

CÓDIGOS PARA SUBFAMÍLIAS DOS GRAFOS CORDAIS

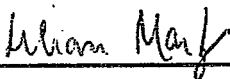
Paulo Renato da Costa Pereira

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

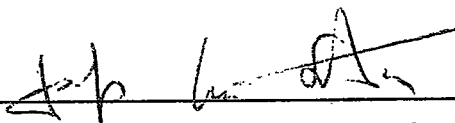
Aprovada por:



Prof. Paulo Roberto Oliveira, Dr.Ing.



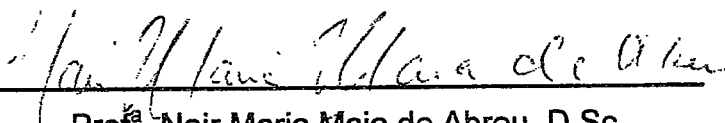
Prof.ª. Lilian Markenzon, D.Sc.



Prof. Jayme Luiz Szwarcfiter, Ph.D.



Prof. Luiz Satoru Ochi, D.Sc.



Prof.ª. Nair Maria Maia de Abreu, D.Sc.



Prof. Fábio Protti, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 2007

PEREIRA, PAULO RENATO DA COSTA

Códigos para Subfamílias dos Grafos Cordais [Rio de Janeiro] 2007

XI, 127 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2007)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Grafos cordais
2. Codificação de grafos
3. k -árvores

I. COPPE/UFRJ II. Título (série)

Agradecimentos

À Deus por possibilitar a realização deste trabalho.

Ao professor Paulo Roberto Oliveira que me recebeu na COPPE Sistemas e sempre confiou no trabalho desenvolvido nesta tese.

À professora Lilian Markenzon que orientou, apoiou, debateu, corrigiu, reclamou, participou; enfim, foi amiga, companheira de trabalho e orientadora.

Ao professor Oswaldo Vernet que colaborou com muitas idéias e debates que ajudaram a delinear nossos estudos nesta tese.

À professora Claudia Justel, colega do IME, pela ajuda e pelos conselhos fornecidos ao longo deste trabalho.

Ao TC Cecilio, chefe do Departamento de Engenharia de Sistemas do IME, que ao longo destes dois últimos anos me ajudou reduzindo minha carga de trabalho viabilizando a conclusão desta tese.

Aos colegas da Seção de Engenharia de Sistemas do IME, que dentro do possível, me ajudaram nesta tarefa.

Aos amigos Gilberto Ferreira da Silva e Maximiliano Pinto Damas, pelos momentos de estudo juntos durante a fase de créditos.

Paulo Renato

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

CÓDIGOS PARA SUBFAMÍLIAS DOS GRAFOS CORDAIS

Paulo Renato da Costa Pereira

Dezembro/2007

Orientadores: Paulo Roberto Oliveira
Lilian Markenzon

Programa: Engenharia de Sistemas e Computação

Este trabalho trata da codificação e do estudo de propriedades estruturais de subfamílias dos grafos cordais. São propostos códigos específicos para os grafos k -caminho e para as k -árvores. Estes códigos são mais compactos do que a representação tradicional por arestas e utilizam algoritmos lineares para codificação e decodificação.

A codificação proposta para os grafos k -caminho viabiliza a obtenção de diversas propriedades estruturais da família. Estas propriedades são fundamentais para a contagem e a geração uniforme de exemplares, bem como para a obtenção de soluções eficientes de problemas tais como o isomorfismo e a determinação do caminho hamiltoniano, problemas estes resolvidos com complexidade $O(n)$.

Utilizando a codificação proposta para as k -árvores, são obtidos algoritmos, também com complexidade $O(n)$, para determinar a coloração ótima de vértices e a seqüência de multiplicidades, conceito que generaliza a seqüência de graus do grafo.

Finalizando, uma nova subfamília dos grafos de intervalo é definida e são delimitadas precisamente as interseções entre diversas subfamílias das k -árvores e dos grafos de intervalo.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

CODES FOR SUBFAMILIES OF CHORDAL GRAPHS

Paulo Renato da Costa Pereira

December/2007

Advisors: Paulo Roberto Oliveira
Lilian Markenzon

Department: System Engineering and Computer Science

This work presents codes and structural properties of subfamilies of chordal graphs. Particular codes for k -path graphs and k -trees are defined. Both codes are more compact than the traditional representation based on edges; linear time algorithms for encoding and decoding are presented.

The code for k -path graphs emphasizes some structural properties of the family. These properties are used for counting and generating instances uniformly. The isomorphism problem and the determination of a hamiltonian path are solved by algorithms with $O(n)$ complexity.

For k -trees, the proposed code allows the development of algorithms, also with $O(n)$ complexity, to determine an optimal coloring of the vertices and the sequence of multiplicities, a new concept that generalizes the sequence of degrees.

Finally, a new subfamily of the interval graphs is defined and the exact intersection among some subfamilies of k -trees and interval graphs are established.

Sumário

Capítulo 1 - Introdução	1
1.1 - Motivação.....	1
1.2 - Estrutura do Trabalho.....	3
1.3 - Conceitos Básicos	4
Capítulo 2 - Codificação de Árvores	10
2.1 - Código de Prüfer	10
2.2 - Contagem e Geração Utilizando o Código de Prüfer.....	16
2.2.1 Contagem e Geração de Árvores Rotuladas sem Restrições.....	17
2.2.2 Geração de Árvores Rotuladas com um Número Mínimo de Folhas Dado	18
2.2.3 Contagem e Geração de Árvores Rotuladas com Seqüência de Graus Dada.....	18
2.2.4 Contagem e Geração de Árvores Rotuladas onde um Vértice Específico tem Grau Dado	20
2.2.5 Contagem e Geração de Árvores com Número de Folhas Dado.....	21
2.3 - Codificações Baseadas no Código de Prüfer.....	24
2.4 - Complexidade dos Algoritmos.....	27
2.5 - Conclusão.....	29
Capítulo 3 - Codificação de Grafos k -caminho	31
3.1 - Noções Básicas.....	31
3.2 - Seqüência de $(k+1)$ -cliques	35
3.3 - Seqüência Reduzida	36
3.4 - Complexidade de Espaço da Codificação	43
3.5 - Validação de uma Seqüência Reduzida	43
3.6 - Codificação da Seqüência Reduzida	47
3.7 - Decodificação da Seqüência Reduzida	48
3.8 - Contagem e Geração de Grafos k -caminho Rotulados.....	50
Capítulo 4 - Propriedades dos Grafos k -caminho	54
4.1 - Propriedades da Seqüência Reduzida.....	54
4.2 - Esquemas de Eliminação Perfeita	60
4.3 - Isomorfismo	61
4.4 - Obtenção de Caminho Hamiltoniano	64

Capítulo 5 - Código de Prüfer para k -árvores Rotuladas	66
5.1 - Definição	66
5.2 - Validação.....	70
5.3 - Propriedades	71
5.4 - Algoritmos Lineares de Codificação e Decodificação	73
Capítulo 6 - Código Compacto para k -árvores Rotuladas	79
6.1 - Definição	80
6.2 - Validação.....	86
6.3 - Obtenção de Esquema de Eliminação Perfeita.....	88
6.4 - Algoritmos de Codificação e Decodificação.....	90
6.5 - Conclusão	94
Capítulo 7 - Aplicações do Código Compacto	95
7.1 - Determinação dos Vértices Simpliciais.....	95
7.2 - Coloração de Vértices	97
7.3 - Seqüência de Multiplicidades em uma k -árvore	100
Capítulo 8 - Relacionamentos entre as Famílias Estudadas	104
8.1 - Famílias Estudadas	104
8.2 - Grafos de Intervalo k -estritos	109
8.3 - Resultado sobre k -intercats.....	113
8.4 - Resultados sobre Grafos de Intervalo e k -árvores.....	114
8.5 - Interseções entre as Famílias.....	118
Capítulo 9 - Conclusão	120
9.1 - Contribuições	120
9.2 - Trabalhos Futuros.....	122
Capítulo 10 - Referências	123

Lista de Figuras

1.1: Exemplo de construção de 2-árvore	6
2.1: Obtenção do código de Prüfer para uma árvore	12
2.2: Codificação de árvore com raiz no vértice de rótulo 5 usando Prüfer	14
2.3: Construção de árvore, dado seu código de Prüfer	16
2.4: Geração aleatória de árvore com 7 vértices e pelo menos 4 folhas.....	18
2.5: Geração aleatória de árvore com seqüência de graus $\langle 1, 1, 1, 2, 4, 1 \rangle$	20
2.6: Todas as árvores com seqüência de graus $\langle 1, 1, 1, 2, 4, 1 \rangle$	20
2.7: Geração aleatória de árvore onde o vértice de rótulo 5 tem grau 3	21
2.8: Permutação de quatro elementos em conjuntos disjuntos não vazios	22
2.9: Triângulo de números de Stirling do segundo tipo	22
2.10: Geração aleatória de árvore com 5 vértices e exatamente 3 folhas.....	23
2.11: Codificação de árvore usando 2º código de Neville.....	25
2.12: Codificação de árvore usando 3º código de Neville.....	25
2.13: Codificação de árvore usando código de Deo e Micikevicius	26
2.14: Árvore da Figura 2.1(a) com pares (x_v, y_v) para as 4 codificações estudadas	29
3.1: 2-árvore e um 2-caminho maximal	32
3.2: Exemplo de construção de 2-árvore que é grafo 2-caminho	33
3.3: 2-caminho com todas as 3-cliques do grafo 2-caminho da Figura 3.2(e)	33
3.4: Representação esquemática de grafo 4-caminho e sua seqüência reduzida.....	37
3.5: Conjuntos L_i e R_i para o grafo 4-caminho da Figura 3.4	41
3.6: Validação da seqüência reduzida do grafo 4-caminho da Figura 3.4.....	45
3.7: Decodificação da seqüência reduzida do grafo 4-caminho da Figura 3.4.....	49
4.1: Representação de grafo 4-caminho	55
4.2: Dois grafos 2-caminho isomorfos	61
5.1: Árvore e seu código redundante de Prüfer	67
5.2: Passo a passo da obtenção do código de Prüfer de uma 2-árvore	69
5.3: Aplicação do Lema 5.1 à três seqüências de conjuntos de tamanho 3	71
5.4: Representação esquemática da codificação e decodificação	75

6.1: 3-árvore com 10 vértices	81
6.2: Obtenção, passo a passo, do código compacto da 3-árvore da Figura 6.1	82
6.3: Obtenção da 3-árvore da Figura 6.1 a partir do seu código compacto	85
6.4: Representação esquemática da codificação e decodificação	90
8.1: 2-caterpillars	108
8.2: 2-árvore estrela	109
8.3: Dois grafos de intervalo e suas representações por intervalo	110
8.4: Representação por intervalos de um grafo 4-caminho	112
8.5: Diagramas com as subclasses das árvores e a dos grafos de intervalo	112
8.6: 2-intercats	114
8.7: 2-caterpillar com asteróide triplo (x,y,z)	115
8.8: Diagramas com subclasses das k -árvores e dos grafos de intervalo	119

Lista de Algoritmos

2.1: Geração do código de Prüfer para árvores	13
2.2: Geração da árvore a partir do código de Prüfer.....	15
2.3: Codificação genérica de árvores.....	29
3.1: Validação de seqüência reduzida.....	46
3.2: Codificação de grafo k -caminho.....	48
3.3: Decodificação da seqüência reduzida de um grafo k -caminho	50
3.4: Geração aleatória de seqüência reduzida de grafo k -caminho.....	53
4.1: Cálculo dos graus dos vértices de um grafo k -caminho	59
4.2: Verificação de isomorfismo de grafos k -caminho.....	63
4.3: Obtenção de caminho hamiltoniano em um grafo k -caminho.....	65
5.1: Codificação de Prüfer para árvores	67
5.2: Codificação de Prüfer para k -árvores	68
5.3: Codificação linear de Prüfer para uma k -árvore.....	76
5.4: Obtenção da seqüência $\langle a_1, \dots, a_{n-k-1} \rangle$ de vértices removidos	77
5.5: Obtenção do vértice a_{n-k} e da clique residual B_{n-k}	77
5.6: Obtenção das arestas de uma k -árvore a partir do seu código redundante	78
6.1: Obtenção do esquema de eliminação perfeita associado.....	89
6.2: Obtenção do código compacto de uma k -árvore	92
6.3: Obtenção do código redundante a partir do código compacto	93
7.1: Determinação dos vértices simpliciais a partir do código compacto	97
7.2: Coloração dos vértices de uma k -árvore a partir de seu código compacto.....	100

Lista de Tabelas

2.1: Todos os códigos de Prüfer para a árvore da Figura 2.1(a).....	14
2.2: Pares (x_v, y_v) associados aos quatro códigos estudados.....	28
3.1: Totais de grafos k -caminho rotulados	52

Capítulo 1 - Introdução

1.1 - Motivação

Em 2002, JOHNSON [24] mostrou como a análise experimental de algoritmos pode ser utilizada de um modo científico para obtenção de resultados confiáveis. Este autor comenta que a geração aleatória de grafos é uma ferramenta de grande valia para esta metodologia, principalmente para a obtenção de instâncias muito grandes.

É possível listar ou catalogar os grafos com pequenas quantidades de vértices. Por exemplo, em [7], são catalogados todos os grafos com 10 vértices. Infelizmente, quando o tamanho dos grafos desejados aumenta, este procedimento torna-se inviável, pois a quantidade de grafos existentes se torna muito grande. Assim, ao invés de gerar todos os grafos, escolhem-se alguns por amostragem. Neste caso, o ideal é a geração aleatória uniforme de grafos.

A geração aleatória uniforme é bem simples em poucos casos, como por exemplo, na geração de grafos rotulados sem características especiais. Neste caso, para gerar um grafo rotulado de modo aleatório, basta que as arestas sejam escolhidas de modo independente com probabilidade de 50%. Entretanto, a geração torna-se muito mais complexa quando se desejam grafos sem rótulos ou grafos rotulados com alguma propriedade especial, como regularidade ou cordalidade, por exemplo. Em [1] e em [32], faz-se a geração de grafos cordais com número de vértices e de arestas desejado utilizando propriedades peculiares desta família. Entretanto, na maioria dos casos, a geração através da inserção de arestas é muito trabalhosa. Por outro lado, a geração utilizando uma codificação da família de grafos é bem mais simples.

De acordo com TINHOFER [46], apesar de terem seguido caminhos históricos distintos, os problemas de contagem e enumeração estão intimamente relacionados aos problemas de geração. Existe uma grande interseção entre essas áreas que utiliza a codificação de grafos. Isso ocorre porque é relativamente mais simples fazer a contagem e a geração aleatória utilizando o código, que é uma representação matemática, do que utilizando propriedades estruturais dos grafos.

Sejam uma família de grafos \mathfrak{S} , um conjunto Φ e uma função bijetora $f: \mathfrak{S} \rightarrow \Phi$. Sejam $G \in \mathfrak{S}$ e $\phi \in \Phi$ tal que $f(G) = \phi$. Diz-se que Φ é uma **codificação** para a família de grafos \mathfrak{S} e que ϕ é o **código** de G . Chama-se **codificação** a obtenção do código $f(G) = \phi$ a partir do grafo G e **decodificação** a obtenção do grafo $f^{-1}(\phi) = G$ a partir do código ϕ . Chama-se de **validação** o processo de verificar se um elemento ϕ pertence a Φ , ou seja, verificar se ϕ é o código de algum grafo da família \mathfrak{S} .

Apesar da simplicidade da definição, a obtenção da codificação Φ ou da função f é uma tarefa difícil para a maioria das famílias de grafos. Normalmente, as famílias que possuem propriedades estruturais bem determinadas possibilitam a definição de codificações. Por exemplo, no Capítulo 2, mostramos o código proposto por Prüfer para árvores em 1918.

O código é uma forma alternativa de representação dos grafos de uma família, em geral mais compacta. Assim, um grafo pode ser representado pelo seu código ao invés do seu conjunto de arestas. As principais vantagens de uma codificação em relação à representação tradicional por arestas de um grafo são:

- Geração mais eficiente;
- Contagem de elementos;
- Representação mais compacta;
- Obtenção de propriedades diretamente do código;

O nosso objetivo principal é o estudo e a definição de codificações mais compactas do que a representação por arestas para subfamílias dos grafos cordais bem como algoritmos eficientes de codificação e de decodificação. Para as subfamílias estudadas, desejamos: obter propriedades do grafo diretamente do seu código, gerar aleatoriamente grafos e fazer a contagem destas subfamílias.

1.2 - Estrutura do Trabalho

No Capítulo 2, revisamos o código de Prüfer [41] para árvores rotuladas, que é uma codificação para esta família, e mostramos alguns resultados sobre contagem de árvores com características específicas apresentados por MOON [35] em 1970. Provamos cada um destes resultados utilizando a codificação de Prüfer e propomos a geração destas árvores de modo aleatório. Além disso, apresentamos três codificações baseadas em Prüfer existentes na literatura.

No Capítulo 3, propomos uma codificação para os grafos k -caminho que foram caracterizados por MARKENZON *et al.* [31] e são uma subfamília dos grafos cordais. Utilizando esta codificação, contamos os grafos desta família e mostramos como fazer a geração destes grafos de modo aleatório. Além disso, propomos os algoritmos de validação, codificação e decodificação e apresentamos propriedades que podem ser obtidas do grafo diretamente do código que representa um grafo k -caminho. No Capítulo 4, mostramos mais algumas propriedades dos grafos k -caminho que são obtidas diretamente da codificação proposta. Estas propriedades possibilitaram a modelagem de algoritmos eficientes para teste de isomorfismo e obtenção de caminho hamiltoniano.

No Capítulo 5, apresentamos o código de Prüfer para k -árvores rotuladas. Em 1970, RÉNYI e RÉNYI [42] definiram esta codificação para k -árvores enraizadas. Reorganizamos esta codificação para k -árvores quaisquer e projetamos algoritmos eficientes para codificação e decodificação. No Capítulo 6, propomos uma nova codificação para as k -árvores e a chamamos de código compacto. Sua principal vantagem é a complexidade $O(n)$ para armazenamento. Também projetamos algoritmos lineares para codificação e decodificação do código compacto. No Capítulo 7, apresentamos aplicações para este novo código.

Finalmente, no Capítulo 8, propomos uma nova subfamília dos grafos de intervalo que contém os grafos k -caminho e delineamos com precisão a interseção entre várias subfamílias dos grafos cordais.

A próxima seção apresenta conceitos básicos que são utilizados neste documento.

1.3 - Conceitos Básicos

Nesta seção, são mostrados resultados e definições da teoria de grafos que são utilizados neste trabalho, que podem ser encontrados em [5, 19, 45]. Um **grafo** $G = (V, E)$ é um conjunto não vazio V de **vértices** e um conjunto de **arestas** E ; cada aresta é um subconjunto de V com 1 ou 2 vértices. Para um grafo $G = (V, E)$, denota-se $|V| = n$ e $|E| = m$. Dada uma aresta $\{v, w\} \in E$, diz-se que os vértices v e w são **extremidades** da aresta e que a aresta **incide** sobre os vértices v e w . Uma aresta da forma $\{v\}$ é denominada **laço**. Os grafos aqui estudados não possuem laços e são finitos, ou seja, possuem um número finito de vértices.

Dois vértices são **adjacentes** ou **vizinhos** quando são extremidades de uma aresta. O **conjunto de vizinhos** de um vértice $v \in V$ é o conjunto $Adj(v) = \{w \in V \mid \{v, w\} \in E\}$. A cardinalidade $|Adj(v)|$ é denominada **grau** de v . Denota-se $d(v)$ o grau do vértice v .

Seja $G = (V, E)$. $G - v$ é o grafo obtido pela remoção do vértice v e de todas as arestas incidentes a ele; $G - \{v, w\}$ é o grafo obtido pela remoção da aresta $\{v, w\}$.

Qualquer grafo $G' = (V', E')$ tal que $V' \subseteq V$ e $E' \subseteq E$ é um **subgrafo** de $G = (V, E)$. Um subgrafo é **induzido** por um subconjunto de vértices $V' \subseteq V$ quando contém todas as arestas de G com extremidades em V' .

Um **caminho** em um grafo $G = (V, E)$ é uma seqüência $\langle v_0, v_1, \dots, v_p \rangle$, $p \geq 0$, tal que $\{v_i, v_{i+1}\} \in E$ para $0 \leq i < p$. Diz-se que p é o **comprimento** deste caminho. Um caminho é dito **simples** se nenhum vértice se repete. Um **caminho hamiltoniano** é um caminho simples com todos os vértices de G . Um **ciclo** é um caminho onde o primeiro e o último vértices coincidem. Um ciclo é dito **simples** se todos os vértices são distintos exceto o primeiro e o último. Um grafo desprovido de ciclos é **acíclico** e, caso contrário, é **cíclico**.

Um grafo é **conexo** quando existe pelo menos um caminho entre cada par de vértices; do contrário, o grafo é dito **desconexo**.

Uma **árvore** é um grafo conexo e acíclico. Em uma árvore $T = (V, E)$, quando existe um vértice especial $r \in V$, denominado **raiz**, a árvore é dita **enraizada**. A árvore

trivial é aquela com apenas um vértice. Para as árvores não triviais diz-se que as **folhas** são os vértices de grau um, exceto a raiz.

A **distância** entre dois vértices u e v é o comprimento mínimo de um caminho que contenha estes dois vértices. A **excentricidade** de um vértice v é a maior distância de v a algum vértice u . O **diâmetro** de um grafo é a maior excentricidade existente e o **raio** de um grafo é a menor excentricidade existente. O **centro** de um grafo é o conjunto de vértices cuja excentricidade é igual ao raio, ou seja, são os vértices que têm excentricidade mínima.

Um grafo **completo** contém as $n(n-1)/2$ arestas possíveis. Uma **clique** do grafo $G = (V, E)$ é um subconjunto de vértices $Q \subseteq V$ tal que o subgrafo induzido por Q é completo. O **tamanho** de uma clique Q é a cardinalidade do conjunto Q . Chama-se de **k -clique** uma clique de tamanho k .

Um vértice $v \in V$ é dito **simplicial** se $Adj(v)$ é uma clique no grafo $G = (V, E)$. Seja $\sigma = \langle v_1, v_2, \dots, v_n \rangle$ uma ordenação de V . Diz-se que σ é um **esquema de eliminação perfeita** se, para $1 \leq i < n$, v_i é um vértice simplicial no subgrafo induzido pelo conjunto de vértices $\{v_i, v_{i+1}, \dots, v_n\}$.

Sejam v e w dois vértices distintos pertencentes a um ciclo C simples que não são adjacentes em C . A aresta $\{v, w\}$, se existir, é chamada **corda**. Um grafo $G = (V, E)$ é **cordal** quando todo ciclo de comprimento maior que 3 possui uma corda.

Teorema 1.1 [19]: Seja $G = (V, E)$ um grafo. As seguintes afirmativas são equivalentes:

- (i) G é cordal;
- (ii) G possui esquema de eliminação perfeita, e qualquer vértice simplicial pode iniciar o esquema.

Observe que um grafo cordal tem vários esquemas de eliminação perfeita e que um esquema pode estar associado a vários grafos. Logo, o esquema de eliminação perfeita não é uma codificação para grafos cordais.

O **grafo de interseção de cliques** de um grafo G é um grafo conexo com pesos nas arestas. Seus vértices são as cliques maximais de G e suas arestas são incidentes a

vértices que correspondem a cliques maximais que possuem interseção não vazia. O peso de cada aresta é igual à cardinalidade da interseção. Na literatura, o grafo de interseção de cliques também é conhecido como **grafo de cliques**. Entretanto, GALINIER *et al.* [15] chamaram de **grafo de cliques** o subgrafo do grafo de interseção de cliques que contém todas as árvores de cliques (definidas abaixo).

Por abuso de linguagem, diz-se que apenas as árvores geradoras de peso máximo do grafo de interseção de cliques de G são **árvores de cliques** de G . De acordo com BLAIR e PEYTON [5], esta definição é equivalente à mais conhecida na literatura onde uma **árvore de cliques** de G é uma árvore cujos vértices são as cliques maximais de G e obedece a **propriedade da interseção**: se Q_1 e Q_2 são cliques maximais de G , a interseção $Q_1 \cap Q_2$ é um subconjunto de todas as cliques maximais de G que estiverem no caminho entre Q_1 e Q_2 na árvore de cliques de G .

A seguir, são apresentadas definições relativas às k -árvores, que são uma subfamília dos grafos cordais. RÉNYI e RÉNYI [42] citam que esta subfamília foi introduzida em 1968 por Harary e Palmer.

Definição 1.2: Uma k -árvore, sendo $k > 0$, é assim definida:

- (1) Um grafo completo com k vértices é uma k -árvore;
- (2) Se $G = (V, E)$ é uma k -árvore, $Q \subseteq V$ é uma k -clique e $v \notin V$, o grafo $G' = (V \cup \{v\}, E \cup \{\{v, w\} \mid w \in Q\})$ é uma k -árvore;
- (3) Nada mais são k -árvores.

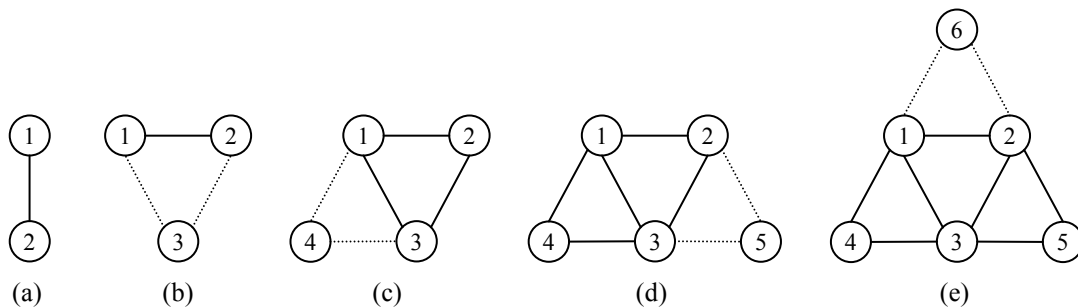


Figura 1.1: Exemplo de construção de 2-árvore.

Na Figura 1.1, constrói-se uma 2-árvore a partir de um grafo completo com dois vértices, que é uma 2-árvore de acordo com o item 1 da definição de k -árvores. Os passos da construção são baseados no item 2 da definição, ou seja, adiciona-se um

vértice adjacente aos vértices de uma 2-clique do grafo. Desse modo, a cada passo, são inseridas duas arestas, que estão indicadas em pontilhado na figura apenas com o intuito de ressaltar o processo de construção do grafo. Os rótulos dos vértices mostram a seqüência de inserções.

RÉNYI e RÉNYI [42] enumeram a quantidade de arestas e a quantidade de cliques de uma k -árvore. Pelo item 2 da Definição 1.2 de k -árvores, a inserção de um novo vértice acarreta a adição de k arestas ao grafo. Assim, são geradas uma nova $(k+1)$ -clique e k cliques de tamanho k . O próximo teorema quantifica o total de arestas e de cliques para uma k -árvore.

Teorema 1.3: Uma k -árvore com n vértices, sendo $n \geq k$, contém:

- a) $k(k-1)/2 + k(n-k)$ arestas;
- b) $k(n-k) + 1$ cliques de tamanho k ;
- c) $n-k$ cliques de tamanho $k+1$.

Prova: A prova será por indução sobre o número de vértices da k -árvore.

- Para $n = k$, a k -árvore é um grafo completo com k vértices e o teorema vale trivialmente.
- Suponha que o resultado vale para k -árvores com $n - 1$ vértices. Assim, elas possuem $k(k-1)/2 + k(n-1-k)$ arestas, $k(n-1-k) + 1$ cliques de tamanho k e $n-1-k$ cliques de tamanho $k+1$.
- A adição de um vértice e k arestas adjacentes aos vértices de uma das k -cliques de uma k -árvore de $n-1$ vértices transforma o grafo em uma k -árvore de n vértices com um total de $k(k-1)/2 + k(n-1-k) + k = k(k-1)/2 + k(n-k)$ arestas. O vértice adicionado forma uma $(k+1)$ -clique com os vértices da k -clique em que foi conectado, logo são $n-1-k+1 = n-k$ cliques de tamanho $k+1$. Como esta $(k+1)$ -clique tem $k+1$ cliques de tamanho k e apenas uma delas já existia no grafo, então são $k(n-1-k) + 1 + k = k(n-k) + 1$ cliques de tamanho k . Desse modo, as três propriedades valem para k -árvores com n vértices e o teorema está provado. \Rightarrow

Os vértices simpliciais são muito importantes principalmente nos grafos cordais. O próximo lema mostra que estes vértices pertencem a uma única clique maximal de um grafo.

Lema 1.4 [5]: Seja o grafo $G=(V,E)$. Um vértice $v \in V$ é simplicial se e somente pertence a uma única clique maximal.

Seja $G=(V,E)$ uma k -árvore com n vértices, sendo $n > k$. Uma **k -folha** de G é um vértice que pertence a uma única $(k+1)$ -clique de G . Particularmente, as k -árvores com k vértices não possuem k -folhas e todos os vértices das k -árvores com $k + 1$ vértices são k -folhas. O próximo lema mostra que, nas k -árvores, as k -folhas são os vértices simpliciais e possuem grau k .

Lema 1.5: Sejam $G=(V,E)$ uma k -árvore com n vértices, sendo $n > k$ e um vértice $v \in V$. As seguintes expressões são equivalentes:

- (1) v é uma k -folha, ou seja, v pertence a apenas uma $(k+1)$ -clique;
- (2) v é simplicial;
- (3) v tem grau k .

Prova: (1) \Leftrightarrow (2). Em G , todas as cliques maximais possuem $k + 1$ vértices. Pelo Lema 1.4, um vértice é simplicial em G se e somente se pertence a apenas uma $(k+1)$ -clique. Logo, v é simplicial em G se e somente se v é uma k -folha.

(1) \Rightarrow (3). Seja v um vértice que pertence a apenas uma $(k+1)$ -clique C . Pela Definição 1.2 de k -árvore, v é adjacente apenas aos demais k vértices de C , ou seja, v possui grau k .

(3) \Rightarrow (1). Seja v um vértice com grau k . Pela Definição 1.2 de k -árvore, v é adjacente a apenas uma k -clique, ou seja, v pertence a apenas uma $(k+1)$ -clique. \Rightarrow

O lema mostrado a seguir quantifica os vértices simpliciais de uma k -árvore e foi comentado no trabalho de RÉNYI e RÉNYI [42]. As k -árvores com k e $k + 1$ vértices são grafos completos e todos os vértices são simpliciais. Portanto, a propriedade quantifica os vértices simpliciais das k -árvores com mais de $k + 1$ vértices.

Lema 1.6: Uma k -árvore com n vértices, sendo $n > k + 1$, contém no mínimo dois e no máximo $n - k$ vértices simpliciais.

Prova: A prova será por indução sobre o número de vértices da k -árvore.

- Para $n = k + 2$, a k -árvore correspondente é formada pela adição de um vértice v adjacente a uma k -clique Q de um grafo completo com $k + 1$ vértices. A k -clique Q está contida nas duas $(k+1)$ -cliques desta k -árvore, logo nenhum dos vértices de Q é simplicial. O vértice v adicionado e o vértice w que já pertencia ao grafo são simpliciais, pois são adjacentes apenas aos vértices da k -clique Q . Como $n - k = k + 2 - k = 2$ e a k -árvore com $k + 2$ vértices possui exatamente dois vértices simpliciais, então o teorema vale para $n = k + 2$.

- Suponha que o teorema vale para uma k -árvore com $n - 1$ vértices. Logo, esta k -árvore tem no mínimo 2 e no máximo $n - k - 1$ vértices simpliciais.

- Pela Definição 1.2, o último vértice adicionado a uma k -árvore é simplicial. Logo, a adição de um vértice a uma k -clique incrementa em um o número de vértices simpliciais, caso seja escolhida uma k -clique que não contenha vértices simpliciais, ou mantém o número de vértices simpliciais, caso seja escolhida uma k -clique com um vértice simplicial. Assim, a k -árvore com n vértices possui no mínimo 2 e no máximo $(n - k - 1) + 1 = n - k$ vértices simpliciais e o teorema vale para k -árvores com n vértices. \Rightarrow

Ao longo da construção de uma k -árvore (Definição 1.2), a partir da k -árvore com $k + 2$ vértices, quando todas as k -cliques Q utilizadas contêm vértices simpliciais, obtém-se uma k -árvore com dois vértices simpliciais, e, quando nenhuma das k -cliques utilizadas contêm vértices simpliciais, obtém-se uma k -árvore com $n - k$ vértices simpliciais.

Capítulo 2 - Codificação de Árvores

A contagem e a codificação de árvores são assuntos estudados há mais de um século. Este capítulo apresenta resultados conhecidos na literatura sobre codificação de árvores e sua utilização na geração e na contagem de árvores rotuladas. Também são vistas a geração e a contagem de árvores com características específicas, como por exemplo, árvores que tenham um número mínimo de folhas.

A Seção 2.1 mostra a codificação proposta por PRÜFER [41] em 1918, que é a mais citada na literatura, em conjunto com os seus algoritmos de codificação e de decodificação. Na Seção 2.2, são exibidas a contagem e a geração de árvores rotuladas utilizando o código de Prüfer. Alguns dos teoremas sobre contagens de árvores rotuladas com características específicas foram levantados por MOON [35] em 1970. Estes resultados são aqui provados novamente, utilizando-se agora a codificação de Prüfer. Com base nessas novas provas, mostramos como fazer a geração destas árvores utilizando as propriedades do código de Prüfer. Na Seção 2.3, são apresentados três outros códigos baseados no esquema de Prüfer: duas codificações distintas de Neville modificadas por MOON e a codificação de DEO e MICIKEVICIUS. Finalmente, na Seção 2.4, é descrita a abordagem proposta recentemente por CAMINITI *et al.* [8], que possibilita a codificação e a decodificação das quatro codificações estudadas neste capítulo com complexidade $O(n)$.

2.1 - Código de Prüfer

Nesta seção, é apresentado o código de Prüfer, a codificação mais utilizada de árvores. São descritos seus algoritmos de codificação e de decodificação assim como algumas de suas aplicações. A representação proposta por Prüfer é uma codificação para árvores rotuladas, pois existe uma relação biunívoca entre sua codificação e as árvores rotuladas.

O código de Prüfer é muito utilizado na geração de árvores devido a sua simplicidade e a eficiência do processo de obtenção do código. Algumas aplicações são também conhecidas, por exemplo, na geração aleatória de árvores e de grafos conexos [26] e na geração aleatória de grafos cordais [2]. Além de permitir a contagem e a

geração de árvores rotuladas, a utilização da codificação possibilita que sejam verificadas propriedades de uma árvore diretamente do seu código. MOON [35] mostrou estudos probabilísticos para o código de Prüfer com o intuito de estimar valores para o diâmetro, o raio e o número de folhas de árvores rotuladas geradas de modo aleatório.

Outra aplicação interessante é a utilização do código de Prüfer em algoritmos genéticos na solução de problemas relacionados a árvores. Os algoritmos genéticos são uma das meta-heurísticas utilizadas na solução de problemas *NP-completos*. Estes algoritmos manipulam cromossomos que são estruturas de dados que representam uma das possíveis soluções do problema. Nos algoritmos genéticos em que cromossomos representam árvores, um cromossomo pode ser representado pelo código de Prüfer correspondente a árvore. Por exemplo, em [14] e em [48], a codificação de Prüfer é aplicada em algoritmos genéticos para encontrar árvores geradoras mínimas com características específicas, tais como certo número de folhas, grau máximo ou diâmetro da árvore. Em 1990, TINHOFER [46] faz um levantamento das técnicas de geração aleatória uniforme de grafos e cita como a codificação de Prüfer é utilizada na geração de árvores rotuladas sem raiz.

Considera-se que os vértices das árvores são rotulados de 1 a n para simplificação dos algoritmos de codificação e decodificação. Os lemas apresentados a seguir são utilizados como base na definição do código de Prüfer.

Lema 2.1: Toda árvore com pelo menos dois vértices possui pelo menos duas folhas.

Lema 2.2: Seja G um grafo com n vértices, sendo $n > 1$. G é uma árvore se e somente se o grafo obtido pela remoção de uma folha for uma árvore com $n - 1$ vértices.

O código de Prüfer de uma árvore é construído sucessivamente pela remoção da folha com menor rótulo e adição do rótulo de seu vizinho ao código. Este procedimento é válido porque o Lema 2.1 garante a existência de pelo menos duas folhas e o Lema 2.2 garante que o grafo obtido é uma árvore. O processo de remoção de folhas é interrompido quando restam dois vértices e o código de Prüfer é composto por uma seqüência de $n - 2$ rótulos de vértices da árvore. Cabe ressaltar que a

árvore trivial não tem representação de Prüfer e o código de Prüfer de uma árvore com dois vértices é uma seqüência vazia.

A Figura 2.1 mostra um exemplo passo a passo de como é obtido o código $\langle 4,7,1,1,3 \rangle$ para o grafo da Figura 2.1(a). Inicia-se com a árvore a ser codificada e com o código de Prüfer vazio. Remove-se a folha de menor rótulo, que é o vértice de rótulo 2, e adiciona-se o rótulo de seu vizinho ao código obtendo o código de Prüfer parcial $\langle 4 \rangle$ e a subárvore exibida na Figura 2.1(b). Agora, remove-se o vértice de rótulo 4, que é a folha de menor rótulo, e adiciona-se o rótulo 7, que é o rótulo de seu vizinho, ao código obtendo o código de Prüfer parcial $\langle 4,7 \rangle$ e a subárvore da Figura 2.1(c). A seguir, são removidos os vértices de rótulos 5 e 6, obtendo o código de Prüfer parcial $\langle 4,7,1,1 \rangle$ e a subárvore da Figura 2.1(e). O vértice de rótulo 1 só passa a ser folha após a remoção do vértice de rótulo 6. Finalmente, remove-se o vértice de rótulo 1 obtendo-se o código de Prüfer $\langle 4,7,1,1,3 \rangle$.

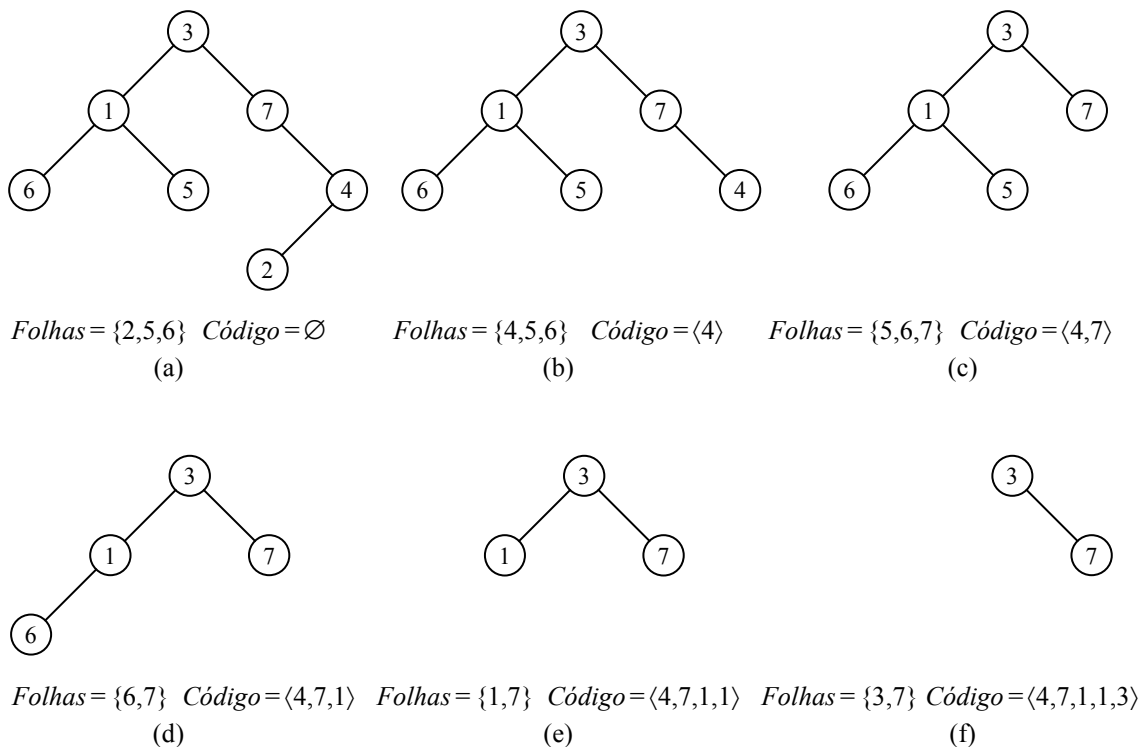


Figura 2.1: Obtenção do código de Prüfer para uma árvore.

O Algoritmo 2.1 obtém o código de Prüfer para uma árvore $T = (V,E)$. A lista *Código* começa vazia e os rótulos das folhas da árvore são inseridos no conjunto *Folhas*. O processo de remoção de folhas é repetido até que restem apenas dois vértices.

De acordo com a definição do código, remove-se a folha com menor rótulo, que é o vértice v do algoritmo, e adiciona-se o rótulo de seu vizinho ao final do código, que é o vértice u do algoritmo. Além disso, removem-se o vértice v e a aresta $\{v,u\}$ do grafo, pois o vértice v não pertence mais à subárvore obtida. Caso o vértice u se torne uma folha na subárvore obtida, adiciona-se seu rótulo ao conjunto das folhas. Ao final do algoritmo, a lista *Código* contém uma seqüência com $n - 2$ rótulos que são o código de Prüfer da árvore $T = (V,E)$.

```

Algoritmo Gera_Código_Prüfer;
Entrada:  Árvore  $T = (V,E)$ , com  $|V| \geq 2$ ;
Saída:   Lista Código com a codificação de Prüfer;
Início
  Código  $\leftarrow \emptyset$ ;
  Folhas  $\leftarrow \{v \in V \mid \text{grau}(v) = 1\}$ ;
  Enquanto  $|V| > 2$  faça
    Encontre  $v \in \text{Folhas}$  com rótulo mínimo;
     $E \leftarrow E - \{v,u\}$ ;
    Adicione  $u$  ao fim da lista Código;
    Se  $\text{grau}(u) = 1$  então  $\text{Folhas} \leftarrow \text{Folhas} \cup \{u\}$ ;
     $V \leftarrow V - \{v\}$ ;
Fim

```

Algoritmo 2.1: Geração do código de Prüfer para árvores.

Para que a codificação de Prüfer seja aplicada a árvores enraizadas, basta estabelecer que o vértice raiz não seja considerado folha mesmo que tenha grau um. O código de Prüfer para árvores enraizadas é composto por $n - 1$ rótulos de vértices, isto é, um a mais que a codificação das árvores sem raiz. Este rótulo a mais é da raiz da árvore e ocupa a última posição do código.

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

A Figura 2.2 mostra um exemplo passo a passo de como é obtido o código de Prüfer $\langle 4,7,1,3,1,1 \rangle$ para a árvore enraizada no vértice de rótulo 5 vista na Figura 2.2(a). Na Figura 2.2, o vértice de rótulo 5 é a raiz da árvore e, por isso, não é considerado folha na codificação. Cabe ressaltar que a Figura 2.2(f) é obtida a partir da Figura 2.2(e) pela remoção sucessiva dos vértices de rótulos 3 e 1 e pela conseqüente adição dos rótulos 1 e 5 à lista *Código*. Na Tabela 2.1, são exibidas todas as possibilidades de enraizar a árvore da Figura 2.1(a) juntamente com os códigos de Prüfer das respectivas árvores.

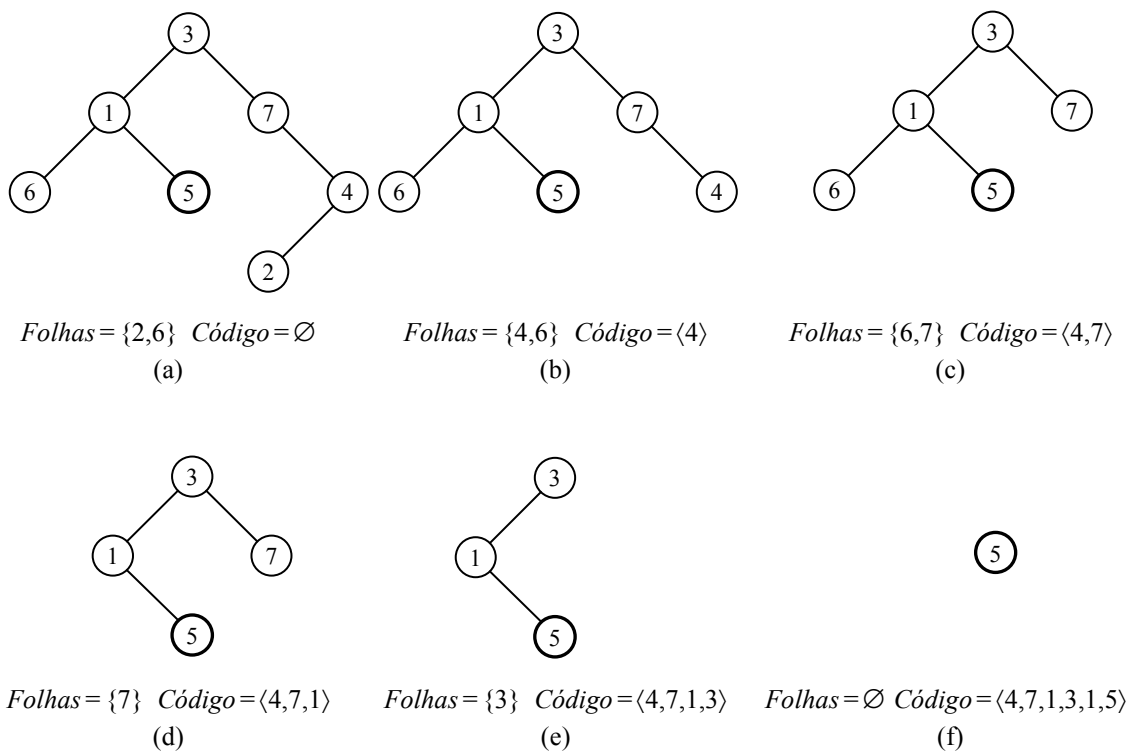


Figura 2.2: Codificação de árvore com raiz no vértice de rótulo 5 usando Prüfer.

Tabela 2.1: Todos os códigos de Prüfer para a árvore da Figura 2.1(a).

<i>Raiz</i>	<i>Código de Prüfer</i>	<i>Raiz</i>	<i>Código de Prüfer</i>
1	$\langle 4,7,1,1,3,1 \rangle$	2	$\langle 1,1,3,7,4,2 \rangle$
3	$\langle 4,7,1,1,3,3 \rangle$	4	$\langle 4,1,1,3,7,4 \rangle$
5	$\langle 4,7,1,3,1,5 \rangle$	6	$\langle 4,7,1,3,1,6 \rangle$
7	$\langle 4,7,1,1,3,7 \rangle$	-	$\langle 4,7,1,1,3 \rangle$

O Algoritmo 2.2 mostra como construir uma árvore a partir do seu código de Prüfer. Como visto antes, este procedimento é chamado de decodificação da árvore. Seja t o tamanho da codificação de Prüfer. Como o código de Prüfer é composto por $n-2$ rótulos de vértices, conclui-se que $n=t+2$ e assume-se que $V = \{1, 2, \dots, t+2\}$. Na codificação, os rótulos das folhas da árvore nunca são inseridos no código de Prüfer, logo os rótulos das folhas são encontrados através do comando $Folhas = V - Código$. Inicia-se com o conjunto de arestas E vazio e constrói-se a árvore pela adição sucessiva de arestas $\{u,v\}$. Para cada um dos $n-2$ rótulos u da lista $Código$, insere-se uma aresta adjacente ao vértice de rótulo u e adjacente à folha v de menor rótulo dentre as ainda não utilizadas. Após a remoção do rótulo u da lista $Código$, caso u não apareça mais nesta lista, então o vértice de rótulo u passa a ser uma folha sendo inserido no respectivo conjunto. Logo após todos os rótulos da lista $Código$ serem analisados, insere-se a última aresta que é adjacente às duas folhas ainda não utilizadas.

```

Algoritmo Gera_Árvore;
Entrada: Lista  $Código$  com o código de Prüfer;
Saída: Conjunto de arestas  $E$ ;
Início
     $t \leftarrow$  tamanho da lista  $Código$ ;
     $V \leftarrow \{1, 2, \dots, t+2\}$ ;
     $E \leftarrow \emptyset$ ;
     $Folhas \leftarrow V - Código$ ;
    Enquanto  $Código \neq \emptyset$  faça
        Retire primeiro elemento  $u$  da lista  $Código$ ;
        Encontre  $v \in Folhas$  com rótulo mínimo;
         $Folhas \leftarrow Folhas - \{v\}$ ;
         $E \leftarrow E \cup \{\{u, v\}\}$ ;
        Se  $u \notin Código$  então  $Folhas \leftarrow Folhas \cup \{u\}$ ;
        Seja  $Folhas = \{u, v\}$ ;
         $E \leftarrow E \cup \{\{u, v\}\}$ ;
    Fim

```

Algoritmo 2.2: Geração da árvore a partir do código de Prüfer.

A Figura 2.3 exemplifica passo a passo como obter a árvore representada pelo código de Prüfer $\langle 4, 7, 1, 1, 3 \rangle$. Os rótulos sublinhados são dos vértices adjacentes à aresta inserida em cada passo do Algoritmo 2.2 de decodificação.

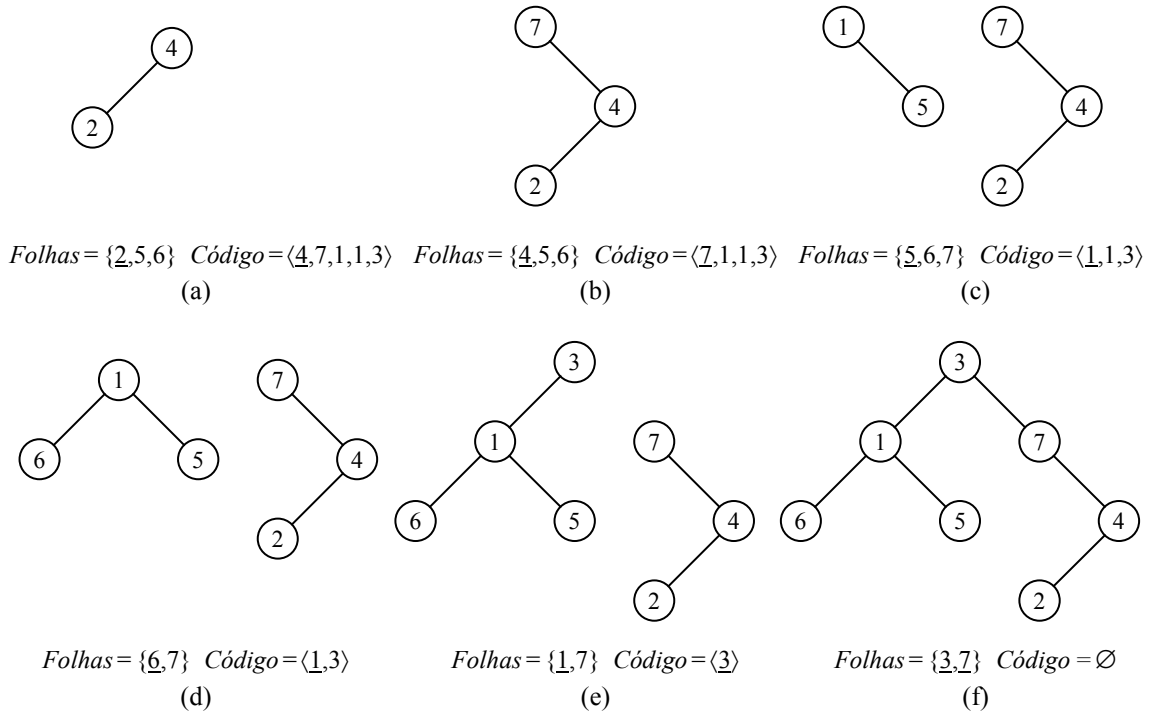


Figura 2.3: Construção de árvore, dado seu código de Prüfer.

2.2 - Contagem e Geração Utilizando o Código de Prüfer

Nesta seção, são apresentadas propriedades obtidas diretamente do código de Prüfer de uma árvore e são descritos cinco resultados sobre contagem e geração de árvores rotuladas. Primeiro, mostra-se a contagem e a geração de árvores sem restrições. Em seguida, mostra-se a geração de árvores com um número mínimo de folhas. Finalmente, são apresentados três resultados sobre contagem e geração de árvores com as seguintes restrições: seqüência de graus dada, vértice específico com grau q e contendo exatamente q folhas. A contagem destas árvores foi levantada por MOON [35]. Aqui refazemos a prova destes resultados utilizando o código de Prüfer e mostramos para cada caso como gerar aleatoriamente estas árvores.

Após a definição do código de Prüfer e dos algoritmos de obtenção do código e de construção da árvore a partir do código, torna-se possível observar algumas propriedades a partir do código de uma árvore. Observa-se, por exemplo, que um código com t vértices corresponde a uma árvore com $t + 2$ vértices. O lema e o corolário vistos a seguir extraem propriedades estruturais da árvore diretamente de seu código de Prüfer e são utilizados na contagem e na geração de árvores com características específicas.

Lema 2.3: Sejam $T=(V,E)$ uma árvore e $d(v)$ o grau do vértice $v \in V$. O rótulo do vértice v ocorre $d(v) - 1$ vezes no código de Prüfer de T .

Prova: No processo de codificação de uma árvore $T=(V,E)$, toda vez que se remove uma folha, adiciona-se o rótulo do vértice adjacente a esta folha ao código. Desse modo, o rótulo de uma folha nunca é inserido no código e os rótulos dos demais vértices são inseridos toda vez que uma folha adjacente a eles é removida. Assim, o rótulo de um vértice é inserido no código até que o vértice se torne folha. Portanto, conclui-se que o rótulo de um vértice v aparece $d(v) - 1$ vezes no código de Prüfer. \Rightarrow

Corolário 2.4: Seja $T=(V,E)$ uma árvore e um vértice $v \in V$. Se o rótulo de v não é incluído no código de Prüfer de T , então v é uma folha.

Prova: Quando um vértice v não é representado no código de Prüfer de T , tem-se que seu rótulo aparece 0 vezes neste código. Desse modo, pelo Lema 2.3, tem-se que $d(v) - 1 = 0$. Logo, $d(v) = 1$, ou seja, o vértice v é uma folha. \Rightarrow

Para a árvore da Figura 2.1(a) cujo código de Prüfer é $\langle 4,7,1,1,3 \rangle$, verifica-se que possui 7 vértices, pois sua representação é composta por 5 rótulos de vértices. Como os rótulos 2, 5 e 6 não estão na seqüência do código, infere-se que os vértices com estes rótulos são folhas. Além disso, observa-se que os vértices de rótulos 3, 4 e 7 têm grau 2, pois estes rótulos aparecem uma única vez no código e que o vértice de rótulo 1 tem grau 3, pois este rótulo aparece duas vezes na seqüência.

2.2.1 Contagem e Geração de Árvores Rotuladas sem Restrições

Denota-se $T(n)$ a quantidade de árvores rotuladas com n vértices. Usualmente atribui-se a fórmula $T(n) = n^{n-2}$ a Cayley (1889). Entretanto, em 1970, MOON [35] comenta que este resultado já havia sido enunciado sem prova por Silvester em 1857 e que um resultado equivalente ao de Cayley foi provado por Borchardt em 1860. Existem várias provas para a fórmula de Cayley, sendo a prova que utiliza a codificação de Prüfer a mais elegante e mais referenciada na literatura: já que existe uma relação biunívoca entre o código de Prüfer e as árvores rotuladas, basta ver que existem n^{n-2} combinações possíveis para esta codificação.

A geração do código de Prüfer de uma árvore sem restrições consiste na geração aleatória de uma seqüência de $n - 2$ rótulos que são escolhidos dentre n rótulos dos vértices.

2.2.2 Geração de Árvores Rotuladas com um Número Mínimo de Folhas Dado

Com base no Corolário 2.4, verifica-se que uma árvore com um número mínimo de folhas q tem pelo menos q vértices não representados em seu código de Prüfer. Este fato pode ser utilizado na geração aleatória de uma árvore com tal característica. O processo de geração é dividido em três passos. Primeiro, o código de Prüfer é iniciado com $n - 2$ posições vazias. Depois, selecionam-se os $n - q$ rótulos de vértices que podem pertencer ao código e os q rótulos de vértices não selecionados são obrigatoriamente folhas. Finalmente, para preencher cada uma das $n - 2$ posições do código, seleciona-se um rótulo qualquer dentre os $n - q$ possíveis.

A Figura 2.4 ilustra a geração de uma árvore com 7 vértices e pelo menos 4 folhas. Observe que, no preenchimento aleatório, apenas os rótulos 2, 4 e 6 podem ser inseridos no código de Prüfer e que, apesar disto, o rótulo 4 não foi inserido no código durante a geração aleatória. Por isso, a árvore gerada possui exatamente 5 folhas e satisfaz a restrição de ter pelo menos 4 folhas. A contagem e a geração de árvores com exatamente q folhas é feita na Subseção 2.2.5.

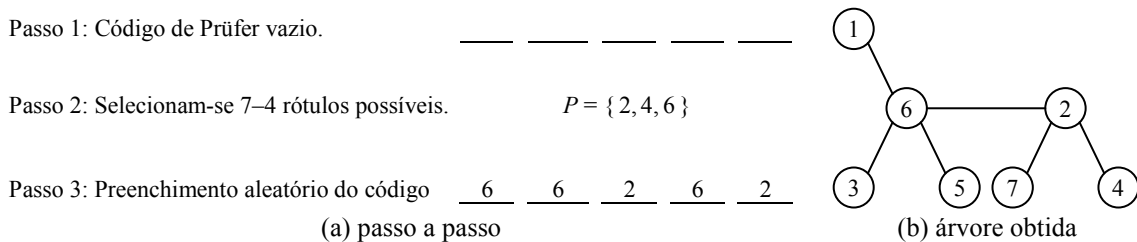


Figura 2.4: Geração aleatória de árvore com 7 vértices e pelo menos 4 folhas.

2.2.3 Contagem e Geração de Árvores Rotuladas com Seqüência de Graus Dada

Seja uma árvore com n vértices rotulados de 1 a n . Diz-se que sua seqüência de graus é a seqüência de inteiros $\langle d(1), \dots, d(n) \rangle$. O próximo teorema quantifica o total de

árvores que tem uma seqüência de graus dada $\langle d_1, \dots, d_n \rangle$. Sabe-se da teoria de grafos que em uma árvore $\sum_{i=1}^n d(i) = 2n - 2$. Diz-se que uma seqüência de graus é **válida** quando satisfaz esta igualdade. O Teorema 2.5 quantifica as árvores com uma seqüência de graus dada.

Teorema 2.5 [35]: A quantidade total de árvores rotuladas com n vértices com a seqüência de graus válida $\langle d_1, \dots, d_n \rangle$ é igual a: $\binom{n-2}{d_1-1, \dots, d_n-1}$.

Prova: A prova é construtiva. Inicia-se com um código de Prüfer com as $n - 2$ posições vazias e adicionam-se rótulos até obter o código para a árvore desejada. Pelo Lema 2.3, cada vértice v é representado $d_v - 1$ vezes no código. Assim, para cada um dos n vértices, atribuem-se seus rótulos a $d_v - 1$ posições ainda não utilizadas no código. Todas as $n - 2$ posições são ocupadas por algum rótulo, pois pela teoria dos grafos sabe-se que em uma árvore $\sum_{i=1}^n d_i = 2n - 2$, e portanto, $\sum_{i=1}^n (d_i - 1) = n - 2$. Existem $(n - 2)!$ códigos de árvores. Como cada vértice é representado $d_v - 1$ vezes, então para cada vértice v existem $(d_v - 1)!$ códigos iguais. Assim, o total de árvores é $\frac{(n - 2)!}{(d_1 - 1)! \dots (d_n - 1)!}$. \square

A geração aleatória de árvores com uma seqüência de graus dada $\langle d_1, \dots, d_n \rangle$ é feita em dois passos de modo análogo ao descrito na prova do Teorema 2.5. Primeiro, inicia-se o código de Prüfer com $n - 2$ posições vazias. Depois, para cada um dos n vértices atribuem-se $d_v - 1$ rótulos a posições ainda não utilizadas no código. Ao final, obtém-se o código de Prüfer de uma árvore com a seqüência de graus dada.

A Figura 2.5 ilustra a geração de uma árvore com seis vértices cuja seqüência de graus é $\langle 1, 1, 1, 2, 4, 1 \rangle$. O total de árvores com esta seqüência de graus é obtido através do Teorema 2.5 assim: $\binom{6-2}{0, 0, 0, 1, 3, 0} = \frac{4!}{0! 0! 0! 1! 3! 0!} = 4$.

Os códigos de Prüfer para as árvores cuja seqüência de graus é $\langle 1, 1, 1, 2, 4, 1 \rangle$ são: $\langle 4, 5, 5, 5 \rangle$, $\langle 5, 4, 5, 5 \rangle$, $\langle 5, 5, 4, 5 \rangle$ e $\langle 5, 5, 5, 4 \rangle$. Estas quatro árvores são exibidas na Figura 2.6.

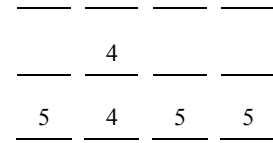
v	1	2	3	4	5	6
$d(v)$	1	1	1	2	4	1
$d(v)-1$	0	0	0	1	3	0

(a) graus dos vértices.

Passo 1: Código de Prüfer vazio.

Passo 2: Preenchimento de 1 posição com o rótulo 4.

Passo 2: Preenchimento de 3 posições com o rótulo 5.



(b) passo a passo

Figura 2.5: Geração aleatória de árvore com seqüência de graus $\langle 1, 1, 1, 2, 4, 1 \rangle$.

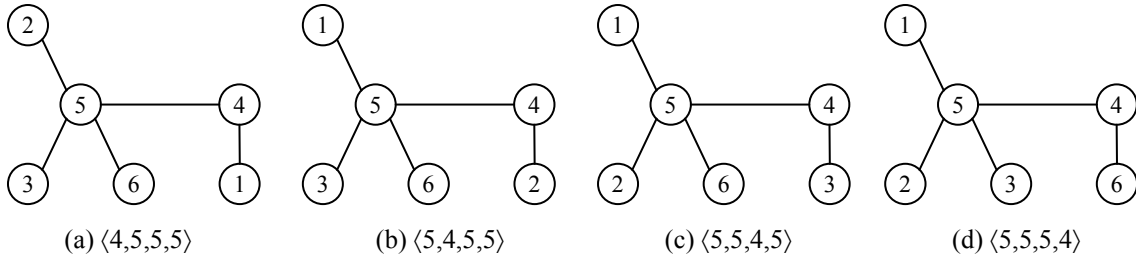


Figura 2.6: Todas as árvores com seqüência de graus $\langle 1, 1, 1, 2, 4, 1 \rangle$.

2.2.4 Contagem e Geração de Árvores Rotuladas onde um Vértice Específico tem Grau Dado

Seja $C(n, q)$ a quantidade de árvores rotuladas com n vértices onde um vértice específico tem grau q . Seja uma árvore cujo vértice de rótulo x tem grau q . No código de Prüfer desta árvore, o rótulo x aparece $q - 1$ vezes. O teorema a seguir calcula o total de árvores rotuladas com essa propriedade.

Teorema 2.6 [35]: $C(n, q) = \binom{n-2}{q-1} (n-1)^{n-q-1}$, para $1 \leq q \leq n-1$.

Prova: A prova é construtiva. Inicia-se com um código de Prüfer com as $n - 2$ posições vazias e adicionam-se rótulos até obter o código para a árvore desejada. Sem perda de generalidade, seja o vértice de rótulo n aquele cujo grau é q . Assim, pelo Lema 2.3, o rótulo n aparece $q - 1$ vezes no código. Logo, atribui-se o rótulo n a $q - 1$ posições do código perfazendo um total de $\binom{n-2}{q-1}$ possibilidades (1). As $n - 2 - (q - 1) = n - q - 1$ posições restantes podem ser ocupadas por qualquer um dos outros $n - 1$ rótulos, logo existem $(n - 1)^{n-q-1}$ possibilidades para terminar de preencher o código gerado (2). Das expressões (1) e (2) obtém-se o resultado do teorema. \square

A geração aleatória de árvores com n vértices onde $d(n) = q$ é feita em três passos de modo análogo ao descrito na prova do Teorema 2.6. Primeiramente, inicia-se o código de Prüfer com as $n - 2$ posições vazias. Depois, aleatoriamente, atribuem-se a $q - 1$ posições do código o valor n . Em seguida, para cada uma das $n - q - 1$ posições restantes, escolhe-se de modo aleatório um valor entre 1 e $n - 1$.

A Figura 2.7 ilustra a geração de uma árvore com 5 vértices onde $d(5) = 3$. Calcula-se o total de árvores com 5 vértices onde $d(5) = 3$, $C(5,3)$, através do Teorema 2.6 assim:

$$C(5,3) = \binom{5-2}{3-1} (5-1)^{5-3-1} = \binom{3}{2} (4)^1 = 12.$$

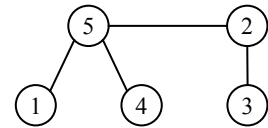
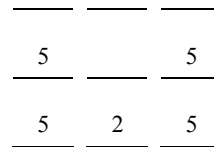
Os códigos de Prüfer para as árvores com 5 vértices onde $d(5) = 3$ são: $\langle 1,5,5 \rangle$, $\langle 2,5,5 \rangle$, $\langle 3,5,5 \rangle$, $\langle 4,5,5 \rangle$, $\langle 5,1,5 \rangle$, $\langle 5,2,5 \rangle$, $\langle 5,3,5 \rangle$, $\langle 5,4,5 \rangle$, $\langle 5,5,1 \rangle$, $\langle 5,5,2 \rangle$, $\langle 5,5,3 \rangle$ e $\langle 5,5,4 \rangle$.

Passo 1: Código de Prüfer vazio.

Passo 2: Preenchimento de 2 posições com o rótulo 5.

Passo 3: Preenchimento aleatório das posições restantes.

(a) passo a passo.



(b) árvore obtida.

Figura 2.7: Geração aleatória de árvore onde o vértice de rótulo 5 tem grau 3.

2.2.5 Contagem e Geração de Árvores com Número de Folhas Dado

Já foi mostrada a geração de árvores com pelo menos q folhas. O Teorema 2.7, visto logo a seguir, trata do total de árvores com exatamente q folhas. Após a prova deste Teorema, mostra-se a geração de árvores com tal característica.

O resultado do Teorema 2.7 utiliza os números de *Stirling* do segundo tipo ou número de *Stirling* para conjuntos. O número de *Stirling* para conjuntos $S(a,b)$ conta o número de maneiras de particionar um conjunto de a elementos em exatamente b subconjuntos não vazios. Calculam-se os valores para $S(a,b)$ assim:

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

A Figura 2.8 ilustra a permutação de um conjunto de quatro elementos em 1, 2, 3 e 4 conjuntos disjuntos não vazios e a Figura 2.9 mostra o triângulo para os números de *Stirling* do segundo tipo. Neste triângulo, a n -ésima linha contém os números de *Stirling* $S(n,1), \dots, S(n,n)$.

1 conjunto:	{1, 2, 3, 4}.
2 conjuntos:	{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 conjuntos:	{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 conjuntos:	{1}, {2}, {3}, {4}.

Figura 2.8: Permutação de quatro elementos em conjuntos disjuntos não vazios.

1
1 1
1 3 1
1 7 6 1
1 15 25 10 1
1 31 90 65 15 1

Figura 2.9: Triângulo de números de *Stirling* do segundo tipo.

Seja $R(n,q)$ a quantidade de árvores rotuladas com n vértices e exatamente q folhas.

Teorema 2.7 [35]: $R(n,q) = \frac{n!}{q!} S(n-2, n-q)$, para $2 \leq q \leq n-1$.

Prova: Para uma árvore com n vértices e exatamente q folhas, o código de Prüfer possui exatamente $n - q$ rótulos de vértices distintos. Seja o código de Prüfer de uma árvore $\langle c_1, \dots, c_{n-2} \rangle$ contendo exatamente $n - q$ rótulos $\{l_1, \dots, l_{n-q}\}$. Para este específico código, definem-se $n - q$ conjuntos P_i formados pelas posições ocupadas pelo rótulo l_i no código, ou seja, $j \in P_i$ se e somente se $c_j = l_i$. Deste modo os conjuntos P_i têm as seguintes propriedades:

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

Observe que cada código corresponde a conjuntos P_i diferentes. Assim, para o código de Prüfer $\langle c_1, \dots, c_{n-2} \rangle$ contendo exatamente $n - q$ rótulos de vértices existe uma

conjugação única onde cada rótulo do conjunto $\{l_1, \dots, l_{n-q}\}$ está relacionado a um dos conjuntos P_1, \dots, P_{n-q} que satisfazem as três condições descritas acima.

O número de *Stirling* para conjuntos $S(a,b)$ conta o número de maneiras de particionar um conjunto de a elementos em exatamente b subconjuntos não vazios. Na contagem das árvores com exatamente q folhas, as $n-2$ posições do código formam um conjunto que deve ser particionado em $n-q$ conjuntos P_i que representam as posições ocupadas pelos rótulos dos vértices. Assim, tem-se um total de $S(n-2, n-q)$ combinações para P_1, \dots, P_{n-q} (1).

Para escolher os rótulos dos q vértices que são folhas e dos $n-q$ vértices que pertencem obrigatoriamente ao código, existem um total de $\frac{n!}{q!}$ combinações (2). Das

expressões (1) e (2) prova-se o teorema. \square

A geração aleatória de árvores com n vértices e exatamente q folhas é feita em quatro passos. Primeiro, inicia-se o código de Prüfer com as $n-2$ posições vazias. Depois, dentre os n vértices, escolhem-se aleatoriamente $n-q$ vértices $\{v_1, \dots, v_{n-q}\}$. Em seguida, para cada um dos $n-q$ vértices, atribui-se seu rótulo a uma posição não ocupada no código. Finalmente, para cada uma das $n-2-(n-q) = q-2$ posições restantes do código, atribui-se o rótulo de um vértice escolhido aleatoriamente dentre os vértices de $\{v_1, \dots, v_{n-q}\}$.

A Figura 2.10 ilustra a geração de uma árvore com 5 vértices e exatamente 3 folhas. Sabendo que $S(3,2) = 3$, calcula-se o total de árvores com 5 vértices e exatamente 3 folhas, $R(5,3)$, através do Teorema 2.7 do seguinte modo:

$$R(5,3) = \left(\frac{5!}{3!}\right)S(5-2, 5-3) = 20S(3,2) = 60.$$

Passo 1: Código de Prüfer vazio.

Passo 2: Escolha dos rótulos que não são das folhas.

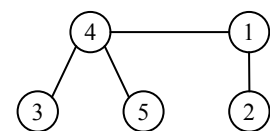
Escolhidos = {1, 4}

Passo 3: Coloca cada rótulo escolhido em 1 posição.

1		4
---	--	---

Passo 4: Preenchimento aleatório das posições restantes.

1	4	4
---	---	---



(a) passo a passo

(b) árvore obtida

Figura 2.10: Geração aleatória de árvore com 5 vértices e exatamente 3 folhas.

2.3 - Codificações Baseadas no Código de Prüfer

O código de Prüfer possibilita a geração de árvores com as mais variadas propriedades. Entretanto, foram propostos outros códigos que permitem identificar importantes características sem construir a árvore. Assim, para a segunda codificação proposta por Neville e para a codificação de Deo e Micikevicius, que são vistas a seguir, existem algoritmos bem simples que possibilitam o cálculo do diâmetro e dos centros da árvore diretamente a partir de seu código.

DEO e MICIKEVICIUS [13] criaram um sistema de classificação para qualquer código que utilize a mesma idéia de Prüfer. Uma codificação de árvores rotuladas é dita **baseada em Prüfer** quando:

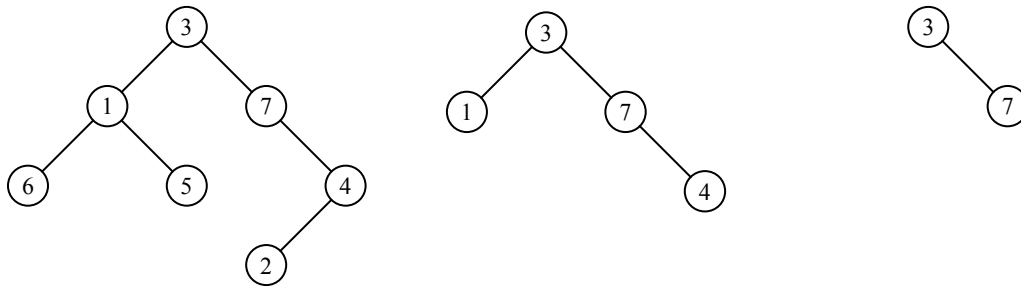
- Iterativamente remove folhas até que reste apenas uma aresta;
- Ao remover uma folha v , o rótulo do vértice adjacente a v é adicionado ao código.

Neste mesmo trabalho [13], os autores fazem um histórico das codificações para árvores rotuladas e citam que existem quatro representações na literatura além da proposta por Prüfer, sendo a mais recente proposta por eles em [11]. Essas codificações de árvores rotuladas existentes são baseadas em Prüfer e diferem apenas no modo de selecionar as folhas que são removidas da árvore a cada iteração do algoritmo.

De acordo com DEO e MICIKEVICIUS [13], em 1953, Neville propôs três codificações para árvores que necessitavam da escolha de um vértice como raiz da árvore e, em 1970, MOON [35] propôs uma modificação nos algoritmos de Neville para que eles não precisassem desta escolha. DEO e MICIKEVICIUS [13] comentam em seu trabalho que, com esta modificação, o primeiro método de Neville é equivalente ao de Prüfer. Por isso, consideram que existem três métodos baseados em Prüfer na literatura: o segundo de Neville, o terceiro de Neville e o método proposto por eles.

No segundo método de Neville, o código é computado em k estágios, onde k é o raio da árvore. Em cada estágio, as folhas da subárvore restante são excluídas na ordem crescente de rótulos, e os rótulos de seus vizinhos são armazenados na seqüência que representará a árvore. O processo é repetido até que restem apenas dois vértices na árvore. A Figura 2.11 ilustra a codificação da árvore vista na Figura 2.11(a). A remoção dos vértices de rótulos 2, 5 e 6, que estão no último nível da árvore, gera a subárvore da Figura 2.11(b), e a adição dos rótulos de seus ancestrais ao código forma

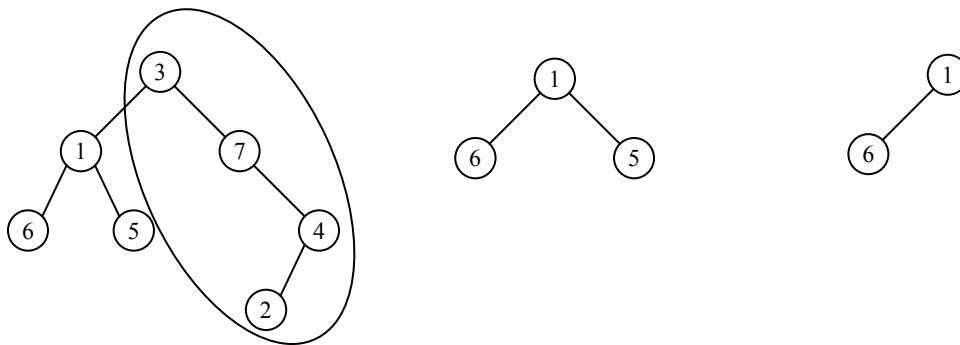
a seqüência $\langle 4,1,1 \rangle$. Para a subárvore da Figura 2.11(b), são removidas as folhas de rótulos 1 e 4, formando o código $\langle 4,1,1,3,7 \rangle$.



Folhas = {2,5,6} *Código* = \emptyset *Folhas* = {1,4} *Código* = $\langle 4,1,1 \rangle$ *Folhas* = \emptyset *Código* = $\langle 4,1,1,3,7 \rangle$
 (a) (b) (c)

Figura 2.11: Codificação de árvore usando 2º código de Neville.

O algoritmo de codificação do 3º código de Neville inicia com a remoção da folha de menor rótulo e adição do rótulo do vértice vizinho a essa folha ao código gerado. Após a remoção de uma folha, o algoritmo verifica se o vértice vizinho à folha removida é uma folha na subárvore gerada. Se o for, este vértice será a próxima folha a ser removida, e, caso contrário, a próxima folha a ser removida será a de menor rótulo. Na verdade, este procedimento faz com que o algoritmo remova um caminho simples, onde um dos extremos é a folha removida. As remoções são feitas até que restem dois vértices. No exemplo da Figura 2.12(a), a folha de rótulo 2 é removida, pois é a de menor rótulo. Removem-se consecutivamente os vértices de rótulos 4, 7 e 3 do grafo, pois formam um caminho. Os rótulos dos ancestrais destes vértices são adicionados ao código e obtém-se a seqüência $\langle 4,7,3,1 \rangle$. Para a subárvore restante, vista na Figura 2.12(b), remove-se a folha com rótulo 5, pois é a de menor rótulo e obtém-se o código $\langle 4,7,3,1,1 \rangle$.



Folhas = {2,5,6} *Código* = \emptyset *Folhas* = {5,6} *Código* = $\langle 4,7,3,1 \rangle$ *Folhas* = {1,6} *Código* = $\langle 4,7,3,1,1 \rangle$
 (a) (b) (c)

Figura 2.12: Codificação de árvore usando 3º código de Neville.

Recentemente, DEO e MICKEVICIUS [11] propuseram um novo código para representar árvores utilizando uma fila para codificar uma árvore e uma pilha para decodificá-la. No algoritmo de codificação de Deo e Micikevicius, inicialmente faz-se o armazenamento dos rótulos das folhas da árvore em uma fila em ordem crescente de rótulos. Em seguida, os rótulos das folhas são sucessivamente removidos desta fila e o vizinho da folha cujo rótulo foi removido é adicionado ao código. A cada passo, após a remoção de uma folha, caso seu vizinho seja uma folha na subárvore gerada, o rótulo deste vértice é adicionado ao final da fila. Repete-se o processo de remoção de rótulos de folhas da fila até que se obtenha uma subárvore com dois vértices. Por exemplo, para a árvore da Figura 2.13(a), os rótulos das folhas são armazenados em ordem crescente na fila. A remoção da folha de rótulo 2 insere o rótulo 4 no fim da fila. Em seguida, após a remoção das folhas de rótulos 5 e 6, o rótulo 1 é adicionado ao final da fila, conforme representado na Figura 2.13(b). Após a remoção das folhas de rótulos 4 e 1, obtém-se o código $\langle 4,1,1,7,3 \rangle$.

Diz-se que uma árvore T_1 está a uma distância k de uma árvore T_2 se existem k arestas de T_1 que não estão em T_2 . DEO e MICKEVICIUS [11] mostraram que é possível obter uma árvore T_2 de distância um de uma árvore T_1 diretamente da codificação da árvore T_1 . Esta propriedade tem aplicação direta em algoritmos genéticos cujos cromossomos representem árvores, tais como os algoritmos genéticos para solução de problemas de árvores geradoras com restrições. A aplicação em algoritmos genéticos e os algoritmos eficientes de codificação e decodificação são as principais vantagens desta codificação.

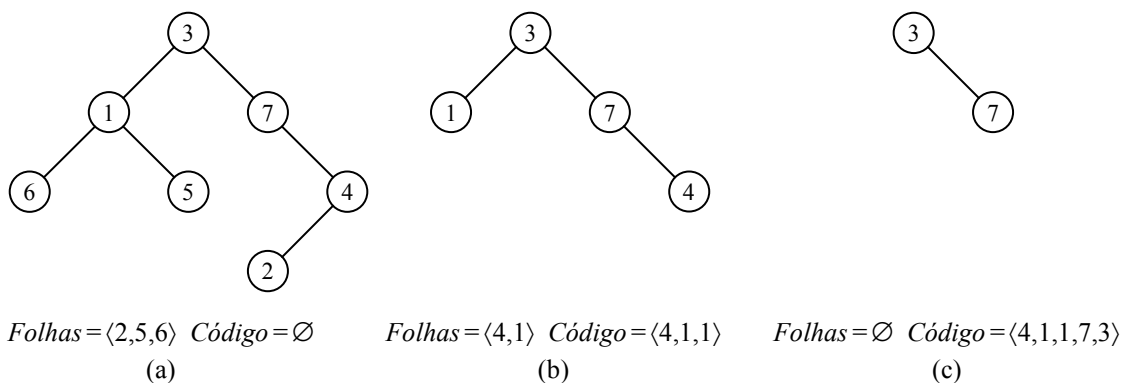


Figura 2.13: Codificação de árvore usando código de Deo e Micikevicius.

2.4 - Complexidade dos Algoritmos

DEO e MICIKEVICIUS [11] propuseram algoritmos de codificação e decodificação lineares junto com a codificação proposta por eles. CHEN e WANG [10] propuseram um algoritmo linear para codificação de Prüfer. Este algoritmo pode ser facilmente adaptado para a terceira codificação de Neville. GREENLAW [20] *et al.* propuseram um algoritmo linear para codificação de Prüfer que pode ser adaptado para a terceira codificação de Neville. Entretanto, para o segundo código de Neville, de acordo com DEO e MICIKEVICIUS [13], até recentemente não existiam algoritmos lineares para sua codificação e decodificação. CAMINITI *et al.* [8] fizeram uma abordagem que resultou em algoritmos genéricos para codificação e decodificação das quatro codificações da literatura: a de Prüfer, a segunda de Neville, a terceira de Neville e a de Deo e Micikevicius. Estes algoritmos têm complexidade linear para os quatro códigos.

Para árvores com raiz, todos os quatro códigos estudados têm tamanho $n - 1$, sendo que o último rótulo do código é o rótulo da raiz. Para árvores sem raiz, o esquema de remoção dos rótulos implicitamente define uma raiz para a árvore. Como comentado anteriormente, na codificação de Prüfer o vértice de rótulo n é implicitamente considerado como raiz. Já para o segundo código de Neville, o esquema de remoção implicitamente considera a raiz como o vértice de maior rótulo dentre os vértices do centro da árvore. No terceiro código de Neville, a folha de maior rótulo não é escolhida para ser removida e, portanto, é considerada a raiz da árvore. No algoritmo de Deo e Micikevicius, considera-se que a raiz é o vértice v do centro da árvore cujo rótulo aparece na posição $n - 2$ da codificação. Isso ocorre porque a última subárvore restante é formada pela aresta $\{v, w\}$ e w já foi inserido na fila de codificação. Resumindo, pode-se dizer que, para árvores sem raiz, o algoritmo de codificação implicitamente define uma raiz assim:

- a) Prüfer – vértice de maior rótulo da árvore;
- b) 2º de Neville – vértice de maior rótulo do centro;
- c) 3º de Neville – folha de maior rótulo;
- d) Deo e Micikevicius – vértice do centro que ocupa a posição $n - 2$ do

código.

A abordagem genérica proposta em [8] utiliza árvores com raiz. Quando se deseja codificar uma árvore sem raiz, o algoritmo genérico de codificação assume a raiz que a codificação implicitamente define conforme visto nos itens a), b), c) e d) acima.

Seja T uma árvore com raiz r e dois vértices quaisquer u e v . Utiliza-se a seguinte notação:

T_v – subárvore de T com raiz em v ;

$dist(u,v)$ – distância entre os vértices u e v , ou seja, número de arestas entre u e v ;

$l(v)$ – distância máxima de v a uma folha de T_v ;

$\mu(v)$ – maior rótulo de T_v ;

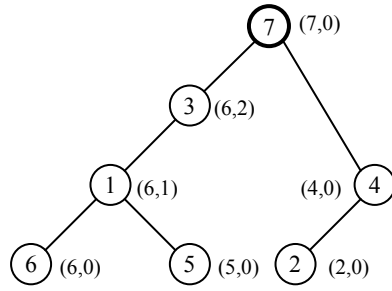
$\lambda(v)$ – maior rótulo entre as folhas de T_v ;

$\chi(v)$ – maior rótulo entre as folhas de T_v que estão a uma distância máxima de v .

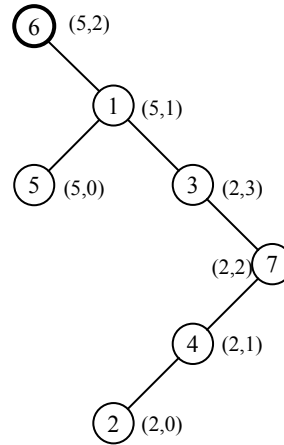
Os algoritmos genéricos de codificação e decodificação propostos utilizam um par de valores (x_v, y_v) para cada uma das quatro codificações como mostra a Tabela 2.2. Nesta tabela, as codificações de Neville estão propositalmente invertidas para facilitar a comparação delas com as outras duas codificações mostradas. Na Figura 2.14, a árvore da Figura 2.1(a) é mostrada com os respectivos pares calculados para os quatro códigos. O Algoritmo 2.3 executa a codificação genérica e tem complexidade $O(n)$, pois os pares (x_v, y_v) são facilmente calculados para cada um dos vértices utilizando um percurso em pós ordem na árvore. A ordenação destes pares é feita com a aplicação da ordenação por distribuição executada duas vezes.

Tabela 2.2: Pares (x_v, y_v) associados aos quatro códigos estudados.

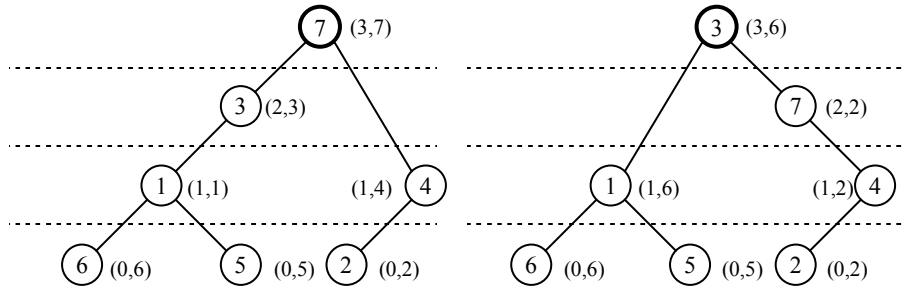
	Prüfer	3º Neville	2º Neville	Deo e Micikevicius
x_v	$\mu(v)$	$\lambda(v)$	$l(v)$	$l(v)$
y_v	$dist(\mu(v), v)$	$dist(\lambda(v), v)$	v	$\chi(v)$



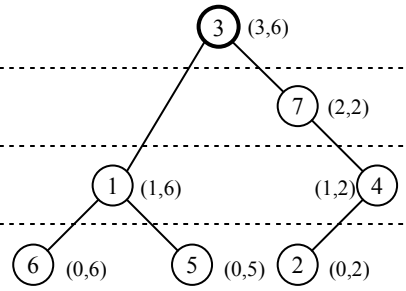
(a) Prüfer



(b) 3º Neville



(c) 2º Neville



(d) Deo e Micikevicius

Figura 2.14: Árvore da Figura 2.1(a) com pares (x_v, y_v) para as 4 codificações estudadas.

Algoritmo Codificação_Unificada;

Entrada: Árvore $T = (V, E)$;

Saída: Lista *Código* com o código de árvore obtido;

Início

Código $\leftarrow \emptyset$;

Para todo $v \in V$ faça

 Calcule (x_v, y_v) ;

 Ordene os pares (x_v, y_v) em ordem crescente de x_v e y_v ;

Para i de 1 até $n-1$ faça

 Adicione o ancestral do vértice da posição i ao *Código*;

Fim

Algoritmo 2.3: Codificação genérica de árvores.

2.5 - Conclusão

O código de Prüfer ainda é muito utilizado apesar de proposto há quase um século. Isso se justifica não somente pelo código ser bem compacto e bem simples, mas principalmente pela eficiência dos algoritmos de codificação e decodificação. O código também é utilizado em conjunto com técnicas mais modernas, como algoritmos genéticos e algoritmos paralelos. Por exemplo, a abordagem genérica [8] vista na Seção 2.4 e os algoritmos apresentados em [12] são utilizados em algoritmos paralelos para os quatro códigos mostrados neste capítulo.

A recente definição de um código por DEO e MICIKEVICIUS [11] e uma tese de doutorado de PICCIOTTO [39] de 1999 dedicada a codificações de árvores mostram como a representação de árvores utilizando codificações baseadas em Prüfer e outras codificações de árvore ainda são objeto de estudo.

No próximo capítulo, propomos uma codificação para os grafos k -caminho e fazemos um estudo similar ao feito com árvores. São apresentados algoritmos de codificação, decodificação e geração aleatória uniforme de grafos k -caminho. Além disso, são apresentadas propriedades destes grafos obtidas diretamente da sua codificação e é feita a contagem dos grafos k -caminho utilizando o código proposto.

Capítulo 3 - Codificação de Grafos k -caminho

No capítulo anterior, foi vista a importância de estabelecer uma codificação para uma família de grafos, quando se está interessado em contar os elementos desta família e conseqüentemente gerá-los de forma eficiente. Neste capítulo, propomos um novo código – que chamamos de **seqüência reduzida** – para os grafos k -caminho [31] que são uma subfamília dos grafos cordais. Ainda neste capítulo, propomos algoritmos com complexidade linear para validação, codificação e decodificação. Além de ser mais compacta do que a representação por arestas, a codificação proposta mostra-se imensamente vantajosa, pois permite: a contagem, a geração aleatória uniforme de modo eficiente e a obtenção de propriedades diretamente do código de um grafo k -caminho.

Na Seção 3.1, são apresentadas noções básicas da literatura relativas aos grafos k -caminho. Ainda na primeira seção, propomos uma nova definição recorrente para esta família. Na Seção 3.2, mostramos que uma seqüência $\langle C_1, C_2, \dots, C_p \rangle$, $p > 0$, de $(k+1)$ -cliques representa um grafo k -caminho. Na Seção 3.3, definimos a seqüência reduzida através da simplificação da seqüência de $(k+1)$ -cliques que representa o grafo. Em seguida, apresentamos algumas de suas propriedades e, utilizando-as, provamos que a seqüência reduzida é uma codificação para os grafos k -caminho. O código proposto tem a vantagem de ser bem compacto, pois sua complexidade de armazenamento é $O(n)$ como visto na Seção 3.4. Nas Seções 3.5, 3.6 e 3.7, propomos os algoritmos de validação, codificação e decodificação, respectivamente. O primeiro algoritmo tem complexidade $O(n)$ e os outros dois têm complexidade $O(m)$. Finalmente, na Seção 3.8, são feitas a contagem e a geração de grafos k -caminho. Os primeiros resultados obtidos para a seqüência reduzida encontram-se em [38].

3.1 - Noções Básicas

Nesta seção, definem-se k -caminhos e grafos k -caminho, que são uma subfamília das k -árvores. O primeiro foi definido por BEINEKE e PIPPERT [4] e o segundo por MARKENZON *et al.* [31]. Além disso, são mostrados alguns resultados relativos aos grafos k -caminho.

Cabe salientar que existem duas definições conhecidas na literatura sobre grafos caminho. Na primeira, que é utilizada nesta tese, os grafos caminho são grafos cujos vértices formam um único caminho. Esta família é extremamente simples. Na segunda, os grafos caminho, também conhecidos como grafos caminho sem direção, são grafos de interseção de caminhos em uma árvore. Esta segunda família contém os grafos de intervalo e são uma subfamília dos cordais. SHAFFER [44] e GAVRIL [18] propuseram algoritmos polinomiais para reconhecimento desta família e BABEL *et al.* [3] provaram que o problema de isomorfismo para esta família é isomorfismo completo.

Um **caminho** em um grafo $G = (V, E)$ é uma seqüência $\langle v_0, e_1, v_1, \dots, e_p, v_p \rangle$, $p > 0$, onde v_i são vértices, e_i são arestas e dois vértices sucessivos v_{i-1} e v_i são incidentes à aresta e_i para $1 \leq i \leq p$. Um k -caminho, definido a seguir, é uma generalização do conceito de caminho que, neste contexto, passa a ser notado 1-caminho.

Definição 3.1: Um k -caminho em um grafo $G = (V, E)$, $k > 0$, com **comprimento** $p > 0$ é uma seqüência $\langle B_0, C_1, B_1, \dots, C_p, B_p \rangle$ tal que:

- (i) $B_j \subset V$, $0 \leq j \leq p$, são k -cliques distintas de G ;
- (ii) $C_i \subseteq V$, $1 \leq i \leq p$, são $(k+1)$ -cliques distintas de G ;
- (iii) $B_{i-1} \subset C_i$, $B_i \subset C_i$ 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 , $1 \leq i \leq p$.

A Figura 3.1(b) mostra o 2-caminho $\langle \{1, 4\}, \{1, 3, 4\}, \{1, 3\}, \{1, 2, 3\}, \{1, 2\}, \{1, 2, 6\}, \{2, 6\} \rangle$ da 2-árvore vista na Figura 3.1(a). Este 2-caminho é maximal, pois não existe um 2-caminho que contenha todas as quatro cliques de tamanho 3 desta 2-árvore.

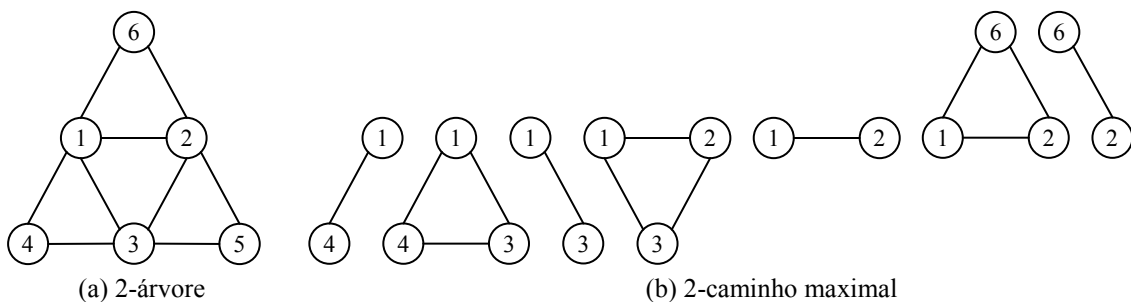


Figura 3.1: 2-árvore e um 2-caminho maximal.

Observe que a seqüência $\langle B_0, C_1, B_1, \dots, C_p, B_p \rangle$ e sua reversa $\langle B_p, C_p, B_{p-1}, \dots, C_1, B_0 \rangle$, para $p > 0$, são o mesmo k -caminho de G , pois as cliques C_i contêm exatamente as mesmas cliques B_{i-1} e B_i nas duas seqüências. Assim, o

2-caminho da Figura 3.1(b) também é descrito pela seqüência $\langle \{2, 6\}, \{1, 2, 6\}, \{1, 2\}, \{1, 2, 3\}, \{1, 3\}, \{1, 3, 4\}, \{1, 4\} \rangle$.

Pelo Teorema 1.3(c), uma k -árvore com n vértices possui $n - k$ cliques de tamanho $k + 1$. Logo, nenhum k -caminho em uma k -árvore pode conter mais do que $n - k$ cliques de tamanho $k + 1$, ou seja, o comprimento máximo de um k -caminho em uma k -árvore é $n - k$. Algumas k -árvores, como a da Figura 3.1(a), não possuem k -caminho de comprimento $n - k$. As k -árvores que possuem um k -caminho com todas as $n - k$ cliques de tamanho $k + 1$ definem uma subfamília chamada de grafos k -caminho. A definição a seguir trata desta subfamília.

Definição 3.2: Seja $G = (V, E)$ uma k -árvore com n vértices, sendo $n > k$. G é um **grafo k -caminho** quando existe em G um k -caminho de comprimento $n - k$.

A Figura 3.2 mostra a construção de uma 2-árvore com 6 vértices que é um grafo 2-caminho, pois existe o 2-caminho $\langle \{3, 4\}, \{1, 3, 4\}, \{1, 3\}, \{1, 2, 3\}, \{2, 3\}, \{2, 3, 5\}, \{2, 5\}, \{2, 5, 6\}, \{5, 6\} \rangle$ que contém todas as $n - k = 6 - 2 = 4$ cliques de tamanho 3 deste grafo. A Figura 3.3 ilustra este 2-caminho.

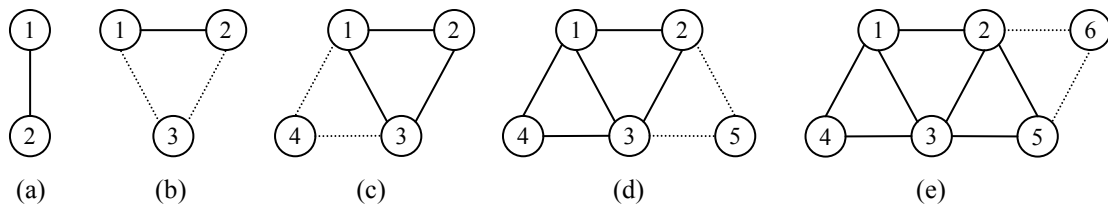


Figura 3.2: Exemplo de construção de 2-árvore que é grafo 2-caminho.

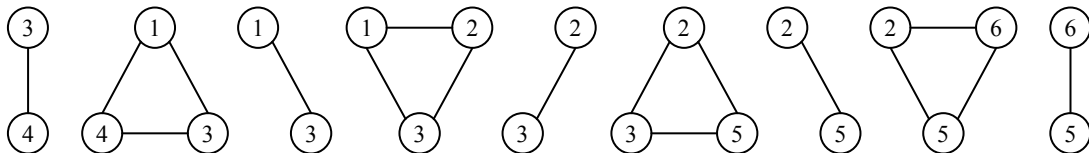


Figura 3.3: 2-caminho com todas as 3-cliques do grafo 2-caminho da Figura 3.2(e).

Observe que a definição de k -caminho não permite a existência de k -caminhos nulos, pois o k -caminho mínimo $\langle B_0, C_1, B_1 \rangle$ tem comprimento um. Como o menor k -caminho tem comprimento um, o menor grafo k -caminho é a menor k -árvore que contém uma clique de tamanho $k + 1$, ou seja, o grafo completo com $k + 1$ vértices.

Para os demais grafos k -caminho, pode ser aplicado o teorema de caracterização demonstrado em [31], que é visto a seguir.

Teorema 3.3 [31]: Seja G uma k -árvore com n vértices, sendo $n > k + 1$. G é um grafo k -caminho se e somente se G possui exatamente dois vértices simpliciais.

Este teorema mostra que os grafos k -caminho são as k -árvores com o mínimo possível de vértices simpliciais já que, pelo Lema 1.6, uma k -árvore tem no mínimo dois vértices simpliciais. Toda k -árvore com $k + 2$ vértices é um grafo k -caminho, pois sempre contém exatamente dois vértices simpliciais de acordo com o Lema 1.6.

Pelo Lema 1.5, os vértices simpliciais de uma k -árvore têm grau k ; logo apenas os dois vértices simpliciais de um grafo k -caminho possuem grau k . Utilizando este fato, o Teorema 3.3 e o algoritmo de reconhecimento de k -árvores proposto por JUSTEL e MARKENZON [25], obtém-se um algoritmo de reconhecimento dos grafos k -caminho.

Em seguida, com base no Teorema 3.3, propomos uma definição recorrente para os grafos k -caminho.

Definição 3.4: Um **grafo k -caminho** é assim definido:

- (1) Um grafo completo com $k + 1$ vértices é um grafo k -caminho;
- (2) Se $G = (V, E)$ é um grafo k -caminho, $Q \subset V$ é uma k -clique que contém pelo menos um vértice simplicial e $v \notin V$, o grafo $G' = (V \cup \{v\}, E \cup \{\{v, w\} \mid w \in Q\})$ é um grafo k -caminho;
- (3) Nada mais são grafos k -caminho.

Esta definição recorrente permite a construção de grafos k -caminho de modo bem simples: inicia-se com um grafo completo de $k + 1$ vértices e, a cada passo, adiciona-se um vértice adjacente aos vértices de uma k -clique que contenha pelo menos um vértice simplicial.

A construção da 2-árvore vista na Figura 3.2 ilustra esse procedimento. Inicia-se com o menor grafo 2-caminho que é o grafo completo de 3 vértices da Figura 3.2(b). Depois, adiciona-se o vértice de rótulo 4 adjacente aos vértices da 2-clique $\{1, 3\}$, deixando o grafo com os vértices simpliciais cujos rótulos são 2 e 4 como visto na Figura 3.2(c). Em seguida, na Figura 3.2(d), escolhe-se a 2-clique $\{2, 3\}$, que contém o vértice simplicial de rótulo 3, e, após a adição do vértice de rótulo 5, o grafo passa a ter

os vértices simpliciais cujos rótulos são 4 e 5. Finalmente, após a adição do vértice de rótulo 6, o grafo 2-caminho fica com os vértices simpliciais cujos rótulos são 4 e 6, pois o vértice de rótulo 5 deixa de ser simplicial após a escolha da clique $\{2,5\}$.

Analisando o 2-caminho da Figura 3.3, nota-se que o grafo da Figura 3.2(e) contém vários 2-caminhos maximais. Isso ocorre porque a 2-clique B_0 pode ser $\{1,4\}$ ou $\{3,4\}$ e a 2-clique B_4 pode ser $\{2,6\}$ ou $\{5,6\}$. Deste modo, existem quatro 2-caminhos maximais contidos no grafo 2-caminho da Figura 3.2(e). O corolário do Teorema 3.3 enunciado a seguir quantifica os k -caminhos de um grafo k -caminho.

Corolário 3.5 [31]: Seja G um grafo k -caminho com n vértices, sendo $n > k + 1$ vértices. A quantidade de k -caminhos maximais em G é k^2 .

3.2 - Seqüência de $(k+1)$ -cliques

Seja $G = (V,E)$ um grafo k -caminho. Pela Definição 3.2, existe um k -caminho maximal $\langle B_0, C_1, B_1, \dots, C_p, B_p \rangle$, $p > 0$, que contém todas as $p = n - k$ cliques de tamanho $k + 1$ de G . Este k -caminho maximal não é a melhor forma de representar o grafo k -caminho, pois ele não é único. Por outro lado, a seqüência $\langle C_1, C_2, \dots, C_p \rangle$, $p > 0$, de $(k+1)$ -cliques obtida pela exclusão das k -cliques de um k -caminho maximal representa um grafo k -caminho de modo único como mostra o Teorema 3.6 a seguir.

Observe que, pela Definição 3.1 de k -caminho, uma k -clique B_i está contida nas $(k+1)$ -cliques C_i e C_{i+1} . Assim, as k -cliques B_i internas, ou seja, para $1 \leq i < p$, são obtidas através da expressão $B_i = C_i \cap C_{i+1}$. Deste ponto em diante, já que todos os k -caminhos utilizados são maximais e representam grafos k -caminho, será considerado que $p = n - k$.

Teorema 3.6: Sejam G um grafo k -caminho e $\langle B_0, C_1, B_1, \dots, C_p, B_p \rangle$, $p > 0$, um k -caminho maximal de G . $T = (\{C_1, \dots, C_p\}, \{\{C_i, C_{i+1}\} \mid 1 \leq i < p\})$ é a única árvore de cliques de G .

Prova: Sabe-se que as cliques maximais de uma k -árvore possuem cardinalidade $k + 1$. Como $C_i \cap C_{i+1} = B_i$, tem-se $|C_i \cap C_{i+1}| = k$, que é a cardinalidade máxima da interseção entre duas cliques maximais de G . Desse modo, T é uma árvore geradora

de peso máximo do grafo de interseção de cliques de G , ou seja, T é uma árvore de cliques de G .

Pela propriedade da interseção das árvores de cliques, a interseção $C_i \cap C_a$ é um subconjunto de todas as cliques que estiverem no caminho entre C_i e C_a na árvore de cliques. Logo, para $a > i + 1$, $C_i \cap C_a \subset C_{i+1}$ e $C_i \cap C_a \subset C_{a-1}$. Pela Definição 3.1, $B_i \subset C_j$, $1 \leq j \leq p$, apenas para $i = j - 1$ e $i = j$. Assim, $C_i \cap C_{i+1} = B_i$, $C_{a-1} \cap C_a = B_{a-1}$ e $B_{a-1} \neq B_i$. Desse modo, $C_i \cap C_a \subset B_i$, ou seja, $|C_i \cap C_a| < k$. Portanto, apenas as interseções de cliques maximais consecutivas no k -caminho maximal possuem cardinalidade k . Logo, T é única. \Rightarrow

Um k -caminho G pode ser representado pela única árvore de cliques T que possui, ou seja, a árvore de cliques é uma codificação para os grafos k -caminho. A árvore de cliques de G pode ser obtida através da seqüência $\langle C_1, C_2, \dots, C_p \rangle$ ou da seqüência $\langle C_p, C_{p-1}, \dots, C_1 \rangle$ de um caminho maximal de G . Desse modo, estas seqüências também representam G .

Apesar de serem uma codificação para os grafos k -caminho, a seqüência $\langle C_1, C_2, \dots, C_p \rangle$, $p > 0$, de $(k+1)$ -cliques e sua reversa não oferecem vantagens em relação à representação tradicional de grafos. Na próxima seção, definimos a seqüência reduzida de um grafo k -caminho que é uma codificação para esta família. A seqüência reduzida possui uma representação compacta e um algoritmo eficiente de validação. Esta seqüência é utilizada na contagem e na geração aleatória uniforme de grafos k -caminho rotulados.

3.3 - Seqüência Reduzida

Nesta seção, definimos a seqüência reduzida de um grafo k -caminho através da simplificação da seqüência $\langle C_1, C_2, \dots, C_p \rangle$, $p > 0$, de $(k+1)$ -cliques e provamos que a seqüência reduzida é uma codificação para esta família. Apesar de serem omitidas as k -cliques B_i da seqüência, pois $B_i = C_i \cap C_{i+1}$, estas k -cliques garantem que duas $(k+1)$ -cliques consecutivas C_i e C_{i+1} têm apenas um vértice distinto em cada uma, pois têm k elementos em comum. A Definição 3.7 trata destes vértices, e, em seguida, define-se a seqüência reduzida de um grafo k -caminho. Após a apresentação de alguns

resultados pertinentes a esta seqüência, provamos que ela é uma codificação para os grafos k -caminho.

As $(k+1)$ -cliques C_i e C_{i+1} têm k elementos em comum, pois a k -clique B_i está contida em C_i e C_{i+1} . Logo, $|C_i - C_{i+1}| = |C_{i+1} - C_i| = 1$. Como todas as $(k+1)$ -cliques são distintas, conclui-se que $C_i - C_{i+1} \neq C_{i+1} - C_i$. Assim, existe um único vértice l_i que está em C_i , mas não está em C_{i+1} e um único vértice r_i que está em C_{i+1} , mas não está em C_i . Estas interseções são válidas para $1 \leq i < p$, ou seja, não existem para $i = p$. A definição a seguir trata dos conjuntos unitários formados pelos vértices l_i e r_i .

Definição 3.7: Seja $\langle C_1, C_2, \dots, C_p \rangle$, $p > 1$, a seqüência de $(k+1)$ -cliques que representa um grafo k -caminho. Para $1 \leq i < p$, $\{l_i\} = C_i - C_{i+1}$ e $\{r_i\} = C_{i+1} - C_i$.

Cabe salientar que os vértices l_i e r_i só são definidos quando existem pelo menos duas cliques na seqüência $\langle C_1, C_2, \dots, C_p \rangle$ de $(k+1)$ -cliques. Por isso, a seqüência de $(k+1)$ -cliques que permite analisar os vértices l_i e r_i deve ter comprimento mínimo dois, ou seja, $p > 1$.

Definição 3.8: Seja $\langle C_1, C_2, \dots, C_p \rangle$, $p \geq 1$, a seqüência de $(k+1)$ -cliques que representa um grafo k -caminho. Denomina-se $S = \langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$ a **seqüência reduzida** deste grafo. Quando $p = 1$, tem-se $S = \langle \rangle$.

$$\begin{aligned} \{l_i\} = C_i - C_{i+1} &\Rightarrow l_1 = b & l_2 = e & l_3 = a & l_4 = g & l_5 = d \\ \{r_i\} = C_{i+1} - C_i &\Rightarrow r_1 = h & r_2 = g & r_3 = f & r_4 = i & r_5 = j \\ B_i = C_i \cap C_{i+1} &\Rightarrow B_1 = \{a, c, d, e\} & B_2 = \{a, c, d, h\} & B_3 = \{c, d, g, h\} & B_4 = \{c, d, f, h\} & B_5 = \{c, f, h, i\} \end{aligned}$$

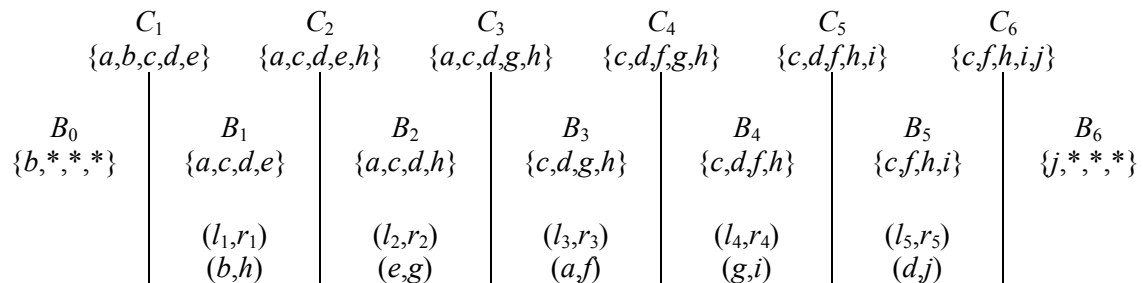


Figura 3.4: Representação esquemática de grafo 4-caminho e sua seqüência reduzida.

Seja, por exemplo, um grafo 4-caminho com 10 vértices representado pela seqüência de 5-cliques $\langle \{a, b, c, d, e\}, \{a, c, d, e, h\}, \{a, c, d, g, h\}, \{c, d, f, g, h\}, \{c, d, f, h, i\},$

$\{c,f,h,i,j\}$ que tem $p = 6$. A Figura 3.4 mostra como obter a seqüência reduzida $S = \langle (b,h), (e,g), (a,f), (g,i), (d,j) \rangle$ deste grafo a partir de sua seqüência de 5-cliques.

Na Figura 3.4, observa-se que $l_2 \in C_2$ e $l_2 \notin B_2$, por definição; assim, $l_2 \notin C_3$. De modo análogo, verifica-se que $r_2 \notin C_2$, pois, da definição, tem-se que $r_2 \notin B_2$ e $r_2 \in C_3$. Essas e outras propriedades obtidas da definição de l_i e r_i são mostradas no Lema 3.9.

Lema 3.9: Sejam $\langle C_1, C_2, \dots, C_p \rangle$, $p > 1$, a seqüência de $(k+1)$ -cliques que representa um grafo k -caminho e $S = \langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$ a sua seqüência reduzida. São válidas as seguintes propriedades:

- a) $l_i \in C_i$ e $r_i \notin C_i$, para $1 \leq i < p$;
- b) $l_i \notin C_{i+1}$ e $r_i \in C_{i+1}$, para $1 \leq i < p$;
- c) $B_i = C_i - \{l_i\} = C_{i+1} - \{r_i\}$, para $1 \leq i < p$;
- d) $C_i = \{l_i\} \cup C_{i+1} - \{r_i\}$ e $C_{i+1} = \{r_i\} \cup C_i - \{l_i\}$, para $1 \leq i < p$;
- e) $r_{i-1} \neq l_i$ para $1 < i < p$.

Prova: Pela Definição 3.7, para $1 \leq i < p$, $\{l_i\} = C_i - C_{i+1}$, logo $l_i \in C_i$ e $l_i \notin C_{i+1}$. Também pela Definição 3.7, tem-se que $\{r_i\} = C_{i+1} - C_i$, logo $r_i \in C_{i+1}$ e $r_i \notin C_i$, para $1 \leq i < p$.

Como todos os elementos de C_i e C_{i+1} são iguais exceto l_i e r_i , então $C_i \cap C_{i+1} = C_i - \{l_i\}$ e $C_i \cap C_{i+1} = C_{i+1} - \{r_i\}$, ou seja, $B_i = C_i - \{l_i\} = C_{i+1} - \{r_i\}$. Desse modo, para $1 \leq i < p$, obtêm-se as seguintes expressões: $C_i = \{l_i\} \cup C_{i+1} - \{r_i\}$ e $C_{i+1} = \{r_i\} \cup C_i - \{l_i\}$.

Todas as $(k+1)$ -cliques C_i são distintas, logo $C_{i-1} \cap C_i \neq C_i \cap C_{i+1}$, para $1 < i < p$. Como $C_{i-1} \cap C_i = C_i - \{r_{i-1}\}$ e $C_i \cap C_{i+1} = C_i - \{l_i\}$, então $C_i - \{r_{i-1}\} \neq C_i - \{l_i\}$ e, portanto, $r_{i-1} \neq l_i$. \Rightarrow

Na Seção 3.2, já foi visto que a seqüência de $(k+1)$ -cliques $\langle C_1, C_2, \dots, C_p \rangle$ e sua reversa $\langle C_p, C_{p-1}, \dots, C_1 \rangle$, $p > 0$, representam o mesmo grafo k -caminho. O próximo teorema mostra que a reversa de uma seqüência reduzida também é seqüência reduzida do mesmo grafo k -caminho.

Teorema 3.10: Seja $\langle C_1, C_2, \dots, C_p \rangle$, $p > 1$, a seqüência de $(k+1)$ -cliques que representa um grafo k -caminho. Apenas as seqüências $S = \langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$ e sua reversa $S' = \langle (r_{p-1}, l_{p-1}), \dots, (r_2, l_2), (r_1, l_1) \rangle$ são seqüências reduzidas deste grafo k -caminho.

Prova: Por definição, r_i e l_i são únicos entre cada par de $(k+1)$ -cliques, logo S é única para $\langle C_1, C_2, \dots, C_p \rangle$. Entretanto, como a seqüência de $(k+1)$ -cliques $\langle C_p, C_{p-1}, \dots, C_1 \rangle$ também representa este grafo k -caminho, então existe outra seqüência reduzida deste grafo. Utilizando a transformação $C_i' = C_{p-i+1}$, obtém-se a seqüência de $(k+1)$ -cliques $\langle C_1', C_2', \dots, C_p' \rangle$ que tem a seqüência reduzida $S' = \langle (l_1', r_1'), \dots, (l_{p-1}', r_{p-1}') \rangle$. Desse modo, $l_i' = C_i' - C_{i+1}'$ e $r_i' = C_{i+1}' - C_i'$, ou seja, $l_i' = C_{p-i+1} - C_{p-i} = r_{p-i}$ e $r_i' = C_{p-i} - C_{p-i+1} = l_{p-i}$. Logo, a seqüência $S' = \langle (l_1', r_1'), \dots, (l_{p-1}', r_{p-1}') \rangle = \langle (r_{p-1}, l_{p-1}), \dots, (r_1, l_1) \rangle$. Desse modo, S e S' são seqüências reduzidas do mesmo grafo k -caminho. \Rightarrow

Por exemplo, a seqüência $S' = \langle (j,d), (i,g), (f,a), (g,e), (h,b) \rangle$, obtida pela reversão de $S = \langle (b,h), (e,g), (a,f), (g,i), (d,j) \rangle$, também é seqüência reduzida do grafo 4-caminho mostrado na Figura 3.4.

No Teorema 3.3, foi visto que os grafos k -caminho com n vértices, sendo $n > k + 1$, são k -árvores que contêm exatamente dois vértices simpliciais. A seqüência reduzida permite a obtenção dos dois vértices simpliciais do grafo k -caminho de modo imediato, como visto no lema a seguir.

Lema 3.11: Seja $S = \langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$, $p > 1$, a seqüência reduzida de um grafo k -caminho. Os vértices l_1 e r_{p-1} são os dois vértices simpliciais desse grafo.

Prova: Sabe-se do Teorema 3.3 que os grafos k -caminho têm exatamente dois vértices simpliciais e do Lema 1.5 que os vértices simpliciais pertencem a apenas uma $(k+1)$ -clique. Pela Definição 3.7, $\{l_1\} = C_1 - C_2$ e $\{r_{p-1}\} = C_p - C_{p-1}$. Como l_1 e r_{p-1} pertencem a apenas uma $(k+1)$ -clique, então l_1 e r_{p-1} são os dois vértices simpliciais do grafo k -caminho. \Rightarrow

Ao remover um vértice simplicial de um grafo k -caminho e as k arestas incidentes a este vértice, obtém-se um grafo k -caminho. Por exemplo, na Figura 3.4, a remoção do vértice b e das k arestas incidentes a ele gera um grafo k -caminho cujos vértices

simpliciais são e e j . Os dois lemas a seguir demonstram que, em um grafo k -caminho, a remoção de um vértice simplicial e de suas arestas incidentes gera um grafo k -caminho.

Lema 3.12: Sejam $\langle C_1, C_2, \dots, C_p \rangle$, $p > 1$, a seqüência de $(k+1)$ -cliques que representa um grafo k -caminho e $S = \langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$ sua seqüência reduzida. A remoção do vértice simplicial l_1 e das k arestas incidentes a ele gera um grafo k -caminho G' cuja representação é $\langle C_2, \dots, C_p \rangle$ e cuja seqüência reduzida é $\langle (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$.

Prova: Pela definição de k -árvore, o grafo G' obtido pela remoção de um vértice simplicial e das k arestas incidentes a ele é uma k -árvore. Ao serem removidos o vértice l_1 e as k arestas incidentes a ele, a clique C_1 deixa de existir no grafo k -caminho G' , que é representado por $\langle C_2, \dots, C_p \rangle$. Devido ao Lema 3.11, l_2 será simplicial em G' . \Rightarrow

Lema 3.13: Sejam $\langle C_1, \dots, C_{p-1}, C_p \rangle$, $p > 1$, a seqüência de $(k+1)$ -cliques que representa um grafo k -caminho e $S = \langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$ sua seqüência reduzida. A remoção do vértice simplicial r_{p-1} e das k arestas incidentes a ele gera um grafo k -caminho G' cuja representação é $\langle C_1, \dots, C_{p-1} \rangle$ e cuja seqüência reduzida é $\langle (l_1, r_1), \dots, (l_{p-2}, r_{p-2}) \rangle$.

Prova: Análoga à prova do Lema 3.12. \Rightarrow

Observe que, de acordo com a prova do Lema 3.12, os vértices l_1, l_2, \dots, l_{p-1} podem ser removidos sucessivamente nesta ordem e os subgrafos obtidos após cada remoção são grafos k -caminho. Um procedimento análogo pode ser feito com os vértices $r_{p-1}, r_{p-2}, \dots, r_1$. O próximo lema utiliza estas remoções sucessivas para provar que estes vértices são distintos.

Lema 3.14: Sejam $\langle C_1, \dots, C_p \rangle$, $p > 1$, a seqüência de $(k+1)$ -cliques que representa um grafo k -caminho e $S = \langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$ sua seqüência reduzida. Tem-se:

- a) $l_1 \neq l_2 \neq \dots \neq l_{p-1}$;
- b) $r_1 \neq r_2 \neq \dots \neq r_{p-1}$.

Prova: Pelo Lema 3.12, os vértices l_i podem ser removidos sucessivamente do grafo k -caminho, logo são distintos. De modo análogo, com base no Lema 3.13, os vértices r_i são removidos do grafo e, portanto, são distintos. \Rightarrow

A seguir, são definidos dois conjuntos para uma seqüência qualquer de pares $S = \langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$, $p > 1$. Estes conjuntos são utilizados na validação de uma seqüência S qualquer, ou seja, são utilizados para verificar se S é uma seqüência reduzida de um grafo k -caminho. Estes conjuntos também são utilizados na obtenção das $(k+1)$ -cliques que representam o grafo k -caminho.

Definição 3.15: Seja $S = \langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$, $p > 1$, uma seqüência de pares de vértices. Definem-se os conjuntos L_i e R_i assim:

$$L_i = \{l_a \mid 0 < a \leq i\}, \text{ para } 0 \leq i < p;$$

$$R_i = \{r_a \mid i \leq a < p\}, \text{ para } 0 < i \leq p.$$

Desse modo, pela Definição 3.15, $L_0 = \emptyset$, $L_1 = \{l_1\}$, $R_{p-1} = \{r_{p-1}\}$ e $R_p = \emptyset$. Além disso, verifica-se que L_p e R_0 não estão definidos. Como L_{p-1} contém todos os vértices l_i e R_1 contém todos os vértices r_i , denota-se $L = L_{p-1}$ e $R = R_1$. Também pela Definição 3.15, observa-se que $L_{i+1} = L_i \cup \{l_{i+1}\}$ e $R_{i-1} = R_i \cup \{r_{i-1}\}$.

Quando S é a seqüência reduzida de um grafo k -caminho, $L_i = \{l_1, \dots, l_i\}$, pois os vértices l_i são distintos pelo Lema 3.14(a). De modo análogo, $R_i = \{r_i, \dots, r_{p-1}\}$ pelo Lema 3.14(b). Assim, tem-se que $|L_i| = i$ e $|R_i| = p - i$.

A Figura 3.5 mostra os conjuntos L_i e R_i para o grafo 4-caminho com 10 vértices da Figura 3.4 cuja seqüência reduzida é $S = \langle (b, h), (e, g), (a, f), (g, i), (d, j) \rangle$ e $p = 6$. Observe que as cardinalidades dos conjuntos são obtidas pelas expressões $|L_i| = i$ e $|R_i| = 6 - i$.

$L_0 = \emptyset$	$R_1 = \{h, g, f, i, j\}$
$L_1 = \{b\}$	$R_2 = \{g, f, i, j\}$
$L_2 = \{b, e\}$	$R_3 = \{f, i, j\}$
$L_3 = \{b, e, a\}$	$R_4 = \{i, j\}$
$L_4 = \{b, e, a, g\}$	$R_5 = \{j\}$
$L_5 = \{b, e, a, g, d\}$	$R_6 = \emptyset$

Figura 3.5: Conjuntos L_i e R_i para o grafo 4-caminho da Figura 3.4.

O subgrafo obtido a partir da remoção dos vértices de L_i e R_j , com $i < j$, é um grafo k -caminho. Por exemplo, na Figura 3.4, o subgrafo obtido pela remoção dos vértices de $L_2 = \{b, e\}$ e $R_4 = \{i, j\}$ é um grafo k -caminho. Quando são removidos os vértices de L_{i-1} e R_i , obtém-se um grafo k -caminho formado pelos vértices da $(k+1)$ -clique C_i . Por exemplo, na Figura 3.4, ao remover os vértices de $L_3 = \{b, e, a\}$ e $R_4 = \{i, j\}$, obtém-se um

grafo completo com os vértices da 5-clique $C_4 = \{c,d,f,g,h\}$. O Teorema 3.16 formaliza este resultado.

Teorema 3.16: Seja $S = \langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$, $p > 1$, a seqüência reduzida de um grafo k -caminho. Para $1 \leq i \leq p$, as seguintes propriedades são observadas:

- a) $C_i = V - (L_{i-1} \cup R_i)$;
- b) $|L_{i-1} \cup R_i| = p - 1$;
- c) $C_i \cup L_{i-1} \cup R_i = V$;
- d) $C_i \cap L_{i-1} \cap R_i = \emptyset$ ou $C_i \cap (L_{i-1} \cup R_i) = \emptyset$.

Prova: Aplicando-se $i - 1$ vezes o Lema 3.12 ao grafo G , removem-se os vértices de L_{i-1} deste grafo, obtendo-se o grafo G' representado por $\langle C_i, C_{i+1}, \dots, C_p \rangle$. Em seguida, aplicando-se $p - i$ vezes o Lema 3.13, os vértices de R_i são removidos do grafo G' , obtendo-se o grafo representado por $\langle C_i \rangle$. Desse modo, conclui-se que $C_i = V - (L_{i-1} \cup R_i)$. Como foram removidos do grafo os $i - 1$ vértices de L_{i-1} e os $p - i$ vértices de R_i , então estes vértices são distintos. Assim, $|L_{i-1} \cup R_i| = p - i + i - 1 = p - 1$.

Os vértices que sobraram acrescidos dos vértices removidos formam o conjunto inicial de vértices, ou seja, $C_i \cup L_{i-1} \cup R_i = V$. Além disso, os vértices restantes são distintos dos vértices que foram excluídos, logo $C_i \cap L_{i-1} \cap R_i = \emptyset$. \Rightarrow

A propriedade (a) do Teorema 3.16 mostra como obter as $(k+1)$ -cliques do grafo k -caminho a partir da seqüência reduzida. Conhecendo as $(k+1)$ -cliques, é possível obter as k -cliques B_i através da interseção $B_i = C_i \cap C_{i+1}$. Entretanto, as k -cliques B_i também podem ser obtidas diretamente da seqüência reduzida como mostra o corolário a seguir.

Corolário 3.17: Seja $S = \langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$, $p > 1$, a seqüência reduzida de um grafo k -caminho. Tem-se: $B_i = V - (L_i \cup R_i)$, para $1 \leq i < p$.

Prova: Sabe-se do Lema 3.9(c) que $B_i = C_i - \{l_i\}$ e do teorema anterior que $C_i = V - (L_{i-1} \cup R_i)$. Logo, conclui-se que $B_i = (V - (L_{i-1} \cup R_i)) - \{l_i\} = V - (L_{i-1} \cup R_i \cup \{l_i\})$. Como $L_i = L_{i-1} \cup \{l_i\}$, então $B_i = V - (L_i \cup R_i)$. \Rightarrow

O Teorema 3.10 prova que um grafo k -caminho possui apenas a seqüência reduzida além de sua reversa. O Teorema 3.16(a) mostra como obter as $(k+1)$ -cliques

que representam o grafo k -caminho a partir de uma seqüência reduzida e do conjunto de vértices. Logo, para um conjunto de vértices dado, existe uma relação biunívoca entre os grafos k -caminho e as seqüências reduzidas, ou seja, uma seqüência reduzida é uma codificação para grafos k -caminho.

3.4 - Complexidade de Espaço da Codificação

Na representação de um grafo k -caminho, a seqüência reduzida é composta por $p - 1 = n - k - 1$ pares de vértices e, portanto ocupa espaço de $O(n)$, enquanto a representação por lista de adjacências ocupa $O(m)$. Como um grafo k -caminho tem $k(k - 1)/2 + k(n - k)$ arestas e a seqüência reduzida tem $2(n - k - 1)$ vértices, conclui-se que a codificação proposta é mais compacta do que a lista de adjacências.

Observe que para caminhos simples é mais apropriado utilizar uma seqüência com n vértices para representar o grafo. Já para grafos k -caminho, $k > 2$, a utilização da seqüência reduzida é a representação mais compacta, pois possui complexidade $O(n)$ enquanto a representação por listas de adjacência possui complexidade $O(m)$.

Além de mais eficiente no armazenamento, a utilização da seqüência reduzida permite a geração aleatória de grafos k -caminho rotulados com complexidade $O(n)$. Obviamente, nem toda seqüência de vértices $S = \langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$, $p > 1$, é uma seqüência reduzida. A próxima seção mostra como verificar se S é uma seqüência reduzida.

3.5 - Validação de uma Seqüência Reduzida

A validação de uma seqüência de pares de vértices $S = \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$ consiste em verificar se S representa um grafo k -caminho, ou seja, se S é seqüência reduzida de um grafo k -caminho. Nesta seção, são mostrados o Teorema 3.18 que possibilita validar uma seqüência de pares de vértices $S = \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$ e o algoritmo de validação com base neste teorema.

Observe que, para $p = 2$, $S = \langle (l_1, r_1) \rangle$ é a seqüência reduzida de um grafo k -caminho G com $k + 2$ vértices e que l_1 e r_1 são distintos por serem os dois vértices simpliciais de G . Assim, qualquer seqüência $S = \langle (l_1, r_1) \rangle$ cujos vértices são distintos representa um grafo k -caminho com $k + 2$ vértices. O próximo teorema mostra como validar seqüências com $p > 2$.

Teorema 3.18: Para $p > 2$, $S = \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$ é a seqüência reduzida de um grafo k -caminho se e somente se $|L_i \cup R_{i-1}| = p + 1$ para $1 < i < p$.

Prova: (\Rightarrow) Infere-se da Definição 3.15 que $L_i = \{l_i\} \cup L_{i-1}$ e $R_{i-1} = \{r_{i-1}\} \cup R_i$. Logo, $L_i \cup R_{i-1} = L_{i-1} \cup R_i \cup \{l_i\} \cup \{r_{i-1}\}$ (1).

Sabe-se que $C_i \cap (L_{i-1} \cup R_i) = \emptyset$, devido ao Teorema 3.16(d) e que $r_{i-1}, l_i \in C_i$, para $1 < i < p$, devido aos itens (a) e (b) do Lema 3.9. Logo, $l_i, r_{i-1} \notin (L_{i-1} \cup R_i)$ (2). De (1) e (2), conclui-se que $|L_i \cup R_{i-1}| = |L_{i-1} \cup R_i| + |\{l_i\} \cup \{r_{i-1}\}|$ (3).

Devido ao Lema 3.9(e), os vértices r_{i-1} e l_i são distintos para $1 < i < p$. Assim, $|\{l_i\} \cup \{r_{i-1}\}| = 2$ (4). Pelo Teorema 3.16(b), sabe-se que $|L_{i-1} \cup R_i| = p - 1$ (5). Substituindo (4) e (5) na expressão (3), obtém-se $|L_i \cup R_{i-1}| = p - 1 + 2 = p + 1$.

(\Leftarrow) Seja V um conjunto qualquer tal que $|V| = n$ e $L \cup R \subseteq V$. A prova é por indução na cardinalidade de S .

- Para $S = \langle (l_1, r_1), (l_2, r_2) \rangle$, tem-se $p = 3$ e $|L_i \cup R_{i-1}| = 3 + 1$ para $1 < i < 3$. Logo, $|L_2 \cup R_1| = 4$, ou seja, $l_1 \neq l_2 \neq r_1 \neq r_2$. Assim, G consiste de três $(k+1)$ -cliques maximais $C_1 = V - \{r_1, r_2\}$, $C_2 = V - \{l_1, r_2\}$ e $C_3 = V - \{l_1, l_2\}$.
- Suponha que o teorema vale para $|S| = p - 2$.
- Seja $S = \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$ tal que $|L_i \cup R_{i-1}| = p + 1$ para $1 < i < p$. Para $i = 2$ e para $i = p - 1$, tem-se que $l_1 \neq l_2 \neq r_1 \neq \dots \neq r_{p-1}$ e $l_1 \neq \dots \neq l_{p-1} \neq r_{p-2} \neq r_{p-1}$. Assim, r_{p-1} não ocorre em $S' = \langle (l_1, r_1), \dots, (l_{p-2}, r_{p-2}) \rangle$. Pela hipótese de indução, existe uma k -árvore $G' = (V', E')$ tal que $V' = V - \{r_{p-1}\}$ e S' é seqüência reduzida de G' . Seja, $\langle C_1, C_2, \dots, C_{p-1} \rangle$ a seqüência de $(k+1)$ -cliques de G' . Por definição, $l_1 \in C_1$ e $r_{p-2} \in C_{p-1}$ e, pelo Lema 3.11, estes dois vértices são simpliciais em G' .

Pelo Teorema 3.16(a), $C_{p-1} = V' - \{l_1, \dots, l_{p-2}\}$. Como $V' = V - \{r_{p-1}\}$ e $l_{p-1} \in V$, então $l_{p-1} \in V'$. Já foi visto que $l_1 \neq \dots \neq l_{p-2} \neq l_{p-1}$; logo, $l_{p-1} \in C_{p-1}$. Seja o grafo

$G = (V, E' \cup \{\{r_{p-1}, v\} \mid v \in C_{p-1} - \{l_{p-1}\}\})$. Obviamente G é uma k -árvore e r_{p-1} é um vértice simplicial. Observe que o vértice r_{p-2} não é simplicial em G , pois seus vizinhos l_{p-1} e r_{p-1} não são adjacentes. Desse modo, G possui apenas dois vértices simpliciais: l_1 e r_{p-1} . Logo, S é a seqüência reduzida do grafo k -caminho G cuja seqüência de $(k+1)$ -cliques é $\langle C_1, \dots, C_{p-1}, \{r_{p-1}\} \cup C_{p-1} - \{l_{p-1}\} \rangle$. \Rightarrow

Para validar uma seqüência $S = \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$, basta verificar se esta seqüência satisfaz o Teorema 3.18. Por exemplo, a Figura 3.6 mostra o processo de validação da seqüência reduzida $S = \langle (b, h), (e, g), (a, f), (g, i), (d, j) \rangle$ que representa o grafo 4-caminho de 10 vértices e $p = 6$ visto na Figura 3.4. Neste caso, basta verificar que $|L_i \cup R_{i-1}| = 7$ para $1 < i < 6$.

$$\begin{aligned} L_2 \cup R_1 &= \{b, e\} \cup \{h, g, f, i, j\} = \{b, e, h, g, f, i, j\} \Rightarrow |L_2 \cup R_1| = 7 \\ L_3 \cup R_2 &= \{b, e, a\} \cup \{g, f, i, j\} = \{b, e, a, g, f, i, j\} \Rightarrow |L_3 \cup R_2| = 7 \\ L_4 \cup R_3 &= \{b, e, a, g\} \cup \{f, i, j\} = \{b, e, a, g, f, i, j\} \Rightarrow |L_4 \cup R_3| = 7 \\ L_5 \cup R_4 &= \{b, e, a, g, d\} \cup \{i, j\} = \{b, e, a, g, d, i, j\} \Rightarrow |L_5 \cup R_4| = 7 \end{aligned}$$

Figura 3.6: Validação da seqüência reduzida do grafo 4-caminho da Figura 3.4.

Como visto na prova do Teorema 3.18, quando $|L_i \cup R_{i-1}| = p + 1$, tem-se que $l_1 \neq \dots \neq l_i \neq r_{i-1} \neq \dots \neq r_{p-1}$. Deste modo, a validação resume-se em verificar se os vértices $l_1, \dots, l_i, r_{i-1}, \dots, r_{p-1}$ são distintos para $1 < i < p$. O Algoritmo 3.1 de validação utiliza este conceito, ou seja, testa se os vértices $l_1, \dots, l_i, r_{i-1}, \dots, r_{p-1}$ são distintos para $i = 2, \dots, p - 1$.

O Algoritmo 3.1 pressupõe $V = \{1, \dots, n\}$, $n > k + 2$, e utiliza como estrutura auxiliar o vetor LR para representar o conjunto $L_i \cup R_{i-1}$ atribuindo a $LR[v]$ a quantidade de vezes que o vértice v aparece na seqüência $\langle l_1, \dots, l_i, r_{i-1}, \dots, r_{p-1} \rangle$. Quando $|L_i \cup R_{i-1}| < p + 1$, esta seqüência tem elementos repetidos e $LR[v] > 1$ para pelo menos um vértice v . Desse modo, para $i = 2, \dots, p - 1$, o algoritmo obtém o conjunto $L_i \cup R_{i-1}$ e verifica se todos os vértices v têm $LR[v] \leq 1$. Caso isto seja verdade, a seqüência dada representa um grafo k -caminho e o algoritmo retorna verdadeiro.

Inicialmente $i = 2$ e o vetor LR é utilizado para encontrar $L_2 \cup R_1$. Assim, o vetor LR é preenchido para as posições dos $p + 1$ vértices $l_1, l_2, r_1, \dots, r_{p-1}$. O algoritmo é

interrompido caso estes vértices não sejam distintos, pois a condição $|L_2 \cup R_1| = p + 1$ não foi satisfeita. Para $i = 3, \dots, p - 1$, preenche-se o vetor LR que representa o conjunto $L_i \cup R_{i-1}$ verificando se $LR[v] \leq 1$.

Observe que o vetor LR não precisa ser recalculado para cada iteração. Basta remover o vértice r_{i-2} e adicionar o vértice l_i ao vetor LR . Este procedimento é suficiente para calcular $L_i \cup R_{i-1}$, pois, no passo anterior, o vetor LR representava o conjunto $L_{i-1} \cup R_{i-2}$ e sabe-se que $L_i = L_{i-1} \cup \{l_i\}$ e $R_{i-1} = R_{i-2} - \{r_{i-2}\}$.

```

Algoritmo Valida_Seqüência;
Entrada:   $n, k, \langle (l_1, r_1), \dots, (l_{n-k-1}, r_{n-k-1}) \rangle$ ;
Saída:   verdadeiro ou falso;
Início
  Para  $j$  de 1 até  $n$  faça  $LR[j] \leftarrow 0$ ;
  Se  $l_1 = l_2$  então retorne falso;
   $LR[l_1] \leftarrow 1$ ;
   $LR[l_2] \leftarrow 1$ ;
  Para  $j$  de 1 até  $n-k-1$  faça
    Se  $LR[r_j] \neq 0$  então retorne falso;
     $LR[r_j] \leftarrow 1$ ;
  Para  $i$  de 3 até  $n-k-1$  faça
     $LR[r_{i-2}] \leftarrow 0$ ;
    Se  $LR[l_i] \neq 0$  então retorne falso;
     $LR[l_i] \leftarrow 1$ ;
  Retorne verdadeiro;
Fim

```

Algoritmo 3.1: Validação de seqüência reduzida.

O Lema 3.19 prova que o Algoritmo 3.1 de validação de uma seqüência reduzida tem complexidade $O(n)$.

Lema 3.19: O algoritmo de validação de uma seqüência reduzida de um grafo k -caminho tem complexidade $O(n)$.

Prova: Primeiro, as n posições do vetor LR são inicializadas com zeros. Em seguida, faz-se o preenchimento inicial do vetor LR em $n - k - 1$ repetições. Finalmente, a repetição que preenche o vetor LR e que verifica se todos os vértices têm $LR[v] \leq 1$ é executada $n - k - 3$ vezes. Assim, conclui-se que este algoritmo tem complexidade $O(n)$. \Rightarrow

3.6 - Codificação da Seqüência Reduzida

O problema de codificação consiste em, dado um grafo k -caminho, obter a seqüência reduzida que o representa. A codificação pode ser feita através da obtenção de um k -caminho maximal do grafo seguida da inspeção das $(k+1)$ -cliques para obter os vértices r_i e l_i da seqüência reduzida. Entretanto, propomos um algoritmo mais eficiente.

Pelo Teorema 3.3, um grafo k -caminho $G = (V, E)$ possui dois vértices simpliciais e, pelo Lema 1.5, estes vértices têm grau k . Assim, basta calcular os graus dos vértices para encontrar os dois vértices simpliciais de G . Pelo Lema 3.11, estes vértices são l_1 e r_{p-1} . Sabe-se do Lema 3.12 que, após a remoção de um vértice simplicial l_i , o vértice l_{i+1} passa a ser simplicial no grafo k -caminho obtido e do Lema 3.13 que, após a remoção de um vértice simplicial r_i , o vértice r_{i-1} passa a ser simplicial no grafo k -caminho obtido. Estes lemas permitem a obtenção de modo eficiente dos vértices da seqüência reduzida.

O Algoritmo 3.2 de codificação pressupõe que $V = \{1, \dots, n\}$, $n > k + 1$. Os graus dos vértices são computados e os dois únicos vértices de grau k são os simpliciais l_1 e r_{p-1} . São então executados dois percursos em sentidos opostos na seqüência reduzida. O primeiro percurso inicia do vértice l_1 e obtém os vértices l_2, \dots, l_{p-1} em $p-2$ iterações. A cada iteração, remove-se o vértice l_i e obtém o vértice l_{i+1} que é o único vértice distinto de r_{p-1} que possui grau k no grafo obtido. De modo similar, o segundo percurso obtém os vértices r_{p-2}, \dots, r_1 em $p-2$ iterações. Inicia-se do grafo k -caminho e a cada iteração remove-se o vértice simplicial r_i e obtém-se o vértice r_{i-1} que é o único vértice distinto de l_1 que possui grau k no grafo obtido.

No Algoritmo 3.2, os graus são armazenados nos vetores $d1$ e $d2$ porque os graus dos vértices são alterados durante os dois percursos. Após a remoção de um vértice l_i , atualiza-se o vetor $d1$ e obtém-se o vértice l_{i+1} que possui grau k . Já o vetor $d2$ é atualizado após a remoção de um vértice r_i e da obtenção do vértice r_{i-1} que possui grau k .

O Algoritmo 3.2 de codificação possui complexidade linear como visto no Lema 3.20 a seguir.

Lema 3.20: O algoritmo de codificação de uma seqüência reduzida de um grafo k -caminho tem complexidade $O(m)$.

Prova: Os graus dos vértices são obtidos e armazenados nos vetores auxiliares com complexidade $O(m)$. Os vértices simpliciais l_1 e r_{p-1} são obtidos percorrendo uma vez o vetor com os graus. Na obtenção dos vértices l_i e r_i , a lista de adjacências do grafo é visitada. Logo, este passo possui complexidade $O(m)$. Assim, conclui-se que este algoritmo tem complexidade $O(m)$, ou seja, $O(n.k)$. \Rightarrow

```

Algoritmo Gera_Seqüência_Reduzida;
Entrada:   $n, k$ , lista de adjacências de grafo  $k$ -caminho;
Saída:     $\langle (l_1, r_1), \dots, (l_{n-k-1}, r_{n-k-1}) \rangle$ ;
Início
   $p \leftarrow n-k$ ;
  Compute os graus dos vértices e armazene em  $d1$  e  $d2$ ;
   $l_1 \leftarrow 0$ ;
  Para  $v$  de 1 até  $n$  faça
    Se  $d1[v] = k$  então
      Se  $l_1 = 0$  então  $l_1 \leftarrow v$ ;
      Senão  $r_{p-1} \leftarrow v$ ;
{obtenção de  $l_2, \dots, l_{p-1}$ }
  Para  $i$  de 1 até  $p-2$  faça
     $d1[l_i] \leftarrow 0$ ;
    Para  $\forall v \in Adj(l_i) \mid d1[v] \neq 0$  faça
       $d1[v] \leftarrow d1[v] - 1$ ;
      Se  $d1[v] = k$  então  $l_{i+1} \leftarrow v$ ;
{obtenção de  $r_{p-2}, \dots, r_1$ }
  Para  $i$  de  $p-1$  até 2 faça
     $d2[r_i] \leftarrow 0$ ;
    Para  $\forall v \in Adj(r_i) \mid d2[v] \neq 0$  faça
       $d2[v] \leftarrow d2[v] - 1$ ;
      Se  $d2[v] = k$  então  $r_{i-1} \leftarrow v$ ;
Fim

```

Algoritmo 3.2: Codificação de grafo k -caminho.

3.7 - Decodificação da Seqüência Reduzida

A decodificação da seqüência reduzida é o processo de obtenção do grafo k -caminho que ela representa. Este procedimento é executado em duas etapas e são obtidas as arestas e a seqüência de $(k+1)$ -cliques a partir de uma seqüência reduzida. Primeiro, obtém-se a $(k+1)$ -clique C_1 utilizando o Teorema 3.16(a), ou seja, $C_1 = V - (L_0 \cup R_1) = V - R$. Na segunda etapa, as demais $(k+1)$ -cliques são obtidas através do Lema 3.9(d), ou seja, para $i = 1, \dots, p-1$, faz-se $C_{i+1} = \{l_i\} \cup C_i - \{r_i\}$.

Seja o grafo 4-caminho de 10 vértices e $p = 6$ representado na Figura 3.4 pela seqüência reduzida $S = \langle (b,h), (e,g), (a,f), (g,i), (d,j) \rangle$. A Figura 3.7 ilustra como decodificar S , ou seja, como obter a seqüência de 5-cliques deste grafo a partir da sua seqüência reduzida.

$$\begin{aligned}
C_1 &= V - (L_0 \cup R_1) \Rightarrow C_1 = V - \{h,g,f,i,j\} = \{a,b,c,d,e\} \\
C_2 &= \{l_1\} \cup C_1 - \{r_1\} \Rightarrow C_2 = \{h\} \cup \{a,b,c,d,e\} - \{b\} = \{a,c,d,e,h\} \\
C_3 &= \{l_2\} \cup C_2 - \{r_2\} \Rightarrow C_3 = \{g\} \cup \{a,c,d,e,h\} - \{e\} = \{a,c,d,g,h\} \\
C_4 &= \{l_3\} \cup C_3 - \{r_3\} \Rightarrow C_4 = \{f\} \cup \{a,c,d,g,h\} - \{a\} = \{c,d,f,g,h\} \\
C_5 &= \{l_4\} \cup C_4 - \{r_4\} \Rightarrow C_5 = \{i\} \cup \{c,d,f,g,h\} - \{g\} = \{c,d,f,h,i\} \\
C_6 &= \{l_5\} \cup C_5 - \{r_5\} \Rightarrow C_6 = \{j\} \cup \{c,d,f,h,i\} - \{d\} = \{c,f,h,i,j\}
\end{aligned}$$

Figura 3.7: Decodificação da seqüência reduzida do grafo 4-caminho da Figura 3.4.

O Algoritmo 3.3 de decodificação pressupõe que $V = \{1, \dots, n\}$, $n > k + 1$. Este algoritmo obtém a seqüência de $(k+1)$ -cliques e o conjunto de arestas de um grafo k -caminho a partir da seqüência reduzida e da quantidade de vértices n . O vetor auxiliar $R^{-1}[v]$ assume valor zero quando $v \notin R$ e valor i quando $v = r_i$. Primeiramente, preenche-se o vetor R^{-1} e, em seguida, encontra-se a $(k+1)$ -clique C_1 com os vértices v que têm $R^{-1}[v] = 0$, pois $C_1 = V - R$. Para esta $(k+1)$ -clique, são determinadas todas as suas $k(k+1)/2$ arestas em G .

Obtêm-se as demais $(k+1)$ -cliques fazendo $C_{i+1} = \{l_i\} \cup C_i - \{r_i\}$, para $i = 1, \dots, p - 1$. Desse modo, todos os vértices de C_i estão em C_{i+1} exceto o vértice l_i que é substituído pelo vértice r_i . Como apenas o vértice r_i foi inserido, determinam-se apenas as k arestas incidentes a ele. Ao fim do processo, todas as $(k+1)$ -cliques e todas as arestas do grafo são obtidas.

O Lema 3.21 prova que o Algoritmo 3.3 de decodificação de uma seqüência reduzida tem complexidade $O(m)$.

Lema 3.21: O algoritmo de decodificação de uma seqüência reduzida de um grafo k -caminho tem complexidade $O(m)$.

Prova: O vetor R^{-1} é inicializado em $2n - k - 1$ iterações. A definição da $(k+1)$ -clique C_1 é executada em n iterações e a determinação de suas arestas consome $k(k+1)/2$ iterações. Para cada uma das $n - k - 1$ cliques restantes, são executadas $k + 1$

iterações para determinar a $(k+1)$ -clique e $k + 1$ iterações para determinar as k arestas que ainda não haviam sido obtidas. Assim são executadas:

$$2n - k - 1 + n + k(k+1)/2 + 2(n-k-1)(k+1) = 2n + k(k+1)/2 + (n-k-1)(2k+3) \text{ iterações.}$$

Desse modo, o algoritmo possui complexidade $O(n) + O(k^2) + O(n.k)$. Como $k < n$, obtém-se a complexidade $O(n.k) = O(m)$. \Rightarrow

```

Algoritmo Gera_Cliques_e_Arestas;
Entrada:  n, k,  $\langle (l_1, r_1), \dots, (l_{n-k-1}, r_{n-k-1}) \rangle$ ;
Saída:     $(k+1)$ -cliques  $C_1, \dots, C_{n-k}$ , conjunto de arestas  $E$ ;
Início
{inicialização do vetor  $R^{-1}$ }
  Para v de 1 até n faça  $R^{-1}[v] \leftarrow 0$ ;
  Para i de 1 até  $n-k-1$  faça  $R^{-1}[r_i] \leftarrow i$ ;
{obtenção da clique  $C_1$  e de suas arestas}
   $C_1 \leftarrow \emptyset$ ;
  Para v de 1 até n faça
    Se  $R^{-1}[v] = 0$  então
       $C_1 \leftarrow C_1 \cup \{v\}$ ;
   $E \leftarrow k(k-1)/2$  arestas da clique  $C_1$ ;
{obtenção da clique  $C_{i+1}$  e de suas arestas que não pertencem a  $C_i$ }
  Para i de 1 até  $n-k-1$  faça
     $C_{i+1} \leftarrow \emptyset$ ;
    Para  $\forall v \in C_i$  faça
      Se  $v = l_i$  então
         $C_{i+1} \leftarrow C_{i+1} \cup \{r_i\}$ ;
      Senão
         $C_{i+1} \leftarrow C_{i+1} \cup \{v\}$ ;
     $E \leftarrow E \cup \{v, r_i\}$ ;
Fim

```

Algoritmo 3.3: Decodificação da seqüência reduzida de um grafo k -caminho.

3.8 - Contagem e Geração de Grafos k -caminho Rotulados

Nessa seção, a seqüência reduzida é utilizada para contagem e geração aleatória de grafos k -caminho rotulados. A geração de uma seqüência reduzida é feita com base no Teorema 3.18, ou seja, em um grafo k -caminho, $|L_i \cup R_{i-1}| = p + 1$, para $1 < i < p$. Assim, é possível analisar as possibilidades de escolha dos vértices na geração de uma seqüência reduzida.

Teorema 3.22: O número de grafos k -caminho rotulados com n vértices, sendo

$$n > k + 1 \text{ vértices é } \frac{n!}{k!} \cdot \frac{k^{n-k-2}}{2}.$$

Prova: A prova é construtiva, ou seja, inicia-se com a seqüência reduzida vazia e escolhem-se vértices dentre os n vértices do grafo para preencher a seqüência reduzida. Para garantir que a seqüência gerada represente um grafo k -caminho, a escolha dos vértices atende a restrição do Teorema 3.18: $|L_i \cup R_{i-1}| = p + 1$, para $1 < i < p$. Inicialmente, no item 1 abaixo, a restrição é atendida para $i = 2$, ou seja, $|L_2 \cup R_1| = p + 1$. Depois, no item 2, a restrição é atendida para $i = 3, \dots, p - 1$.

1) Para atender a restrição $|L_2 \cup R_1| = p + 1$ é necessário que $r_1 \neq \dots \neq r_{p-1} \neq l_1 \neq l_2$. Assim, são escolhidos $p + 1 = n - k + 1$ vértices distintos dentre os n vértices de V . Desse modo, obtém-se um total de $\frac{n!}{(n - (n - k + 1))!} = \frac{n!}{(k - 1)!}$ permutações possíveis.

2) Os vértices l_3, l_4, \dots, l_{p-1} são escolhidos nesta ordem. Ao escolher o vértice l_i , verifica-se a restrição $|L_i \cup R_{i-1}| = p + 1$ e sabe-se que $l_1 \neq \dots \neq l_{i-1} \neq r_{i-2} \neq \dots \neq r_{p-1}$, pois já foi verificado que $|L_{i-1} \cup R_{i-2}| = p + 1$. Desse modo, o vértice l_i é o único elemento do conjunto $L_i \cup R_{i-1} = \{l_1, \dots, l_{i-1}, l_i, r_{i-1}, \dots, r_{p-1}\}$ que não foi selecionado. Como p elementos de $L_i \cup R_{i-1}$ já estão definidos e l_i deve ser diferente destes elementos, então existem $n - p = k$ possibilidades de escolha para l_i . São definidos $n - k - 3$ vértices l_3, l_4, \dots, l_{p-1} ; portanto são feitas k^{n-k-3} combinações na escolha dos l_i .

3) Como as seqüências $\langle (l_1, r_1), (l_2, r_2), \dots, (l_{p-1}, r_{p-1}) \rangle$ e $\langle (r_{p-1}, l_{p-1}), \dots, (r_2, l_2), (r_1, l_1) \rangle$ representam o mesmo grafo k -caminho, então cada grafo foi contado 2 vezes.

$$\text{De 1), 2) e 3) obtém-se: } \frac{n!}{(k-1)!} \cdot \frac{k^{n-k-3}}{2}.$$

O resultado do teorema é obtido ao multiplicar e dividir a fórmula acima por k . \Rightarrow

Observe que, para $k = 1$, obtém-se a quantidade de caminhos simples rotulados distintos, ou seja, $n!/2$ caminhos. Para $n = k + 1$, pela definição, os grafos k -caminho são grafos completos e a fórmula não pode ser usada, pois só vale para $n > k + 1$. Isso ocorre porque, na obtenção da fórmula, é considerada a existência de pelo menos

duas $(k+1)$ -cliques. A tabela a seguir mostra alguns resultados para valores de n quando $k=2$ e $k=3$.

Tabela 3.1: Totais de grafos k -caminho rotulados.

<i>Fórmula</i>	<i>n</i>	<i>k</i>	<i>Total</i>	<i>n</i>	<i>k</i>	<i>Total</i>
$n = k + 2, n(n-1)/2$	4	2	6	5	3	10
$n = k + 3, n(n-1)(n-2)k/2$	5	2	60	6	3	180
$n = k + 4, n(n-1)(n-2)(n-3)k^2/2$	6	2	720	7	3	3.780
$n = k + 5, n(n-1)(n-2)(n-3)(n-4)k^3/2$	7	2	10.080	8	3	90.720

A geração aleatória de grafos k -caminho é feita com base na prova do Teorema 3.22. Primeiro, inicia-se com a seqüência reduzida vazia e escolhem-se $n - k$ vértices distintos para $\{l_1, r_1, \dots, r_{p-1}\}$. Em seguida, são escolhidos l_2, \dots, l_{p-1} nessa ordem. Para $1 < i < p$, todos os elementos de $L_i \cup R_{i-1} = \{l_1, \dots, l_{i-1}, l_i, r_{i-1}, \dots, r_{p-1}\}$ já estão selecionados, exceto l_i . Assim, a cada iteração escolhe-se um l_i dentre os vértices que não estão contidos em $\{l_1, \dots, l_{i-1}, r_{i-1}, \dots, r_{p-1}\}$.

O Algoritmo 3.4 de geração aleatória uniforme de grafos k -caminho pressupõe que $V = \{1, \dots, n\}$, $n > k + 1$, e utiliza o vetor auxiliar *livre* de n posições. Este vetor armazena os vértices que podem ser escolhidos nas últimas posições. São utilizadas: as últimas $n - i$ posições para os vértices r_i , as últimas $k + 1$ posições para o vértice l_1 e as últimas k posições para os demais vértices l_i . Cabe ressaltar que, na escolha do vértice l_i , as k posições do vetor *livre* contêm os vértices que não pertencem a $\{l_1, \dots, l_{i-1}, r_{i-1}, \dots, r_{p-1}\}$ e, na escolha do vértice l_{i+1} , contêm os vértices que não pertencem a $\{l_1, \dots, l_i, r_i, \dots, r_{p-1}\}$. Assim, na iteração i , permuta-se o vértice l_i (que acabou de ser escolhido) com o vértice r_{i-1} (que fará parte nas opções de escolha da próxima iteração).

Lema 3.23: O algoritmo de geração aleatória uniforme de uma seqüência reduzida de um grafo k -caminho tem complexidade $O(n)$.

Prova: A inicialização do vetor *livre* é feita em n iterações e a escolha dos vértices r_i em $n - k - 1$ iterações. O vértice l_1 é obtido em uma iteração e os demais vértices l_i são obtidos em $n - k - 2$ iterações. Logo, o algoritmo possui complexidade $O(n)$. \Rightarrow

```

Algoritmo Geração_Aleatória_Sequência_Reduzida;
Entrada:  $n, k$ ;
Saída:  $\langle (l_1, r_1), \dots, (l_{n-k-1}, r_{n-k-1}) \rangle$ ;
Início
  Para  $v$  de 1 até  $n$  faça
     $livre[v] \leftarrow v$ ;
  {escolha dos vértices  $r_1, \dots, r_{p-1}$ }
    Para  $j$  de 1 até  $n-k-1$  faça
      escolha  $x$  aleatório entre  $j$  e  $n$ ;
       $r_j \leftarrow livre[x]$ ;
       $livre[x] \leftarrow livre[j]$ ;
  {escolha do vértice  $l_1$ }
    escolha  $x$  aleatório entre  $n-k$  e  $n$ ;
     $l_1 \leftarrow livre[x]$ ;
     $livre[x] \leftarrow livre[n-k]$ ;
  {escolha dos vértices  $l_2, \dots, l_{p-1}$ }
    Para  $i$  de 2 até  $n-k-1$  faça
      escolha  $x$  aleatório entre  $n-k+1$  e  $n$ ;
       $l_i \leftarrow livre[x]$ ;
       $livre[x] \leftarrow r_{i-1}$ ;
Fim

```

Algoritmo 3.4: Geração aleatória de sequência reduzida de grafo k -caminho.

Capítulo 4 - Propriedades dos Grafos k -caminho

Neste capítulo, apresentamos algumas propriedades dos grafos k -caminho que são obtidas diretamente da seqüência reduzida e as utilizamos em soluções eficientes para alguns problemas. Os dois problemas mais importantes solucionados com complexidade $O(n)$ são o isomorfismo e a obtenção de caminho hamiltoniano. Além disso, verificamos a pertinência de uma aresta com complexidade $O(1)$ e obtemos o grau de um vértice com complexidade $O(1)$. Isto é possível através de uma representação computacional adequada para a seqüência reduzida.

Na Seção 4.1, mostra-se a relação de pertinência dos vértices da seqüência reduzida com as cliques da seqüência de $(k+1)$ -cliques. Também é visto como obter os graus dos vértices e como verificar a pertinência de uma aresta diretamente da seqüência reduzida. Na Seção 4.2, são contados o total de esquemas de eliminação perfeita de um grafo k -caminho. Nas Seções 4.3 e 4.4, são propostos algoritmos eficientes para verificar o isomorfismo de grafos k -caminho e para obter um caminho hamiltoniano. Estes dois algoritmos possuem complexidade $O(n)$ e têm como entradas seqüências reduzidas de grafos k -caminho.

4.1 - Propriedades da Seqüência Reduzida

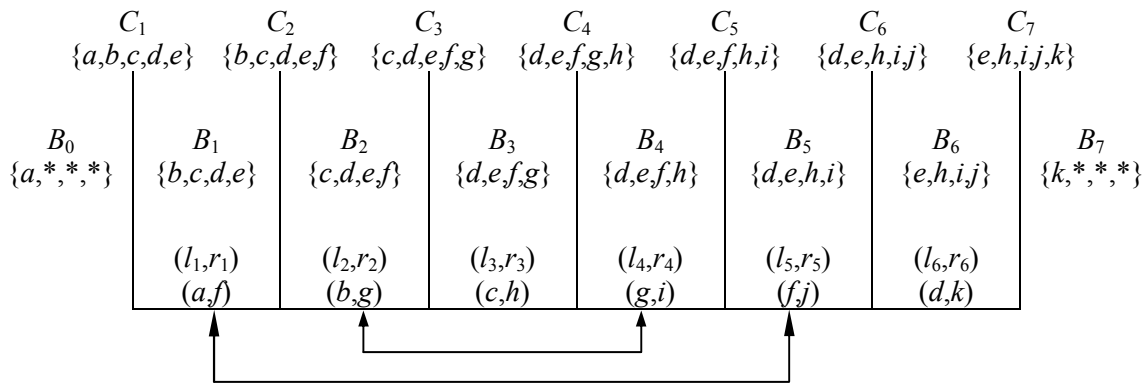
Nesta seção, são apresentadas propriedades dos grafos k -caminho que relacionam os vértices da seqüência reduzida com as cliques maximais da seqüência de $(k+1)$ -cliques. Estas propriedades são utilizadas na obtenção dos graus de todos os vértices com complexidade $O(n)$ diretamente da seqüência reduzida e na verificação da existência de uma aresta com complexidade $O(1)$.

Seja $G = (V, E)$ um grafo k -caminho. O Teorema 4.1 mostra que todo vértice de V pertence a uma subseqüência de $(k+1)$ -cliques consecutivas do grafo G , representado por $\langle C_1, C_2, \dots, C_p \rangle$, $p > 1$.

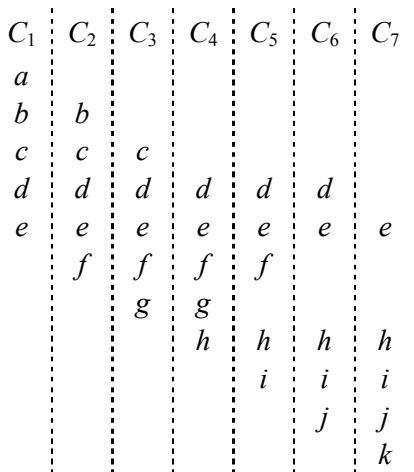
Teorema 4.1: Sejam $G = (V, E)$ um grafo k -caminho, $\langle C_1, C_2, \dots, C_p \rangle$, $p > 1$, sua seqüência de $(k+1)$ -cliques e um vértice $v \in V$. As $(k+1)$ -cliques que contêm o vértice v formam uma subseqüência $\langle C_x, C_{x+1}, \dots, C_y \rangle$, $1 \leq x \leq y \leq p$, de $(k+1)$ -cliques consecutivas.

Prova: No Teorema 3.6, provamos que $T = (\{C_1, \dots, C_p\}, \{\{C_i, C_{i+1}\} \mid 1 \leq i < p\})$ é a única árvore de cliques de G . Pela propriedade da interseção das árvores de cliques, todo vértice v pertencente a C_x e C_y , $x < y$, está no caminho entre C_x e C_y em T , ou seja, v pertence as cliques consecutivas C_x, C_{x+1}, \dots, C_y . \Rightarrow

A Figura 4.1(a) mostra um esquema com as cliques do grafo 4-caminho com comprimento $p = 7$ cuja seqüência reduzida é $S = \langle (a,f), (b,g), (c,h), (g,i), (f,j), (d,k) \rangle$. A Figura 4.1(b) ilustra as 5-cliques e a Figura 4.1(c) exhibe as subseqüências consecutivas de 5-cliques para cada um dos vértices deste grafo. As subseqüências podem ser formadas utilizando $(k+1)$ -cliques: (i) mais à esquerda, (ii) mais à direita, (iii) do centro ou (iv) todas. Por exemplo, na Figura 4.1(c), verifica-se que os vértices são distribuídos pelos quatro casos assim: (i) a, b, c, d , (ii) h, i, j, k , (iii) f, g , (iv) e .



a) Representação esquemática.



(b) 5-cliques.

Vértice	Subseqüência
a	$\langle C_1 \rangle$
b	$\langle C_1, C_2 \rangle$
c	$\langle C_1, C_2, C_3 \rangle$
d	$\langle C_1, C_2, C_3, C_4, C_5, C_6 \rangle$
e	$\langle C_1, C_2, C_3, C_4, C_5, C_6, C_7 \rangle$
f	$\langle C_2, C_3, C_4, C_5 \rangle$
g	$\langle C_3, C_4 \rangle$
h	$\langle C_4, C_5, C_6, C_7 \rangle$
i	$\langle C_5, C_6, C_7 \rangle$
j	$\langle C_6, C_7 \rangle$
k	$\langle C_7 \rangle$

(c) Subseqüências de 5-cliques.

Figura 4.1: Representação de grafo 4-caminho.

Já foi visto no Lema 3.14 que $l_1 \neq l_2 \neq \dots \neq l_{p-1}$ e $r_1 \neq r_2 \neq \dots \neq r_{p-1}$. Assim, verifica-se que um vértice aparece no máximo duas vezes na seqüência reduzida, uma vez em L e outra em R . São quatro combinações de pertinência de um vértice na seqüência

reduzida: (i) $v \in L$ e $v \notin R$, (ii) $v \notin L$ e $v \in R$, (iii) $v \in L$ e $v \in R$ e (iv) $v \notin L$ e $v \notin R$. O Teorema 4.2 mostra que estes casos estão diretamente relacionados com os quatro casos das subsequências de $(k+1)$ -cliques consecutivas descritas anteriormente.

Teorema 4.2: Sejam um grafo k -caminho $G=(V,E)$, sua seqüência $\langle C_1, C_2, \dots, C_p \rangle$, $p > 1$, de $(k+1)$ -cliques e $S = \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$ a sua seqüência reduzida. As seguintes propriedades são observadas:

- i) $v = l_i$ e $v \notin R$ se e somente se $v \in C_a$, para $1 \leq a \leq i$;
- ii) $v \notin L$ e $v = r_j$ se e somente se $v \in C_a$, para $j < a \leq p$;
- iii) $v = l_i$ e $v = r_j$ se e somente se $v \in C_a$, para $j < a \leq i$ sendo $j \neq i-1$;
- iv) $v \notin L$ e $v \notin R$ se e somente se $v \in C_a$, para $1 \leq a \leq p$.

Prova: Na prova das quatro propriedades, utiliza-se o Teorema 3.16(a), ou seja, para $1 \leq a \leq p$, os vértices da $(k+1)$ -clique C_a são obtidos pela expressão $C_a = V - (L_{a-1} \cup R_a)$. Assim, conclui-se que $v \in C_a$ se e somente se $v \notin L_{a-1} \cup R_a$.

i) (\Rightarrow) $v \notin R$, logo $v \in C_a = V - (L_{a-1} \cup R_a)$ se $v \in V - L_{a-1}$. Como $v = l_i \notin L_{a-1}$ quando $a-1 < i$, conclui-se que $v \in C_a$ quando $a \leq i$.

(\Leftarrow) Sabe-se que $v \in C_a$ se e somente se $v \notin L_{a-1} \cup R_a$. Desse modo, $v \in C_1$ se e somente se $v \notin L_0 \cup R_1 = R$, logo $v \notin R$. Assim, $v \in C_a = V - (L_{a-1} \cup R_a)$ se $v \in V - L_{a-1}$. Como $v \in C_i$ e $v \notin C_{i+1}$, então $v \notin L_{i-1}$ e $v \in L_i$. Sabendo-se que $L_i = \{l_i\} \cup L_{i-1}$, conclui-se que $v = l_i$.

ii) (\Rightarrow) $v \notin L$, logo $v \in C_a = V - (L_{a-1} \cup R_a)$ se $v \in V - R_a$. Como $v = r_j \notin R_a$ quando $j < a$, conclui-se que $v \in C_a$ quando $j < a$.

(\Leftarrow) Sabe-se que $v \in C_a$ se e somente se $v \notin L_{a-1} \cup R_a$. Deste modo, $v \in C_p$ se e somente se $v \notin L_{p-1} \cup R_p = L$, logo $v \notin L$. Assim, $v \in C_a = V - (L_{a-1} \cup R_a)$ se $v \in V - R_a$. Como $v \notin C_j$ e $v \in C_{j+1}$, então $v \in R_j$ e $v \notin R_{j+1}$. Sabendo-se que $R_j = \{r_j\} \cup R_{j+1}$, conclui-se que $v = r_j$.

iii) (\Rightarrow) Sabe-se que $v = l_i \notin L_{a-1}$ para $a \leq i$ e que $v = r_j \notin R_a$ para $j < a$. Assim, $v \notin L_{a-1} \cup R_a$ quando $j < a \leq i$. Como $v \in C_a$ se e somente se $v \notin L_{a-1} \cup R_a$, conclui-se que $v \in C_a$ quando $j < a \leq i$. Sabe-se do Lema 3.9(e) que $r_{a-1} \neq l_a$. Como $r_j = l_i$, tem-se $j \neq i-1$.

(\Leftarrow) Sabe-se que $v \in C_a$ se e somente se $v \notin L_{a-1} \cup R_a$. Como $v \in C_i$ e $v \notin C_{i+1}$, então $v \notin L_{i-1} \cup R_i$ e $v \in L_i \cup R_{i+1}$. Sabendo-se que $L_i = \{l_i\} \cup L_{i-1}$ e $R_{i+1} \subset R_i$, conclui-se

que $v = l_i$. De modo análogo, $v \in C_{j+1}$ e $v \notin C_j$, então $v \notin L_j \cup R_{j+1}$ e $v \in L_{j-1} \cup R_j$. Sabendo-se que $R_j = \{r_j\} \cup R_{j+1}$ e $L_{j-1} \subset L_j$, conclui-se que $v = r_j$. Como $r_{a-1} \neq l_a$ e $r_j = l_i$, tem-se que $j \neq i - 1$.

iv) (\Rightarrow) Sabe-se que $v \in C_a$ se e somente se $v \notin L_{a-1} \cup R_a$. Já que $v \notin R$ e $v \notin L$, tem-se que $v \notin (R_{a-1} \cup L_a)$, para $1 \leq a \leq p$. Assim, $v \in C_a$, para $1 \leq a \leq p$.

(\Leftarrow) Sabe-se que $v \in C_a$ se e somente se $v \notin L_{a-1} \cup R_a$. Sabe-se que $v \in C_1$ e $v \in C_p$. Assim, $v \notin L_0 \cup R_1 = R$, então $v \notin R$ e $v \notin L_{p-1} \cup R_p = L$, então $v \notin L$. \Rightarrow

A pertinência de um vértice a uma subsequência de $(k+1)$ -cliques vai nos permitir determinar seu grau no grafo. Denota-se por $q(v)$ a quantidade de $(k+1)$ -cliques a que o vértice v pertence. A partir da Definição 3.4 de grafos k -caminho, verifica-se que, ao inserir um vértice v no grafo, adjacente aos vértices da k -clique Q , os vértices de Q têm os seus graus incrementados de um e passam a pertencer a mais uma $(k+1)$ -clique. O próximo teorema mostra a relação entre o grau de um vértice e a quantidade de $(k+1)$ -cliques a que ele pertence.

Teorema 4.3: Seja um grafo k -caminho $G = (V, E)$. Para $v \in V$, $d(v) = q(v) + k - 1$.

Prova: Pela Definição 3.4, um grafo k -caminho pode ser obtido iniciando de um grafo completo com $k + 1$ vértices e recursivamente adicionando vértices adjacentes a k -cliques do grafo. Assim, a prova é feita por indução no número de vértices.

- Para $n = k + 1$, o grafo correspondente tem exatamente uma $(k+1)$ -clique e, para todos os seus vértices, $d(v) = k$ e $q(v) = 1$. Logo, o teorema é válido para o menor grafo k -caminho.
- Suponha que o resultado vale para os grafos k -caminho com $n - 1$ vértices.
- Quando um vértice v é inserido, adicionam-se k arestas adjacentes a uma k -clique Q . O vértice v pertence a exatamente uma $(k+1)$ -clique e tem grau k , portanto o teorema vale para ele. Para os vértices pertencentes a k -clique Q , o teorema também é válido, pois tanto o grau destes vértices como a quantidade de $(k+1)$ -cliques a qual pertencem são incrementados de um como decorrência da inserção do vértice v . Desse modo, a propriedade vale para grafos k -caminho com n vértices e o teorema está provado. \Rightarrow

Os graus dos vértices de um grafo k -caminho representado por sua seqüência reduzida podem ser obtidos pela decodificação desta seqüência seguida de uma contagem de arestas. Entretanto, os graus dos vértices são obtidos diretamente da seqüência reduzida como mostra o lema visto a seguir.

Lema 4.4: Sejam um grafo k -caminho $G = (V, E)$, sua seqüência $\langle C_1, C_2, \dots, C_p \rangle$, $p > 1$, de $(k+1)$ -cliques e $S = \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$ a sua seqüência reduzida. As seguintes propriedades são observadas:

- i) Se $v = l_i$ e $v \notin R$, então $q(v) = i$ e $d(v) = i + k - 1$;
- ii) Se $v \notin L$ e $v = r_j$, então $q(v) = p - j$ e $d(v) = n - j - 1$;
- iii) Se $v = l_i$ e $v = r_j$, então $q(v) = j - i$ e $d(v) = k - 1 + i - j$;
- iv) Se $v \notin L$ e $v \notin R$, então $q(v) = p$ e $d(v) = n - 1$.

Prova: Pelo Teorema 4.2, tem-se que:

- i) $v = l_i$ e $v \notin R$ se e somente se $v \in C_a$, para $1 \leq a \leq i$. Logo, $q(v) = i$;
- ii) $v \notin L$ e $v = r_j$ se e somente se $v \in C_a$, para $j < a \leq p$. Logo, $q(v) = p - j$;
- iii) $v = l_i$ e $v = r_j$ se e somente se $v \in C_a$, para $j < a \leq i$. Logo, $q(v) = i - j$;
- iv) $v \notin L$ e $v \notin R$ se e somente se $v \in C_a$, para $1 \leq a \leq p$. Logo, $q(v) = p$.

Os graus dos vértices são obtidos pela simples substituição de $q(v)$ na expressão obtida no teorema anterior. \Rightarrow

O algoritmo para cálculo dos graus dos vértices de um grafo através da análise de cada uma das arestas do grafo tem complexidade $O(m)$. Entretanto, propomos o Algoritmo 4.1 que computa os graus dos vértices de um grafo k -caminho diretamente de sua seqüência reduzida com complexidade $O(n)$, conforme provamos no Lema 4.5.

O Algoritmo 4.1 pressupõe que $V = \{1, \dots, n\}$ e utiliza os vetores auxiliares L^{-1} e R^{-1} . O elemento $L^{-1}[v]$ assume valor $p = n - k$ quando $v \notin L$ e valor i quando $v = l_i$. De modo similar, $R^{-1}[v]$ assume valor zero quando $v \notin R$ e valor i quando $v = r_i$. Assim, para os quatro casos do Lema 4.4, tem-se $q(v) = L^{-1}[v] - R^{-1}[v]$ e, portanto, $d(v) = k - 1 + L^{-1}[v] - R^{-1}[v]$.

Lema 4.5: O algoritmo para cálculo dos graus dos vértices de um grafo k -caminho diretamente da seqüência reduzida tem complexidade $O(n)$.

Prova: A inicialização e o preenchimento dos vetores auxiliares são feitos em $O(n)$. Como o cálculo dos graus também é feito em $O(n)$, o algoritmo tem complexidade $O(n)$. \Rightarrow

```

Algoritmo Calcula_Graus;
Entrada:   $n, k, \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$ ;
Saída:   vetor  $d[n]$  com os graus dos  $n$  vértices;
Início
  Para  $v$  de 1 até  $n$  faça
     $L^{-1}[v] \leftarrow n-k$ ;
     $R^{-1}[v] \leftarrow 0$ ;
  Para  $i$  de 1 até  $n-k-1$  faça
     $L^{-1}[l_i] \leftarrow i$ ;
     $R^{-1}[r_i] \leftarrow i$ ;
  Para  $v$  de 1 até  $n$  faça
     $d[v] \leftarrow k-1 + L^{-1}[v] - R^{-1}[v]$ ;
Fim

```

Algoritmo 4.1: Cálculo dos graus dos vértices de um grafo k -caminho.

Existe uma preocupação do ponto de vista algorítmico em verificar se uma dada aresta existe no grafo de modo eficiente. Esta operação possui complexidade $O(n)$ no pior caso quando se utiliza listas de adjacências e complexidade $O(1)$ quando se utiliza matriz de adjacência. Utilizando os vetores L^{-1} e R^{-1} , também é possível verificar se uma aresta $\{v, w\}$ pertence a um grafo k -caminho com complexidade $O(1)$. Apesar da mesma complexidade de execução, a seqüência reduzida é mais eficiente no armazenamento, pois possui complexidade de memória $O(n)$ enquanto a matriz de adjacência possui complexidade $O(n^2)$.

Teorema 4.6: Sejam um grafo k -caminho $G = (V, E)$ e $S = \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$ a sua seqüência reduzida. Sejam $L^{-1}[v] = n-k$ se $v \notin L$ e $L^{-1}[v] = i$ quando $v = l_i$; $R^{-1}[v] = 0$ se $v \notin R$ e $R^{-1}[v] = i$ quando $v = r_i$. Verifica-se que:

a aresta $\{v, w\} \in E$ se e somente se $R^{-1}[v] < L^{-1}[w]$ e $R^{-1}[w] < L^{-1}[v]$.

Prova: A aresta $\{v, w\} \in E$ se e somente se os vértices v e w pertencem a pelo menos uma $(k+1)$ -clique C_i . Os vértices $v, w \in C_i$ se e somente se $R^{-1}[v] < i \leq L^{-1}[v]$ e $R^{-1}[w] < i \leq L^{-1}[w]$. Logo, $R^{-1}[v] < i \leq L^{-1}[w]$ e $R^{-1}[w] < i \leq L^{-1}[v]$, ou seja, $R^{-1}[v] < L^{-1}[w]$ e $R^{-1}[w] < L^{-1}[v]$ e o teorema está provado. \Rightarrow

4.2 - Esquemas de Eliminação Perfeita

Um grafo cordal possui vários esquemas de eliminação perfeita. Nesta seção, utilizando as propriedades estruturais bem definidas dos grafos k -caminho, contamos a quantidade de esquemas de eliminação perfeita de um grafo k -caminho.

Para grafos k -caminho com $n > k + 1$ vértices, sabe-se do Teorema 3.3 de caracterização proposto por MARKENZON *et al.* [31] que possuem exatamente dois vértices simpliciais e verifica-se a partir da Definição 3.4 de grafo k -caminho que ao remover um vértice simplicial obtém-se um grafo k -caminho com $n - 1$ vértices. No próximo teorema, utilizam-se estes dois resultados na contagem dos esquemas de eliminação perfeita de um grafo k -caminho.

Teorema 4.7: Existem $2^{n-k-1}(k+1)!$ esquemas de eliminação perfeita em um grafo k -caminho $G=(V,E)$ com n vértices.

Prova: A prova é construtiva. A cada iteração remove-se um vértice simplicial v_i do grafo k -caminho até que se obtenha o esquema de eliminação perfeita $\langle v_1, \dots, v_n \rangle$. A prova é dividida em duas partes: na primeira, obtém-se v_1, \dots, v_{n-k-1} e, na segunda, v_{n-k}, \dots, v_n .

Para $i = 1 \dots n - k - 1$, escolhe-se dentre um dos dois vértices simpliciais do grafo k -caminho para ser o vértice v_i removido. Como são removidos $n - k - 1$ vértices, existem 2^{n-k-1} modos de escolher os vértices v_1, \dots, v_{n-k-1} (1).

Como o grafo induzido pelos $k + 1$ vértices restantes é um grafo completo, então os vértices v_{n-k}, \dots, v_n podem ser removidos em qualquer ordem. Logo, existem $(k + 1)!$ modos de escolher os vértices v_{n-k}, \dots, v_n (2). Das expressões (1) e (2) obtém-se o resultado do teorema. \Rightarrow

Observe que o raciocínio utilizado na prova do Teorema 4.10 possibilita a geração simples e eficiente de todos os $2^{n-k-1}(k+1)!$ esquemas de eliminação perfeita de um grafo k -caminho a partir de sua seqüência reduzida.

4.3 - Isomorfismo

O isomorfismo de grafos é um dos poucos problemas que não se sabe se é NP-completo ou se existe solução polinomial. Apesar disto, a maioria dos pesquisadores acha que este problema não é polinomial. Em [30], LUEKER e BOOTH provam que se os grafos cordais possuírem algoritmo polinomial para o isomorfismo, então o problema de isomorfismo é polinomial. Portanto, espera-se que não exista algoritmo polinomial para isomorfismo de grafos cordais. Entretanto para algumas classes, como a dos grafos planares [23] e a dos grafos de intervalo [30], existem algoritmos lineares. Nesta seção, propomos um algoritmo eficiente com complexidade $O(n)$ para verificação de isomorfismo de grafos k -caminho a partir da seqüência reduzida.

Dois grafos são ditos isomorfos quando têm a mesma forma apesar de os conjuntos de arestas serem distintos. A Figura 4.2 ilustra dois grafos 2-caminho isomorfos. A definição clássica de isomorfismo é mostrada a seguir.

Definição 4.8: Dois grafos $G = (V_1, E_1)$ e $H = (V_2, E_2)$ são **isomorfos** quando existe uma bijeção $f: V_1 \rightarrow V_2$ tal que $\{v, w\} \in E_1 \Leftrightarrow \{f(v), f(w)\} \in E_2, \forall v, w \in V_1$.



Figura 4.2: Dois grafos 2-caminho isomorfos.

No Teorema 4.9, provamos as condições necessárias e suficientes para que dois grafos k -caminho $G = (V_1, E_1)$ e $H = (V_2, E_2)$ sejam isomorfos através de suas seqüências reduzidas $S_1 = \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$ e $S_2 = \langle (t_1, u_1), \dots, (t_{p-1}, u_{p-1}) \rangle$. Já foram definidos os vetores auxiliares L^{-1} e R^{-1} para S_1 tal que $L^{-1}[v] = n - k$ se $v \notin L$ e $L^{-1}[v] = i$ quando $v = l_i$ e $R^{-1}[v] = 0$ se $v \notin R$ e $R^{-1}[v] = i$ quando $v = r_i$. Assim, de modo similar, tem-se $T^{-1}[v] = n - k$ se $v \notin T$ e $T^{-1}[v] = i$ quando $v = t_i$ e $U^{-1}[v] = 0$ se $v \notin U$ e $U^{-1}[v] = i$ quando $v = u_i$.

Teorema 4.9: Sejam dois grafos k -caminho com n vértices $G = (V_1, E_1)$ e $H = (V_2, E_2)$ cujas seqüências reduzidas são $S_1 = \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$ e $S_2 = \langle (t_1, u_1), \dots, (t_{p-1}, u_{p-1}) \rangle$. Estes grafos são isomorfos se e somente se:

(i) $\forall i, j \ l_i = r_j \Leftrightarrow t_i = u_j$ ou

(ii) $\forall i, j \ l_i = r_j \Leftrightarrow u_{p-i} = t_{p-j}$.

Prova: (\Rightarrow) Seja uma função de isomorfismo $f: V_1 \rightarrow V_2$. Como f é uma permutação de vértices, tem-se que $S = \langle (f(l_1), f(r_1)), \dots, (f(l_{p-1}), f(r_{p-1})) \rangle$ é seqüência reduzida de H . Além disso, sua reversa $S' = \langle (f(r_{p-1}), f(l_{p-1})), \dots, (f(r_1), f(l_1)) \rangle$ também é seqüência reduzida de H . Logo, $S_2 = S$ ou $S_2 = S'$.

Sabe-se que $\forall i, j \ l_i = r_j \Leftrightarrow f(l_i) = f(r_j)$. No caso em que $S_2 = S$, tem-se $f(l_i) = t_i$ e $f(r_j) = u_j$, ou seja, $\forall i, j \ l_i = r_j \Leftrightarrow t_i = u_j$ (i). No caso em que $S_2 = S'$, tem-se $f(l_i) = u_{p-i}$ e $f(r_j) = t_{p-j}$, ou seja, $\forall i, j \ l_i = r_j \Leftrightarrow u_{p-i} = t_{p-j}$ (ii).

(\Leftarrow) Nos dois casos, utiliza-se o fato de que $l_a \neq l_b$, $r_a \neq r_b$, $t_a \neq t_b$ e $u_a \neq u_b$, para $a \neq b$ e utiliza-se o Teorema 4.6 que permite verificar a existência de uma aresta pela análise dos vetores L^{-1} e R^{-1} , ou seja, a aresta $\{v, w\} \in E_1$ se e somente se $R^{-1}[v] < L^{-1}[w]$ e $R^{-1}[w] < L^{-1}[v]$. De modo similar, uma aresta $\{v, w\} \in E_2$ se e somente se $U^{-1}[v] < T^{-1}[w]$ e $U^{-1}[w] < T^{-1}[v]$.

(i) $\forall i, j \ l_i = r_j \Leftrightarrow t_i = u_j$. Logo, $\forall i \ l_i \notin R \Leftrightarrow t_i \notin U$ e $\forall i \ r_i \notin L \Leftrightarrow u_i \notin T$. Assim, $|S_1| = |S_2|$ e existe uma função bijetora $f_1: S_1 \rightarrow S_2$ tal que $\forall i \ f_1(l_i) = t_i$ e $f_1(r_i) = u_i$. Seja $f_2: V_1 - S_1 \rightarrow V_2 - S_2$ uma função bijetora qualquer e $f: V_1 \rightarrow V_2$ a função composta por f_1 e f_2 . Desse modo, $L^{-1}[v] = T^{-1}[f(v)] = i$ caso $v = l_i$ e $L^{-1}[v] = T^{-1}[f(v)] = p$ caso $v \notin L$. De modo similar, tem-se que $R^{-1}[v] = U^{-1}[f(v)] = i$ caso $v = r_i$ e $R^{-1}[v] = U^{-1}[f(v)] = 0$ caso $v \notin R$. Logo, $R^{-1}[v] < L^{-1}[w]$ e $R^{-1}[w] < L^{-1}[v]$ se e somente se $U^{-1}[f(v)] < T^{-1}[f(w)]$ e $U^{-1}[f(w)] < T^{-1}[f(v)]$, ou seja, $\{v, w\} \in E_1 \Leftrightarrow \{f(v), f(w)\} \in E_2$.

(ii) $\forall i, j \ l_i = r_j \Leftrightarrow u_{p-i} = t_{p-j}$. Utilizando raciocínio análogo ao feito no primeiro caso, conclui-se que existe a função bijetora $f_1: S_1 \rightarrow S_2$ tal que $\forall i \ f_1(l_i) = u_{p-i}$ e $f_1(r_i) = t_{p-i}$. Seja $f: V_1 \rightarrow V_2$ a função composta por f_1 e por uma função bijetora qualquer $f_2: V_1 - S_1 \rightarrow V_2 - S_2$. Desse modo, $L^{-1}[v] = i$ e $U^{-1}[f(v)] = p - i$ caso $v = l_i$ e $L^{-1}[v] = p$ e $U^{-1}[f(v)] = 0$ caso $v \notin L$. De modo similar, tem-se que $R^{-1}[v] = i$ e $T^{-1}[f(v)] = p - i$ caso $v = r_i$ e $R^{-1}[v] = 0$ e $T^{-1}[f(v)] = p$ caso $v \notin R$. Logo, $R^{-1}[v] < L^{-1}[w]$ e $R^{-1}[w] < L^{-1}[v]$ se e somente se $T^{-1}[f(v)] > U^{-1}[f(w)]$ e $T^{-1}[f(w)] > U^{-1}[f(v)]$, ou seja, $\{v, w\} \in E_1 \Leftrightarrow \{f(v), f(w)\} \in E_2$.

Desse modo, se um dos dois casos for satisfeito, a função f satisfaz a Definição 4.8 de isomorfismo, ou seja, os grafos k -caminho G e H são isomorfos. \Rightarrow

Sem perda de generalidade, assume-se que a entrada do Algoritmo 4.2 de isomorfismo é composta pela seqüência reduzida de dois grafos k -caminho $G = (V, E_1)$ e $H = (V, E_2)$ tal que $V = \{1, \dots, n\}$. Este algoritmo retorna falso caso os grafos não sejam isomorfos ou retorna verdadeiro caso uma das duas condições do Teorema 4.9 seja satisfeita.

O Algoritmo 4.2 computa os vetores auxiliares L^{-1} , T^{-1} e U^{-1} definidos anteriormente. Como $l_1 \neq \dots \neq l_{p-1}$ e $r_1 \neq \dots \neq r_{p-1}$, para cada r_j existem duas opções: $r_j = l_i$ ou $r_j \notin \{l_1, \dots, l_{p-1}\}$. Assim, pelo Teorema 4.9, os grafos são isomorfos quando um dos dois casos ocorre:

- Para $1 \leq j < p$, $(l_i = r_j \wedge t_i = u_j) \vee (r_j \notin L \wedge u_j \notin T)$. Desse modo, $(L^{-1}[r_j] = T^{-1}[u_j] = i) \vee (L^{-1}[r_j] = T^{-1}[u_j] = 0)$, ou seja, $L^{-1}[r_j] = T^{-1}[u_j]$.
- Para $1 \leq j < p$, $(l_i = r_j \wedge u_{p-i} = t_{p-j}) \vee (r_j \notin L \wedge t_{p-j} \notin U)$. Desse modo, $(L^{-1}[r_j] = i \wedge U^{-1}[t_{p-j}] = p - i) \vee (L^{-1}[r_j] = p \wedge U^{-1}[t_{p-j}] = 0)$, ou seja, $L^{-1}[r_j] + U^{-1}[t_{p-j}] = p$.

No Algoritmo 4.2, as variáveis *resp1* e *resp2* mantêm o valor verdadeiro caso os vértices r_1, \dots, r_{p-1} satisfaçam o primeiro caso ou o segundo caso, respectivamente. Assim, os grafos são isomorfos quando uma das duas variáveis for verdadeira. O Lema 4.10 prova que este algoritmo tem complexidade $O(n)$.

```

Algoritmo Isomorfismo_Grafos_k_caminho;
Entrada:   $n, k, \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle, \langle (t_1, u_1), \dots, (t_{p-1}, u_{p-1}) \rangle$ ;
Saída:   verdadeiro ou falso;
Início
   $p \leftarrow n - k$ ;
  Para  $v$  de 1 até  $n$  faça  $L^{-1}[v] \leftarrow T^{-1}[v] \leftarrow p; U^{-1}[v] \leftarrow 0$ ;
  Para  $i$  de 1 até  $p-1$  faça  $L^{-1}[l_i] \leftarrow T^{-1}[t_i] \leftarrow i; U^{-1}[u_i] \leftarrow i$ ;
   $resp1 \leftarrow resp2 \leftarrow$  verdadeiro;
  Para  $j$  de 1 até  $p-1$  faça
    Se  $L^{-1}[r_j] \neq T^{-1}[u_j]$  então  $resp1 \leftarrow$  falso;
    Se  $L^{-1}[r_j] + U^{-1}[t_{p-j}] \neq p$  então  $resp2 \leftarrow$  falso;
   $resp \leftarrow resp1 \vee resp2$ ;
  retorne  $resp$ ;
Fim

```

Algoritmo 4.2: Verificação de isomorfismo de grafos k -caminho.

Lema 4.10: A verificação de isomorfismo entre dois grafos k -caminho a partir de suas seqüências reduzidas possui complexidade $O(n)$.

Prova: Tanto a inicialização dos vetores auxiliares quanto os testes com estes vetores são feitos com complexidade $O(n)$. Portanto, o Algoritmo 4.2 possui claramente complexidade $O(n)$. \Rightarrow

4.4 - Obtenção de Caminho Hamiltoniano

A obtenção de um caminho hamiltoniano em um grafo é um problema muito importante em teoria dos grafos. Nesta seção, provamos que todo grafo k -caminho possui caminho hamiltoniano e apresentamos um algoritmo que utiliza a seqüência reduzida na obtenção de caminhos hamiltonianos.

Teorema 4.11: Todo grafo k -caminho possui caminho hamiltoniano.

Prova: A prova é construtiva, ou seja, é obtido um caminho hamiltoniano a partir da seqüência reduzida $S = \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$ de um grafo k -caminho $G = (V, E)$. Já foi visto na prova do Teorema 3.18 que $l_1 \neq \dots \neq l_{i-1} \neq l_i \neq r_i \neq r_{i+1} \neq \dots \neq r_{p-1}$. Sabe-se do Lema 3.9 que $r_j \neq l_{j+1}$ e que $C_j = \{l_j\} \cup C_{j+1} - \{r_j\}$ e $C_{j+1} = \{l_{j+1}\} \cup C_{j+2} - \{r_{j+1}\}$. Desse modo, tem-se $C_j = \{l_j, l_{j+1}\} \cup C_{j+2} - \{r_j, r_{j+1}\}$. Como l_j e l_{j+1} pertencem a C_j e os vértices r_j e r_{j+1} pertencem a C_{j+2} , então existem as arestas $\{l_j, l_{j+1}\}$ e $\{r_j, r_{j+1}\}$. Logo, existem os caminhos $\langle l_1, l_2, \dots, l_i \rangle$ e $\langle r_i, \dots, r_{p-1} \rangle$.

Sabe-se do Corolário 3.17 que $B_i = V_i - (L_i \cup R_i) = V - \{l_1, l_2, \dots, l_i, r_i, r_{i+1}, \dots, r_{p-1}\}$. Seja $B_i = \{v_1, \dots, v_k\}$. Logo, $\langle v_1, \dots, v_k \rangle$ é um caminho. Por definição, $B_i = C_i \cap C_{i+1}$, $l_i = C_i - C_{i+1}$ e $r_i = C_{i+1} - C_i$. Logo, l_i e r_i são adjacentes aos vértices de B_i e existem as arestas $\{l_i, v_1\}$ e $\{v_k, r_i\}$. Desse modo, $\langle l_1, l_2, \dots, l_i, v_1, v_2, \dots, v_k, r_i, \dots, r_{p-1} \rangle$ é um caminho hamiltoniano de G . \Rightarrow

O Algoritmo 4.3 obtém o caminho hamiltoniano diretamente da prova do Teorema 4.11 a partir da seqüência reduzida e do índice i . Na verdade, basta obter os vértices da k -clique B_i através do vetor auxiliar $BI[1..n]$. Armazena-se 1 na posição $BI[v]$ para indicar que v pertence a B_i e 0 caso contrário. O caminho hamiltoniano é então obtido pela junção de três subcaminhos: $\langle l_1, l_2, \dots, l_i \rangle$, $\langle v_1, v_2, \dots, v_k \rangle$ e $\langle r_i, \dots, r_{p-1} \rangle$.

```

Algoritmo Caminho_Hamiltoniano_Grafo_k_caminho;
Entrada:  $n, k, i, \langle (l_1, r_1), \dots, (l_{p-1}, r_{p-1}) \rangle$ ;
Saída: caminho hamiltoniano  $\langle u_1, u_2, \dots, u_n \rangle$ ;
Início
  Para  $v$  de 1 até  $n$  faça  $BI[v] \leftarrow 1$ ;
  Para  $j$  de 1 até  $i$  faça  $BI[l_j] \leftarrow 0$ ;  $u_j \leftarrow l_j$ ;
  Para  $j$  de  $i$  até  $n-k-1$  faça  $BI[r_j] \leftarrow 0$ ;  $u_{j+k+1} \leftarrow r_j$ ;
  Para  $v$  de 1 até  $n$  faça
    Se  $BI[v] = 1$  então
       $i \leftarrow i + 1$ ;
       $u_i \leftarrow v$ ;
Fim

```

Algoritmo 4.3: Obtenção de caminho hamiltoniano em um grafo k -caminho.

O Algoritmo 4.3 que obtém o caminho hamiltoniano de um grafo k -caminho a partir da seqüência reduzida possui claramente complexidade $O(n)$.

Capítulo 5 - Código de Prüfer para k -árvores Rotuladas

Na Seção 2.1, foi visto o código proposto por PRÜFER [41] para representar as árvores rotuladas, que são 1-árvores pela definição de k -árvores. Em 1970, RÉNYI e RÉNYI [42] expandiram esta codificação para k -árvores rotuladas que possuem uma clique formada pelos k maiores vértices (denominadas k -árvores de Rényi) e para k -árvores enraizadas. Neste capítulo, a abordagem destes autores é revista com o intuito de codificar k -árvores quaisquer. Procuramos organizar da melhor forma possível o que foi apresentado por RÉNYI e RÉNYI [42] para facilitar a coerência e encadeamento de idéias. Por isso, os resultados já existentes na literatura encontram-se, algumas vezes, intercalados com nossos resultados neste capítulo. Portanto, cabe salientar que além da revisão dos códigos para k -árvores sem raiz, contribuímos com algumas novas propriedades na Seção 5.3 e, principalmente, com os algoritmos de codificação e decodificação vistos na Seção 5.4.

Na Seção 5.1, inicialmente apresentamos a extensão do código de Prüfer para árvores denominada código redundante de Prüfer e, em seguida, mostramos a generalização do código de Prüfer e do código redundante de Prüfer para k -árvores quaisquer. Mais uma vez, cabe salientar que estes códigos foram definidos por RÉNYI e RÉNYI [42] apenas para k -árvores enraizadas. Na Seção 5.2, são vistas as regras propostas por RÉNYI e RÉNYI [42] que permitem verificar se uma seqüência de $n - k - 1$ cliques de tamanho k é o código de Prüfer de uma k -árvore e, na Seção 5.3, propriedades que podem ser obtidas a partir da codificação. Não encontramos algoritmos lineares para o código de Prüfer de k -árvores na literatura. Desse modo, na Seção 5.4, definimos uma estrutura de dados chamada lista de prioridades (L,U) -limitada e propomos algoritmos lineares de codificação e decodificação que a utilizam. Sem perda de generalidade, considera-se $V = \{1, \dots, n\}$ neste capítulo.

5.1 - Definição

O **código de Prüfer** $\langle b_1, b_2, \dots, b_{n-2} \rangle$ de uma árvore é obtido pela sucessiva remoção da menor folha a_i e adição de seu vizinho b_i ao código (Seção 2.1). As folhas são removidas até que reste apenas a aresta $\{a_{n-1}, b_{n-1}\}$ onde a_{n-1} é o menor vértice desta

aresta e b_{n-1} o maior. Chamamos b_{n-1} de **vértice residual**. Cabe lembrar que a árvore trivial não tem representação de Prüfer e o código de Prüfer de uma árvore com dois vértices é uma seqüência vazia.

O código de Prüfer pode ser naturalmente estendido, originando uma representação matricial $\begin{pmatrix} a_1 & a_2 & \dots & a_{n-1} \\ b_1 & b_2 & \dots & b_{n-1} \end{pmatrix}$ denominada **código redundante de Prüfer** onde a_i é a i -ésima folha removida e b_i seu vértice adjacente. O Algoritmo 5.1 mostra a construção deste código para uma árvore.

Observe que $\{a_1, b_1\}, \dots, \{a_{n-1}, b_{n-1}\}$ são as $n - 1$ arestas da árvore. Por isso, a decodificação do código tradicional $\langle b_1, b_2, \dots, b_{n-2} \rangle$ resume-se à obtenção do código redundante. Observe também que a aresta $\{a_{n-1}, b_{n-1}\}$ é formada pelos vértices que não foram removidos na decodificação. Logo, $\{a_{n-1}, b_{n-1}\} = V - \{a_1, \dots, a_{n-2}\}$. Particularmente, o vértice residual b_{n-1} é o maior vértice de V , pois toda árvore possui pelo menos duas folhas e, em todos os passos da codificação, existe uma folha menor que o maior vértice de V para ser removida.

```

Algoritmo Codificação_de_Prüfer_para_Árvores;
Entrada:  Árvore  $T = (V, E)$  tal que  $V = \{1, \dots, n\}$  e  $n > 2$ ;
Saída:   Código de Prüfer  $\langle b_1, \dots, b_{n-2} \rangle$ ,
        Código redundante  $\langle a_1, \dots, a_{n-1} \rangle, \langle b_1, \dots, b_{n-1} \rangle$ ;
Início
  Para  $i$  de 1 até  $n-2$  faça
     $a_i \leftarrow$  menor vértice  $v \in V$  com grau 1 (menor folha);
     $b_i \leftarrow$  vértice adjacente a  $a_i$ ;
     $V \leftarrow V - \{a_i\}$ ;
     $E \leftarrow E - \{a_i, b_i\}$ ;
   $a_{n-1} \leftarrow \min(V)$ ;
   $b_{n-1} \leftarrow \max(V)$ ;
Fim

```

Algoritmo 5.1: Codificação de Prüfer para árvores.

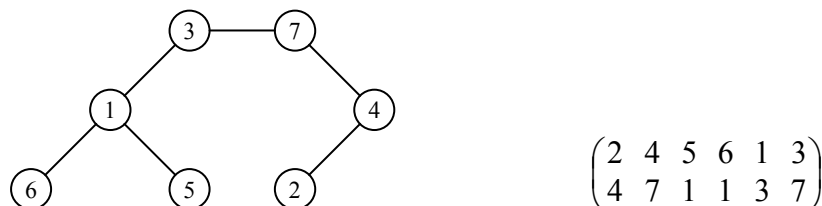


Figura 5.1: Árvore e seu código redundante de Prüfer.

A árvore cujo código de Prüfer é $\langle 4, 7, 1, 1, 3 \rangle$ é vista na Figura 5.1 juntamente com seu código redundante. As seis arestas $\{2,4\}$, $\{4,7\}$, $\{1,5\}$, $\{1,6\}$, $\{1,3\}$ e $\{3,7\}$ desta árvore podem ser obtidas diretamente do código redundante.

Seja $G = (V, E)$ uma k -árvore e $n > k + 1$. O **código de Prüfer** $\langle B_1, B_2, \dots, B_{n-k} \rangle$ de G é obtido pela sucessiva remoção da menor k -folha a_i (e de suas k arestas adjacentes) e adição da clique B_i , formada pelos k vértices adjacentes a a_i , ao código. A remoção das k -folhas é executada até que reste apenas a $(k+1)$ -clique $C_{n-k} = \{a_{n-k}\} \cup B_{n-k}$ onde a_{n-k} é o menor vértice de C_{n-k} . A clique B_{n-k} é chamada de **k -clique residual**. Cabe ressaltar que as k -árvores com k vértices não têm representação de Prüfer e o código de Prüfer das k -árvores com $k + 1$ vértices é uma seqüência vazia.

De modo similar ao feito para árvores, o código de Prüfer para k -árvores pode ser naturalmente estendido, originando uma representação matricial $\begin{pmatrix} a_1 & a_2 & \dots & a_{n-k} \\ B_1 & B_2 & \dots & B_{n-k} \end{pmatrix}$ denominada **código redundante de Prüfer para k -árvores** (a partir de agora referido como código redundante) onde a_i é a i -ésima k -folha removida e B_i é a clique formada pelos k vértices adjacentes a a_i . No texto, utiliza-se o par de seqüências $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$ ao invés da representação matricial. Note-se que o código de Prüfer e o código redundante para k -árvores contêm os apresentados para árvores; basta considerar $k = 1$. O Algoritmo 5.2 mostra a codificação de Prüfer e a codificação redundante para uma k -árvore.

```

Algoritmo Codificação_de_Prüfer_para_k-Árvores;
Entrada:  $k$ -árvore  $T = (V, E)$  tal que  $V = \{1, \dots, n\}$  e  $n > k + 1$ ;
Saída: Código de Prüfer  $\langle B_1, \dots, B_{n-k} \rangle$ ,
        Código redundante  $\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle$ ;
Início
    Para  $i$  de 1 até  $n - k - 1$  faça
         $a_i \leftarrow$  menor vértice  $v \in V$  com grau  $k$  (menor  $k$ -folha);
         $B_i \leftarrow$   $k$  vértices adjacentes a  $a_i$ ;
         $V \leftarrow V - \{a_i\}$ ;
        Remover de  $E$  as  $k$  arestas incidentes a  $a_i$ ;
     $a_{n-k} \leftarrow \min(V)$ ;
     $B_{n-k} \leftarrow V - \{a_{n-k}\}$ ;
Fim

```

Algoritmo 5.2: Codificação de Prüfer para k -árvores.

A Figura 5.2 apresenta um exemplo: a obtenção do código de Prüfer $\langle \{1,2\}, \{1,4\}, \{2,4\} \rangle$ e do código redundante $(\langle 3,5,1,2 \rangle, \langle \{1,2\}, \{1,4\}, \{2,4\}, \{4,6\} \rangle)$ para a 2-árvore da Figura 5.2(a). A cada passo, tanto a menor 2-folha a_i quanto suas arestas adjacentes são removidas e estão indicadas em pontilhado enquanto a 2-clique B_i , formada pelos vértices adjacentes a a_i , está em negrito. Observe que a Figura 5.2(d) é utilizada apenas na obtenção do código redundante. O vértice $a_4 = 2$ é o menor vértice da 3-clique $\{2,4,6\}$ e $B_4 = \{4,6\}$ é a 2-clique residual.

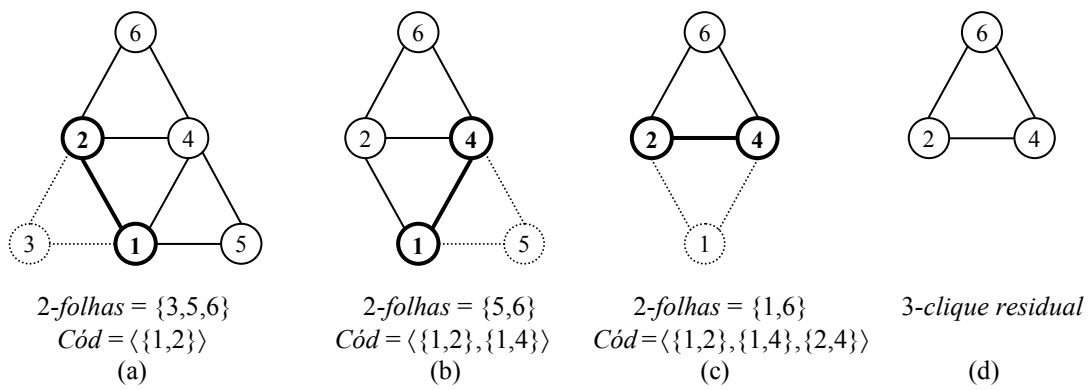


Figura 5.2: Passo a passo da obtenção do código de Prüfer de uma 2-árvore.

Uma das principais propriedades do código redundante é a obtenção direta das $n - k$ cliques maximais C_i de uma k -árvore, através da expressão $C_i = \{a_i\} \cup B_i$, para $1 \leq i \leq n - k$. Por exemplo, a 2-árvore da Figura 5.2 possui as quatro cliques maximais $\{1,2,3\}$, $\{1,4,5\}$, $\{1,2,4\}$ e $\{2,4,6\}$ que são facilmente visualizadas na representação

$$\text{matricial} \begin{pmatrix} 3 & 5 & 1 & 2 \\ \{1,2\} & \{1,4\} & \{2,4\} & \{4,6\} \end{pmatrix}.$$

A decodificação do código de Prüfer para k -árvores $\langle B_1, \dots, B_{n-k-1} \rangle$ consiste basicamente na obtenção do código redundante $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$, pois toda aresta da k -árvore está contida em pelo menos uma clique maximal. O lema a seguir mostra como determinar o código redundante a partir do código de Prüfer de uma k -árvore.

Lema 5.1: Sejam uma k -árvore $G = (V, E)$, $n > k > 0$ e $\langle B_1, \dots, B_{n-k-1} \rangle$ o código de Prüfer de G . O código redundante $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$ é obtido da seguinte forma:

- 1) a_1 é o menor vértice de $V - (B_1 \cup B_2 \cup \dots \cup B_{n-k-1})$;
- 2) a_j é o menor vértice de $V - \{a_1, \dots, a_{j-1}\} - (B_j \cup \dots \cup B_{n-k-1})$ para $1 < j < n - k$;
- 3) a_{n-k} é o menor vértice de $V - \{a_1, \dots, a_{n-k-1}\}$;
- 4) $B_{n-k} = V - \{a_1, \dots, a_{n-k}\}$.

Prova: Na codificação, os vértices a_1, \dots, a_{n-k-1} são removidos até que reste a $(k+1)$ -clique $\{a_{n-k}\} \cup B_{n-k}$. Logo, todo vértice de V ocorre uma vez em $a_1, \dots, a_{n-k}, B_{n-k}$. Sabe-se que uma k -folha é um vértice que pertence a apenas uma $(k+1)$ -clique e que as $n - k$ cliques maximais de uma k -árvore são obtidas pela expressão $\{a_i\} \cup B_i$. Assim, as k -folhas são os vértices que ocorrem apenas em $a_1, \dots, a_{n-k}, B_{n-k}$, ou seja, são os vértices de $V - (B_1 \cup \dots \cup B_{n-k-1})$. Como a_1 é a menor k -folha de G , obtém-se o item 1.

Seja G_j a k -árvore obtida pela remoção dos vértices a_1, \dots, a_{j-1} . Os vértices de $V - \{a_1, \dots, a_{j-1}\}$ ocorrem uma vez em $a_j, \dots, a_{n-k}, B_{n-k}$. Utilizando raciocínio análogo ao do parágrafo anterior, verifica-se que as k -folhas de G_j são os vértices que ocorrem apenas em $a_j, \dots, a_{n-k}, B_{n-k}$, ou seja, são os vértices de $V - \{a_1, \dots, a_{j-1}\} - (B_j \cup \dots \cup B_{n-k-1})$. Como a_j é a menor k -folha de G_j , obtém-se o item 2.

Na codificação, os vértices a_1, \dots, a_{n-k-1} são removidos até que reste a $(k+1)$ -clique $C_{n-k} = \{a_{n-k}\} \cup B_{n-k}$, ou seja, $C_{n-k} = V - \{a_1, \dots, a_{n-k-1}\}$. Como a_{n-k} é o menor vértice de C_{n-k} , obtém-se o item 3. O item 4 é obtido utilizando a expressão $B_{n-k} = C_{n-k} - \{a_{n-k}\}$. \square

5.2 - Validação

Como visto na Seção 2.1, qualquer seqüência $\langle b_1, \dots, b_{n-2} \rangle$ de $n - 2$ vértices é o código de Prüfer de uma árvore com n vértices. Diferentemente das árvores, nem toda seqüência $\langle B_1, \dots, B_{n-k-1} \rangle$ de conjuntos de tamanho k é o código de Prüfer de uma k -árvore. Por esta razão é necessário validar uma seqüência de conjuntos de tamanho k para garantir que esta seqüência represente uma k -árvore.

A validação de $P = \langle B_1, \dots, B_{n-k-1} \rangle$ é feita em dois passos. Primeiro, caso seja possível, obtém-se o par de seqüências $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$ através do Lema 5.1. No segundo passo, verifica-se as condições do Teorema 5.2, ou seja, se cada conjunto B_i é uma k -clique da k -árvore obtida após a remoção dos vértices a_1, \dots, a_i .

Teorema 5.2 [42]: A seqüência $\langle B_1, \dots, B_{n-k-1} \rangle$ de conjuntos de tamanho k é o código de Prüfer de uma k -árvore $G = (V, E)$ se e somente se $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$ pode ser obtido pelo Lema 5.1 e para $1 \leq i < n - k$, existe um inteiro j , $i < j \leq n - k$, tal que $B_i \subset C_j = \{a_j\} \cup B_j$.

Sejam três seqüências de conjuntos $P_a = \langle \{1,6,8\}, \{3,7,8\}, \{3,4,5\}, \{1,6,7\} \rangle$, $P_b = \langle \{1,6,8\}, \{3,7,8\}, \{1,3,5\}, \{1,3,8\} \rangle$ e $P_c = \langle \{1,6,8\}, \{3,7,8\}, \{1,3,6\}, \{1,3,8\} \rangle$ que representam ou não uma 3-árvore com 8 vértices. A Figura 5.3 mostra o par (ainda não reconhecido como válido) obtido pelo Lema 5.1 para cada uma destas seqüências. P_a não é um código de Prüfer porque não é possível determinar o vértice a_2 de acordo com o Lema 5.1, pois $V - \{a_1\} \cup B_2 \cup B_3 \cup B_4 = \emptyset$. Apesar de o Lema 5.1 ser satisfeito para P_b , esta seqüência não é um código de Prüfer porque a 3-clique $B_1 = \{1,6,8\}$ não está contida em nenhuma 4-clique. Finalmente, P_c é o código de Prüfer de uma 3-árvore porque, além de satisfazer o Lema 5.1, satisfaz o Teorema 5.2, pois $B_1 \subset C_4$, $B_2 \subset C_5$, $B_3 \subset C_4$ e $B_4 \subset C_5$.

$$\left(\begin{array}{ccccc} 2 & & & & \\ \{1,6,8\} & \{3,7,8\} & \{3,4,5\} & \{1,6,7\} & \{ _, _, _ \} \end{array} \right)$$

a) P_a não satisfaz o Lema 5.1.

$$\left(\begin{array}{ccccc} 2 & 4 & 6 & 5 & 1 \\ \{1,6,8\} & \{3,7,8\} & \{1,3,5\} & \{1,3,8\} & \{3,7,8\} \end{array} \right)$$

b) P_b não satisfaz o Teorema 5.2.

$$\left(\begin{array}{ccccc} 2 & 4 & 5 & 6 & 1 \\ \{1,6,8\} & \{3,7,8\} & \{1,3,6\} & \{1,3,8\} & \{3,7,8\} \end{array} \right)$$

c) P_c é o código de Prüfer de uma 3-árvore com oito vértices.

Figura 5.3: Aplicação do Lema 5.1 à três seqüências de conjuntos de tamanho 3.

5.3 - Propriedades

Algumas propriedades de uma k -árvore podem ser obtidas diretamente de seu código de Prüfer ou de seu código redundante sem que haja necessidade de conhecer o conjunto de suas arestas. Por exemplo, as cliques maximais de uma k -árvore são determinadas de modo trivial pela simples inspeção do seu código redundante. Nesta seção, são vistos lemas, corolários e teoremas que expressam propriedades de uma k -árvore e resultam diretamente de seu código. Alguns destes resultados são de RÉNYI e RÉNYI [42] (assinalados explicitamente no texto) e outros são contribuições nossas.

O Lema 5.3 mostra que, em uma k -árvore, existe uma relação entre o grau de um vértice e a quantidade de ocorrências deste vértice no código de Prüfer de G . A partir da inspeção do mesmo, o lema mostra como obter os graus dos vértices e o corolário, seus vértices simpliciais.

Lema 5.3 [42]: Sejam uma k -árvore $G = (V, E)$, $n > k$ e $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$ o seu código redundante. Um vértice $v \in V$ aparece uma vez em $\langle a_1, \dots, a_{n-k}, B_{n-k} \rangle$ e $d(v) - k$ vezes em $\langle B_1, \dots, B_{n-k-1} \rangle$.

Corolário 5.4: Sejam uma k -árvore $G = (V, E)$, $n > k$ e $\langle B_1, \dots, B_{n-k-1} \rangle$ o seu código de Prüfer. Um vértice $v \in V$ é simplicial quando não ocorre no código de Prüfer de G .

Prova: Pelo Lema 1.5, sabe-se que os vértices simpliciais de uma k -árvore têm grau k . Pelo lema anterior, este vértice ocorre 0 vezes em B_1, \dots, B_{n-k-1} , ou seja, v não ocorre no código de Prüfer de G . \square

A k -folha a_i é um vértice simplicial no subgrafo obtido pela remoção dos vértices a_1, \dots, a_{i-1} . Este fato é utilizado no próximo teorema para obter um esquema de eliminação perfeita a partir do código redundante de uma k -árvore.

Teorema 5.5: Sejam uma k -árvore $G = (V, E)$, $n > k$ e $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$ o código redundante de G . A seqüência $\langle a_1, \dots, a_{n-k}, B_{n-k} \rangle$ é um esquema de eliminação perfeita.

Prova: Pelo Lema 5.3, todo vértice de V ocorre uma vez em $\langle a_1, \dots, a_{n-k}, B_{n-k} \rangle$. Logo, estes n vértices são distintos entre si. Pela definição do código de Prüfer, o vértice a_i é uma k -folha no subgrafo obtido pela remoção dos vértices a_1, \dots, a_{i-1} , ou seja, o vértice a_i é simplicial no subgrafo induzido pelos vértices $a_i, a_{i+1}, \dots, a_{n-k}, B_{n-k}$. Como o subgrafo induzido por B_{n-k} é um grafo completo, qualquer ordenação de seus vértices é um esquema de eliminação perfeita. \square

RÉNYI e RÉNYI [42] definiram o grau de uma k -clique que, neste texto, será chamado de **multiplicidade**. O Lema 5.7 mostra a relação entre a multiplicidade de uma k -clique em uma k -árvore e a quantidade de vezes que esta k -clique aparece no seu código de Prüfer.

Definição 5.6 [42]: Sejam uma k -árvore $G = (V, E)$, $n > k$ e uma k -clique Q de G . A multiplicidade de Q , denotada $\mathit{mult}(Q)$, é o número de $(k+1)$ -cliques que contêm Q .

Lema 5.7 [42]: Sejam uma k -árvore $G = (V, E)$, $n > k$ e $\langle B_1, \dots, B_{n-k-1} \rangle$ o seu código de Prüfer. Uma k -clique Q de G aparece $\mathit{mult}(Q) - 1$ vezes em $\langle B_1, \dots, B_{n-k-1} \rangle$.

Por exemplo, na 2-árvore da Figura 5.2(a), a 2-clique $\{1,2\}$ possui multiplicidade 2 porque está contida nas 3-cliques $\{1,2,3\}$ e $\{1,2,4\}$ enquanto a 2-clique $\{1,3\}$ possui multiplicidade 1 porque está contida apenas em $\{1,2,3\}$. As multiplicidades destas 2-cliques podem ser determinadas pela inspeção do código de Prüfer $\langle \{1,2\}, \{1,4\}, \{2,4\} \rangle$, onde $\{1,2\}$ ocorre uma vez e não há ocorrência da clique $\{1,3\}$.

O código redundante fornece um esquema de construção de uma k -árvore G com base na Definição 1.2 de k -árvores. Inicia-se com as arestas da $(k+1)$ -clique $\{a_{n-k}\} \cup B_{n-k}$ e, para $n - k - 1 \leq i \leq 1$, adicionam-se as k arestas adjacentes a a_i e aos k vértices de B_i . Desse modo, nesta construção de G , cada clique B_i é escolhida $\mathit{mult}(B_i) - 1$ vezes.

5.4 - Algoritmos Lineares de Codificação e Decodificação

Não encontramos algoritmos lineares para codificação e decodificação do código de Prüfer de k -árvores na literatura. A implementação direta da definição, utilizando lista de prioridades, leva a algoritmos com complexidade $O(n \log n)$. Em [33], definimos um caso particular de listas de prioridades que chamamos de **lista de prioridade (L, U) -limitada** e a utilizamos na modelagem de algoritmos lineares para o código de Prüfer de k -árvores de Rényi, que são k -árvores cujos k maiores vértices formam uma k -clique. Nesta seção, apresentamos estes algoritmos com as alterações necessárias para serem aplicados a k -árvores quaisquer.

Uma lista de prioridades Q é uma coleção de elementos onde cada um deles possui uma prioridade inteira. Em Q , são possíveis as seguintes operações:

- *Cria_Lista(Q)*: inicializa a lista vazia.
- *Inserir(Q, pri, el)*: insere o elemento el cuja prioridade é pri .
- *Remove_Min(Q, pri, el)*: retorna o elemento de Q cuja prioridade é mínima.

Uma lista de prioridades (L,U) -limitada é um caso particular de lista de prioridades onde as prioridades estão definidas apenas entre dois inteiros L e U . Além disso, uma seqüência de operações é **válida** em uma lista de prioridades Q quando:

- Inicia com a operação $Cria_Lista(Q, L, U)$;
- O i -ésimo $Remove_Min$ é precedido de pelo menos i operações $Inserir$;
- Imediatamente após cada $Remove_Min$, ocorre no máximo uma operação $Inserir$.

Em [33], provamos que uma seqüência válida de operações em uma lista de prioridades possui complexidade linear em relação à quantidade de operações e ao tamanho do intervalo de prioridades no pior caso. O Teorema 5.8 ilustra este resultado.

Teorema 5.8 [33]: Em uma lista de prioridades (L,U) -limitada, qualquer seqüência válida contendo exatamente t operações $Inserir$ pode ser executada com complexidade $O(t+U-L)$ no pior caso.

Observamos que nos algoritmos de codificação e decodificação vistos adiante ocorrerão no máximo n operações $Inserir$ em uma lista de prioridades $(1,n)$ -limitada Q . Então, podemos adiantar que, pelo Teorema 5.8, a complexidade da seqüência de operações envolvendo Q nestes algoritmos é $O(n)$ no pior caso.

A codificação consiste em obter o código de Prüfer $\langle B_1, \dots, B_{n-k-1} \rangle$ a partir da k -árvore, que aqui consideraremos representada por listas de adjacência. Além do código de Prüfer, o Algoritmo 5.3 computa as k -folhas $\langle a_1, \dots, a_{n-k-1} \rangle$ removidas durante a codificação; esta saída do algoritmo será utilizada na Seção 6.4 do próximo capítulo.

A decodificação consiste em obter as arestas da k -árvore a partir do código de Prüfer. É feita em duas etapas. Primeiro, obtém-se o código redundante de Prüfer utilizando o Algoritmo 5.4 e o Algoritmo 5.5. Em seguida, utiliza-se o Algoritmo 5.6 para obter as arestas da k -árvore. A Figura 5.4 resume a utilização destes algoritmos na codificação e decodificação.

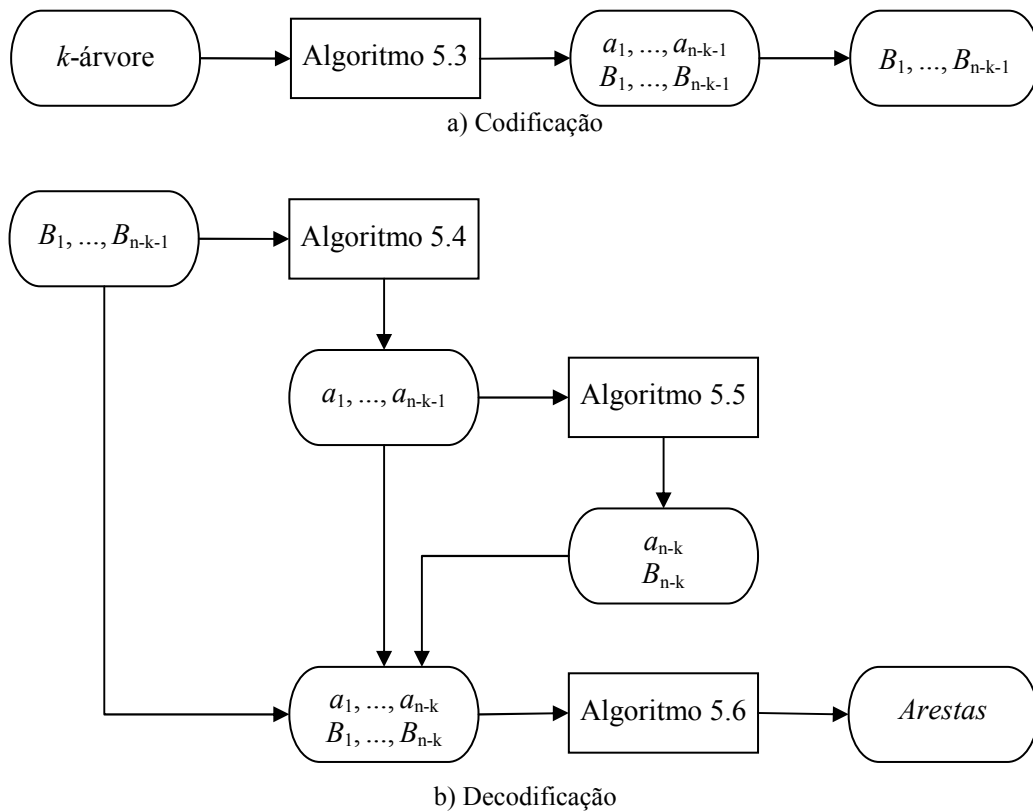


Figura 5.4: Representação esquemática da codificação e decodificação.

Codificação

O Algoritmo 5.3 de codificação utiliza a lista de prioridades $(1, n)$ -limitada Q para armazenar as k -folhas da k -árvore, que são os vértices de grau k . Inicialmente, calculam-se os graus dos vértices e todos os que possuem grau k são inseridos em Q . Cada k -folha v é armazenada em Q com prioridade v . Assim, na iteração i , a remoção de um vértice da lista de prioridade Q fornece a menor k -folha a_i enquanto a k -clique B_i é formada pelos k vértices adjacentes a a_i no grafo corrente. Após remover o vértice a_i , caso um dos vértices de B_i possua grau k , insere-se este vértice na lista Q das k -folhas.

Atribui-se grau zero aos vértices a_i removidos da k -árvore para sinalizar que estes vértices não pertencem mais à k -árvore. Desse modo, na iteração i , dentre os vértices da lista de adjacências de a_i , apenas os k vértices de B_i possuem grau diferente de zero.

Como foi provado em [33] que a seqüência de operações na lista Q tem complexidade $O(n)$ e na iteração principal $n - k - 1$ vértices têm sua lista de adjacências visitada, portanto $O(m)$, então o Algoritmo 5.3 possui complexidade $O(m)$.

Algoritmo Codificação_Linear_de_Prüfer_para_k-Árvores;
Entrada: $n \geq k+1$, $Adj(v) \forall v \in V$ de uma k -árvore;
Saída: cód Prüfer $\langle B_1, \dots, B_{n-k-1} \rangle$, vértices $\langle a_1, \dots, a_{n-k-1} \rangle$;
Início
 Para v de 1 até n faça $grau[v] \leftarrow |Adj(v)|$;
 Cria_Lista($Q, 1, n$);
 Para i de 1 até n faça
 Se $grau[v] = k$ então
 Inserir(Q, v, v);
 Para i de 1 até $n-k-1$ faça
 Remove_Min(Q, a_i, a_i);
 $grau[a_i] \leftarrow 0$;
 $B_i \leftarrow \emptyset$;
 Para todo $v \in Adj(a_i) \mid grau[v] > k$ faça
 $B_i \leftarrow B_i \cup \{v\}$;
 $grau[v] \leftarrow grau[v]-1$;
 Se ($grau[v] = k$) \wedge ($i \neq n-k-1$) então
 Inserir(Q, v, v);
Fim

Algoritmo 5.3: Codificação linear de Prüfer para uma k -árvore.

Decodificação

O Algoritmo 5.4 obtém os vértices $\langle a_1, \dots, a_{n-k-1} \rangle$ a partir do código de Prüfer $\langle B_1, \dots, B_{n-k-1} \rangle$ com base nos itens 1 e 2 do Lema 5.1. Utiliza-se o vetor $freq[1..n]$ para armazenar a quantidade de vezes que cada vértice aparece nos conjuntos $B_i, B_{i+1}, \dots, B_{n-k-1}$. A lista de prioridades $(1,n)$ -limitada Q armazena os vértices com frequência zero, ou seja, os vértices que não pertencem ao conjunto $B_i \cup \dots \cup B_{n-k-1}$. Assim, na iteração i , o vértice removido de Q é o vértice a_i que satisfaz o Lema 5.1.

Como foi provado em [33] que a seqüência de operações na lista Q tem complexidade $O(n)$ e na iteração principal as $n - k - 1$ cliques B_i têm seus vértices explorados, então o Algoritmo 5.4 possui complexidade $O(m)$.

O Algoritmo 5.5 obtém o vértice a_{n-k} e a clique residual B_{n-k} a partir dos vértices a_1, \dots, a_{n-k-1} utilizando os itens 3 e 4 do Lema 5.1. Este algoritmo possui claramente complexidade $O(n)$, pois todas as repetições possuem no máximo n iterações. O vetor auxiliar $aux[1..n]$ possui valor 1 quando $v \in \{a_1, \dots, a_{n-k-1}\}$ e 0 caso contrário. Assim, a $(k+1)$ -clique $C_{n-k} = V - \{a_1, \dots, a_{n-k-1}\}$ é formada pelos vértices que possuem valor 0 no vetor aux . Dentre os vértices de C_{n-k} , a_{n-k} é o menor e a clique B_{n-k} é formada pelos demais vértices.

Algoritmo Obtenção_da_seqüência_de_vértices_removidos;
Entrada: código de Prüfer $\langle B_1, \dots, B_{n-k-1} \rangle$;
Saída: vértices removidos $\langle a_1, \dots, a_{n-k-1} \rangle$;
Início
 Para v de 1 até n faça $freq[v] \leftarrow 0$;
 Para i de 1 até $n-k-1$ faça
 Para todo $v \in B_i$ faça
 $freq[v] \leftarrow freq[v] + 1$;
 Cria_Lista($Q, 1, n$);
 Para i de 1 até n faça
 Se $freq[v] = 0$ então
 Inserir(Q, v, v);
 Para i de 1 até $n-k-1$ faça
 Remove_Min(Q, a_i, a_i);
 Para todo $v \in B_i$ faça
 $freq[v] \leftarrow freq[v] - 1$;
 Se $freq[v] = 0 \wedge (i \neq n-k-1)$ então
 Inserir(Q, v, v);
Fim

Algoritmo 5.4: Obtenção da seqüência $\langle a_1, \dots, a_{n-k-1} \rangle$ de vértices removidos.

Algoritmo Obtenção_de_ a_{n-k} _e_ B_{n-k} ;
Entrada: vértices $\langle a_1, \dots, a_{n-k-1} \rangle$;
Saída: vértice a_{n-k} , clique residual B_{n-k} ;
Início
 Para v de 1 até n faça $aux[v] \leftarrow 0$;
 Para i de 1 até $n-k-1$ faça $aux[a_i] \leftarrow 1$;
 $B_{n-k} \leftarrow \emptyset$;
 $a_{n-k} \leftarrow 0$;
 Para v de 1 até n faça
 Se $aux[v] = 0$
 Se $a_{n-k} = 0$ então
 $a_{n-k} \leftarrow v$;
 Senão
 $B_{n-k} \leftarrow B_{n-k} \cup \{v\}$;
Fim

Algoritmo 5.5: Obtenção do vértice a_{n-k} e da clique residual B_{n-k} .

Sabe-se que o reverso do esquema de eliminação perfeita fornece um esquema de construção de um grafo cordal. O Algoritmo 5.6 obtém as arestas de uma k -árvore a partir do código redundante com complexidade $O(n.k) = O(m)$ utilizando o inverso do esquema de eliminação perfeita mostrado no Teorema 5.5. Inicialmente, inserem-se todas as $k(k-1)/2$ arestas relativas à k -clique B_{n-k} . Em seguida, para $n-k \leq i \leq 1$, inserem-se k arestas adjacentes ao vértice a_i e aos vértices da k -clique B_i . Cabe ressaltar que as $k(k-1)/2$ arestas relativas a k -clique B_i já pertencem ao conjunto de arestas quando são inseridas as k arestas adjacentes ao vértice a_i e aos vértices da k -clique B_i .

Algoritmo Obtenção_das_Arestas;
Entrada: código redundante $\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle$;
Saída: conjunto de arestas E ;
Início
 $E \leftarrow \emptyset$;
Para todo $u \in B_{n-k}$ faça
 Para todo $v \in B_{n-k}$ faça
 Se $u < v$ então
 $E \leftarrow E \cup \{u, v\}$;
 Para i de $n-k$ até 1 faça
 Para todo $v \in B_i$ faça
 $E \leftarrow E \cup \{v, a_i\}$;
Fim

Algoritmo 5.6: Obtenção das arestas de uma k -árvore a partir do seu código redundante.

No próximo capítulo, propomos uma nova codificação para k -árvores e a chamamos de código compacto. O Algoritmo 5.3, o Algoritmo 5.5 e o Algoritmo 5.6 são utilizados na Seção 6.4 do próximo capítulo, que trata da codificação e da decodificação do novo código.

Capítulo 6 - Código Compacto para k -árvores Rotuladas

A codificação de uma família de grafos tem tradicionalmente como principais objetivos a redução do espaço de armazenamento, a contagem e a geração aleatória uniforme de exemplares. Para alguns códigos, é possível determinar propriedades que resolvam problemas de modo mais eficiente do que utilizando a representação tradicional de grafos. Neste capítulo, propomos uma nova codificação para as k -árvores rotuladas, que chamamos de **código compacto**, a partir de uma simplificação do código redundante. A principal vantagem do código proposto é o fato de ser bem conciso, pois tem complexidade $O(n)$ para espaço de armazenamento enquanto o código de Prüfer, o código redundante e a representação tradicional de grafos têm complexidade $O(m) = O(n.k)$.

Paralelamente ao desenvolvimento de nossa pesquisa, CAMINITI *et al.* [9] publicaram em 2007 um trabalho sobre codificação de k -árvores. Estes autores propõem outro código que também possui complexidade de armazenamento $O(n)$. Entretanto, além da obtenção das cliques maximais, seus algoritmos de codificação e decodificação efetuam uma re-rotulação dos vértices com vistas a obter uma k -árvore de Rényi, além de necessitar de outros passos intermediários. A codificação e a decodificação do nosso código apresentados nesse capítulo, além de mais intuitivas, são mais simples e eficientes porque utilizam apenas o código redundante de Prüfer. Além disso, nossa codificação é utilizada na solução de problemas e na obtenção de propriedades de uma k -árvore diretamente de seu código.

Na Seção 6.1, apresentamos teoremas que fundamentam a definição do código compacto. Ainda nesta seção, definimos o código compacto e mostramos sua decodificação, ou seja, como obter o código redundante a partir do código compacto. Na Seção 6.2, propomos regras que permitem verificar a validade do código compacto de uma k -árvore. Finalmente, na Seção 6.3, apresentamos algoritmos lineares para codificação e decodificação. Sem perda de generalidade, considera-se $V = \{1, \dots, n\}$ neste capítulo. A utilização do código para solucionar de maneira eficiente alguns problemas é assunto do próximo capítulo. Os primeiros resultados obtidos para o código compacto podem ser encontrados em [37].

6.1 - Definição

O Teorema 6.1 e o Corolário 6.2 apresentados a seguir sustentam a definição do código compacto a partir de uma simplificação do código redundante.

Teorema 6.1: Sejam uma k -árvore $G = (V, E)$ e $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$ seu código redundante. Então, para cada k -clique $B_i \neq B_{n-k}$, existe um único inteiro j , $i < j \leq n-k$, tal que $a_j \in B_i$ e $B_i - \{a_j\} \subset B_j$.

Prova: Sejam $B_i \neq B_{n-k}$ e y o maior inteiro tal que $B_i = B_y$. Assim, $i \leq y < n-k$. Pelo Teorema 5.2, $B_y \subset \{a_j\} \cup B_j$ para algum j tal que $y < j \leq n-k$. Pela definição de y , $B_y \neq B_j$. Logo, $a_j \in B_y$ e $B_y - \{a_j\} \subset B_j$.

A prova da unicidade é por contradição. Suponha por contradição que além do j existe outro inteiro j' tal que $a_{j'} \in B_i$ e $B_i - \{a_{j'}\} \subset B_{j'}$. Como $j \neq j'$ e os vértices a_i são distintos, tem-se que $a_j \neq a_{j'}$. Assim, $a_j \in B_i - \{a_{j'}\} \subset B_{j'}$ e $a_{j'} \in B_i - \{a_j\} \subset B_j$, ou seja, $a_j \in B_{j'}$ e $a_{j'} \in B_j$. Isso é um absurdo porque, pelo item 2 do Lema 5.1, $a_{j'} \notin B_j$ se $j' < j$ e $a_j \notin B_{j'}$ se $j < j'$. Logo j é único. \square

O corolário a seguir mostra que, além de o índice j ser único, a_j é o primeiro vértice da k -clique B_i a aparecer em $\langle a_{i+1}, \dots, a_{n-k} \rangle$.

Corolário 6.2: Sejam a clique B_i e um inteiro j que satisfaz as condições do Teorema 6.1. Então, $j = \min \{t \mid i < t \leq n-k \wedge a_t \in B_i\}$.

Prova: Durante a codificação, logo após os vértices a_{i+1}, \dots, a_{j-1} serem removidos do grafo, remove-se o vértice a_j que é adjacente aos vértices da k -clique B_j . Logo, $a_{i+1}, \dots, a_j \notin B_j$. Pelo Teorema 6.1, $a_j \in B_i$ e $B_i - \{a_j\} \subset B_j$. Assim, conclui-se que $a_{i+1}, \dots, a_{j-1} \notin B_i$. Como $a_j \in B_i$, j é o mínimo indicado no enunciado. \square

Observe que $B_i - B_j$ e $B_j - B_i$ são conjuntos unitários, pois $B_i \cap B_j = B_i - \{a_j\}$ e $|B_i \cap B_j| = k - 1$. Na definição a seguir, por abuso de notação, utilizamos o vértice ao invés do conjunto unitário que o contém.

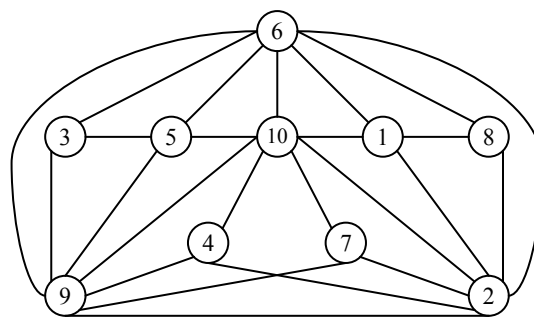
Definição 6.3: Sejam uma k -árvore $G = (V, E)$, $n > k + 1$ e seu código redundante $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$. O **código compacto** de G , $CC(G)$, é composto pela k -clique B e pela seqüência de pares de vértices $(\langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$, obtidos do seguinte modo:

- $B = B_{n-k}$.
- Para $i = 1, \dots, n - k - 1$:
 - Se $B_i = B_{n-k}$, então $(\alpha_i, \beta_i) = (a_{n-k}, a_{n-k})$;
 - Se $B_i \neq B_{n-k}$, então $(\alpha_i, \beta_i) = (B_i - B_j, B_j - B_i)$, sendo $j = \min\{t \mid i < t \leq n - k \wedge a_t \in B_i\}$.

Cabe ressaltar que o código compacto não é definido para k -árvores com k vértices e o código compacto para as k -árvores com $k + 1$ vértices é formado pela k -clique B e por uma seqüência vazia.

Observe que, para $B_i \neq B_{n-k}$, $\alpha_i = a_j$. Isto ocorre, pois, pela definição de j , $a_j \in B_i$ e $a_j \notin B_j$ porque o vértice a_j é adjacente aos vértices de B_j . Na obtenção do código compacto, esta relação dispensa o cálculo de $B_i - B_j$. Além disso, utiliza-se esta igualdade nas provas de alguns teoremas.

A Figura 6.1 ilustra uma 3-árvore com 10 vértices e seu código redundante enquanto a Figura 6.2 mostra a obtenção, passo a passo, do código compacto $(\{6,9,10\}, \langle (5,10), (2,6), (2,2), (2,6), (1,10), (2,9) \rangle)$ desta 3-árvore. No passo i , o vértice a_j é o primeiro vértice de B_i que aparece na subseqüência $\langle a_{i+1}, \dots, a_7 \rangle$. Por exemplo, no passo 1, o vértice 5 é o primeiro vértice de $\{5,6,9\}$ a ocorrer em $\langle 4, 5, 7, 8, 1, 2 \rangle$ e, no passo 5, o vértice 1 é o primeiro vértice de $\{1,2,6\}$ a ocorrer em $\langle 1, 2 \rangle$.



a) representação gráfica

$$\left(\begin{array}{cccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{5,6,9\} & \{2,9,10\} & \{6,9,10\} & \{2,9,10\} & \{1,2,6\} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

b) código redundante

Figura 6.1: 3-árvore com 10 vértices.

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{5,6,9\} & \{2,9,10\} & \{6,9,10\} & \{2,9,10\} & \{1,2,6\} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

$$(\alpha_1, \beta_1) = (B_1 - B_3, B_3 - B_1) = (5, 10)$$

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{5,6,9\} & \{2,9,10\} & \{6,9,10\} & \{2,9,10\} & \{1,2,6\} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

$$(\alpha_2, \beta_2) = (B_2 - B_7, B_7 - B_2) = (2, 6)$$

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{5,6,9\} & \{2,9,10\} & \{6,9,10\} & \{2,9,10\} & \{1,2,6\} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

$$(\alpha_3, \beta_3) = (a_7, a_7) = (2, 2)$$

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{5,6,9\} & \{2,9,10\} & \{6,9,10\} & \{2,9,10\} & \{1,2,6\} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

$$(\alpha_4, \beta_4) = (B_4 - B_7, B_7 - B_4) = (2, 6)$$

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{5,6,9\} & \{2,9,10\} & \{6,9,10\} & \{2,9,10\} & \{1,2,6\} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

$$(\alpha_5, \beta_5) = (B_5 - B_6, B_6 - B_5) = (1, 10)$$

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{5,6,9\} & \{2,9,10\} & \{6,9,10\} & \{2,9,10\} & \{1,2,6\} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

$$(\alpha_6, \beta_6) = (B_6 - B_7, B_7 - B_6) = (2, 9)$$

$$CC(G) = \langle \{6,9,10\}, (5,10), (2,6), (2,2), (2,6), (1,10), (2,9) \rangle$$

Figura 6.2: Obtenção, passo a passo, do código compacto da 3-árvore da Figura 6.1.

O vértice α_{n-k-1} do código compacto possui a peculiaridade de ser sempre igual ao vértice a_{n-k} do código redundante. No próximo lema, provamos esta relação, que é muito utilizada nas provas de alguns teoremas e em alguns algoritmos.

Lema 6.4: Sejam uma k -árvore $G = (V, E)$, $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$ seu código redundante e $(B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ o seu código compacto. Então, $a_{n-k} = \alpha_{n-k-1}$.

Prova: Seja $i = n - k - 1$. Pela Definição 6.3 de código compacto existem dois casos.

- $B_{n-k-1} = B_{n-k}$ e $(\alpha_{n-k-1}, \beta_{n-k-1}) = (a_{n-k}, a_{n-k})$. Logo, $\alpha_{n-k-1} = a_{n-k}$.
- $B_{n-k-1} \neq B_{n-k}$, $(\alpha_{n-k-1}, \beta_{n-k-1}) = (B_{n-k-1} - B_j, B_j - B_{n-k-1})$ e $j = \min\{t \mid n-k-1 < t \leq n-k \wedge a_t \in B_{n-k-1}\}$. Logo, $j = n-k$, $a_{n-k} \in B_{n-k-1}$ e $\{\alpha_{n-k-1}\} = B_{n-k-1} - B_{n-k}$. Como $a_{n-k} \notin B_{n-k}$ e $a_{n-k} \in B_{n-k-1}$, tem-se que $\alpha_{n-k-1} = a_{n-k}$. \square

O Teorema 6.1 e seu Corolário 6.2 permitem determinar inequivocamente os pares (α_i, β_i) do código compacto a partir do código redundante. Desse modo, existe um único código compacto correspondendo a um código redundante. O Teorema 6.6 mostra que a recíproca é verdadeira; o Lema 6.5, visto a seguir, é utilizado para auxiliar na sua prova.

Lema 6.5: Sejam uma k -árvore $G = (V, E)$, $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$ o seu código redundante e $(B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ seu código compacto. Então, para $1 \leq i < n-k$:

$$B_i \cup \dots \cup B_{n-k} \cup \{a_{n-k}\} = \{\alpha_i\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B.$$

Prova: É feita por indução no tamanho da seqüência com base em uma variação regressiva do índice i . As igualdades $a_{n-k} = \alpha_{n-k-1}$ provada no lema anterior e $B = B_{n-k}$ da definição de código compacto são amplamente utilizadas nesta prova.

- Seja $i = n-k-1$. Existem dois casos pela Definição 6.3 de código compacto.
 - $B_{n-k-1} = B_{n-k}$ e obviamente $B_{n-k-1} \cup B_{n-k} \cup \{a_{n-k}\} = \{\alpha_{n-k-1}\} \cup B$.
 - $B_{n-k-1} \neq B_{n-k}$ e $\{\alpha_{n-k-1}\} = B_{n-k-1} - B_{n-k}$. Assim, por manipulação algébrica verifica-se que:
$$\{\alpha_{n-k-1}\} \cup B = \{\alpha_{n-k-1}\} \cup B_{n-k} \cup \{a_{n-k}\}$$

$$\{\alpha_{n-k-1}\} \cup B = (B_{n-k-1} - B_{n-k}) \cup B_{n-k} \cup \{a_{n-k}\}$$

Como $(X - Y) \cup Y = X \cup Y$, tem-se:

$$\{\alpha_{n-k-1}\} \cup B = B_{n-k-1} \cup B_{n-k} \cup \{a_{n-k}\}.$$

Logo, nos dois casos, $B_{n-k-1} \cup B_{n-k} \cup \{a_{n-k}\} = \{\alpha_{n-k-1}\} \cup B$ e o lema vale para $i = n-k-1$.

- Suponha que o resultado vale para um $i+1$ tal que $1 \leq i < n-k-1$. Desse modo, tem-se: $B_{i+1} \cup \dots \cup B_{n-k} \cup \{a_{n-k}\} = \{\alpha_{i+1}\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B$. (1)
- Pela Definição 6.3 de código compacto, existem dois casos.
 - $B_i = B_{n-k} = B$ e $\alpha_i = a_{n-k}$. Fazendo a união de B_i e $\{a_{n-k}\}$ ao lado esquerdo da expressão (1) e de $\{\alpha_i\}$ e B ao lado direito, obtém-se o resultado do teorema.

• $B_i \neq B_{n-k}$ e $\{\alpha_i\} = B_i - B_j$. Fazendo a união de $B_i - B_j$ ao lado esquerdo da expressão (1) e de $\{\alpha_i\}$ ao lado direito, obtém-se:

$$(B_i - B_j) \cup B_{i+1} \cup \dots \cup B_j \cup \dots \cup B_{n-k} \cup \{a_{n-k}\} = \{\alpha_i\} \cup \{\alpha_{i+1}\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B.$$

Como $(B_i - B_j) \cup B_j = B_i \cup B_j$, obtém-se o resultado do teorema. \square

Teorema 6.6: Sejam uma k -árvore $G = (V, E)$ e $(B, \langle(\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1})\rangle)$ o seu código compacto. O código redundante $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$ de G é unicamente determinado do seguinte modo:

- 1) $B_{n-k} = B$;
- 2) a_1 é o menor vértice de $V - (\{\alpha_1\} \cup \{\alpha_2\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B)$;
- 3) a_j é o menor vértice de $V - \{a_1, \dots, a_{j-1}\} - (\{\alpha_1\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B)$, para $1 < j < n - k$;
- 4) $a_{n-k} = \alpha_{n-k-1}$;
- 5) $B_i = \{\alpha_i\} \cup B_j - \{\beta_i\}$, onde j é tal que $\alpha_i = a_j$ para $n - k > i \geq 1$.

Prova: O item 1 é obtido diretamente da Definição 6.3. No lema anterior, quando $i = 1$ tem-se $B_1 \cup \dots \cup B_{n-k} \cup \{a_{n-k}\} = \{\alpha_1\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B$. Pelo Lema 5.1, a_1 é o menor vértice de $V - (B_1 \cup \dots \cup B_{n-k-1})$. Infere-se do Teorema 5.5 que $a_1 \notin B_{n-k} \cup \{a_{n-k}\}$. Logo, obtém-se o item 2.

Para $1 < j < n - k$, pelo Lema 6.5, $B_j \cup \dots \cup B_{n-k} \cup \{a_{n-k}\} = \{\alpha_j\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B$ e do Lema 5.1 que a_j é o menor vértice de $V - \{a_1, \dots, a_{j-1}\} - (B_j \cup \dots \cup B_{n-k-1})$. Infere-se do Teorema 5.5 que $a_j \notin B_{n-k} \cup \{a_{n-k}\}$, para $1 \leq j < n - k$. Logo, obtém-se o item 3. O item 4 é válido devido ao Lema 6.4.

Para o item 5 são analisados os dois casos da Definição 6.3 de código compacto.

- $B_i = B_{n-k}, j = n - k$ e $\alpha_i = \beta_i = a_{n-k}$. Logo, o item 5 vale trivialmente.
- $B_i \neq B_{n-k}, \{\alpha_i\} = B_i - B_j$ e $\{\beta_i\} = B_j - B_i$. Assim, $B_i \cap B_j = B_i - \{\alpha_i\} = B_j - \{\beta_i\}$. Logo, $B_i = \{\alpha_i\} \cup B_j - \{\beta_i\}$ e o item 5 é válido. \square

A Figura 6.3 mostra como obter o código redundante da 3-árvore da Figura 6.1 a partir do seu código compacto $(\{6,9,10\}, \langle(5,10), (2,6), (2,2), (2,6), (1,10), (2,9)\rangle)$. Inicialmente, obtém-se os vértices a_1, \dots, a_{n-k} utilizando os itens 2, 3 e 4 do Teorema 6.6 e, em seguida, calcula-se as cliques B_i utilizando o item 5. Como $\alpha_i = a_j$, procura-se pelo vértice α_i em a_{i+1}, \dots, a_{n-k} para obter o índice j .

$$\begin{aligned}
a_1 &= \min (\{1, \dots, 10\} - \{5\} \cup \{2\} \cup \{2\} \cup \{2\} \cup \{1\} \cup \{2\} \cup \{6,9,10\}), & \text{logo } a_1 &= 3. \\
a_2 &= \min (\{1, \dots, 10\} - \{3\} \cup \{2\} \cup \{2\} \cup \{2\} \cup \{1\} \cup \{2\} \cup \{6,9,10\}), & \text{logo } a_2 &= 4. \\
a_3 &= \min (\{1, \dots, 10\} - \{3,4\} \cup \{2\} \cup \{2\} \cup \{1\} \cup \{2\} \cup \{6,9,10\}), & \text{logo } a_3 &= 5. \\
a_4 &= \min (\{1, \dots, 10\} - \{3,4,5\} \cup \{2\} \cup \{1\} \cup \{2\} \cup \{6,9,10\}), & \text{logo } a_4 &= 7. \\
a_5 &= \min (\{1, \dots, 10\} - \{3,4,5,7\} \cup \{1\} \cup \{2\} \cup \{6,9,10\}), & \text{logo } a_5 &= 8. \\
a_6 &= \min (\{1, \dots, 10\} - \{3,4,5,7,8\} \cup \{2\} \cup \{6,9,10\}), & \text{logo } a_6 &= 1. \\
a_7 &= \alpha_6, & \text{logo } a_7 &= 2.
\end{aligned}$$

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{_ _ _ \} & \{_ _ _ \} & \{_ _ _ \} & \{_ _ _ \} & \{_ _ _ \} & \{_ _ _ \} & \{6,9,10\} \end{array} \right)$$

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{_ _ _ \} & \{_ _ _ \} & \{_ _ _ \} & \{_ _ _ \} & \{_ _ _ \} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

$$B_6 = \alpha_6 \cup B_7 - \beta_6 = \{2\} \cup \{6,9,10\} - \{9\} = \{2,6,10\}$$

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{_ _ _ \} & \{_ _ _ \} & \{_ _ _ \} & \{_ _ _ \} & \{1,2,6\} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

$$B_5 = \alpha_5 \cup B_6 - \beta_5 = \{1\} \cup \{2,6,10\} - \{10\} = \{1,2,6\}$$

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{_ _ _ \} & \{_ _ _ \} & \{_ _ _ \} & \{2,9,10\} & \{1,2,6\} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

$$B_4 = \alpha_4 \cup B_7 - \beta_4 = \{2\} \cup \{6,9,10\} - \{6\} = \{2,9,10\}$$

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{_ _ _ \} & \{_ _ _ \} & \{6,9,10\} & \{2,9,10\} & \{1,2,6\} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

$$B_3 = \alpha_3 \cup B_7 - \beta_3 = \{2\} \cup \{6,9,10\} - \{2\} = \{6,9,10\}$$

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{_ _ _ \} & \{2,9,10\} & \{6,9,10\} & \{2,9,10\} & \{1,2,6\} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

$$B_2 = \alpha_2 \cup B_7 - \beta_2 = \{2\} \cup \{6,9,10\} - \{6\} = \{2,9,10\}$$

$$\left(\begin{array}{ccccccc} 3 & 4 & 5 & 7 & 8 & 1 & 2 \\ \{5,6,9\} & \{2,9,10\} & \{6,9,10\} & \{2,9,10\} & \{1,2,6\} & \{2,6,10\} & \{6,9,10\} \end{array} \right)$$

$$B_1 = \alpha_1 \cup B_3 - \beta_1 = \{5\} \cup \{6,9,10\} - \{10\} = \{5,6,9\}$$

Figura 6.3: Obtenção, passo a passo, da 3-árvore da Figura 6.1 a partir do seu código compacto.

6.2 - Validação

Do mesmo modo que nem toda seqüência $\langle B_1, \dots, B_{n-k-1} \rangle$ de conjuntos de tamanho k é o código de Prüfer de uma k -árvore, nem todo par $P = (B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ é um código compacto de uma k -árvore. Por esta razão, mostramos como verificar se um par P qualquer é o código compacto de uma k -árvore.

A validação é dividida em três etapas. As etapas consistem em verificar se:

- P possui a estrutura de um código introduzido pela Definição 6.3 de código compacto;
- é possível obter $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$ utilizando o Teorema 6.6;
- $(\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle)$ é o código redundante de uma k -árvore.

Cada um dos três teoremas vistos a seguir é utilizado em uma das etapas da validação.

Teorema 6.7: Sejam uma k -árvore $G = (V, E)$ e $P = (B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ o seu código compacto. Então, $|B| = k$, $n \in B$ e para $1 \leq i < n - k$:

- α_i e $\beta_i \in V$;
- $\alpha_i = \beta_i = \alpha_{n-k-1}$ ou $\alpha_i \neq \beta_i$;
- $\alpha_i \notin B$.

Prova: Como B é a k -clique residual, então $|B| = k$. Como toda k -árvore possui pelo menos duas k -folhas, então sempre existe uma k -folha menor do que n para ser removida durante a codificação. Logo, $n \in B$.

Existem dois casos na Definição 6.3 de código compacto.

- $B_i = B_{n-k}$ e $(\alpha_i, \beta_i) = (a_{n-k}, a_{n-k})$. Logo, α_i e $\beta_i \in V$ e $\alpha_i = \beta_i = \alpha_{n-k-1}$;
- $B_i \neq B_{n-k}$ e $(\alpha_i, \beta_i) = (B_i - B_j, B_j - B_i)$ tal que $\alpha_i = a_j$. Logo, α_i e β_i são vértices de uma k -clique de G , então α_i e $\beta_i \in V$. Como $B_i \neq B_j$, então $\alpha_i \neq \beta_i$.

Inferese do Teorema 5.5 que $\{a_1, \dots, a_{n-k}\} \cap B_{n-k} = \emptyset$, ou seja, $a_i \notin B_{n-k}$. No primeiro caso da na Definição 6.3 de código compacto, $\alpha_i = a_{n-k}$ e, no segundo caso, $\alpha_i = a_j$. Logo, $\alpha_i \notin B$. \square

Os itens 1, 2, 3 e 4 do Teorema 6.6, que trata da obtenção do código redundante, são sempre satisfeitos devido à cardinalidade dos conjuntos envolvidos. O Teorema 6.8

visto a seguir define condições necessárias para que os conjuntos $B_i = \{\alpha_i\} \cup B_j - \{\beta_i\}$, obtidos através do item 5 do Teorema 6.6, possuam tamanho k .

Teorema 6.8: Sejam um par $P = (B, \langle(\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1})\rangle)$ que satisfaz o teorema anterior e $\langle(a_1, \dots, a_{n-k}), \langle B_1, \dots, B_{n-k}\rangle\rangle$ obtido pelo Teorema 6.6. Para $1 \leq i < n-k$, os conjuntos B_i possuem cardinalidade k se:

- $\alpha_i = \beta_i = \alpha_{n-k-1}$ ou $(\alpha_i = a_j \text{ e } \beta_i \in B_j)$.

Prova: Os itens 1, 2, 3 e 4 do Teorema 6.6 são sempre satisfeitos devido à cardinalidade dos conjuntos envolvidos. Pelo item 3, observa-se que os vértices a_1, \dots, a_{n-k-1} são distintos e que nenhum destes vértices pertence a $\{\alpha_{n-k-1}\} \cup B = \{a_{n-k}\} \cup B_{n-k}$. Como P satisfaz o teorema anterior, então $\alpha_i \notin B$ e mais especificamente $\alpha_{n-k-1} \notin B$, ou seja, $a_{n-k} \notin B_{n-k}$. Logo, $\{a_1, \dots, a_{n-k}\} \cap B_{n-k} = \emptyset$. Como $\alpha_i \notin B$, então sempre existe um j tal que $\alpha_i = a_j$. São dois casos na obtenção da clique B_i .

- $\alpha_i = \beta_i$. Pelo lema anterior, $\alpha_i = \beta_i = \alpha_{n-k-1} = a_{n-k}$. Logo, $j = n-k$. Como, $a_{n-k} \notin B_{n-k}$, tem-se que $B_i = \{a_{n-k}\} \cup B_{n-k} - \{a_{n-k}\} = B_{n-k}$. Pelo teorema anterior, $B_{n-k} = B$ possui cardinalidade k .
- $\alpha_i \neq \beta_i$. Inference-se do Lema 6.5 que $B_j \subset \{\alpha_j\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B$ e do item 3 do Teorema 6.6 que $a_j \notin \{\alpha_j\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B$. Logo, $a_j \notin B_j$. Desse modo, $\{\alpha_i\} \cup B_j = \{a_j\} \cup B_j$ possui cardinalidade $k+1$. Logo, $B_i = \{\alpha_i\} \cup B_j - \{\beta_i\}$ possui cardinalidade k quando $\beta_i \in B_j$. \square

A prova do Teorema 6.6 é baseada no Lema 5.1 que obtém o código redundante a partir do código de Prüfer exceto pelo fato de considerar $a_{n-k} = \alpha_{n-k-1}$ e $B_{n-k-1} = B$. No item 5 do Teorema 6.6, $B_i = \{a_j\} \cup B_j - \{\beta_i\}$, ou seja, $B_i \subset \{a_j\} \cup B_j$. Assim, a obtenção da clique B_i implicitamente testa a condição $B_i \subset \{a_j\} \cup B_j$ do Teorema 5.2 de validação do código de Prüfer. Desse modo, para completar a validação basta verificar se o vértice $\alpha_{n-k-1} = a_{n-k}$ e a k -clique $B = B_{n-k}$ são coerentes com o código redundante da k -árvore obtida.

Teorema 6.9: Seja $P = (B, \langle(\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1})\rangle)$ satisfazendo o Teorema 6.7 e o Teorema 6.8. Seja $\langle(a_1, \dots, a_{n-k}), \langle B_1, \dots, B_{n-k}\rangle\rangle$ obtido pelo Teorema 6.6. P é o código compacto de uma k -árvore quando:

- α_{n-k-1} é menor que os vértices de B ;
- β_{n-k-1} é o maior vértice do conjunto $V - B_{n-k-1}$.

Prova: Ao final da codificação, resta a $(k+1)$ -clique $C_{n-k} = \{a_{n-k}\} \cup B_{n-k}$ onde a_{n-k} é o menor vértice de C_{n-k} e B_{n-k} é a k -clique residual. Logo, $\alpha_{n-k-1} = a_{n-k}$ é menor que os vértices de $B = B_{n-k}$.

Pelo Teorema 5.2, $B_{n-k-1} \subset C_{n-k}$. Durante a codificação, os vértices de B_{n-k-1} não são k -folhas, pois estão contidos em duas cliques maximais: C_{n-k-1} e C_{n-k} . Como toda k -árvore possui pelo menos duas k -folhas, então, durante a codificação, existe um vértice menor que o maior vértice de $V - B_{n-k-1}$ para ser removido. Logo, $C_{n-k} = B_{n-k-1} \cup maior(V - B_{n-k-1})$.

Do item 5 do Teorema 6.6, para $i = n - k - 1$, tem-se $B_{n-k-1} = \{a_{n-k}\} \cup B_{n-k} - \{\beta_{n-k-1}\}$. Assim, $B_{n-k-1} \cup \{\beta_{n-k-1}\} = \{a_{n-k}\} \cup B_{n-k} = C_{n-k}$. Logo, β_{n-k-1} é o maior vértice do conjunto $V - B_{n-k-1}$. \square

6.3 - Obtenção de Esquema de Eliminação Perfeita

Nesta seção, apresentamos um algoritmo que encontra um esquema de eliminação perfeita de uma k -árvore a partir do seu código compacto com complexidade $O(n)$. Na próxima seção, este algoritmo será utilizado como passo intermediário da decodificação.

Sabe-se do Teorema 5.5 que $\langle a_1, \dots, a_{n-k}, B_{n-k} \rangle$ é um esquema de eliminação perfeita de uma k -árvore. Apesar de todo grafo cordal possuir mais de um esquema de eliminação perfeita, o obtido pelo Teorema 5.5 é único, a menos da permuta dos k vértices de B_{n-k} . Como $B = B_{n-k}$, a partir deste ponto, $\langle a_1, \dots, a_{n-k}, B \rangle$ será referido como **esquema de eliminação perfeita associado** ao código compacto $(B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$.

O Algoritmo 6.1, visto a seguir, obtém o esquema de eliminação perfeita associado utilizando os itens 2, 3 e 4 do Teorema 6.6. No Algoritmo 6.1, utiliza-se o vetor $freq[1..n]$ e a lista de prioridades $(1, n)$ -limitada Q . Na iteração i , a posição $freq[v]$ armazena a quantidade de vezes que o vértice v ocorre nos conjuntos $\{\alpha_i\}, \{\alpha_{i+1}\}, \dots, \{\alpha_{n-k}\}, B$ e a lista de prioridades $(1, n)$ -limitada Q armazena os vértices com frequência zero, ou seja, os vértices que não pertencem ao conjunto $\{\alpha_i\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B$. Assim, na primeira iteração, o vértice removido de Q é o

menor vértice de $V - (\{\alpha_1\} \cup \{\alpha_2\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B)$, ou seja, é o vértice a_1 que satisfaz o item 2 do Teorema 6.6. Nas iterações seguintes, após obter a_1, \dots, a_{j-1} , o vértice removido de Q é o menor vértice de $V - \{a_1, \dots, a_{j-1}\} - (\{\alpha_j\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B)$, ou seja, é o vértice a_j que satisfaz o item 3. O vértice a_{n-k} é obtido diretamente do item 4, ou seja, $a_{n-k} = \alpha_{n-k-1}$.

```

Algoritmo Esquema_Eliminação_Perfeita;
Entrada: código compacto  $(B, S = \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ ;
Saída:  $\langle a_1, \dots, a_{n-k}, B \rangle$ ;
Início
     $k \leftarrow |B|$ ;
     $n \leftarrow k + |S| + 1$ ;
    Para todo  $v \in V$  faça  $freq[v] \leftarrow 0$ ;
    Para todo  $v \in B$  faça  $freq[v] \leftarrow 1$ ;
    Para  $i$  de 1 até  $n-k-1$  faça
         $freq[\alpha_i] \leftarrow freq[\alpha_i] + 1$ ;
        Cria_Lista( $Q, 1, n$ );
    Para todo  $v \in V$  faça
        Se  $freq[v] = 0$  então
            Inserir( $Q, v, v$ );
    Para  $j$  de 1 até  $n-k-1$  faça
        Remove_Min( $Q, a_j, a_j$ );
         $freq[\alpha_j] \leftarrow freq[\alpha_j] - 1$ ;
        Se  $freq[\alpha_j] = 0$  então
            Inserir( $Q, \alpha_j, \alpha_j$ );
     $a_{n-k} \leftarrow \alpha_{n-k-1}$ ;
Fim

```

Algoritmo 6.1: Obtenção do esquema de eliminação perfeita associado.

Teorema 6.10: O algoritmo que obtém o esquema de eliminação perfeita associado ao código compacto de uma k -árvore tem complexidade $O(n)$.

Prova: Na iteração j , foram removidos exatamente j vértices de Q e apenas os vértices de $\{\alpha_{j+1}\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B$ não foram inseridos em Q . Assim, no mínimo, foram inseridos $j+1$ vértices e a lista Q não fica vazia. Além disso, após cada remoção, existe no máximo um vértice α_j com frequência zero que pode ser inserido em Q . Logo, esta seqüência de operações na lista de prioridades Q é válida. Até a iteração $n-k-1$, todos os vértices são inseridos em Q exceto os vértices de B . Logo, são inseridos exatamente $n-k$ vértices. Pelo Teorema 5.8, esta seqüência de operações pode ser executada com complexidade $O(n-k+n) = O(n)$.

Cada passo tem no máximo n repetições e as operações com a lista de prioridades têm complexidade $O(n)$. Logo, o algoritmo possui complexidade $O(n)$. \square

6.4 - Algoritmos de Codificação e Decodificação

Nesta seção, mostramos como codificar e decodificar o código compacto com complexidade $O(m)$. Para tal utilizamos os algoritmos já desenvolvidos, apresentamos novos algoritmos e dividimos tanto a codificação quanto a decodificação em duas etapas.

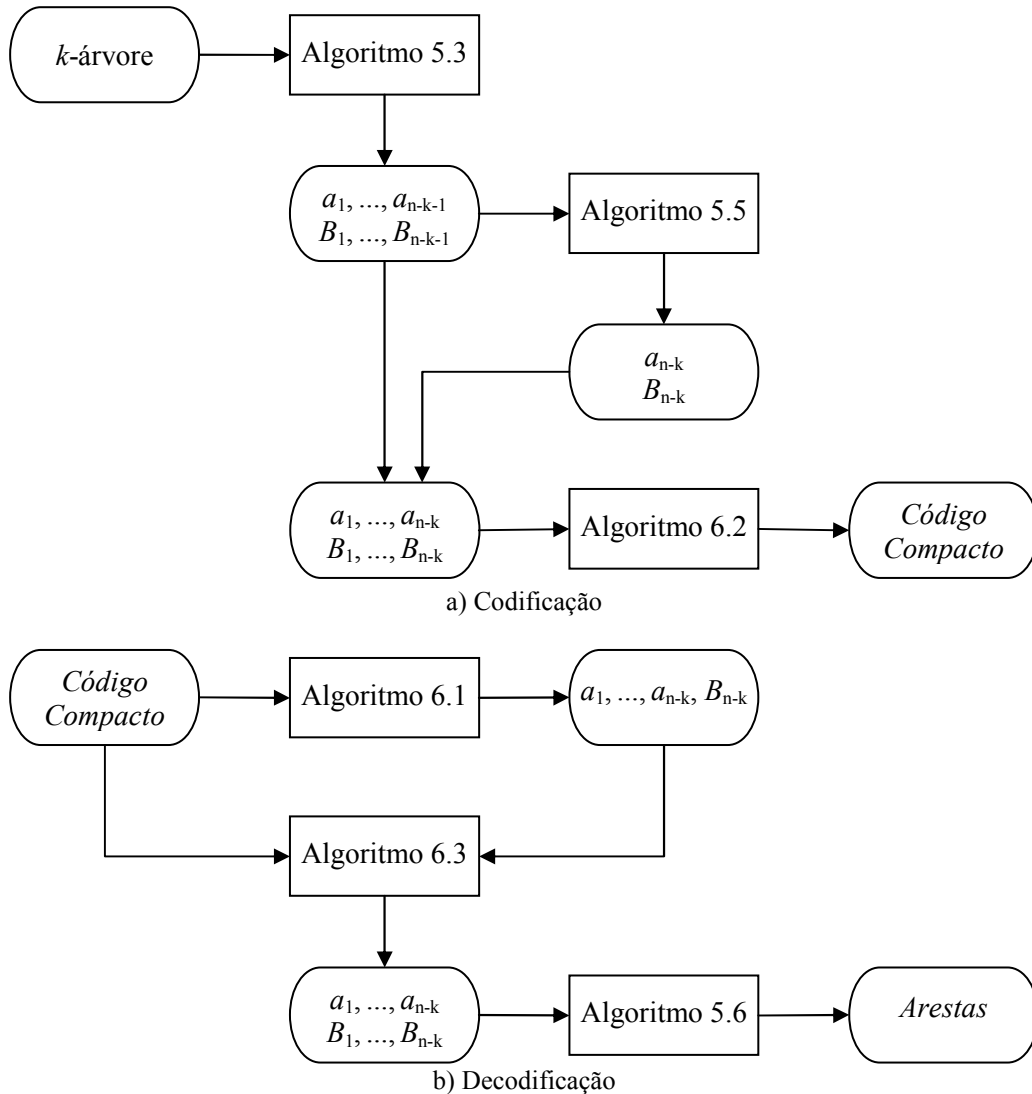


Figura 6.4: Representação esquemática da codificação e decodificação.

A primeira etapa da codificação utiliza algoritmos já desenvolvidos na Seção 5.4 para obter o código redundante e, na segunda etapa, obtém-se o código compacto através do Algoritmo 6.2 que será visto nesta seção. Na primeira etapa da decodificação, obtém-se o esquema de eliminação perfeita utilizando o Algoritmo 6.1 visto na seção anterior e, em seguida, utiliza-se o Algoritmo 6.3 visto nesta seção para obter o código

redundante. Finalmente, na segunda etapa da decodificação, obtém-se as arestas da k -árvore utilizando o Algoritmo 5.6 visto na Seção 5.4. A Figura 6.4 resume os processos de codificação e decodificação que são descritos detalhadamente a seguir.

Codificação

A codificação é dividida em duas etapas. Na primeira etapa, os Algoritmos 5.3 e 5.5, vistos na Seção 5.4, são utilizados para obter o código redundante de G e, na segunda, dado o código redundante, o Algoritmo 6.2 visto aqui obtém o código compacto diretamente de sua definição.

No Algoritmo 6.2, utilizam-se os vetores auxiliares $pos_a[1..n]$ e $Clique[1..k]$. Inicialmente, calcula-se o vetor auxiliar $pos_a[v]$ que armazena i se $v = a_i$ e $n - k$ caso $v \in B_{n-k}$. Na obtenção do vértice α_i , obtém-se $j = \min\{pos_a[v] \mid v \in B_i\}$. Do mesmo modo que na Definição 6.3 de código compacto, existem dois casos:

- $B_i = B_{n-k}$. Todos os vértices de $B_i = B_{n-k}$ possuem valor $n - k$ no vetor pos_a e tem-se $j = n - k$. Logo, $\alpha_i \leftarrow a_{n-k}$.
- $B_i \neq B_{n-k}$. $j = \min\{pos_a[v] \mid v \in B_i\} = \min\{t \mid i < t \leq n - k \wedge a_t \in B_i\}$. Logo, $\alpha_i \leftarrow a_j$.

Durante a obtenção do vértice β_i , utiliza-se o vetor auxiliar $Clique[1..n]$. A posição $Clique[v]$ assume valor i se $v \in B_i$ e permanece com outro valor caso contrário. Existem dois casos quando B_i e B_j são comparados:

- $B_i = B_j = B_{n-k}$. Para todo vértice $v \in B_j$, tem-se $clique[v] = i$. Logo, β_i permanece igual a a_{n-k} de acordo com o primeiro caso da Definição 6.3.
- $B_i \neq B_{n-k}$. Pela Definição 6.3, $\beta_i = B_j - B_i$. Logo, $\beta_i \in B_j$ e $\beta_i \notin B_i$. Assim, apenas para β_i tem-se $clique[\beta_i] \neq i$. Logo, β_i é o vértice $v \in B_j$ tal que $clique[v] \neq i$.

Teorema 6.11: O algoritmo que obtém o código compacto de uma k -árvore a partir do seu código redundante tem complexidade $O(m)$.

Prova: A inicialização dos vetores auxiliares no Algoritmo 6.2 é feita com complexidade $O(n)$. Durante o cálculo de cada vértice α_i , obtém-se o índice j com complexidade $O(k)$. A obtenção do vértice β_i é feita através da verificação dos vértices de B_i (utilizando o vetor auxiliar $Clique$) e de B_j . Logo, obtém-se β_i com complexidade $O(k)$. Como estes cálculos são feitos para $n - k - 1$ pares (α_i, β_i) , a obtenção destes é feita com complexidade $O(n.k) = O(m)$. Logo, o algoritmo possui complexidade $O(m)$. \square

```

Algoritmo Codificação_Compacta;
Entrada: código redundante  $\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle$ ;
Saída: código compacto  $(B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ ;
Início
   $k \leftarrow |B_1|$ ;
   $n \leftarrow k + |\{a_1, \dots, a_{n-k}\}|$ ;
  Para  $v$  de 1 até  $n$  faça
     $pos\_a[v] \leftarrow n-k$ ;
     $Clique[v] \leftarrow -1$ ;
  Para  $i$  de 1 até  $n-k$  faça
     $pos\_a[a_i] \leftarrow i$ ;
  Para  $i$  de 1 até  $n-k-1$  faça
     $j \leftarrow n-k$ ;  $\{j \leftarrow \text{mínimo}(pos\_a[v] \mid v \in B_i)\}$ 
    Para todo  $v \in B_i$  faça
      Se  $pos\_a[v] < j$  então
         $j \leftarrow pos\_a[v]$ ;
     $\alpha_i \leftarrow \beta_i \leftarrow a_j$ ;
    Para todo  $v \in B_i$  faça  $Clique[v] \leftarrow i$ ;
    Para todo  $v \in B_j$  faça
      Se  $Clique[v] \neq i$  então
         $\beta_i \leftarrow v$ ;
Fim

```

Algoritmo 6.2: Obtenção do código compacto de uma k -árvore.

Como visto na Seção 5.4, os Algoritmos 5.3 e 5.5 possuem complexidades $O(m)$ e $O(n)$ respectivamente. No Teorema 6.11, provamos que o Algoritmo 6.2 possui complexidade $O(m)$. Desse modo, a codificação é feita com complexidade $O(m)$.

Decodificação

Como citado anteriormente, a decodificação é dividida em duas etapas. Na primeira etapa, determina-se o código redundante a partir da aplicação do Teorema 6.6 em dois algoritmos. O Algoritmo 6.1 computa o esquema de eliminação perfeita $\langle a_1, \dots, a_{n-k}, B_{n-k} \rangle$ utilizando os itens 2, 3 e 4 deste teorema e o Algoritmo 6.3 obtém o código de Prüfer $\langle B_1, \dots, B_{n-k-1} \rangle$ utilizando o item 5. Na segunda etapa da decodificação, as arestas da k -árvore são determinadas utilizando-se o Algoritmo 5.6 visto no Capítulo 5.

O Algoritmo 6.3 utiliza o vetor auxiliar pos_a já visto anteriormente na codificação. A posição $pos_a[v]$ armazena i se $v = a_i$ e $n-k$ caso $v \in B_{n-k}$. Como $\alpha_i = a_j$, tem-se $j = pos_a[\alpha_i]$. Sabe-se do item 5 do Teorema 6.6 que $B_i = \{\alpha_i\} \cup B_j - \{\beta_i\}$. Existem dois casos na obtenção das cliques B_i .

- $\alpha_i = \beta_i = a_{n-k}$. Como $j = n - k$ e $a_{n-k} \notin B_{n-k}$, todos os vértices de $B_j = B_{n-k}$ são inseridos em B_i , ou seja, faz-se $B_i = B_{n-k}$.
- $\alpha_i \neq \beta_i$. Todos os vértices de B_j são inseridos em B_i exceto o vértice β_i que é substituído pelo vértice α_i , ou seja, faz-se $B_i = \{\alpha_i\} \cup B_j - \{\beta_i\}$.

Algoritmo Decodificação_Compacta;
Entrada: $(B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle), EEP = \langle a_1, \dots, a_{n-k}, B_{n-k} \rangle$;
Saída: Código Redundante $\langle a_1, \dots, a_{n-k} \rangle, \langle B_1, \dots, B_{n-k} \rangle$;
Início
 $k \leftarrow |B|$;
 $n \leftarrow |EEP|$;
Para v de 1 até n faça $pos_a[v] \leftarrow n-k$;
Para i de 1 até $n-k$ faça $pos_a[a_i] \leftarrow i$;
Para i de $n-k-1$ até 1 faça
 $j \leftarrow pos_a[\alpha_i]$;
 $B_i \leftarrow \emptyset$;
Para todo $v \in B_j$ faça
Se $v \neq \beta_i$ então
 $B_i \leftarrow B_i \cup \{v\}$;
Senão
 $B_i \leftarrow B_i \cup \{\alpha_i\}$;
Fim

Algoritmo 6.3: Obtenção do código redundante a partir do código compacto e dos vértices $\langle a_1, \dots, a_{n-k}, B \rangle$.

Teorema 6.12: O algoritmo que obtém o código redundante de uma k -árvore a partir do seu código compacto e do seu esquema de eliminação perfeita associado tem complexidade $O(m)$.

Prova: A obtenção do vetor pos_a é feita com complexidade $O(n)$ e a obtenção das $n - k - 1$ cliques B_i com complexidade $O(n.k) = O(m)$. Logo, o algoritmo possui complexidade $O(m)$. \square

Como visto na Seção 5.4 e na seção anterior, o Algoritmo 5.6 e o Algoritmo 6.1 possuem complexidades $O(m)$ e $O(n)$, respectivamente. No Teorema 6.12, provamos que a complexidade do Algoritmo 6.3 é $O(m)$. Desse modo, a decodificação possui complexidade $O(m)$.

6.5 - Conclusão

RÉNYI e RÉNYI [42] expandiram o código de Prüfer para k -árvores rotuladas que possuíam uma clique formada pelos k maiores vértices e para k -árvores enraizadas. A abordagem vista na Seção 5.1 para k -árvores quaisquer tem como principais vantagens:

- ser a expansão natural do código de Prüfer para k -árvores;
- codificar sem precisar conhecer uma k -clique da k -árvore;
- possuir um único código para uma k -árvore.

O código compacto proposto neste capítulo tem como principal motivação uma representação computacional reduzida, já que é mais conciso do que o código de Prüfer para k -árvores e possui algoritmos eficientes de codificação e decodificação. No próximo capítulo, mostramos aplicações que utilizam o código compacto.

Capítulo 7 - Aplicações do Código Compacto

Neste capítulo, são vistos alguns problemas cuja solução é particularmente eficiente quando, na implementação, a k -árvore é representada pelo seu código compacto. Em [34], é apresentado um algoritmo com complexidade $O(m)$ para a determinação de todos os vértices simpliciais de um grafo cordal. Na Seção 7.1, mostramos como obter os vértices simpliciais de uma k -árvore a partir do seu código compacto com complexidade $O(n)$. GAREY e JOHNSON [16] citam que, apesar de a coloração de vértices ser um problema NP -completo, existem algoritmos polinomiais para algumas famílias de grafos. Particularmente, para os grafos cordais, GAVRIL [17] propôs um algoritmo com complexidade $O(m)$ utilizando o esquema de eliminação perfeita. Na Seção 7.2, apresentamos um algoritmo com complexidade $O(n)$ para obter a coloração ótima dos vértices de uma k -árvore qualquer representada pelo seu código compacto. Na Seção 5.3, foi definido o conceito de multiplicidade que generaliza o conceito de grau. Na Seção 7.3, apresentamos o conceito de seqüência de multiplicidades que generaliza o conceito de seqüência de graus e projetamos um algoritmo que obtém a seqüência de multiplicidades de uma k -árvore com complexidade $O(n)$.

7.1 - Determinação dos Vértices Simpliciais

Os vértices simpliciais são muito importantes nos grafos cordais. A determinação eficiente de todos os vértices simpliciais ocorre com freqüência como etapa intermediária na solução de problemas envolvendo grafos cordais. Particularmente, nas k -árvores com mais de k vértices, apenas os vértices simpliciais possuem grau k e são obtidos através do cálculo dos graus, ou seja, com complexidade $O(m)$. Nesta seção, mostramos como obter estes vértices através da inspeção do código compacto com complexidade $O(n)$.

Observe que os vértices $\alpha_i = a_j = B_i - B_j$ não são simpliciais, pois pertencem às cliques maximais C_i e C_j . No Lema 7.1, mostramos que o vértice β_{n-k-1} é o único vértice de C_{n-k} que pode ser simplicial. O Teorema 7.2, visto em seguida, possibilita a determinação dos vértices simpliciais, sendo dividido em dois casos. O primeiro caso

abrange todos os vértices exceto o vértice β_{n-k-1} e o segundo especifica condições apenas para o vértice β_{n-k-1} .

Lema 7.1: Seja o código compacto $(B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ de uma k -árvore $G=(V,E)$. O vértice β_{n-k-1} é o único vértice de $\{\alpha_{n-k-1}\} \cup B$ que pode ser simplicial.

Prova: Sabe-se que $\alpha_{n-k-1} = a_{n-k}$, $B_{n-k} = B$ e que $C_{n-k} = \{a_{n-k}\} \cup B_{n-k}$. Logo, $C_{n-k} = \{\alpha_{n-k-1}\} \cup B$. Além disso, sabe-se do Teorema 5.2 que $B_{n-k-1} \subset C_{n-k}$ e do Lema 1.5 que um vértice simplicial pertence a apenas uma clique maximal. Desse modo, o vértice do conjunto unitário $\{s\} = C_{n-k} - B_{n-k-1}$ é o único vértice de C_{n-k} que pode ser simplicial. Pela Definição 6.3 existem dois casos:

- $B_{n-k} = B_{n-k-1}$ e $\alpha_{n-k-1} = \beta_{n-k-1} = a_{n-k}$. Assim, $\{s\} = \{\beta_{n-k-1}\} \cup B_{n-k} - B_{n-k-1} = \{\beta_{n-k-1}\}$.
- $B_{n-k} \neq B_{n-k-1}$ e $\beta_{n-k-1} = B_{n-k} - B_{n-k-1}$. Assim, $\beta_{n-k-1} \notin B_{n-k-1}$ e $\beta_{n-k-1} \in C_{n-k}$. Logo, $\{s\} = \{\beta_{n-k-1}\}$.

Portanto, β_{n-k-1} é o único vértice de $C_{n-k} = \{\alpha_{n-k-1}\} \cup B$ que pode ser simplicial. \square

Teorema 7.2: Seja o código compacto $(B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ de uma k -árvore $G=(V,E)$. Um vértice $v \in V$ é simplicial quando:

- $v \neq \beta_{n-k-1}$ e $v \notin \{\alpha_1\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B$;
- $v = \beta_{n-k-1}$ e para $1 \leq i < n-k-1$: $\alpha_i \neq \alpha_{n-k-1}$ ou $(\alpha_i, \beta_i) = (\alpha_{n-k-1}, \beta_{n-k-1})$.

Prova: Nos dois casos, utiliza-se o Corolário 5.4, ou seja, um vértice v é simplicial quando $v \notin B_1 \cup \dots \cup B_{n-k-1}$.

Seja $v \neq \beta_{n-k-1}$. Pelo Lema 7.1, β_{n-k-1} é o único vértice de $B_{n-k} \cup \{a_{n-k}\}$ que pode ser simplicial, então, $v \neq \beta_{n-k-1}$ é simplicial quando $v \notin B_1 \cup \dots \cup B_{n-k-1} \cup B_{n-k} \cup \{a_{n-k}\}$. Sabe-se do Lema 6.5 que $B_1 \cup \dots \cup B_{n-k-1} \cup B_{n-k} \cup \{a_{n-k}\} = \{\alpha_1\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B$. Logo, $v \neq \beta_{n-k-1}$ é simplicial quando $v \notin \{\alpha_1\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B$.

No segundo caso, β_{n-k-1} é simplicial quando $\beta_{n-k-1} \notin B_1 \cup \dots \cup B_{n-k-1}$. Sabe-se do Teorema 5.2 que $B_i \subset C_j$ para algum $j > i$. Assim, para $1 \leq i < n-k-1$, cada clique B_i satisfaz a um dos dois casos:

- $B_i \not\subset C_{n-k}$. $B_i \subset C_j$ e $j \neq n-k$. Logo, $\alpha_i = a_j \neq a_{n-k} = \alpha_{n-k-1}$, ou seja, $\alpha_i \neq \alpha_{n-k-1}$.
- $B_i \subset C_{n-k}$. $\beta_{n-k-1} \notin B_i$ e $j = n-k$. Logo, $\alpha_i = a_{n-k} = \alpha_{n-k-1}$ e $B_i = \{\alpha_i\} \cup B_{n-k} - \{\beta_i\}$, ou seja, $B_i = \{\alpha_{n-k-1}\} \cup B - \{\beta_i\}$. Como $\beta_{n-k-1} \notin B_i$ e $\beta_{n-k-1} \in \{\alpha_{n-k-1}\} \cup B$, tem-se $\beta_i = \beta_{n-k}$. Logo, $(\alpha_i, \beta_i) = (\alpha_{n-k-1}, \beta_{n-k-1})$. \square

Por exemplo, inspecionando o código compacto $(\{6,9,10\}, \langle(5,10), (2,6), (2,2), (2,6), (1,10), (2,9)\rangle)$ da 3-árvore da Figura 6.1, verifica-se que $\{\alpha_1\} \cup \dots \cup \{\alpha_6\} \cup B = \{1, 2, 5, 6, 9, 10\}$ e que $\beta_6 = 9$. Pelo item 1 do Teorema 7.2, os vértices 3, 4, 7 e 8 são simpliciais. Já o vértice $\beta_6 = 9$ não é simplicial, pois existe o par $(2,6) = (\alpha_4, \beta_4) \neq (\alpha_6, \beta_6) = (2,9)$ que contradiz o item 2.

Com base no Teorema 7.2, o Algoritmo 7.1 determina os vértices simpliciais de uma k -árvore a partir do seu código compacto. A saída é o vetor $simpl[1..n]$, onde $simpl[v]$ possui valor um quando v é simplicial e zero caso contrário. O vetor é inicializado de modo que todos os vértices são considerados simpliciais. Em seguida, os vértices $\{\alpha_1\} \cup \dots \cup \{\alpha_{n-k-1}\} \cup B$ são excluídos dos vértices simpliciais. Finalmente, a segunda condição do Teorema 7.2 é verificada. Caso não seja satisfeita para algum par (α_i, β_i) , o vértice β_{n-k-1} é excluído dos simpliciais. Este algoritmo possui claramente complexidade $O(n)$ já que todas as repetições são executadas no máximo n vezes.

```

Algoritmo Determinar_Simpliciais;
Entrada: código compacto  $(B, S = \langle(\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1})\rangle)$ ;
Saída: vetor  $simpl[1..n]$  onde  $simpl[v]=1$  se  $v$  é simplicial;
Início
  Para  $i$  de 1 até  $n$  faça  $simpl[i] \leftarrow 1$ ;
  Para  $i$  de 1 até  $n-k-1$  faça  $simpl[\alpha_i] \leftarrow 0$ ;
  Para todo  $v \in B$  faça  $simpl[v] \leftarrow 0$ ;
   $simpl[\beta_{n-k-1}] \leftarrow 1$ ;
  Para  $i$  de 1 até  $n-k-2$  faça
    Se  $(\alpha_i = \alpha_{n-k-1}) \wedge (\beta_i \neq \beta_{n-k-1})$  então
       $simpl[\beta_{n-k-1}] \leftarrow 0$ ;
Fim

```

Algoritmo 7.1: Determinação dos vértices simpliciais a partir do código compacto.

7.2 - Coloração de Vértices

O problema de coloração de vértices possui as mais diversas aplicações. Um exemplo clássico em compiladores é a otimização da alocação de variáveis em registradores. Recentemente, PEREIRA e PALSBERG [36] implementaram compiladores eficientes através da coloração de grafos cordais. Estes autores utilizam o algoritmo guloso de complexidade $O(m)$ proposto por GAVRIL [17]. Este algoritmo colore os vértices de um grafo cordal na ordem inversa do esquema de eliminação perfeita. Nesta seção, propomos um algoritmo com complexidade $O(n)$ para coloração de vértices de uma k -árvore com n vértices a partir do seu código compacto. Primeiro,

provamos que toda k -árvore é unicamente $(k+1)$ -colorível. Em seguida, provamos que os vértices a_i e β_i devem possuir a mesma cor e que é possível colorir os vértices na ordem inversa do esquema de eliminação perfeita associado ao código compacto.

Definição 7.3: Seja um grafo $G = (V, E)$. Uma **x -coloração** é uma função $C : V \rightarrow \{1, \dots, x\}$ de forma que $\forall \{v, w\} \in E$ tem-se $C(v) \neq C(w)$. Quando existe uma única x -coloração, diz-se que G é **unicamente x -colorível**.

O próximo lema mostra que toda k -árvore é unicamente $(k+1)$ -colorível.

Lema 7.4: Toda k -árvore com pelo menos $k + 1$ vértices é unicamente $(k+1)$ -colorível.

Prova: A prova será por indução sobre o número de vértices da k -árvore.

- Para $n = k + 1$, a k -árvore é um grafo completo e o teorema vale trivialmente.
- Suponha que o resultado vale para k -árvores com $n - 1$ vértices.
- Ao adicionar um vértice v e k arestas adjacentes aos vértices de uma k -clique Q de uma k -árvore com $n - 1$ vértices, obtém-se uma k -árvore com n vértices. Como os vértices da k -clique Q estão coloridos com k cores distintas, o vértice v possui obrigatoriamente a única cor não presente nos vértices de Q . Desse modo, as k -árvores com n vértices possuem uma única $(k+1)$ -coloração. \square

Os dois lemas vistos a seguir permitem que seja obtida uma $(k+1)$ -coloração dos vértices de uma k -árvore utilizando os vértices β_i e o esquema de eliminação perfeita $\langle a_1, \dots, a_{n-k}, B \rangle$ associado ao código compacto. O Lema 7.5 garante que os vértices a_i e β_i possuem a mesma cor e o Lema 7.6 que o vértice β_i é colorido antes do vértice a_i .

Lema 7.5: Sejam uma k -árvore $G = (V, E)$ e $\langle a_1, \dots, a_{n-k}, B \rangle$ o esquema de eliminação perfeita associado ao código compacto $(B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ de G . Em uma $(k+1)$ -coloração de G , os vértices a_i e β_i , $1 \leq i < n - k$, possuem a mesma cor.

Prova: Pela Definição 6.3 de código compacto, existem dois casos.

- $B_i = B_{n-k}$ e $\alpha_i = \beta_i = a_{n-k}$. Sabe-se que a_{n-k} é adjacente aos vértices da k -clique B_{n-k} . Logo, $\beta_i = a_{n-k}$ é adjacente aos vértices da k -clique $B_i = B_{n-k}$.

• $B_i \neq B_{n-k}$, $\{\alpha_i\} = B_i - B_j$ e $\{\beta_i\} = B_j - B_i$. Assim, por manipulação algébrica, obtém-se $\{\alpha_i\} \cup B_j = (B_i - B_j) \cup B_j = B_i \cup B_j$ e $\{\beta_i\} \cup B_i = (B_j - B_i) \cup B_i = B_j \cup B_i$. Logo, $\{\alpha_i\} \cup B_j = \{\beta_i\} \cup B_i$. Como $C_j = \{a_j\} \cup B_j$ e $\alpha_i = a_j$, verifica-se que $C_j = \{\alpha_i\} \cup B_j = \{\beta_i\} \cup B_i$. Assim, o vértice β_i é adjacente aos vértices da k -clique B_i .

Nos dois casos, o vértice β_i é adjacente aos vértices da k -clique B_i . Como são $k + 1$ cores, o vértice β_i possui a cor que não ocorre nos vértices de B_i . Como $C_i = B_i \cup \{a_i\}$, o vértice a_i possui a cor que não ocorre nos vértices de B_i . Logo, os vértices a_i e β_i possuem a mesma cor. \square

Lema 7.6: Sejam uma k -árvore $G = (V, E)$ e $\langle a_1, \dots, a_{n-k}, B \rangle$ o esquema de eliminação perfeita associado ao código compacto $(B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ de G . Então, $\beta_i \in \{a_{i+1}, \dots, a_{n-k}\} \cup B$.

Prova: Pela Definição 6.3 de código compacto, existem dois casos.

- $\beta_i = a_{n-k}$ e o teorema vale trivialmente.
- $\beta_i = B_j - B_i$ para um j tal que $i < j$. Logo, $\beta_i \in B_j$. Pela definição de código redundante, verifica-se que os vértices $a_1, \dots, a_j \notin B_j$. Logo, $\beta_i \notin \{a_1, \dots, a_j\}$. Como todo vértice ocorre uma vez no esquema de eliminação perfeita, conclui-se que $\beta_i \in \{a_{j+1}, \dots, a_{n-k}\} \cup B$. Como $i < j$, tem-se que $\beta_i \in \{a_{i+1}, \dots, a_j\} \cup \{a_{j+1}, \dots, a_{n-k}\} \cup B$, ou seja, $\beta_i \in \{a_{i+1}, \dots, a_{n-k}\} \cup B$. \square

O Algoritmo 7.2 obtém uma coloração ótima dos vértices a partir do código compacto. Os vértices são coloridos na ordem inversa do esquema de eliminação perfeita $\langle a_1, \dots, a_{n-k}, B \rangle$ obtido através do Algoritmo 6.1. Primeiro, são atribuídas $k + 1$ cores distintas aos vértices de $\{a_{n-k}\} \cup B$. Após colorir os vértices $B, a_{n-k}, \dots, a_{i+1}$, atribui-se a a_i a cor de β_i conforme o Lema 7.5. Isto é possível porque o vértice β_i já foi colorido, pois $\beta_i \in \{a_{i+1}, \dots, a_{n-k}\} \cup B$ de acordo com o Lema 7.6.

Teorema 7.7: O algoritmo que atribui uma coloração ótima aos vértices de uma k -árvore a partir do seu código compacto $(B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ tem complexidade $O(n)$.

Prova: O Algoritmo 6.1 que computa o esquema de eliminação perfeita $\langle a_1, \dots, a_{n-k}, B \rangle$ a partir do código compacto possui complexidade $O(n)$. A atribuição de

cores distintas aos $k+1$ vértices de $\{a_{n-k}\} \cup B$ é feita em $k+1$ iterações e a atribuição de cores aos vértices a_{n-k-1}, \dots, a_1 em $n-k-1$ passos. Assim, o algoritmo que obtém uma coloração ótima dos vértices possui complexidade $O(n)$. \square

Algoritmo Coloração_Vértices;
Entrada: código compacto $(B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$;
Saída: coloração dos vértices;
Início
 Obter $\langle a_1, \dots, a_{n-k}, B \rangle$ utilizando o Algoritmo 6.1;
 Atribuir $k+1$ cores distintas aos vértices de $\{a_{n-k}\} \cup B$;
 Para i de $n-k-1$ até 1 faça
 Atribuir a a_i a cor de β_i ;
Fim

Algoritmo 7.2: Coloração dos vértices de uma k -árvore a partir de seu código compacto.

7.3 - Seqüência de Multiplicidades em uma k -árvore

Na Seção 5.3, a Definição 5.6 de multiplicidade de uma k -clique generaliza o conceito de grau de vértice. Assim, para $k=1$, a multiplicidade (grau) de uma 1-clique (vértice) v de uma 1-árvore (árvore) é a quantidade de 2-cliques (arestas) que contêm v . A seqüência de graus de um grafo é formada pelos graus dos n vértices de um grafo em ordem crescente de graus. Portanto, a **seqüência de multiplicidades** em uma k -árvore G é formada pelas multiplicidades das $k(n-k)+1$ cliques de tamanho k de G em ordem crescente de multiplicidades.

A 3-árvore da Figura 6.1 possui a seqüência de cliques maximais $\langle C_1, C_2, C_3, C_4, C_5, C_6, C_7 \rangle = \langle \{3,5,6,9\}, \{2,4,9,10\}, \{5,6,9,10\}, \{2,7,9,10\}, \{1,2,6,8\}, \{1,2,6,10\}, \{2,6,9,10\} \rangle$. Pelo Teorema 1.3, esta 3-árvore possui 22 cliques de tamanho 3. Observe que apenas a 3-clique $\{2,9,10\}$ possui multiplicidade 3, pois está contida nas 4-cliques C_2, C_4 e C_7 enquanto as 3-cliques $\{5,6,9\}, \{6,9,10\}, \{1,2,6\}$ e $\{2,6,10\}$ possuem multiplicidade 2, pois estão contidas nas cliques C_1 e C_3, C_3 e C_7, C_5 e C_6, C_6 e C_7 , respectivamente. As demais 17 cliques de tamanho 3 possuem multiplicidade 1. Assim, a seqüência de multiplicidades desta 3-árvores é $\langle 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,3 \rangle$ que pode ser representada de modo simplificado como $\langle 1^{17}, 2^4, 3^1 \rangle$.

Nesta seção, apresentamos dois resultados sobre multiplicidade de k -cliques em k -árvores. O primeiro sobre seu somatório e o segundo relaciona as multiplicidades das

k -cliques do código de Prüfer $\langle B_1, \dots, B_{n-k-1} \rangle$ com os pares (α_i, β_i) do código compacto. Com base neste último resultado, apresentamos um algoritmo com complexidade $O(n)$ para obtenção da seqüência de multiplicidades de uma k -árvore com n vértices.

É bem conhecido na literatura que o somatório dos graus dos vértices de um grafo é igual a duas vezes a quantidade de arestas. No teorema a seguir, provamos que o somatório das multiplicidades de uma k -árvore G é igual a $k + 1$ vezes a quantidade de $(k+1)$ -cliques de G .

Teorema 7.8: Seja $\langle mult(1), \dots, mult(k[n - k] + 1) \rangle$ a seqüência de multiplicidades de uma k -árvore $G = (V, E)$. Então,

$$\sum_{i=1}^{k(n-k)+1} mult(i) = (k + 1)(n - k).$$

Prova: Sabe-se do Teorema 1.3 que uma k -árvore possui $n - k$ cliques de tamanho $k + 1$. Como cada $(k+1)$ -clique possui $k + 1$ cliques de tamanho k , então cada $(k+1)$ -clique contribui com $k + 1$ unidades no somatório. \square

Já foi comentado que seqüência de multiplicidades da 3-árvore da Figura 6.1 é $\langle 1^{17}, 2^4, 3^1 \rangle$. O somatório das multiplicidades é igual a 28 e pode ser obtido diretamente do Teorema 7.8.

O lema a seguir mostra que existe uma relação biunívoca entre a clique B_i e o par (α_i, β_i) , $1 \leq i < n - k$. Assim, a quantidade de ocorrências da clique B_i no código de Prüfer $\langle B_1, \dots, B_{n-k-1} \rangle$ é igual à quantidade de vezes que o par (α_i, β_i) aparece no código compacto. Este lema é utilizado na prova do Teorema 7.10 que determina a multiplicidade de B_i através da inspeção do código compacto.

Lema 7.9: Sejam uma k -árvore $G = (V, E)$, $(B, \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ o código compacto e $\langle (a_1, \dots, a_{n-k}), \langle B_1, \dots, B_{n-k} \rangle \rangle$ o seu código redundante. Para $1 \leq i < q < n - k$:

$$B_i = B_q \text{ se e somente se } (\alpha_i, \beta_i) = (\alpha_q, \beta_q).$$

Prova: (\Rightarrow) Pela Definição 6.3 de código compacto, existem dois casos.

- $B_i = B_q = B_{n-k}$ e $(\alpha_i, \beta_i) = (\alpha_q, \beta_q) = (a_{n-k}, a_{n-k})$.
- $B_i = B_q \neq B_{n-k}$ e $j = \min \{t \mid i < t \leq n - k \wedge a_t \in B_i\} = \min \{t \mid q < t \leq n - k \wedge a_t \in B_q\}$.
Logo, $(\alpha_i, \beta_i) = (\alpha_q, \beta_q) = (B_i - B_j, B_j - B_i)$.

(\Leftarrow) Pelo item 5 do Teorema 6.6, $B_i = \{\alpha_i\} \cup B_j - \{\beta_i\}$ tal que $\alpha_i = a_j$. Como $\alpha_i = \alpha_q = a_j$ e $\beta_i = \beta_q$, tem-se que $B_i = B_q = \{\alpha_q\} \cup B_j - \{\beta_q\}$. \square

Teorema 7.10: Sejam uma k -árvore $G = (V, E)$, $(B, \langle(\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1})\rangle)$ o seu código compacto e $\langle B_1, \dots, B_{n-k-1} \rangle$ o seu código de Prüfer. A multiplicidade da k -clique B_i é igual a um mais o número de vezes que o par (α_i, β_i) aparece no código compacto.

Prova: Pelo Lema 5.7, uma k -clique Q de G aparece $mult(Q) - 1$ vezes em $\langle B_1, \dots, B_{n-k-1} \rangle$. Pelo lema anterior, a k -clique B_i aparece em $\langle B_1, \dots, B_{n-k-1} \rangle$ toda vez que o par (α_i, β_i) aparece no código compacto. Logo, a multiplicidade de B_i é igual a um mais o número de vezes que o par (α_i, β_i) aparece no código compacto. \square

Pelo Lema 5.7, uma k -clique Q de G aparece $mult(Q) - 1$ vezes em $\langle B_1, \dots, B_{n-k-1} \rangle$. Assim, todas as k -cliques de G que não ocorrem no código de Prüfer possuem multiplicidade igual a um. Pelo Teorema 7.10, a multiplicidade da k -clique B_i é igual a um mais o número de vezes que o par (α_i, β_i) aparece no código compacto. Desse modo, para obter as multiplicidades que são maiores que um, basta analisar os pares (α_i, β_i) .

No Algoritmo 7.3, ordenam-se lexicograficamente em ordem crescente os pares (α_i, β_i) . Para cada par (α, β) , armazena-se na variável *num* a quantidade de vezes consecutivas que este par ocorre nesta ordenação. Logo, pelo Teorema 7.10, a k -clique associada a cada par possui multiplicidade $num + 1$.

Sabe-se que uma k -árvore possui no máximo $n - k$ cliques maximais e, portanto, a maior multiplicidade possível é $n - k$. Desse modo, utiliza-se o vetor $q[1 \dots n - k]$ para descrever a seqüência de multiplicidades $\langle 1^{q[1]}, \dots, n - k^{q[n-k]} \rangle$. Nesta representação, a posição $q[i]$ armazena a quantidade de k -cliques que possuem multiplicidade i . A variável *qtd* armazena o total de k -cliques que possuem multiplicidade maior que um. A subtração desta variável do total de k -cliques que a k -árvore possui fornece a quantidade de k -cliques que possuem multiplicidade um.

Teorema 7.11: O algoritmo que determina a seqüência de multiplicidades em uma k -árvore a partir do seu código compacto tem complexidade $O(n)$.

Prova: A ordenação dos $n - k - 1$ pares (α_i, β_i) é feita com a aplicação da ordenação por distribuição executada duas vezes. Logo, este passo tem complexidade

$O(n)$. O vetor é inicializado em $n - k$ passos. Percorre-se uma única vez a ordenação crescente de pares para obter as multiplicidades e, portanto, este passo tem complexidade $O(n)$. Logo, o algoritmo tem complexidade $O(n)$. \square

```

Algoritmo Seqüência_de_Multiplicidades;
Entrada: Código Compacto  $(B, S = \langle (\alpha_1, \beta_1), \dots, (\alpha_{n-k-1}, \beta_{n-k-1}) \rangle)$ ;
Saída: Seqüência de Multiplicidades  $\langle 1^{q[1]}, \dots, n-k^{q[n-k]} \rangle$ ;
Início
    Ordene lexicograficamente em ordem crescente os pares  $(\alpha_i, \beta_i)$ ;
    Para  $i$  de 2 até  $n-k$  faça
         $q[i] \leftarrow 0$ ;
     $qtd \leftarrow 0$ ;
    Para cada par  $(\alpha, \beta)$  distinto na ordenação crescente faça
         $qtd \leftarrow qtd + 1$ ;
         $num \leftarrow$  número de pares iguais consecutivos na ordenação;
         $q[num+1] \leftarrow q[num+1] + 1$ ;
     $q[1] \leftarrow n * (n-k) + 1 - qtd$ ;
Fim

```

Algoritmo 7.3: Determinação da seqüência de multiplicidades a partir do código compacto.

As soluções dos problemas vistos neste capítulo também podem ser aplicadas a uma k -árvore representada pela sua lista de adjacências. Para isto, basta utilizar os algoritmos de codificação vistos na Seção 6.4 antes de aplicar os algoritmos vistos nesta seção. Este passo a mais, aumenta a complexidade dos algoritmos de $O(n)$ para $O(m)$.

Para concluir, vale ressaltar que KUMAR e MADAVAM [28] utilizam um conceito diferente para multiplicidades. Particularmente, para as k -árvores, a definição destes autores leva a uma unidade a menos que a definição utilizada nesta tese. Em [28], quando aplicado a k -árvores, o algoritmo proposto pelos autores possui complexidade $O(m \log n)$ para obter as cliques e suas multiplicidades. A obtenção da seqüência de multiplicidades utilizando este algoritmo também possui complexidade $O(m \log n)$, sendo portanto menos eficiente do que o algoritmo proposto nesta seção.

8 - Relacionamentos entre as Famílias Estudadas

Este capítulo tem como objetivos principais propor uma nova subfamília dos grafos de intervalo e delinear com precisão a interseção entre várias subfamílias dos grafos cordais. Algumas destas famílias já foram apresentadas durante esta tese enquanto outras são apresentadas neste capítulo. O inter-relacionamento entre as famílias decorre de propriedades agrupadas aqui em três tipos: as que são relativas a resultados da literatura, as que foram propostas em capítulos anteriores e as que são propostas neste capítulo.

Na Seção 8.1, são revistas as definições das famílias já citadas na tese e de outras que aparecem apenas neste capítulo. Na Seção 8.2, definimos e propomos uma caracterização para os grafos de intervalo k -estritos, que são uma nova subfamília dos grafos de intervalo. Além disso, provamos que os grafos k -caminho (definido no Capítulo 3) são grafos de intervalo $(k+1)$ -estritos e que os caterpillars (definido na Seção 8.1) são exatamente a família dos grafos de intervalo 2-estritos. Na Seção 8.3, apresentamos novas propriedades relativas aos k -intercats (definido na Seção 8.1). Em seguida, na Seção 8.4, obtemos novos resultados sobre grafos de intervalo e k -árvores. Provamos que os k -intercats são a interseção entre a família das k -árvores e a dos grafos de intervalo e que os grafos k -caminho são a interseção entre a subfamília das SC k -árvores (definida na Seção 8.1) e a dos grafos de intervalo. Finalmente, na Seção 8.5, utilizamos os resultados obtidos nas seções anteriores para delimitar as interseções entre as subfamílias dos grafos cordais aqui estudadas.

8.1 - Famílias Estudadas

Nesta seção, são revistas as definições das subfamílias dos grafos cordais cujas propriedades estruturais foram estudadas neste trabalho. São também apresentadas outras famílias, conhecidas na literatura, e para as quais estabelecemos novas relações de pertinência. As subfamílias são agrupadas em três grandes grupos: grafos de intervalo, subfamílias dos grafos planares e subfamílias das k -árvores.

Grafos de Intervalo

Definição 8.1: Um **grafo** é dito **de intervalo** quando para cada vértice pode ser atribuído um intervalo no eixo real de modo que exista uma interseção entre dois intervalos se e somente se existir uma aresta entre os vértices correspondentes.

HARARY E SCHWENK [22] foram pioneiros no estudo de propriedades da família dos caterpillars. Eles citam que A. HOBBS definiu os caterpillars como as árvores com pelo menos três vértices cuja remoção das folhas resulta em um caminho. Na definição de caterpillars vista a seguir, consideramos as árvores com 1 e 2 vértices como caterpillars.

Definição 8.2: Seja $G = (V, E)$ uma árvore com n vértices. Para $n = 1$ e $n = 2$, G é um **caterpillar** de **corpo vazio**. Para $n > 2$, G é um **caterpillar** de **corpo P** quando o grafo P obtido pela remoção das folhas de G é um caminho.

O Lema 8.3, visto a seguir, proposto por TROTTER e HARARY [47] em 1979, é um resultado bem conhecido na literatura e caracteriza os caterpillars como as árvores que são grafos de intervalo, ou seja, prova que os caterpillars são a interseção exata entre a família das árvores e a dos grafos de intervalo.

Lema 8.3 [47]: Uma árvore é grafo de intervalo se e somente se é um caterpillar.

Subfamílias dos Grafos Planares

Os grafos periplanares maximais (MOP - *maximal outerplanar*) e os grafos cordais planares maximais definidos a seguir são duas importantes subfamílias dos grafos planares que também pertencem à família dos grafos cordais.

Definição 8.4: Seja $G = (V, E)$ um grafo planar. O grafo G é dito **periplanar** se existe uma representação plana em que todos os vértices de V estão na face externa. Um grafo G periplanar é dito **periplanar maximal** se, para qualquer aresta $e \notin E$, o grafo $H = (V, E \cup \{e\})$ não é periplanar.

Definição 8.5: Seja $G = (V, E)$ um grafo cordal e planar. O grafo G é dito **cordal planar maximal** se, para qualquer aresta $e \notin E$, o grafo $H = (V, E \cup \{e\})$ não é cordal ou não é planar.

k-árvores e Subfamílias

Após relembrar as definições de *k*-árvore e de grafo *k*-caminho, são vistas as definições de quatro subfamílias das *k*-árvores: as SC *k*-árvores, os *k*-caterpillars, os *k*-intercats e as *k*-árvores estrela. As três primeiras já foram definidas na literatura e as *k*-árvores estrela são aqui definidas pela primeira vez.

Definição 8.6 (Definição 1.2): Uma *k*-árvore, sendo $k > 0$, é assim definida:

- (1) Um grafo completo com k vértices é uma *k*-árvore;
- (2) Se $G = (V, E)$ é uma *k*-árvore, $Q \subseteq V$ é uma *k*-clique e $v \notin V$, o grafo $G' = (V \cup \{v\}, E \cup \{\{v, w\} \mid w \in Q\})$ é uma *k*-árvore;
- (3) Nada mais são *k*-árvores.

Definição 8.7 (Definição 3.4): Um grafo *k*-caminho, sendo $k > 0$, é assim definido:

- (1) Um grafo completo com $k + 1$ vértices é um grafo *k*-caminho;
- (2) Se $G = (V, E)$ é um grafo *k*-caminho, $Q \subset V$ é uma *k*-clique que contém pelo menos um vértice simplicial e $v \notin V$, o grafo $G' = (V \cup \{v\}, E \cup \{\{v, w\} \mid w \in Q\})$ é um grafo *k*-caminho;
- (3) Nada mais são grafos *k*-caminho.

As SC *k*-árvores (*Simple Clique k-trees*) são uma subfamília das *k*-árvores definida por MARKENZON *et al.* [31]. Estes autores provaram que esta família contém duas subfamílias bastante estudadas: os grafos periplanares maximais, que são as SC 2-árvores, e os grafos cordais planares maximais, que são as SC 3-árvores. Além disso, mostraram que os grafos *k*-caminho são uma subfamília das SC *k*-árvores.

Definição 8.8: Uma SC *k*-árvore, sendo $k > 0$, é assim definida:

- (1) Um grafo completo com $k + 1$ vértices é uma SC *k*-árvore;
- (2) Se $G = (V, E)$ é uma SC *k*-árvore, $Q \subset V$ é uma *k*-clique contida em uma única clique maximal e $v \notin V$, o grafo $G' = (V \cup \{v\}, E \cup \{\{v, w\} \mid w \in Q\})$ é uma SC *k*-árvore;
- (3) Nada mais são SC *k*-árvores.

A primeira referência à família dos *k*-caterpillars foi feita por PROSKUROWSKI [40]. O *k*-caterpillar é uma generalização do conceito de caterpillar que, neste contexto,

passa a ser notado 1-caterpillar. Assim, os k -caterpillars são as k -árvores cuja remoção das k -folhas resultam em um grafo k -caminho. Introduzimos a Definição 8.9 de corpo de um grafo cordal para simplificar as definições vistas em seguida. Note-se que algumas definições encontradas na literatura não eram absolutamente precisas quanto a condições de contorno; neste caso, elas foram reescritas com esta preocupação em vista.

Definição 8.9: Seja $G = (V, E)$ um grafo cordal e S o seu conjunto de vértices simpliciais. O grafo $G[V - S]$ é chamado **corpo** de G .

Definição 8.10: Sejam $G = (V, E)$ uma k -árvore e P o seu corpo. Diz-se que G é um **k -caterpillar** quando P é: (i) um grafo vazio ou (ii) um grafo completo ou (iii) um grafo k -caminho.

Por exemplo, as duas 2-árvores da Figura 8.1 são 2-caterpillars cujos corpos estão em negrito e as k -folhas e arestas removidas na obtenção dos respectivos corpos estão em pontilhado.

A Definição 8.11 trata dos **k -caterpillars interiores**, ou simplesmente **k -intercats**, que foram definidos por PROSKUROWSKI em [40]. Recentemente, em outro trabalho deste autor [21], esta mesma família recebeu o nome de **k -caminhos completos**. Optamos pela nomenclatura inicialmente proposta, por acharmos ser a mais apropriada para esta subfamília dos k -caterpillars.

Antes da definição dos k -intercats, são revistas propriedades dos grafos k -caminho estudadas no Capítulo 3. Seja $G = (V, E)$ um grafo k -caminho e $\langle B_0, C_1, B_1, \dots, C_p, B_p \rangle$, $p > 1$, um k -caminho maximal de G . Já foi visto no Teorema 3.6 que todo k -caminho maximal de G contém a mesma seqüência de $(k+1)$ -cliques $\langle C_1, C_2, \dots, C_p \rangle$. Como $B_i = C_i \cap C_{i+1}$, as k -cliques B_1, \dots, B_{p-1} ocorrem em qualquer k -caminho maximal de G , ou seja, os k -caminhos maximais de G diferem apenas nas k -cliques B_0 e B_p dos extremos. Pelo Corolário 3.5 [31], para $n > k + 1$, existem k^2 k -caminhos maximais de G . Por exemplo, o corpo do 2-caterpillar da Figura 8.1(a) possui os 2^2 2-caminhos maximais:

$$\begin{aligned} & \langle \{a, e\}, \{a, b, e\}, \{a, b\}, \{a, b, c\}, \{b, c\}, \{b, c, d\}, \{b, d\} \rangle; \\ & \langle \{a, e\}, \{a, b, e\}, \{a, b\}, \{a, b, c\}, \{b, c\}, \{b, c, d\}, \{c, d\} \rangle; \\ & \langle \{b, e\}, \{a, b, e\}, \{a, b\}, \{a, b, c\}, \{b, c\}, \{b, c, d\}, \{b, d\} \rangle; \\ & \langle \{b, e\}, \{a, b, e\}, \{a, b\}, \{a, b, c\}, \{b, c\}, \{b, c, d\}, \{c, d\} \rangle. \end{aligned}$$

Definição 8.11: Sejam G um k -caterpillar e P o seu corpo. Diz-se que G é um k -intercat quando: (i) P é um grafo vazio ou (ii) P é um grafo completo com k vértices ou (iii) quando existe um k -caminho maximal $\langle B_0, C_1, B_1, \dots, C_p, B_p \rangle$ de P tal que todo vértice simplicial de G é adjacente aos k vértices de uma k -clique B_i .

Em uma k -árvore, por definição, uma k -folha v é adjacente aos k vértices de uma k -clique Q_v . Em um k -intercat, Q_v é uma das k -cliques B_0, \dots, B_p que ocorrem em um k -caminho maximal do corpo P . Por exemplo, no 2-intercat da Figura 8.1(a), Q_x é uma 2-clique B_i que ocorre em todos os 2-caminhos maximais do corpo P enquanto Q_y, Q_z e Q_w são 2-cliques do 2-caminho maximal $\langle \{a,e\}, \{a,b,e\}, \{a,b\}, \{a,b,c\}, \{b,c\}, \{b,c,d\}, \{c,d\} \rangle$ de P . Por outro lado, verifica-se que o 2-caterpillar da Figura 8.1(b) não é um 2-intercat por dois motivos. Primeiro, porque Q_x não é uma 2-clique B_i em nenhum 2-caminho maximal de P e segundo porque nenhum 2-caminho maximal de P contém Q_z e Q_w como 2-cliques B_i ao mesmo tempo.

Observe que, no caso em que $k = 1$, todo caterpillar é um intercat. Isso ocorre, porque existe um único caminho maximal $\langle B_0, C_1, B_1, \dots, C_p, B_p \rangle = \langle v_0, e_1, v_1, \dots, e_p, v_p \rangle$ do corpo P e, obviamente, toda folha é adjacente a um vértice v_i (1-clique B_i) do corpo P .

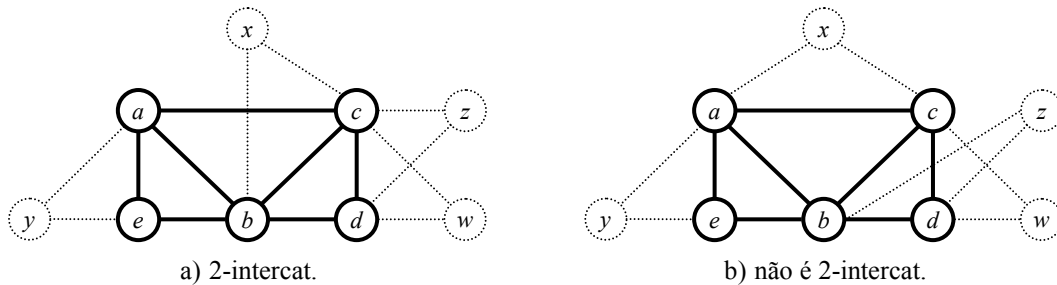


Figura 8.1: 2-caterpillars.

As árvores com n vértices que possuem $n - 1$ folhas são chamadas de árvores estrela. Na próxima definição, generalizamos este conceito para k -árvores.

Definição 8.12: Seja $G = (V, E)$ uma k -árvore com $n > k + 1$ vértices que possui f vértices simpliciais. Diz-se que G é uma **k -árvore estrela** quando $f = n - k$.

Sabe-se do Lema 1.6 que toda k -árvore, para $n > k + 1$, possui no mínimo 2 e no máximo $n - k$ vértices simpliciais. Além disso, sabe-se do Teorema 3.3 que as k -árvores com exatamente dois vértices simpliciais são os grafos k -caminho. Portanto, os grafos

k -caminho e as k -árvores estrela são os casos extremais em relação à quantidade de vértices simpliciais. Cabe salientar que, diferente dos grafos k -caminho que possuem vários exemplares não isomorfos com $n > k + 2$ vértices, existe apenas uma k -árvore estrela com n vértices.

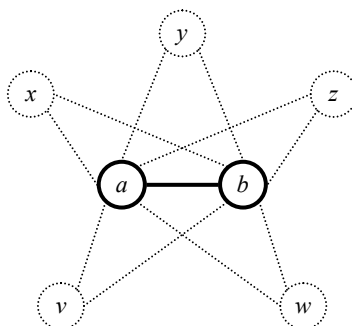


Figura 8.2: 2-árvore estrela.

Observe que as k -árvores estrela sempre possuem como corpo um grafo completo com k vértices. Logo, pela Definição 8.11, as k -árvores estrela são k -intercats. Por exemplo, a 2-árvore estrela da Figura 8.2 possui $n - k = 7 - 2 = 5$ vértices simpliciais e seu corpo é o grafo completo com 2 vértices que está em negrito.

8.2 - Grafos de Intervalo k -estritos

Nesta seção, introduzimos uma nova subfamília dos grafos de intervalo, que chamamos de grafos de intervalo k -estritos, e propomos uma caracterização para esta subfamília com base na cardinalidade de suas cliques maximais. Além disso, provamos dois resultados importantes sobre grafos de intervalo $(k+1)$ -estritos. Primeiro, que eles contêm a interseção entre os grafos de intervalo e as k -árvores e segundo, que eles contêm os grafos k -caminho. Mais adiante, na Seção 8.4, provamos a interseção exata entre eles e as k -árvores. Finalizando a sessão, provamos que os caterpillars são equivalentes à família dos grafos de intervalo 2-estritos.

Antes de propormos a definição dos grafos de intervalo k -estritos, relembremos a caracterização clássica dos grafos de intervalo que utiliza uma ordenação das cliques maximais do grafo. Observe que, utilizando o Lema 8.13 e a propriedade da interseção da árvore de cliques, verifica-se que apenas os grafos de intervalo possuem uma árvore de cliques que é um caminho.

Lema 8.13: Um grafo $G = (V, E)$ é de intervalo se e somente se as cliques maximais de G podem ser linearmente ordenadas de maneira que, para todo $v \in V$, as cliques maximais contendo v são consecutivas.

Definição 8.14: Um grafo de **intervalo k -estrito** é um grafo de intervalo que possui uma associação de vértices a intervalos do eixo real tal que todo número real pertence a 0 ou k intervalos.

A Figura 8.3 mostra dois grafos de intervalo. O grafo de intervalo da Figura 8.3(a) não é um grafo de intervalo k -estrito, pois não é possível atribuir-lhe intervalos que satisfaçam a Definição 8.14. Já o grafo da Figura 8.3(b) é um grafo de intervalo 3-estrito porque possui uma representação por intervalos onde todos os valores que pertencem a algum intervalo pertencem a exatamente 3 intervalos.

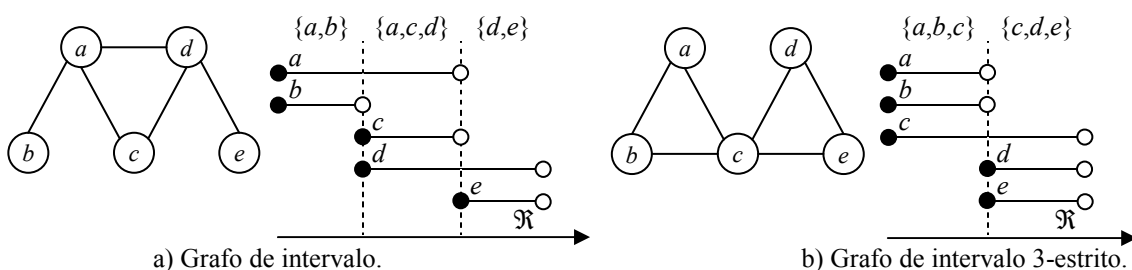


Figura 8.3: Dois grafos de intervalo e suas representações por intervalo.

Em seguida, propomos um teorema de caracterização para os grafos de intervalo k -estritos com base na cardinalidade de suas cliques maximais.

Teorema 8.15: Um grafo é de intervalo k -estrito se e somente se é grafo de intervalo e todas as suas cliques maximais possuem cardinalidade k .

Prova: (\Rightarrow) Seja G um grafo de intervalo k -estrito. Seja uma associação de vértices a intervalos do eixo real que satisfaz a Definição 8.14. Assim, todo número real que pertence a um intervalo, pertence a k intervalos I_1, \dots, I_k cujos vértices associados são v_1, \dots, v_k . Logo, para $1 \leq i < j \leq k$, v_i e v_j são adjacentes, ou seja, $\{v_1, \dots, v_k\}$ é uma k -clique de G . Desse modo, todas as cliques maximais de G têm cardinalidade k .

(\Leftarrow) Seja G um grafo de intervalo que possui todas as cliques maximais com cardinalidade k . Sejam $\gamma(v) = \min\{j \mid v \in C_j\}$ e $\eta(v) = \max\{i \mid v \in C_i\}$. Pelo Lema 8.13,

existe uma ordenação $\langle D_1, D_2, \dots, D_p \rangle$ tal que, $\forall v \in V$, as k -cliques maximais contendo o vértice v são consecutivas. Logo, apenas as cliques consecutivas $D_{\chi(v)}, D_{\chi(v)+1}, \dots, D_{\eta(v)}$ contêm o vértice v . Desse modo, os intervalos $I_v = [\chi(v), \eta(v)+1[$ satisfazem a definição de grafo de intervalo. Observe que para $x \notin [1, p+1[$ não existe nenhum intervalo I_x associado. Além disso, para $x \in [1, p+1[$, x está contido nos k intervalos associados aos vértices da k -clique D_i onde i é a parte inteira de x . Logo, G é grafo de intervalo k -estrito. \Rightarrow

Os resultados que se seguem mostram relações existentes entre os grafos de intervalo k -estritos e as famílias já estudadas. No Teorema 8.16, provamos que os grafos de intervalo $(k+1)$ -estritos contêm a interseção entre a família das k -árvores e a dos grafos de intervalo. Já foi provado no Teorema 3.6 do Capítulo 3 que os grafos k -caminho possuem uma única árvore de cliques que é um caminho. Isto equivale a dizer que os grafos k -caminho são grafos de intervalo. No Corolário 8.17, mostramos que os grafos k -caminho são na verdade grafos de intervalo $(k+1)$ -estritos.

Teorema 8.16: Seja G uma k -árvore com n vértices, sendo $n > k$. Se G é grafo de intervalo, então G é grafo de intervalo $(k+1)$ -estrito.

Prova: Quando $n > k$, todas as cliques maximais de uma k -árvore G têm cardinalidade $k + 1$. Pelo Teorema 8.15, se G é grafo de intervalo e suas cliques maximais têm cardinalidade $k + 1$, então, G é grafo de intervalo $(k+1)$ -estrito. \Rightarrow

Corolário 8.17: Todo grafo k -caminho é grafo de intervalo $(k+1)$ -estrito.

A Figura 8.4 mostra uma representação por intervalos para o grafo 4-caminho visto na Figura 4.1 cuja seqüência de cliques maximais é $\langle C_1, \dots, C_7 \rangle = \langle \{a,b,c,d,e\}, \{b,c,d,e,f\}, \{c,d,e,f,g\}, \{d,e,f,g,h\}, \{d,e,f,h,i\}, \{d,e,h,i,j\}, \{e,h,i,j,k\} \rangle$. Sabe-se do Corolário 8.17 que este grafo é um grafo de intervalo 5-estrito. Observe que os intervalos vistos na Figura 8.4 são obtidos utilizando $I_v = [\chi(v), \eta(v)+1[$, onde $\chi(v) = \min\{j \mid v \in C_j\}$ e $\eta(v) = \max\{i \mid v \in C_i\}$. Por exemplo, para o vértice f , verifica-se que $\chi(f) = 2$ e $\eta(f) = 5$; portanto, obtém-se $I_f = [2,6[$.

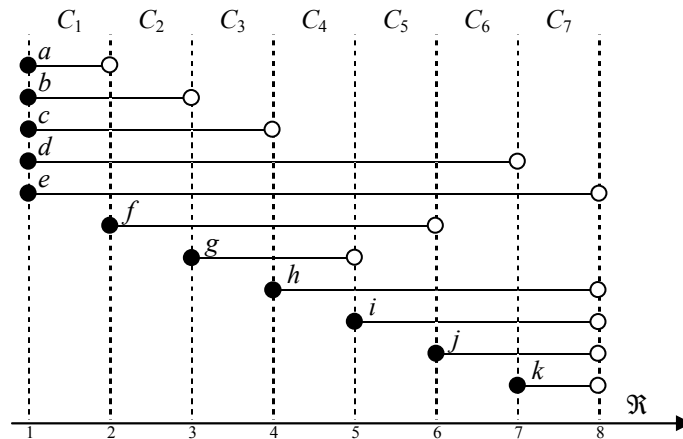


Figura 8.4: Representação por intervalos de um grafo 4-caminho.

No Teorema 8.18 visto a seguir, provamos que os grafos de intervalo 2-estritos e os caterpillars são equivalentes. Portanto, os grafos de intervalo 2-estritos são uma subfamília das árvores. Apesar disso, para $k > 1$, existem grafos de intervalo $(k+1)$ -estritos que não são k -árvores. Por exemplo, o grafo de intervalo 3-estrito da Figura 8.3(b) não é uma 2-árvore.

Teorema 8.18: Um grafo é caterpillar se e somente se é grafo de intervalo 2-estrito.

Prova: (\Rightarrow) Seja G um caterpillar. Pelo Lema 8.3, G é árvore e grafo de intervalo. Pelo Teorema 8.16, G é grafo de intervalo 2-estrito.

(\Leftarrow) Seja G um grafo de intervalo 2-estrito. Pela Definição 8.14, G é grafo de intervalo e, pelo Teorema 8.15, todas as cliques maximais de G têm cardinalidade 2. Sabe-se do Lema 8.13 que as 2-cliques (arestas) de G podem ser linearmente ordenadas tal que, $\forall v \in V$, as 2-cliques (arestas) contendo v são consecutivas. Logo, G não possui ciclos, ou seja, G é uma árvore. Pelo Lema 8.3, G é caterpillar. \Rightarrow

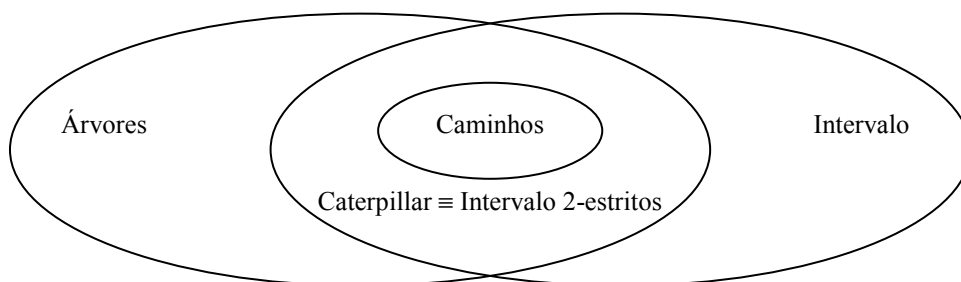


Figura 8.5: Diagramas com as subclasses das árvores e a dos grafos de intervalo.

O diagrama da Figura 8.5 adiciona esta equivalência ao clássico resultado do Lema 8.3 que define os caterpillars como a interseção entre as famílias das árvores e a dos grafos de intervalo.

8.3 - Resultado sobre k -intercats

Nesta seção, apresentamos um novo resultado sobre k -intercats, mostrando que k -árvores com certo número de vértices simpliciais pertencem obrigatoriamente a esta família.

Já foi comentado que os grafos k -caminho e as k -árvores estrela são os casos extremos de k -árvores em relação à quantidade de vértices simpliciais. No próximo teorema, provamos que os k -intercats contêm estas duas subfamílias das k -árvores.

Teorema 8.19: Seja $G = (V, E)$ uma k -árvore com n vértices que possui f vértices simpliciais. Se $f=2$ ou $f=n-k$, então G é um k -intercat.

Prova: Quando $f = n - k$, o corpo é um grafo completo com k vértices. Pela Definição 8.10, G é um k -caterpillar e, pela Definição 8.11, G é um k -intercat.

Quando $f = 2$, G é um grafo k -caminho. Se $n = k + 2$, então G é um k -intercat cujo corpo é um grafo completo com k vértices. Seja $\langle B_0, C_1, B_1, \dots, C_p, B_p \rangle$, $p > 2$, um k -caminho maximal de G . Sabe-se que l_1 e r_{p-1} são os vértices simpliciais de G e que $C_1 = B_1 \cup \{l_1\}$ e $C_p = B_{p-1} \cup \{r_{p-1}\}$. Assim, o corpo P de G é um grafo k -caminho e $\langle B_1, C_2, B_2, \dots, C_{p-1}, B_{p-1} \rangle$, $p > 2$, é um k -caminho maximal de P . Logo, G é um k -caterpillar. Como os vértices simpliciais de G são adjacentes as k -cliques B_1 e B_{p-1} , tem-se que G é um k -intercat. \Rightarrow

Seja G uma k -árvore com n vértices. Para $n \leq k + 3$ e k fixo, existem apenas cinco k -árvores não rotuladas. Para $n = k$, $n = k + 1$ e $n = k + 2$ existe um único exemplar e para $n = k + 3$ existem dois. Estas cinco k -árvores são k -intercats e podem ser separadas em quatro casos:

- $n = k$ e $n = k + 1$. São k -intercats cujo corpo é o grafo vazio.

- $n = k + 2$. É um k -intercat cujo corpo é o grafo completo com k vértices e possui dois vértices simpliciais, ou seja, pertence às famílias dos grafos k -caminho e das k -árvores estrela.
- $n = k + 3$ e $f = 2$. É um grafo k -caminho cujo corpo possui $k + 1$ vértices.
- $n = k + 3$ e $f = n - k = 3$. É uma k -árvore estrela cujo corpo possui k vértices.

A Figura 8.6 ilustra as k -árvores com $k + 2$ e $k + 3$ vértices para o caso particular em que $k = 2$. Nesta figura, os corpos dos 2-intercats estão em negrito e as 2-folhas e arestas removidas na obtenção dos respectivos corpos estão em pontilhado.

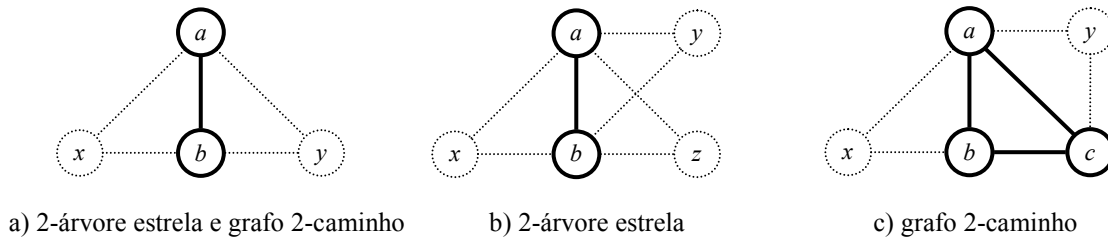


Figura 8.6: 2-intercats.

8.4 - Resultados sobre Grafos de Intervalo e k -árvores

Como uma árvore é grafo de intervalo se e somente se é um caterpillar (Lema 8.3), a extensão intuitiva deste lema seria que uma k -árvore é grafo de intervalo se e somente se é um k -caterpillar. Apesar de KUMAR e MADHAVAN [27] terem apresentado esta generalização, este resultado não é válido, pois, para $k > 1$, existem k -caterpillars que não são grafos de intervalo. Na verdade, em [27], estes autores definiram os k -intercats ao invés dos k -caterpillars.

Nesta seção, primeiramente, provamos que os k -intercats são a interseção entre as k -árvores e os grafos de intervalo. Em seguida, provamos que os grafos k -caminho são a interseção entre a família das SC k -árvores e a dos grafos de intervalo. Como consequência direta deste resultado sobre grafos k -caminho, obtemos a interseção entre os grafos de intervalo e duas importantes famílias dos cordais: os grafos periplanares maximais e os grafos cordais planares maximais.

A seguir, são apresentados o conceito de asteróide triplo e um teorema de reconhecimento de grafos de intervalo. Estes resultados da literatura são utilizados para mostrar que, para $k > 1$, nem todo k -caterpillar é grafo de intervalo através de um contra-exemplo: um 2-caterpillar que não é grafo de intervalo.

Definição 8.20: Três vértices (x,y,z) de um grafo formam um **asteróide triplo** quando existe um caminho entre qualquer par de vértices desta tripla que não contém vértices adjacentes ao terceiro.

Por exemplo, no grafo da Figura 8.7, a tripla (x,y,z) é um asteróide triplo porque os caminhos $\langle x,a,y \rangle$, $\langle x,b,z \rangle$ e $\langle y,c,z \rangle$ não contêm vértices adjacentes a z , y e x , respectivamente. O próximo teorema, proposto por LEKKERKERKER e BOLAND [29] em 1962, utiliza este conceito para verificar se um grafo cordal é de intervalo.

Teorema 8.21 [29]: Um grafo é de intervalo se e somente se é cordal e não possui asteróide triplo.

Com base neste teorema, conclui-se que o grafo da Figura 8.7 não é de intervalo, pois contém o asteróide triplo (x,y,z) . É fácil verificar que este grafo é uma 2-árvore, construída a partir da 2-árvore formada pelos vértices $\{a,b,c\}$ pela adição dos vértices x , y e z adjacentes aos vértices das 2-cliques $\{a,b\}$, $\{a,c\}$ e $\{b,c\}$, respectivamente. Esta 2-árvore possui os vértices simpliciais x , y e z e seu corpo é um grafo completo com os vértices a , b e c . Desse modo, conclui-se que o grafo da Figura 8.7 é um 2-caterpillar, mas não é um grafo de intervalo.

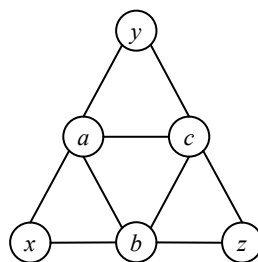


Figura 8.7: 2-caterpillar com asteróide triplo (x,y,z) .

Observe que o 2-caterpillar da Figura 8.7 não é um 2-intercat porque nenhum dos 2-caminhos maximais $P_1 = \langle \{a,c\}, \{a,b,c\}, \{b,c\} \rangle$, $P_2 = \langle \{a,b\}, \{a,b,c\}, \{b,c\} \rangle$ e $P_3 = \langle \{a,b\}, \{a,b,c\}, \{a,c\} \rangle$ do seu corpo satisfaz a Definição 8.11 de k -intercat, já que os vértices x , y e z não são adjacentes a nenhuma 2-clique B_i de P_1 , P_2 e P_3 , respectivamente.

No próximo teorema, temos um importante resultado que caracteriza os k -intercats como as k -árvores que são grafos de intervalo. Este resultado delimita com exatidão a interseção entre a família das k -árvores e a dos grafos de intervalo.

Teorema 8.22: Uma k -árvore é um grafo de intervalo se e somente se é um k -intercat.

Prova: As k -árvores com k e $k + 1$ vértices são grafos completos e, portanto, são grafos de intervalo. Como possuem corpo vazio, são k -intercats. Assim, basta provar que o teorema é válido para k -árvores com mais de $k + 1$ vértices.

Sabe-se do Lema 1.4 que os vértices simpliciais de um grafo cordal pertencem a apenas uma clique maximal e do Lema 1.5 que toda clique maximal de uma k -árvore contém no máximo um vértice simplicial. Estes resultados são utilizados durante a prova do teorema.

(\Rightarrow) Seja G uma k -árvore que é grafo de intervalo. Seja $\langle C_1, C_2, \dots, C_p \rangle, p > 1$, uma ordenação das cliques maximais que satisfaz o Lema 8.13. Como esta seqüência define uma árvore de cliques que é um caminho, então $|C_i \cap C_{i+1}| = k$, para $1 \leq i < p$.

Quando as cliques C_i não possuem vértices simpliciais para $1 < i < p$, G é um grafo k -caminho, que é um k -intercat. Seja uma clique maximal $C_i = \{v_i\} \cup B_i$, $1 < i < p$, contendo um vértice simplicial v_i . Como $v_i \notin C_{i-1}$, $v_i \notin C_{i+1}$ e $|C_{i-1} \cap C_i| = |C_i \cap C_{i+1}| = k$, então $C_{i-1} \cap C_{i+1} = B_i$. Logo, na k -árvore $G - \{v_i\}$, nenhum vértice de B_i é simplicial, pois B_i está contido em duas cliques maximais C_{i-1} e C_{i+1} .

Seja P^* a k -árvore obtida pela remoção dos vértices simpliciais contidos nas cliques C_i para $1 < i < p$. Como agora apenas C_1 e C_p possuem vértices simpliciais, então P^* é um grafo k -caminho. Ao remover os vértices simpliciais l_1 e r_{p-1} de P^* , obtém-se o corpo P de G que é um grafo k -caminho. Seja $\langle B_1, C_a, B_a, C_b, B_b, \dots, B_m, C_m, B_{p-1} \rangle$ um k -caminho maximal de P onde $B_1 = C_1 - \{l_1\}$ e $B_{p-1} = C_p - \{r_{p-1}\}$. Como todo vértice simplicial de G é adjacente a uma k -clique B_i deste k -caminho maximal de P , então G é um k -intercat.

Como caso particular, se todas as cliques C_i possuem vértices simpliciais, então G é uma k -árvore estrela, ou seja, é um k -intercat cujo corpo é um grafo completo com os k vértices de B_i .

(\Leftarrow) Seja G um k -intercat. Seja $\langle B_0, C_1, B_1, C_2, B_2, \dots, B_{p-1}, C_p, B_p \rangle$ um k -caminho maximal do corpo P tal que os vértices simpliciais são adjacentes aos k vértices de alguma clique B_i . Sabe-se do Corolário 8.17 que os grafos k -caminho são grafos de intervalo. Logo, pelo Lema 8.13, $\langle C_1, \dots, C_p \rangle$ é uma ordenação de cliques tal que $\forall v \in V$ as cliques C_i contendo v são consecutivas.

Sejam os vértices simpliciais v_{i1}, \dots, v_{ix} adjacentes aos vértices de uma clique B_i . A seqüência $\langle C_1, C_2, \dots, C_i, \{v_{i1}\} \cup B_i, \dots, \{v_{ix}\} \cup B_i, C_{i+1}, \dots, C_p \rangle$ mantém a propriedade de todas as cliques contendo um vértice v serem consecutivas, pois os k vértices comuns a C_i e C_{i+1} são os vértices de B_i que estão presentes nas cliques inseridas e cada vértice simplicial v_{ij} ocorre apenas em uma clique. Repetindo, o processo para todos os vértices simpliciais, obtém-se uma ordenação onde as cliques contendo um vértice v são consecutivas. Logo, pelo Lema 8.13, G é grafo de intervalo. \Rightarrow

No próximo teorema, provamos que os grafos k -caminho são a interseção entre a família das SC k -árvores e a dos grafos de intervalo.

Teorema 8.23: Uma SC k -árvore é grafo de intervalo se e somente se é grafo k -caminho.

Prova: (\Rightarrow) Seja G uma SC k -árvore que é grafo de intervalo. Utilizando raciocínio análogo ao utilizado na prova do Teorema 8.22, obtém-se a seqüência $S = \langle C_1, B_1, C_2, B_2, \dots, B_{p-1}, C_p \rangle$ onde $B_i = C_i \cap C_{i+1}$. Como G é uma SC k -árvore, então $B_1 \neq \dots \neq B_p$. Para $1 < i < p$, já foi visto na prova do Teorema 8.22 que B_i não possui vértice simplicial. Observe que $C_i = B_{i-1} \cup B_i$, $1 < i < p$. Logo, apenas C_1 e C_p possuem um vértice simplicial cada; portanto, G é um grafo k -caminho.

(\Leftarrow) Já foi provado por MARKENZON *et al.* [31] que os grafos k -caminho são uma subfamília das SC k -árvores e já foi visto no Corolário 8.17 que os grafos k -caminho são grafos de intervalo. Logo, um grafo k -caminho é uma SC k -árvore que é grafo de intervalo. \Rightarrow

Como decorrência deste teorema temos os dois corolários vistos a seguir. No primeiro, provamos que os grafos 2-caminho são a interseção entre os grafos periplanares maximais e os grafos de intervalo e, no segundo, que os grafos 3-caminho são a interseção entre os grafos cordais planares maximais e os grafos de intervalo.

Cabe ressaltar que, como provado no Corolário 8.17, os grafos k -caminho destas interseções são grafos de intervalo 3-estrito e 4-estrito, respectivamente.

Corolário 8.24: Um grafo periplanar maximal é grafo de intervalo se e somente se é grafo 2-caminho.

Corolário 8.25: Um grafo cordal planar maximal é grafo de intervalo se e somente se é grafo 3-caminho.

8.5 - Interseções entre as Famílias

Nesta seção, utilizamos os diversos resultados obtidos nesta tese para delimitar com precisão as interseções entre as famílias aqui estudadas.

No Teorema 8.22, provamos que os k -intercats são a interseção exata entre as k -árvores e os grafos de intervalo. Na verdade, podem ser consideradas duas famílias menores nessa interseção: os k -caterpillars e os grafos de intervalo $(k+1)$ -estritos. Isso porque, pelo Teorema 8.16, as k -árvores que são grafos de intervalo são grafos de intervalo $(k+1)$ -estritos e, pela Definição 8.11, os k -intercats constituem uma subfamília dos k -caterpillars. Desse modo, os k -intercats são a interseção entre os k -caterpillars e os grafos de intervalo $(k+1)$ -estritos.

Cabe salientar que, para $k > 1$, a família dos grafos de intervalo $(k+1)$ -estritos e a dos k -caterpillars possuem elementos que não são k -intercats. Por exemplo, o grafo de intervalo 3-estrito da Figura 8.3(b) e o 2-caterpillar da Figura 8.7 não são 2-intercats.

No Teorema 8.19, provamos que os grafos k -caminho e as k -árvores estrela são subfamílias dos k -intercats. Como visto na Seção 8.3, para um k fixo, $k > 1$, existe uma única k -árvore com $k+2$ vértices e esta k -árvore é a interseção entre as k -árvores estrela e os grafos k -caminho.

No Teorema 8.23, provamos que os grafos k -caminho são a interseção entre as SC k -árvores e os grafos de intervalo. Como os k -intercats são as k -árvores que são grafos de intervalo, os grafos k -caminho são, na verdade, a interseção entre as SC k -árvores e os k -intercats. Cabe também ressaltar que existem k -árvores que não são grafos k -caminho na interseção entre as SC k -árvores e os k -caterpillars. Por exemplo, a

2-árvore da Figura 8.7 pertence à família das SC 2-árvores e à família dos 2-caterpillars e não é um grafo 2-caminho.

O diagrama da Figura 8.8 ilustra as várias interseções de famílias de grafos delimitadas neste capítulo. Todos os grafos utilizados como exemplo no diagrama são 2-árvores exceto o grafo de intervalo 3-estrito e o grafo de intervalo que estão mais à direita na figura. Apesar de existirem vários exemplares em cada interseção entre as subfamílias, para cada interseção é ilustrado apenas um exemplar minimal em relação ao número de vértices. Entretanto, vale a pena lembrar que para um determinado k a interseção entre os grafos k -caminho e as k -árvores estrela possui um único elemento que é a k -árvore com $k + 2$ vértices.

Resumindo as interseções do capítulo, para um dado $k, k > 1$, tem-se:

- Grafos k -caminho \subset k -intercats \subset k -caterpillars \subset k -árvores;
- Grafos k -caminho \subset k -intercats \subset grafos de intervalo $(k+1)$ -estritos \subset grafos de intervalo;
- SC k -árvore \cap k -intercats = SC k -árvore \cap grafos de intervalo = grafos k -caminho;
- SC k -árvores \cap k -caterpillar $\neq \emptyset$;
- k -árvores estrela \subset k -intercats.

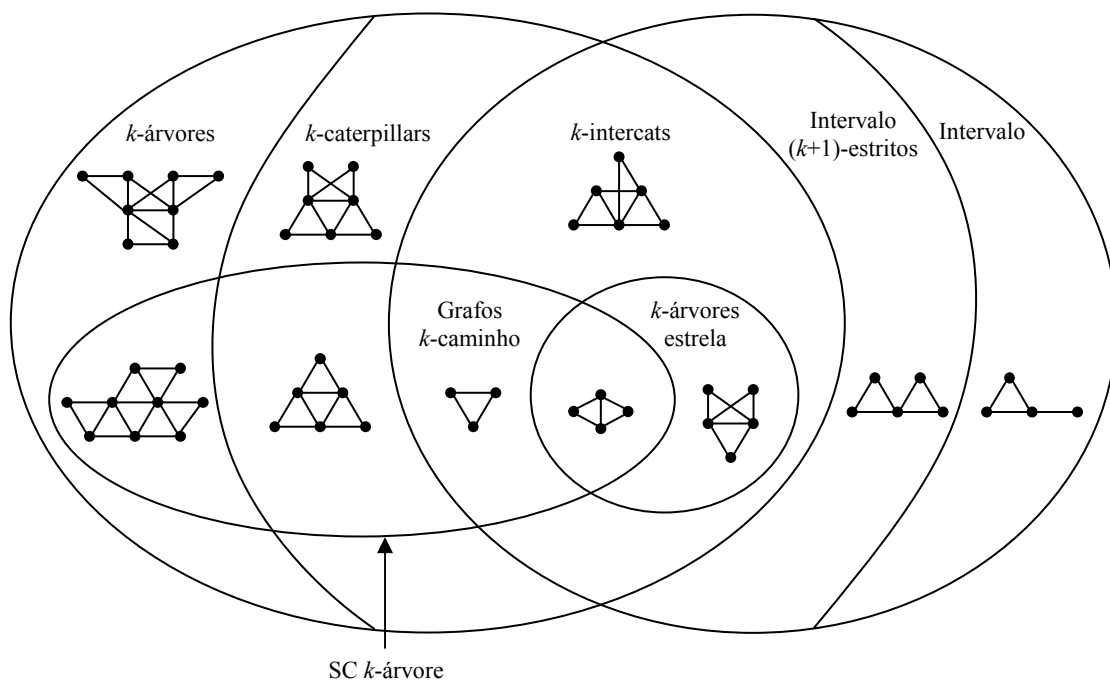


Figura 8.8: Diagramas com subclasses das k -árvores e dos grafos de intervalo.

Capítulo 9 - Conclusão

O código é uma forma alternativa de representação dos grafos de uma família. É desejável, mas não necessário, que uma codificação seja mais compacta do que a representação por arestas. Por outro lado, é fundamental que seja possível obter o código a partir do grafo e vice-versa, ou seja, que existam algoritmos de codificação e de decodificação. Para que a codificação possua aplicações algorítmicas, é preciso que esses algoritmos sejam eficientes. Por exemplo, o código para árvores proposto por Prüfer em 1918 possui diversas aplicações devido aos seus algoritmos eficientes de codificação e decodificação.

O estudo de códigos para árvores iniciou um pouco antes de Prüfer. MOON [35] citou um trabalho de representação e contagem de árvores publicado por BORCHADT [6] em 1860. Apesar de ser um assunto antigo, existe uma grande quantidade de publicações recentes que abordam tanto a definição de novas codificações para árvores quanto a utilização de codificações na solução de problemas cuja modelagem é feita por árvores.

9.1 - Contribuições

Em 1970, RÉNYI E RÉNYI [42] expandiram o código de Prüfer para k -árvores enraizadas. Nesta tese, contribuímos com a reorganização deste assunto definindo o código de Prüfer para k -árvores sem raiz. Além disso, projetamos algoritmos eficientes de codificação e decodificação para k -árvores quaisquer.

Normalmente, as famílias que possuem propriedades estruturais bem definidas possibilitam a obtenção de codificações. Nesta tese, estudamos propriedades estruturais e propomos códigos para duas subfamílias dos grafos cordais: os grafos k -caminho e as k -árvores. Apesar dos grafos k -caminho constituírem uma subfamília das k -árvores, as propriedades específicas utilizadas permitiram a definição de um código para os grafos k -caminho cuja obtenção é mais simples e possui mais aplicações. As codificações propostas para as duas famílias são bem compactas já que possuem complexidade $O(n)$ de armazenamento. Além disso, projetamos algoritmos eficientes para codificação e

decodificação das duas famílias. Estes algoritmos lineares viabilizam a utilização dos códigos desenvolvidos na tese.

Normalmente, a codificação é utilizada para contagem e geração de elementos da família. Durante nosso estudo, percebemos que a codificação possui outras duas importantes utilizações. A primeira é a obtenção de propriedades estruturais da família que, a princípio, não são perceptíveis pela representação por arestas e, às vezes, até pela definição da família. A segunda é a solução de problemas de modo mais eficiente ao utilizar o código ao invés da representação por arestas.

A codificação proposta para os grafos k -caminho possibilitou a contagem e a geração aleatória e uniforme. Também obtivemos diversas propriedades diretamente do código tais como: determinação de qualquer uma das cliques maximais e verificação da pertinência de uma aresta. Além disso, resolvemos dois importantes problemas para grafos k -caminho com complexidade $O(n)$: teste de isomorfismo e obtenção de caminho hamiltoniano.

Utilizando o código compacto de uma k -árvore, obtivemos um esquema de eliminação perfeita e colorimos os vértices com complexidade $O(n)$. A determinação da seqüência de multiplicidades de modo eficiente é outro resultado obtido com complexidade $O(n)$. De um modo geral, os algoritmos propostos na tese são mais eficientes do que os baseados no conjunto de arestas cuja complexidade mínima é $O(m)$.

As propriedades estruturais estudadas utilizando a seqüência reduzida permitiram verificar que os grafos k -caminho são grafos de intervalo com características bem peculiares. Este foi o primeiro passo para a identificação e definição de uma nova subfamília de grafos: a dos grafos de intervalo k -estritos. O estudo de propriedades estruturais de subfamílias das k -árvores permitiu delimitar com precisão o inter-relacionamento destas subfamílias com os grafos de intervalo. Ressaltamos a delimitação precisa entre os grafos de intervalo e três subfamílias dos grafos cordais: (i) k -árvores, (ii) grafos periplanares maximais e (iii) grafos cordais planares maximais.

9.2 - Trabalhos Futuros

O trabalho desenvolvido até agora nos fornece três diretivas distintas a serem abordadas para a continuação do mesmo.

A primeira, mais imediata, é a busca da solução de outros problemas utilizando os códigos já desenvolvidos. Espera-se encontrar novas aplicações que sejam mais eficientes do que as soluções baseadas na representação tradicional de grafos. Já está sendo estudada, por exemplo, a obtenção de ciclos hamiltonianos para grafos k -caminho.

Tendo em vista que a estrutura particular dos grafos k -caminho e das k -árvores permitiu a obtenção de códigos de armazenamento extremamente eficientes, a segunda diretiva consiste em buscar outras famílias com propriedades estruturais bem definidas que possibilitem a obtenção de códigos. A primeira família a ser estudada será a dos k -caterpillars, já que possuem aplicações na área de redes de computadores.

Finalmente, a terceira diretiva consiste em explorar melhor as relações mostradas na Figura 8.8, permitindo assim, novas caracterizações e algoritmos de reconhecimento eficientes para as famílias já estudadas.

Capítulo 10 - Referências

- [1] ARAÚJO, L. H., *Algoritmos Dinâmicos para Manutenção de Grafos Cordais e Periplanares*. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2004.
- [2] ARAÚJO, L. H., MARKENZON, L., VERNET, O., “Generation of Chordal Graphs Through Successive Edge Insertions”. In: *XII Congreso Latino-Iberoamericano de Investigación de Operaciones - CLAIO 2004*, T84, Havana, Cuba, 2004.
- [3] BABEL, L., PONOMARENKO, I. N., TINHOFER, G., “The Isomorphism Problem for Directed Path Graphs and Rooted Directed Path Graphs”, *Journal of Algorithms*, v. 21, pp. 542-564, 1996.
- [4] BEINEKE, L. W., PIPPERT, R. E., “Properties and Characterizations of k -trees”. *Mathematika*, v. 18, pp. 141-151, 1971.
- [5] BLAIR J. R. S., PEYTON, B., “An Introduction to Chordal Graphs and Clique Trees”. In: *Graph Theory and Sparse Matrix Computation, IMA 56*, pp. 1-29, 1993.
- [6] BORCHARDT, C. W., “Ueber eine der Interpolation entsprechende Darstellung der Eliminations Resultante”. *Journal für die reine und angewandte Mathematik*, v. 57, pp. 111-121, 1860.
- [7] CAMERON, R. D., COLBOURN C. J., READ, R. C., WORMALD, N. C., “Cataloguing the graphs on 10 vertices”. *Journal of Graph Theory*, v. 9, pp. 551-562, 1985.
- [8] CAMINITI, S., FINOCCHI, I., PETRESCHI, R., “A Unified Approach to Coding Labeled Trees”. In: *Proceedings of 6th Latin American Symposium on Theoretical Informatics*, v. 4, pp. 339-348, Buenos Aires, Argentina, 2004.
- [9] CAMINITI, S., FUSCO, E., PETRESCHI, R., “A Bijective Code for k -trees with Linear Time Encoding and Decoding”. In: *Proceedings of International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies (ESCAPE 07)*, pp. 408-420, Hangzhou, China, 2007.

- [10] CHEN, H. C., WANG, Y. L., “An Efficient Algorithm for Generating Prüfer Codes from Labelled Trees”. *Theory of Computing Systems*, v. 33, pp. 97-105, 2000.
- [11] DEO, N., MICIKEVICIUS, P., “A New Encoding for Labeled Trees Employing a Stack and a Queue”. *Bulletin of the Institute of Combinatorics and its Applications*, v. 34, pp. 77-85, 2002.
- [12] DEO, N., MICIKEVICIUS, P., *Parallel Algorithms for Computing Prüfer-Like Codes of Labeled Trees*, Computer Science Technical Report CS-TR-01-06, 2001.
- [13] DEO, N., MICIKEVICIUS, P., “Prüfer Like-Codes for Labeled Trees”. *Congressus Numerantium*, v. 151, pp. 65-73, 2001.
- [14] EDELSON, W., GARGANO, M. L., “Feasible Encodings for GA Solutions of Constrained Minimal Spanning Tree Problems”. *Genetic and Evolutionary Computation Conference (GECCO 2000)*, poster 754, 2000.
- [15] GALINIER, P., HABIB, M., PAUL, C., “Chordal Graphs and their Clique Graphs”. In: *Proceedings of Workshop on Graph-Theoretic Concepts in Computer Science (WG'95)*, LNCS 1017, pp. 358-371, 1995.
- [16] GAREY, R., JOHNSON, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco, 1979.
- [17] GAVRIL, F., “Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph”, *SIAM Journal on Computing*, v. 1, pp. 180-187, 1972.
- [18] GAVRIL, F., “A Recognition Algorithm for the Intersection Graph of Paths in Trees”, *Discrete Mathematics*, v. 23, pp. 211-227, 1978.
- [19] GOLUBIC, M. C., *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.
- [20] GREENLAW, R., HALLDORSSON, M. M., PETRESCHI, R., “Prüfer Codes and their Corresponding Trees Optimally”. In: *Proceedings of Secondes Journées de l'Informatique Messine (JIM'2000)*, Metz, France, 2000.

- [21] GUPTA, A., NISHIMURA, N., PROSKUROWSKI, A., RAGDE, P., “Embeddings of k -connected Graphs of Pathwidth k ”. *Discrete Applied Mathematics* v. 145, pp 242-265, 2005.
- [22] HARARY, F., SCHWENK, A. J., “Trees with Hamiltonian Squares”, *Mathematika*, v. 18, pp. 138-140, 1971.
- [23] HOPCROFT J., WONG J., “A Linear Time Algorithm for Isomorphism of Planar Graphs”. In: *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*, pp. 310-324, 1974.
- [24] JOHNSON, D. S., “A Theoretician's Guide to the Experimental Analysis of Algorithms”. In: *Data Structures, Near Neighbor Searches, and Methodology: 5th and 6th DIMACS Implementation Challenges*, pp. 215-250, 2002.
- [25] JUSTEL, C. M., MARKENZON, L., “Lexicographic Breadth First Search and k -Trees”. In: *Proceedings of Secondes Journées de l'Informatique Messine (JIM'2000)*, pp. 23-28, Metz, France, 2000.
- [26] KUMAR, V., DEO, N., KUMAR, N., “Parallel Generation of Random Trees and Connected graphs”. *Congressus Numerantium*, v. 130, pp. 7-18, 1998.
- [27] KUMAR, P. S., MADHAVAN, C. E. V., “Clique Tree Generalization and new Subclasses of Chordal Graphs”. *Discrete Applied Mathematics*, v. 117, pp. 109-131, 2002.
- [28] KUMAR, P. S., MADHAVAN, C. E. V., “Minimal Vertex Separators of Chordal Graphs”. *Discrete Applied Mathematics*, v. 89, pp. 155-168, 1998.
- [29] LEKKERKERKER, C. G., BOLAND, J. C., “Representation of a Finite Graph by a Set of Intervals on the Real Line”, *Fund. Math.*, 51, pp. 45-64, 1962.
- [30] LUEKER G. S., BOOTH, K. S., “A Linear Time Algorithm for Deciding Interval Graph Isomorphism”, *Journal of the ACM*, v. 26, 1979, pp. 183-195.
- [31] MARKENZON, L., JUSTEL, C. M., PACIORNIK, N., “Subclasses of k -Trees: Characterization and Recognition”. *Discrete Applied Mathematics*, v. 154, pp. 818-825, 2006.

- [32] MARKENZON, L., VERNET, O., “Weighted Perfect Elimination Orderings and the Generation of Chordal Graphs”. In: *Proceedings of the Génération Aléatoire de Structures Combinatoires (GASCOM 2006)*, 2006, Dijon.
- [33] MARKENZON, L., VERNET, O., PEREIRA, P. R. C., “ (L,U) -Bounded Priority Queues and the Codification of Rényi k -trees”, Relatório Técnico NCE 05/05, 2005.
- [34] MARKENZON, L., VERNET, O., PEREIRA, P. R. C., “Determinação Eficiente de Vértices Simpliciais em Grafos Cordais”, In: *Anais Eletrônicos do XXXVIII Simpósio Brasileiro de Pesquisa Operacional*, pp. 2254-2260, Goiânia, Brasil, 2006.
- [35] MOON, J. W., *Counting Labelled Trees*, Canadian Mathematical Monographs, Montreal, 1970.
- [36] PEREIRA, F. M. Q., PALSBERG, J. “Register Allocation via Coloring of Chordal Graphs”, In: *Proceedings of Asian Symposium on Programming Languages and Systems (APLAS'05)*, pp. 315-329, Tsukuba, Japão, 2005.
- [37] PEREIRA, P. R. C., MARKENZON, L., VERNET, O., “The Reduced Prüfer Code for Rooted Labelled Trees”, *Electronic Notes in Discrete Mathematics*, v. 22, pp. 135-139, 2005.
- [38] PEREIRA, P. R. C., MARKENZON, L., VERNET, O., “A Clique-difference Encoding Scheme for Labelled k -Trees”. *Discrete Applied Mathematics*, aguardando publicação.
- [39] PICCIOTTO, S., *How to Encode a Tree*. Ph.D. dissertation, University of California, San Diego, USA, 1999.
- [40] PROSKUROWSKI, A., “Separating Subgraphs in k -Trees: Cables and Caterpillars”. *Discrete Mathematics*, v. 49, pp. 275-285, 1984.
- [41] PRÜFER, H., “Neuer Beweis eines Satzes über Permutationen”, *Archiv für Mathematik und Physik*, v. 27, pp. 142-144, 1918.

- [42] RÉNYI, C., RÉNYI, A., “The Prüfer Code for k -Trees”. In: *P. Erdős et al., editor, Combinatorial Theory and Its Applications*, pp. 945-971. North-Holland, Amsterdam, 1970.
- [43] ROSE, D. J., TARJAN, R. E., LUEKER, G. S., “Algorithmic Aspects of Vertex Elimination on Graphs”. *SIAM Journal on Computing*, v. 5, pp. 266-283, 1976.
- [44] SCHAFFER, A. A., “A Faster Algorithm to Recognize Undirected Path Graphs”, *Discrete Applied Mathematics* v. 43, pp. 261-295, 1993.
- [45] SZWARCFITER, J. L., *Grafos e Algoritmos Computacionais*. Editora Campus, 1986.
- [46] TINHOFER, M. G., “Generating Graphs Uniformly at Random”. *Computing Supplementum*, v. 7, pp. 235-255, 1990.
- [47] TROTTER, W. T., HARARY, F., “On Double and Multiple Interval Graphs”. *Journal of Graph Theory*, v. 3, pp. 205-211, 1979.
- [48] ZHOU, G., GEN, M., “Genetic Algorithm Approach on Multi-Criteria Minimum Spanning Tree Problem”. *European Journal of Operational Research*, v. 114, n. 1, pp. 141-152, 1999.