitchell, Linear algorithms to recognize outerplanar and maximal outerplanar graphs, *Information Processing Letters*, **9** (1979), 229-232.

[23] J.W. Moon, "Counting Labelled Trees", Canadian Mathematical Monographs, Montreal, 1970.

[24] B.S. Panda, New linear time algorithms for generating perfect elimination orderings of chordal graphs, *Information Processing Letters*, **38** (1996), 111-115.

[25] P.R.C. Pereira, L. Markenzon e O. Vernet, A compact representation for labelled k-path graphs, Proceedings of the Workshop on Graphs and Combinatorial Optimization, CTW 2005, pp.68-73, 2005.

[26] P.R.C. Pereira, L. Markenzon e O. Vernet, A clique-difference encoding scheme for labelled k-path graphs, artigo submetido, 2005.

[27] A. Prüfer, Neuer beweis eines satzes über permutationen, *Archiv der Mathematik und Physik*, **27** (1918), 142-144.

[28] D.J. Rose, R.E. Tarjan e G. Lueker, Algorithmic aspects of vertex elimination on graphs, *SIAM J.Comput.* **5** (1976) 266-283.

[29] M.M. Syslo, Outerplanar graphs: characterizations, testing, coding and counting, *Bull. Acad. Pol. Sci. Ser. Math.*, **26** (1978), 675-684.

[30] J.L. Szwarcfiter, "Grafos e Algoritmos Computacionais", Editora Campus, 1984.

[31] J.L. Szwarcfiter e L. Markenzon, "Estruturas de Dados e seus Algoritmos", 2a. ed., LTC - Livros Técnicos e Científicos, Rio de Janeiro, 1994.

[32] R.E. Tarjan, "Data Structures and Network Algorithms", SIAM, Philadelphia, 1983.

[33] R.E. Tarjan e M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, *Siam J. Comput.*, **13** (1984), 566-579.

Índice

Notas em Matemática Aplicada

1. Restauração de Imagens com Aplicações em Biologia e Engenharia
   Geraldo Cidade, Antônio Silva Neto e Nilson Costa Roberty
2. Fundamentos, Potencialidades e Aplicações de Algoritmos Evolutivos
   Leandro dos Santos Coelho
3. Modelos Matemáticos e Métodos Numéricos em Águas Subterrâneas
   Edson Wendlander
4. Métodos Numéricos para Equações Diferenciais Parciais
   Maria Cristina de Castro Cunha e Maria Amélia Novais Schleicher
5. Modelagem em Biomatematica
   Joyce da Silva Bevilacqua, Marat Rafikov e Cláudia de Lello Courtouke Guedes
6. Métodos de Otimização Randômica: algoritmos genéticos e "simulated annealing"
   Sezimária F. Pereira Saramago
7. "Matemática Aplicada à Fisiologia e Epidemiologia"
   H.M. Yang, R. Sampaio e A. Sri Ranga
8. Uma Introdução à Computação Quântica
   Renato Portugal, Carlile Campos Lavor, Luiz Mariano Carvalho e Nelson Maculan
9. Aplicações de Análise Fatorial de Correspondências para Análise de Dados
   Dr. Homero Chaib Filho, Embrapa
10. Modelos Matemáticos baseados em autômatos celulares para Geoprocessamento
    Marilton Sanchotene de Aguiar, Fábia Amorim da Costa, Graçaliz Pereira Dimuro e Antônio Carlos da Rocha Costa

11. Computabilidade: os limites da Computação
    Regivan H. N. Santiago e Benjamín R. C. Bedregal
12. Modelagem Multiescala em Materiais e Estruturas
    Fernando Rochinha e Alexandre Madureira
13. Modelagem em Biomatemática (Coraci Malta ed.)
    1 - "Modelagem matemática do comportamento elétrico de neurônios e algumas aplicações"
        Reynaldo D. Pinto
    2 - "Redes complexas e aplicações nas Ciências"
        José Carlos M. Mombach
    3 - "Possíveis níveis de complexidade na modelagem de sistemas biológicos"
        Henrique L. Lenzi, Waldemiro de Souza Romanha e Marcelo Pelajo-Machado
14. A lógica na construção dos argumentos
    Angela Cruz e José Eduardo de Almeida Moura
15. Modelagem Matemática e Simulação Numérica em Dinâmica dos Fluidos
    Valdemir G. Ferreira, Hélio A. Navarro, Magda K. Kaibara
16. Introdução ao Tratamento da Informação nos Ensinos Fundamental e Médio
    Marcilia Andrade Campos, Paulo Figueiredo Lima
17. Teoria dos Conjuntos Fuzzy com Aplicações
    Rosana Sueli da Motta Jafelice, Laércio Carvalho de Barros, Rodney Carlos Bassanezi
18. Introdução à Construção de Modelos de Otimização Linear e Inteira
    Socorro Rangel
19. Observar e Pensar, antes de Modelar
    Flavio Shigeo Yamamoto, Sérgio Alves, Edson P. Marques Filho, Amauri P. de Oliveira
20. Frações Contínuas: Propriedades e Aplicações
    Eliana Xavier Linhares de Andrade, Cleonice Fátima Bracciali