UDESC – Universidade do Estado de Santa Catarina

DCC – Departamento de Ciência da Computação

Curso: BCC

Disciplina: Teoria dos Grafos – TEG0001

Notas de Aula – Parte IV

Referências bibliográficas:

1. Cormen, T. Introduction to Algorithms. Third edition, 2009. The MIT Press
2. Bondy, J.A. Graph Theory with applications. Fifth edition.
3. Rosen K. Matemática discreta e suas aplicações, sexta edição, 2009
4. Gersting, Judith L. Fundamentos Matemáticos para a Ciência da Computação. Rio de Janeiro. 5a Ed. Editora. LTC, 2004
5. WEST, Douglas, B. Introduction to Graph Theory, second edition, Pearson, 2001.
6. SEDGEWICK, R. Algorithms in C – part 5 – Graph Algorithms, third edition, 2002, Addison-Wesley.
7. ROSEN, K. Discrete Mathematics and its applications, seventh edition, McGraw Hill, 2011.

8. GOLDBARG, M., GOLDBARG E., Grafos: Conceitos, algoritmos e aplicações. Editora Elsevier, 2012.

9. FEOFILOFF, P., KOHAYAKAWA, Y., WAKABAYASHI, Y., uma introdução sucinta à teoria dos grafos. 2011. (www.ime.usp.br/~pf/teoriadosgrafos)

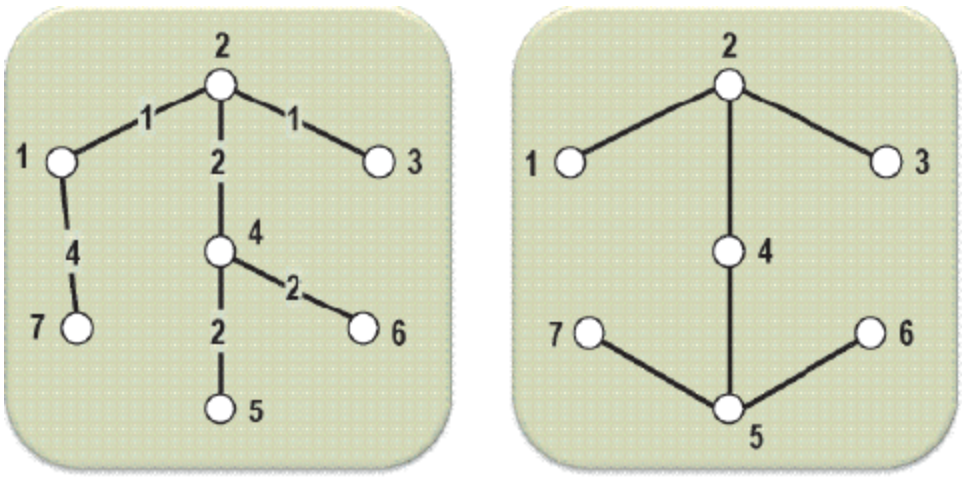
10. DIESTEL, R. Graph Theory, second edition, springer, 2000.

11. Kleinberg, J, Tardos E., Algorithm Design, Pearson, 2006

12. Bastante material disponível na internet.

4. Árvores

Uma árvore consiste em um grafo **conexo e acíclico** com um nó especial, denominado de raiz da árvore [Gersting, 2004]. Como uma árvore não pode ter um ciclo simples, não pode conter laços, portanto **qualquer árvore deve ser um grafo simples**.



(1) Árvore ponderada (2) Árvore não ponderada

Fig. 4.1 árvores ponderadas e não ponderadas

Algumas propriedades das árvores:

. Todo grafo G conexo possui pelo menos uma sub árvore que contêm todos os seus vértices;

. Toda árvore é um grafo planar

. Toda árvore finita com n > 1 possui pelo menos dois vértices terminais.

Um grafo G não orientado com n vértices é uma árvore, se e somente se, existir um único caminho entre cada par de nós de G e G possui (n-1) arestas.

Teorema de Caley: O número de árvores distintas em um grafo completo com n vértices é dada por: n^{n-2}.

Grafo Caminho: um grafo caminho é uma árvore com n vértices, se e somente se, seu grau máximo for igual a 2, e possui apenas dois vérticers com grau 1.

Como uma árvore é um **grafo conexo e acíclico** existe um caminho da raiz para qualquer outro nó na árvore e esse **caminho é único**. A remoção de qualquer aresta do grafo o torna desconexo.

Uma árvore com raiz é uma árvore na qual um vértice tenha sido escolhido como raiz e toda aresta esteja orientada no sentido que se afasta da raiz.

Escolhas diferentes de raízes produzem árvores com raízes diferentes. Vide figura 4.1

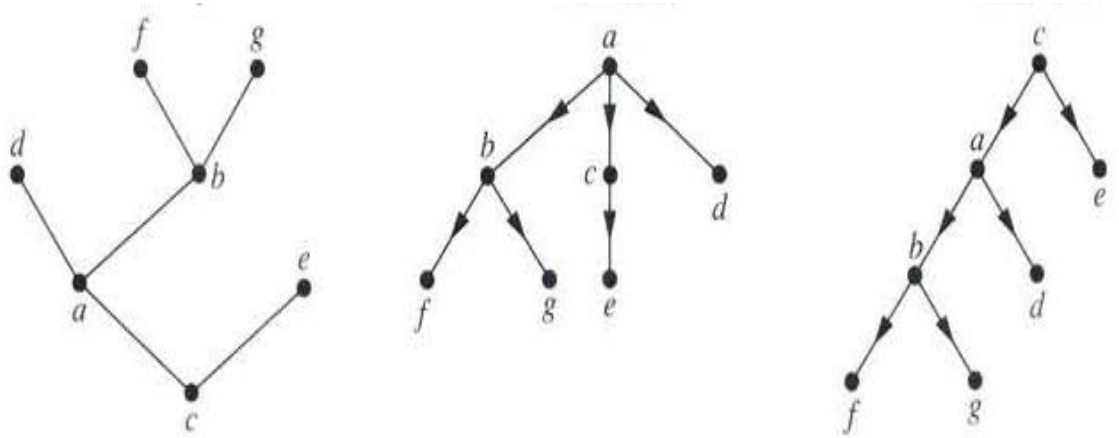


Fig. 4.1 uma árvore e árvores enraizadas

Árvores disjuntas: dada a figura 4.2 se T_1 , T_2 ,..., T_n são **árvores disjuntas** com raízes r_1 , r_2 ,..., r_n , o grafo formado colocando-se um nó r ligado, por um único arco, a cada um dos nós r_1 , r_2 ,..., r_i , é uma árvore com raiz r. Os nós r_1 , r_2 ,..., r_i correspondem aos filho r, e portanto, r é considerado o pai destes nós.

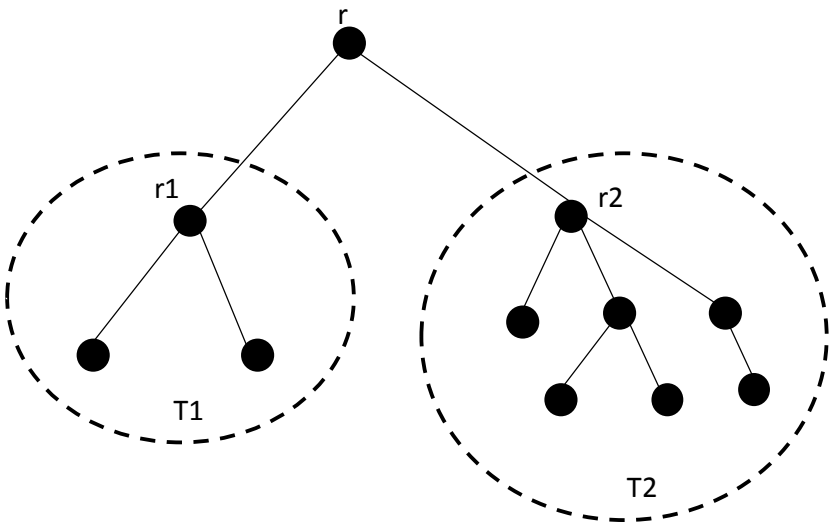


Fig. 4.2 – árvores disjuntas

Folha de uma árvore: Uma folha de uma árvore é um vértice v tal que d(v) = 1. Toda árvore possui pelo menos duas folhas, n > 1

Exemplo: Dado os grafos da figura 4.3 quais são árvores ?

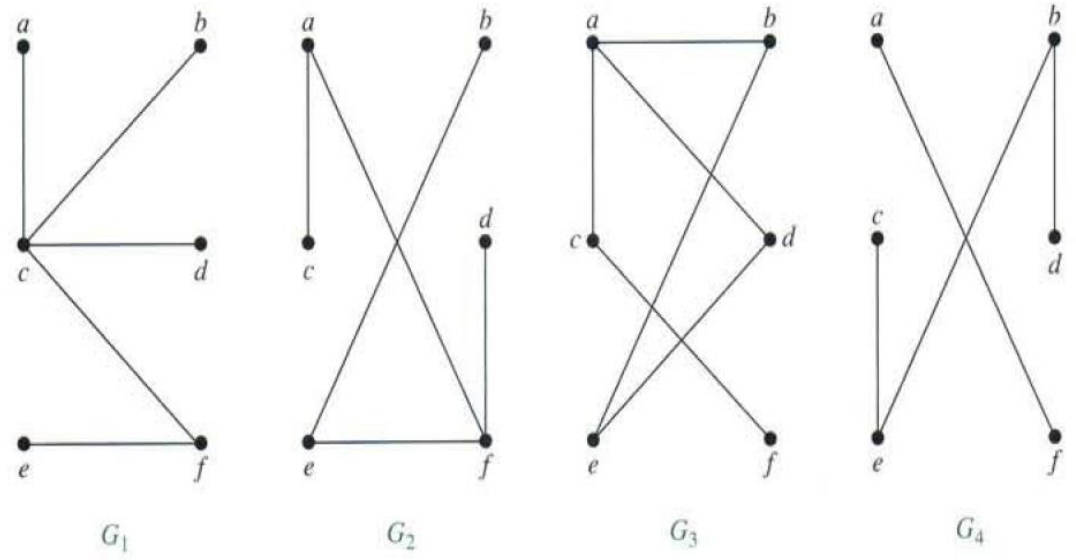


Fig. 4.3 – exemplos de grafos

Somente G1 e G2 são arvores. G3 possui um ciclo e G4 não é conexo.

Abaixo apresentamos a figura 4.4 com várias árvores, todas com 6 (seis) nós.

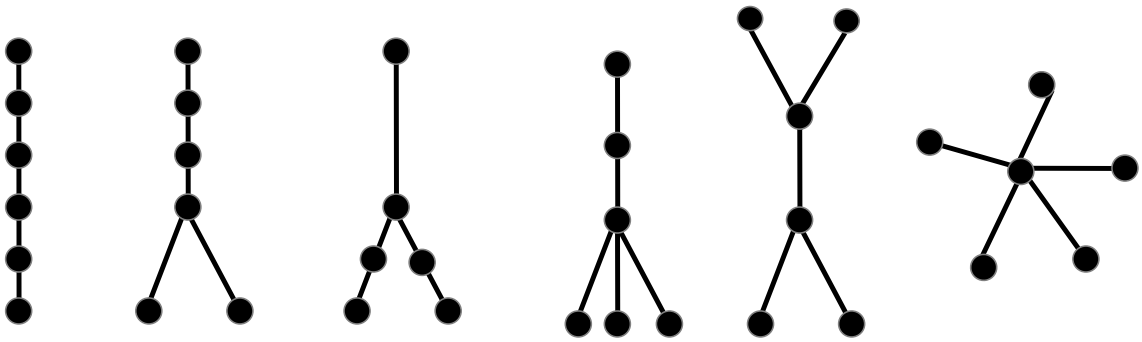


Fig. 4.4 – exemplos de árvores

Árvore enraizada: Uma árvore enraizada é dita uma árvore de grau m se todo vértice interno não tiver mais que m filhos. Uma árvore é dita de grau m cheia se cada vértice interno tiver exatamente m filhos.

Uma árvore de grau m, com m=2, é chamada de árvore binária.

Exemplo: Dada a figura 4.5 quais são os seus respectivos graus.

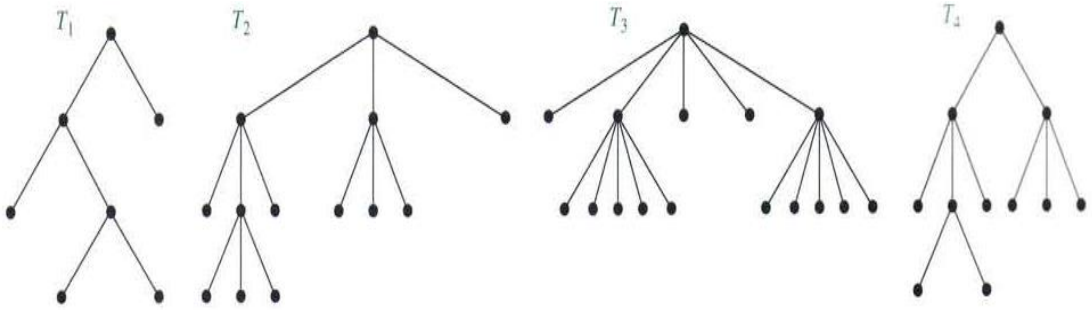


Fig. 4.5 – Quatro árvores enraizadas.

Vértices internos: Os vértices que possuem filhos são denominados de vértices internos.

Exemplo Computação paralela. Uma rede de sete processadores conectada através de uma árvore é representada na figura 4.6. Por exemplo, esta rede de processadores pode ser utilizada para somar oito números (x1 + x2 + ... + xn) usando três passos.

No primeiro passo soma-se x1+x2 usando P4 e assim sucessivamente.

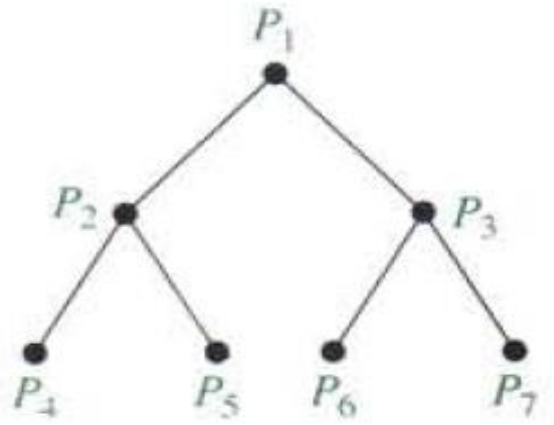


Fig. 4.6 – rede de sete computadores interligados em árvore

Prove por indução que uma árvore com n vértices tem n-1 arestas.

Prove que uma árvore de grau m cheia com i vértices internos contêm n = mi + 1

Prove que um grafo conexo é uma árvore se e somente se toda aresta é uma ponte

Seja T uma árvore com pelo menos 3 nós. Seja T` a árvore obtida de T pela exclusão de todas as suas folhas. Entao T e T` possuem o **mesmo centro**.

Uma árvore de grau **m cheia** com:

a. **n vértices tem:**

i = (n-1)/m vértices internos

L=[(m-1)n +1] /m folhas

b. i vértices internos tem:

n = m*i + 1 vértices

L=(m-1)*i + 1 folhas

c. L folhas tem:

n=[(m*L-1)/(m-1)] vértices

i = [(L-1)/(m-1)] vértices internos

Exemplo: Suponha que alguém inicie uma corrente de cartas. É solicitado a cada pessoa que recebe a carta que a envie para quatro outras pessoas.

Algumas pessoas fazem isso, mas outras não mandam nenhuma carta. Quantas pessoas viram a carta, incluindo a primeira pessoa, se ninguém recebeu mais de uma carta e se a corrente termina depois de ter havido 100 pessoas que leram, mas não mandaram a carta? Quantas pessoas mandaram a carta?

Resposta:

A corrente de carta pode ser representada usando uma árvore, por exemplo, de grau 4. Os vértices internos correspondem as pessoas que enviaram a carta, e as folhas correspondem a pessoas que não a mandaram.

Como 100 pessoas não mandaram a carta, o número de folhas nesta árvore enraizada é L=100. Logo, considerando a equação do item c temos que:

N = (4*100 – 1)/(4-1) = 133.

Além disso, o número de vértices internos é 133 – 100 = 33.

Então 33 pessoas enviaram a carta.

Nível de um vértice. O nível de um vértice v em uma árvore enraizada é o comprimento do único caminho da raiz a este vértice. O nível da raiz é definido como zero. Um vértice Vi é dito estar no nível i da árvore se Vi está a uma distância i da raiz

Altura de uma árvore enraizada: corresponde ao máximo dos níveis dos vértices, ou seja, corresponde ao comprimento do caminho mais longo da raiz a qualquer vértice.

Exemplo: Encontre o nível de cada vértice da árvore da figura 4.7:

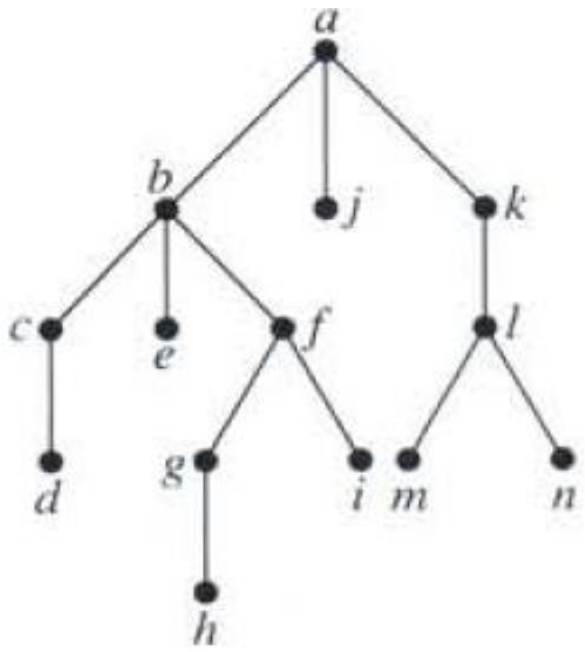


Fig. 4.7 – níveis de cada vértice

Raiz: 0
b,j,k = 1
c,e,f e l = 2
d,g,i,m e n = 3
h = 4
altura da árvore = 4.

Árvore enraizada balanceada: uma árvore de grau m enraizada de altura h é balanceada se todas as folhas estiverem nos níveis h ou h-1.

Exemplo: Quais das árvores enraizadas da figura 4.8 são balanceadas ?

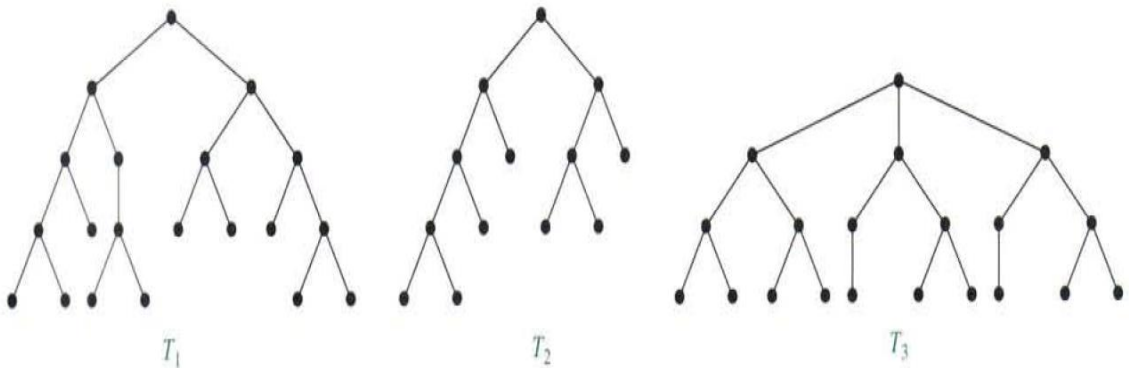
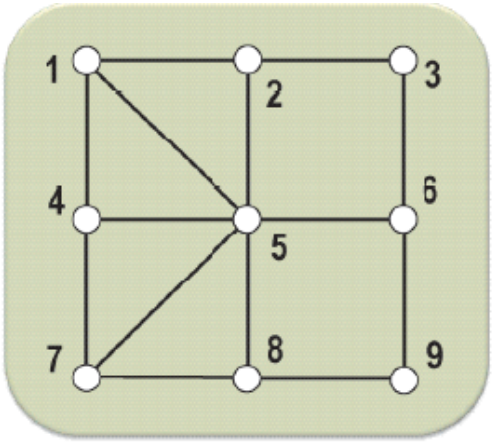


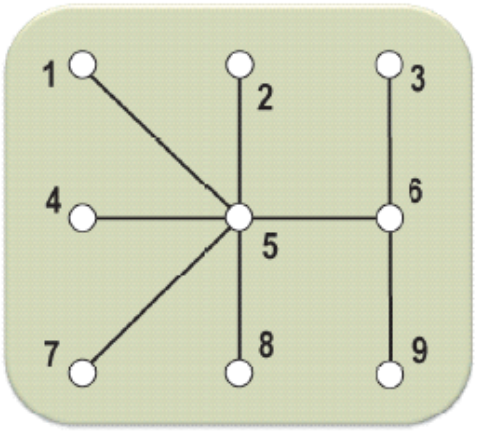
Fig. 4.8 – exemplos de árvores enraizadas

Distâncias entre árvores: A distância entre duas árvores geradoras T1 = (N, M1) e T2 = (N,M2) de um grafo G = (N,M), d(T1,T2), é dada pelo módulo da diferença entre o conjunto de arestas de T1 e T2, d(T1, T2) = | (M1– M2) |.

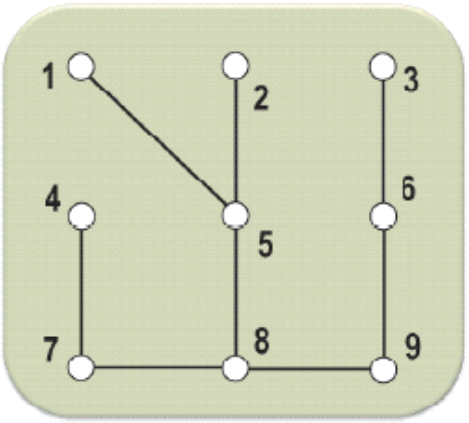
Vide exemplo da figura 4.9.



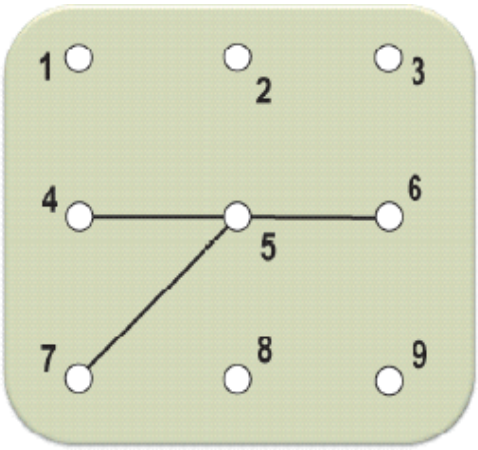
(1) Grafo exemplo



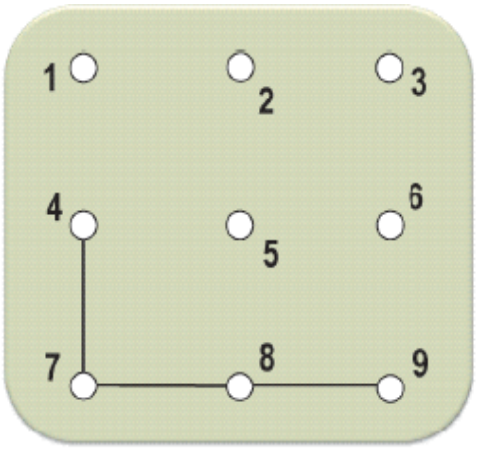
(2) Árvore T_1



(3) Árvore T_2



(4) $d(T_1,T_2) = 3$



(5) $d(T_2,T_1) = 3$

Fig. 4.9 – distâncias entre árvores.

Centro de uma árvore: um vértice é chamado de centro, se nenhum outro vértice na árvore tiver excentricidade menor do

que este vértice. A excentricidade de um vértice em uma árvore sem raiz corresponde ao comprimento do caminho simples mais longo que começa neste vértice.

Dado o grafo da figura 4.9, os vértices centrais da árvore estão sinalizados em preto.

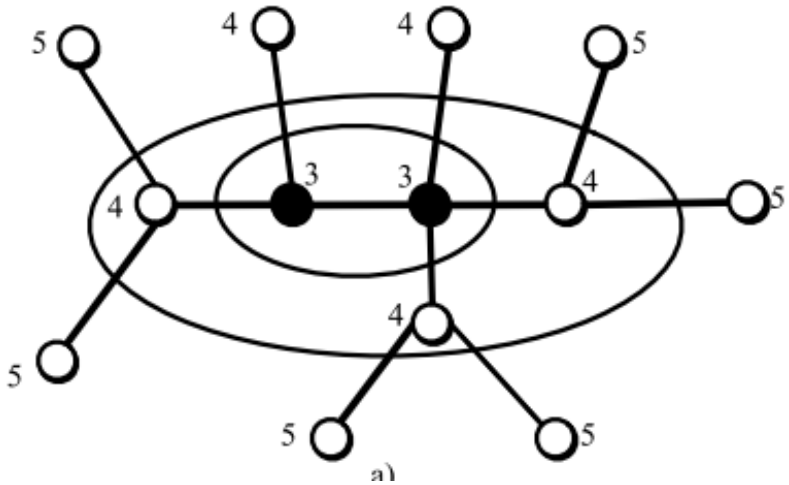


Fig. 4.9 – centralidade de uma árvore

Resolução de exercícios

Prove que existem no máximo m^h folhas em uma árvore de grau m de altura h *(vide resolução do exercício 19).*

Se uma árvore de grau m, de altura h e possui L folhas, então:

h >=$(log_m L)$, considerando que h é um inteiro.

A **profundidade de um** nó em uma árvore corresponde ao comprimento do caminho raiz ao nó. A raiz tem profundidade 0.

A **profundidade ou altura de uma árvore** corresponde a maior profundidade dos nós na árvore, ou seja, é o comprimento do caminho mais comprido da raiz até um dos nós.

Uma **folha de uma árvore** corresponde a um nó sem filhos, de grau 1, e todos os nós que não são folhas são considerados nós internos à árvore.

Uma **floresta** é considerada um grafo acíclico, não necessariamente conexo. Logo, uma floresta corresponde a uma coleção de árvores disjuntas, ou seja, sem vértices em comum A figura 4.11 é considerada uma floresta.

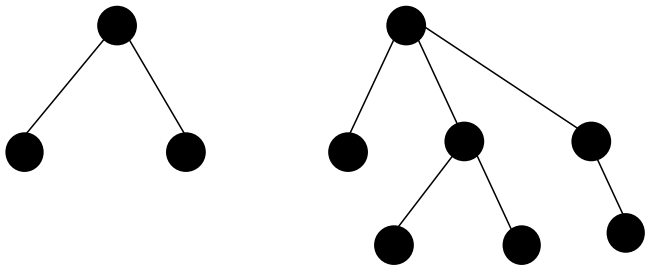
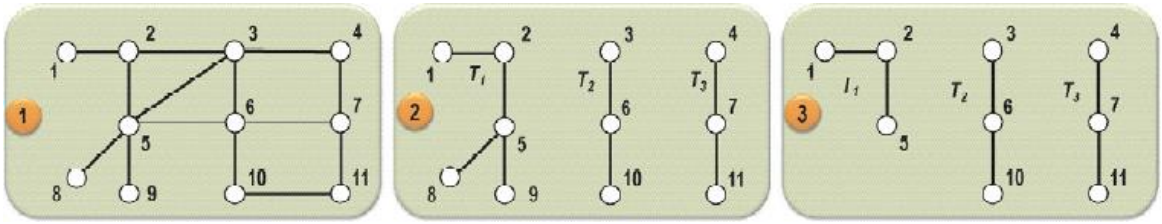


Fig. 4.11 – exemplo de uma floresta

Uma **floresta geradora** corresponde a um conjunto de árvores que contêm todos os vértices de G. Ou seja, cada componente conexo é uma árvore e cada vértice do grafo original está em alguma árvore.



A **orientação de uma árvore** é definida por: pai, filhos, descendentes e ancestrais.

Árvore geradora:

Dado um grafo G, uma **árvore geradora** (do inglês ***spanning tree***) corresponde a um sub grafo que contém todos os vértices de G e é uma árvore. Ou seja, é uma árvore que alcança todos os vértices. Um grafo que é uma árvore possui exatamente uma árvore geradora. Vide figura 4.10.

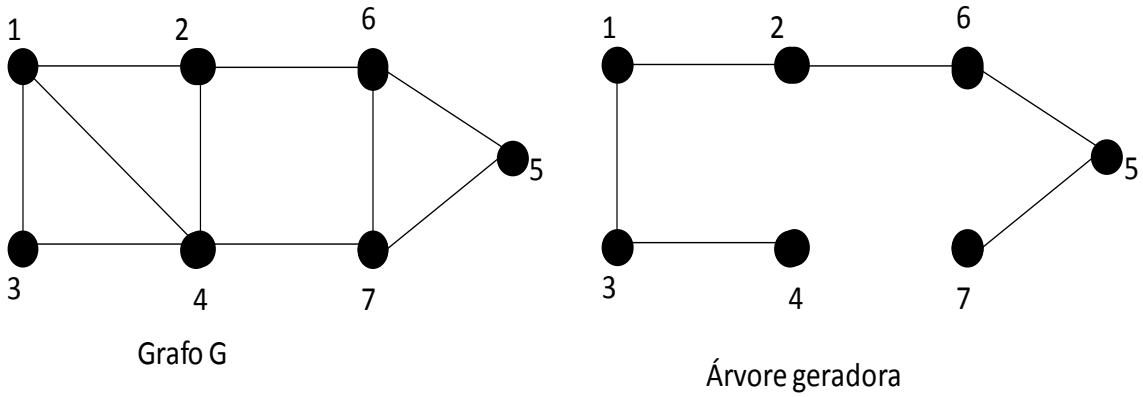
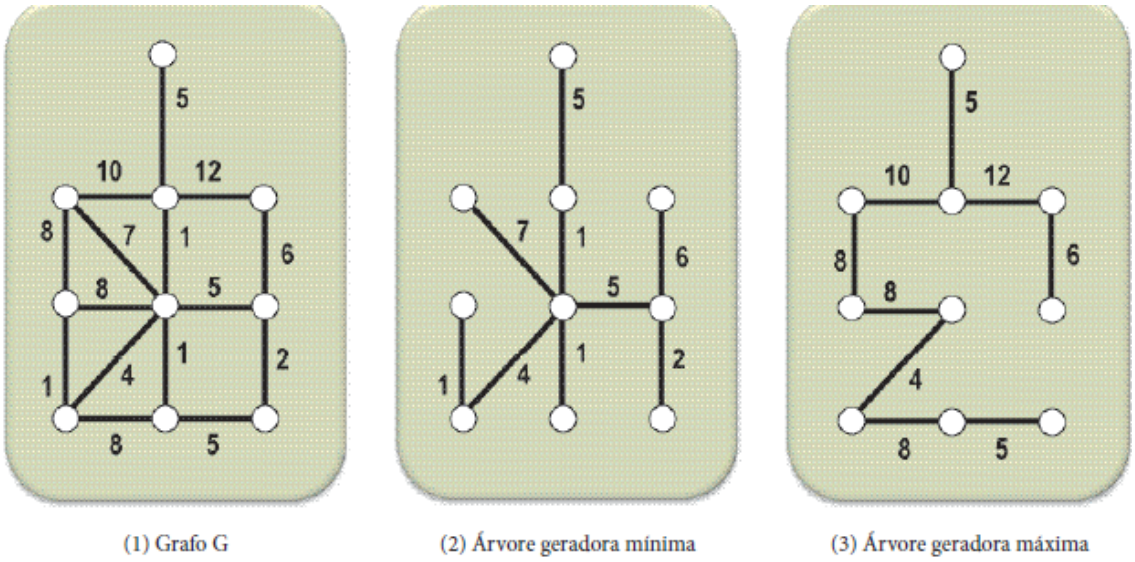


Fig. 4.10 – árvore geradora

Árvore geradora mínima: uma árvore geradora mínima corresponde a uma árvore de menor custo, dentre todas as possíveis em G. O custo de uma árvore geradora T de um grafo ponderado G é dado pelo somatório dos custos das arestas de T.

Árvore geradora máxima: uma árvore geradora máxima corresponde a árvore geradora de maior custo em G.



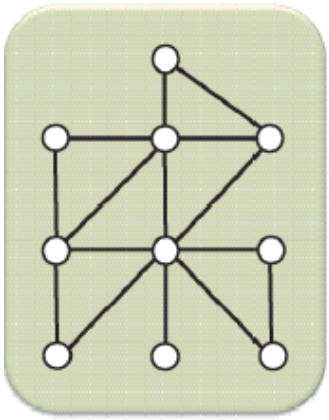
(1) Grafo G (2) Árvore geradora mínima (3) Árvore geradora máxima

Fig. 4.11 – árvore geradora mínima e máxima

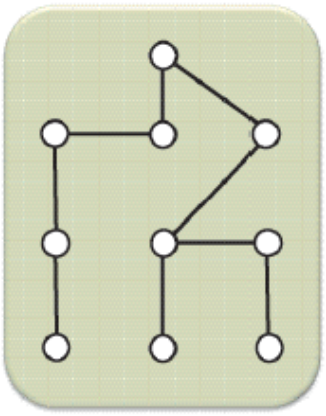
Árvore geradora de grau mínimo: corresponde a uma árvore desenvolvida em um grafo não ponderado e que possui o menor grau máximo possível.

Árvore geradora mínima com mínimo grau: corresponde a árvore geradora mínima de um grafo ponderado a qual possui o menor grau máximo possível

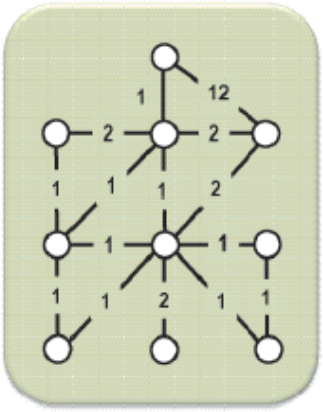
Árvore geradora de grau restrito: considerando um grafo G e um inteiro positivo K é uma árvore geradora de G tal que o máximo grau de seus vértices seja menor ou igual a k.



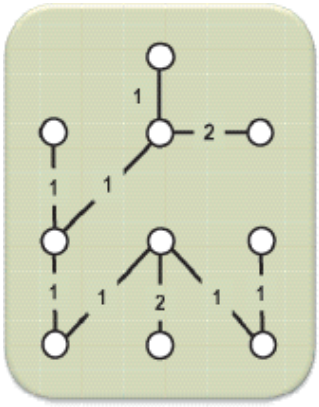
(1) Grafo G



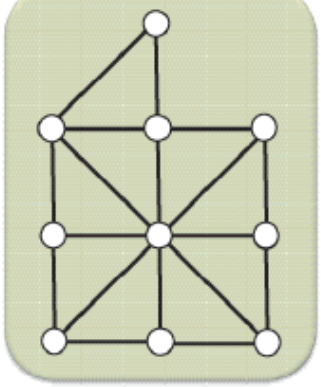
(2) AGGM do grafo (1)



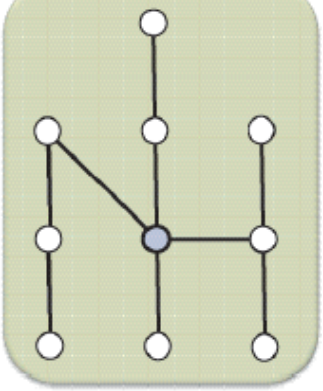
(3) Grafo G ponderado



(4) AGMMG do grafo (3)



(5) Grafo G



(6) AGM-GR de (5) com k = 4

Resolução exercícios

Prove o teorema que um grafo simples é conexo se e somente se ele tem uma árvore geradora. (vide exercício 22 da terceira lista)

Prove que seja T uma árvore geradora de um grafo conexo G e seja a uma aresta de G, a $\notin$ T. Então T + a contém um único ciclo (vide exercício 23 da terceira lista***).***

Árvore binária:

Uma **árvore binária** é um tipo especial de arvore enraizada e corresponde a árvore em que cada nó possui no máximo dois filhos (esquerdo e direito). As sub árvores descendentes dos filhos são denominadas árvores da esquerda e da direita.

Cada vértice da árvore binária possui exatamente 2 filhos, ou seja, existe apenas um vértice com grau 2 (raíz) e os outros vértices possuem grau 1 ou 3

A figura 4.12 apresenta uma árvore binária de altura 4.

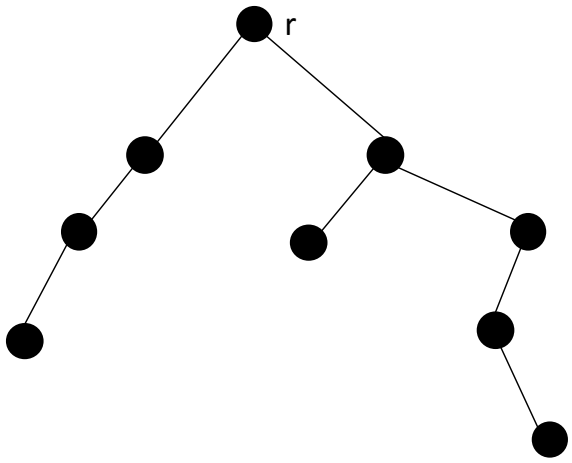


Fig. 4.12 – árvore binária

Uma **árvore binária cheia** é aquele em que todos os nós internos com dois filhos e onde todas as folhas estão à mesma

profundidade. A figura 4.13 apresenta uma árvore binária cheia de altura 3.

O número de nós em cada nível de uma árvore binária cheia segue uma progressão geométrica, ou seja: 1 no nó nível 0; 2^1 no nível 1; 2^2 no nível 2; e assim por diante. Portanto, em uma árvore binária cheia de altura d, o número total de nós é:

$$\mathbf{1 + 2^1 + 2^2 + 2^3 + \ldots + 2^d = 2^{d+1} - 1}$$

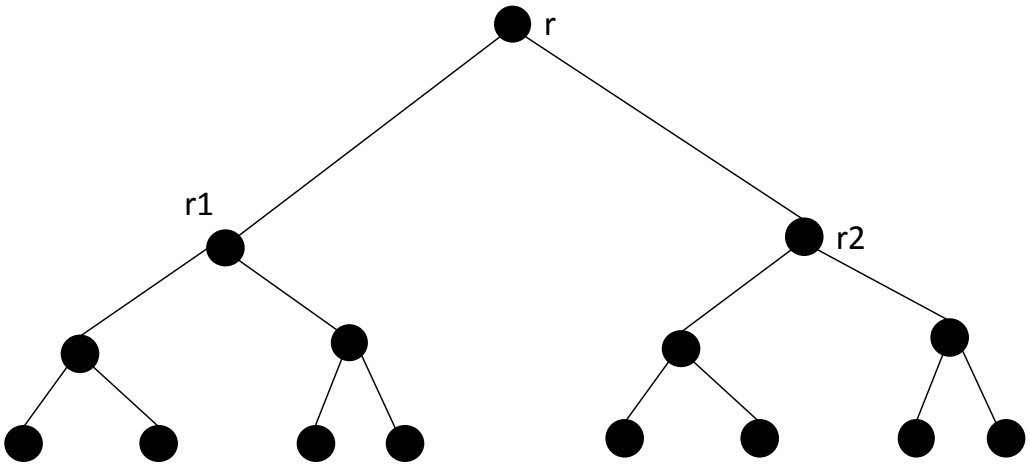
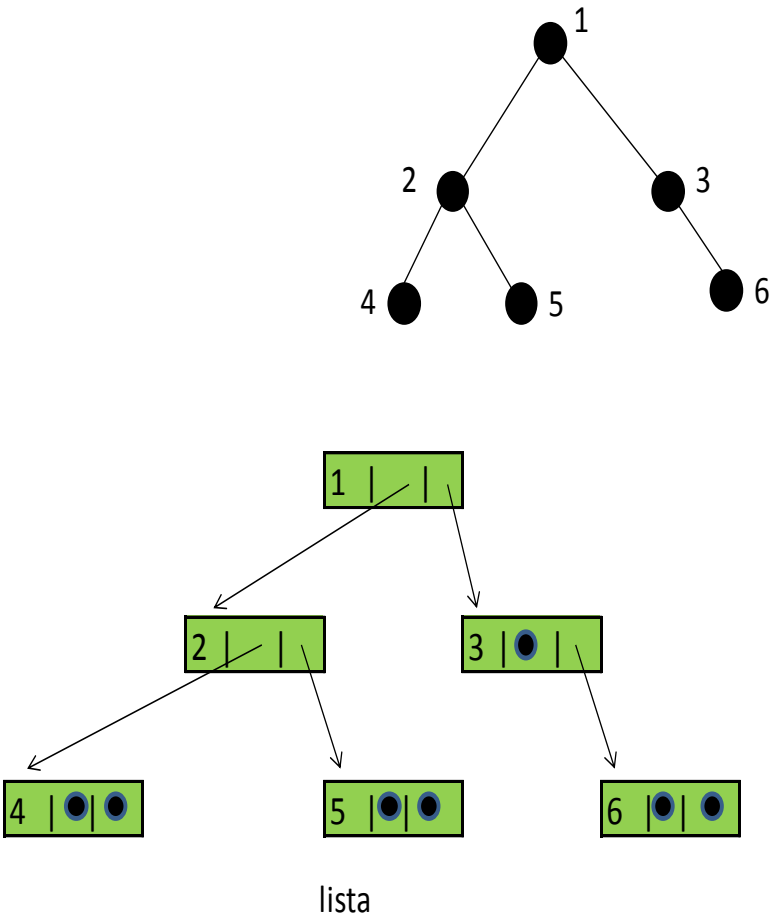


Fig. 4.13 – árvore binária cheia.

Representação de árvores binárias: A representação de árvores binárias pode ser realizada através de **matrizes ou listas adjacentes**.

A figura 4.14 apresenta as duas estruturas de dados, em que os filhos do lado esquerdo e direito são devidamente identificados nas estruturas.



	Esquerdo	Direito
1	**2**	**3**
2	**4**	**5**
3	**0**	**6**
4	**0**	**0**
5	**0**	**0**
6	**0**	**0**
	0	**1**
	0	**0**
	0	**1**
	0	**0**

matriz

Fig. 4.14 – representação de árvores.

Exemplos de utilização de árvores para armazenar informações.

A busca ou exploração (“passeio”) sistemática das informações armazenadas em uma árvore tem como objetivo encontrar a informação requisitada.

. Organograma de uma empresa;

. Estrutura de diretórios de arquivos;

. Organização de uma biblioteca;

. Árvore genealógica;

. Estrutura de um compilador;

. Expressões algébricas.

A figura 4.15 apresenta a utilização de uma árvore para representar uma expressão algébrica.

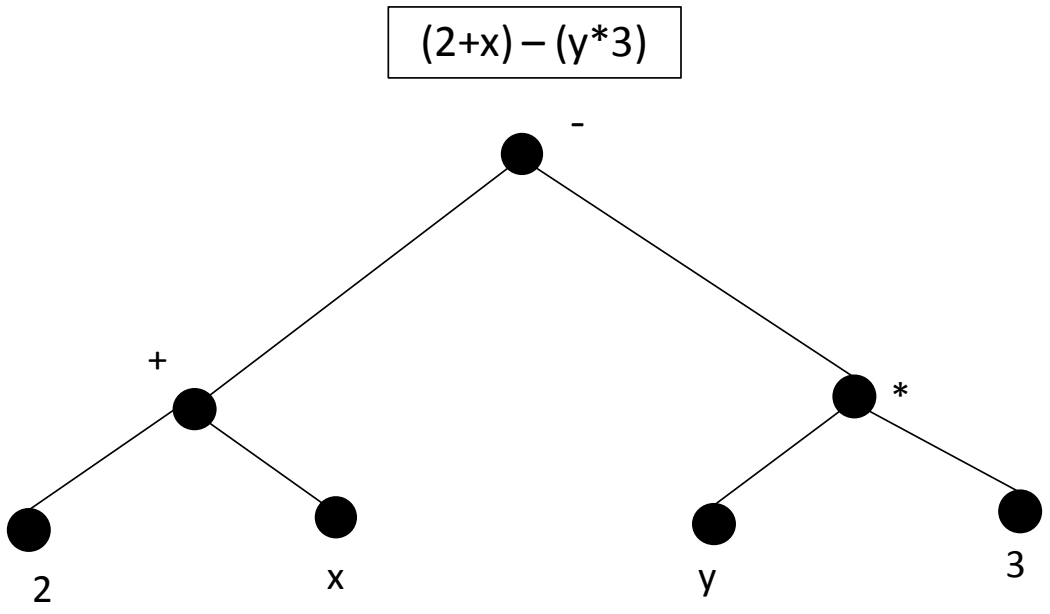


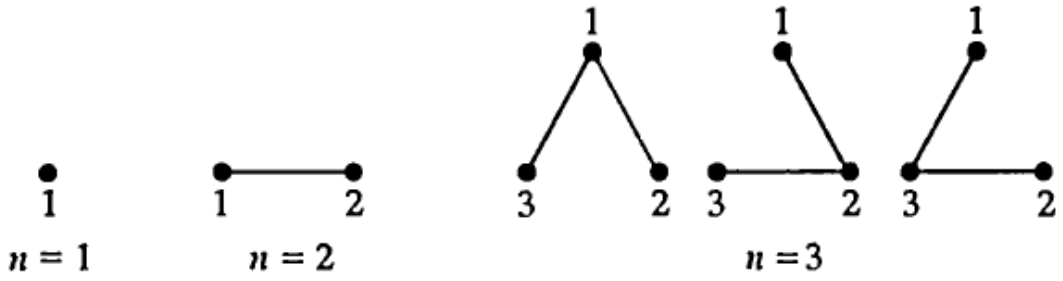
Fig. 4.15 – representação de expressões algébricas.

Teorema de Cayley: Os vértices de um grafo podem ou não ser ordenados. Dado n vértices existem n**(n-2) possíveis árvores com vértices ordenados que podem ser criadas.

A tabela abaixo apresenta o número de árvores ordenadas ou não que podem ser criadas à partir de n vértices.

número de vértices	1	2	3	4	5	6	7	8	9
Árvores sem vértices ordenados	1	1	1	2	3	6	11	23	47
Árvores com vértices ordenados	1	1	3	16	125	1296	16807	262144	4782969

Apresenta-se abaixo valores de n=1,2 e 3 respectivamente, e as possíveis árvores que podem ser criadas.



Ordenação topológica

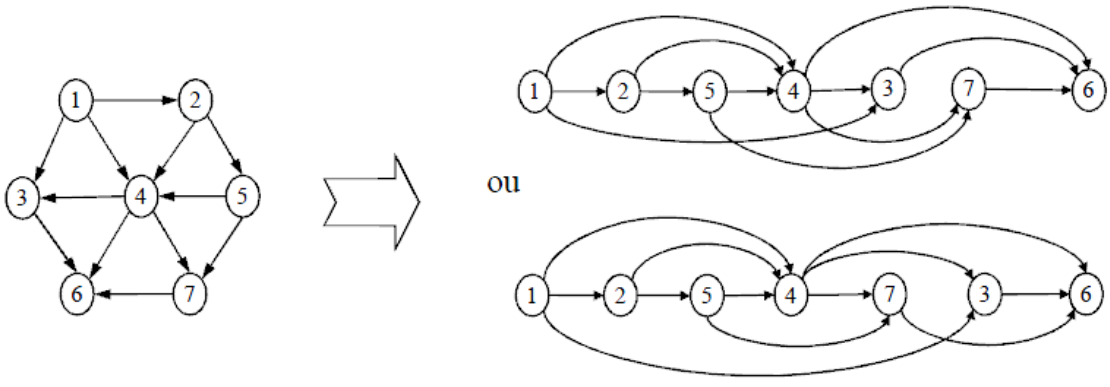
O critério da ordenação depende da aplicação em questão. Por exemplo, os vértices podem ser ordenados de acordo com os seus respectivos graus, e as arestas podem ser ordenadas de acordo com os pesos atribuídos a cada uma delas.

Uma ordenação topológica é uma permutação dos vértices v1,..,vn, de um digrafo acíclico, de forma que para qualquer aresta (vi ; vj), i < j. Qualquer caminho entre vi e vj não passa por vk se k < i ou k > j.

Dígrafos com ciclos não admitem ordenação topológica, enquanto todo DAG admite ordenação topológica.

Uma Ordenação Topológica de um grafo acíclico direcionado é uma ordenação linear de seus vértices, na qual cada vértice aparece antes de seus descendentes.

Exemplo1: **A ordenação não é necessariamente única**.



Exemplo2: O grafo abaixo permite diferentes ordenações topológicas:

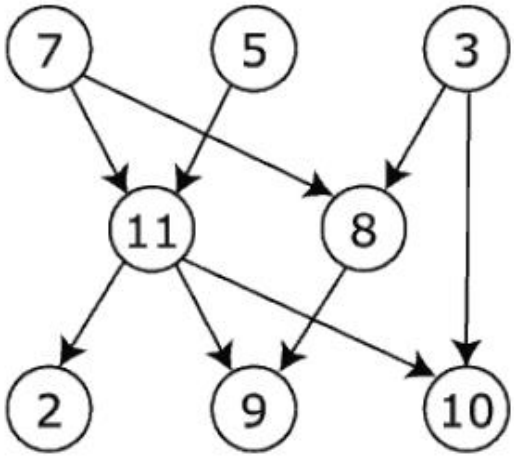


Fig. 4.16 – exemplo de ordenação topológica

a.) Ordenação tendo como prioridade o movimento da esquerda para a direita;
Sequencia: 7,5,3,11,8,2,9,10.

b.) Ordenação tendo como prioridade os vértices de menor número;
Sequencia: 3,5,7,8,11,2,9,10 ou 3,7,8,5,11,10,2

c.) Ordenação tendo como prioridade vértices com o menor número de arestas.

Sequencia: 5,7,3,8,11,10,9,2

Aplicações da ordenação topológica:

. Fluxograma de disciplinas de um curso, respeitando os pre-requisitos, na qual a ordenação representa as disciplinas que o aluno está apto a cursar.

. Os vértices do DAG podem representar tarefas a serem realizadas, e as arestas representam restrições de dependência entre as tarefas.

Se numa ordenação todos os pares de vértices consecutivos forem conectados por arestas, essas arestas formam um caminho hamiltoniano dirigido no DAG.

Se existe um caminho Hamiltoniano a ordenação topológica é única.

Se uma ordenação topológica não formar um caminho Hamiltoniano, o DAG terá mais do que uma ordenação, pois é possível trocar dois vértices consecutivos não ligados por uma aresta.

Algorítmos para ordenação topológica

Existem alguns algoritmos que realizam a ordenação topológica, tais como o algoritmo proposto em 1962 por A. B. Khan: “Topological sorting in large scale networks”, e a algoritmo DFS que será abordado a seguir.

O Algoritmo de Khan possui como princípio determinar os vértices que não possuam arestas de entrada e inserir na solução. A cada vértice inserido na solução, todas as arestas relacionadas são removidas do grafo G. O algoritmo é dado por:

. Identificar os vértices de um grafo G com grau de entrada zero ou vértices fonte;

. Atribuir os vértices a um conjunto V, que pode ser uma fila uma pilha;

. Considere que o grafo é acíclico, e portanto, que pelo menos um vértice desses deve existir;

. Se se os vértices fonte e suas arestas de saída forem removidos, o grafo remanescente é também um DAG:

. Remover do conjunto S sucessivamente os vértices fontes, rotulando os em ordem de remoção.

Algoritmos de percursos em árvores

Considera-se que uma árvore T tem uma raíz r e quaisquer sub árvores a partir de T são chamadas da esquerda para a direita de T1,T2,...,Tn. A raiz tem ramificações para as raízes de suas sub árvores.

Os algorítmos de percurso referem-se a procedimentos sistemáticos para visitar todos os vértices de uma árvore enraizada ordenada.

Endereçamento global: Procedimentos para percorrer todos os vértices de uma árvore enraizada ordenada baseiam-se na ordem dos filhos. Nas árvores enraizadas ordenadas, os filhos de um vértice interno são mostrados da esquerda para a direita nos desenhos que representam estes grafos orientados.

Segue abaixo procedimento para ordenar os vértices de uma árvore.

1. Rotule a raiz com o número 0. A seguir rotule seus K filhos (no nível 1) da esquerda para a direita, com 1,2,3,...,k.
2. Para cada vértice **v no nível n com rótulo A, rotule os seus Kij filhos da esquerda para a direita, com A.1;A.2...,A.n.**

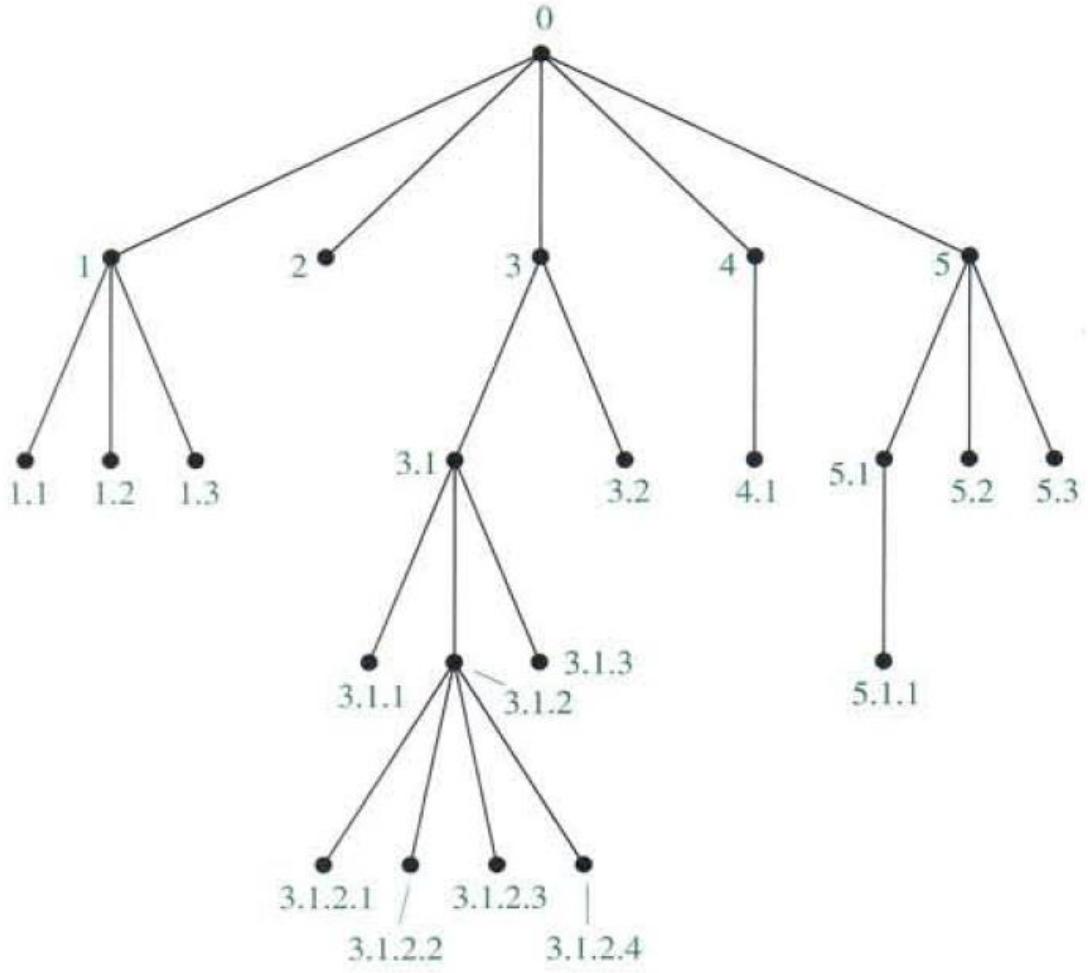


Fig.4.17 Sistema de endereçamento global

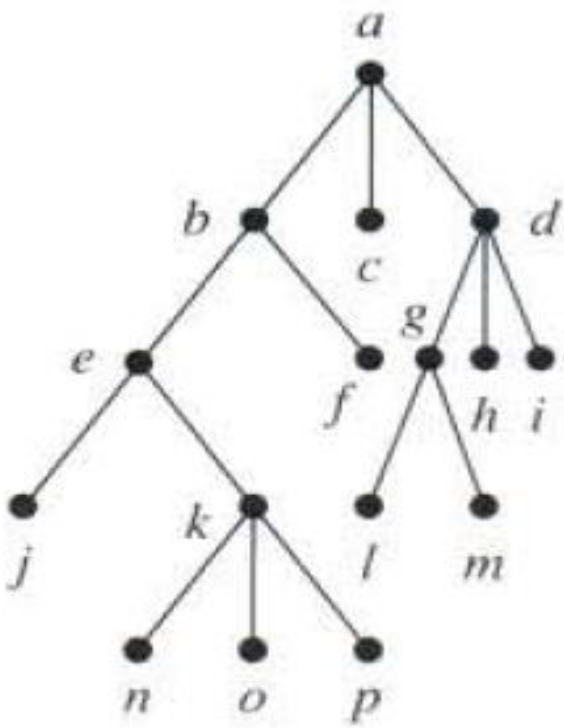
Três dos mais comumente usados algoritmos de percurso são:

. **Percursos em pré-ordem**: a raiz é visitada por primeiro e depois processam-se as sub árvores, da esquerda para a direita. Portanto, seja T uma árvore enraizada ordenada com raiz r. Se T consistir apenas em r, então r é o percurso em pré-ordem de T.

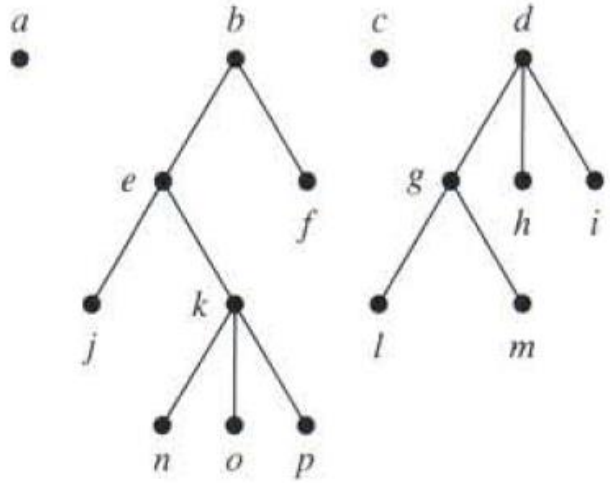
Caso contrário, suponha que T1, T2,..., Tn sejam sub árvores em r da esquerda para a direita. O percurso em **pré-ordem** começa visitando r, depois na sequencia T1, T2,...,Tn.

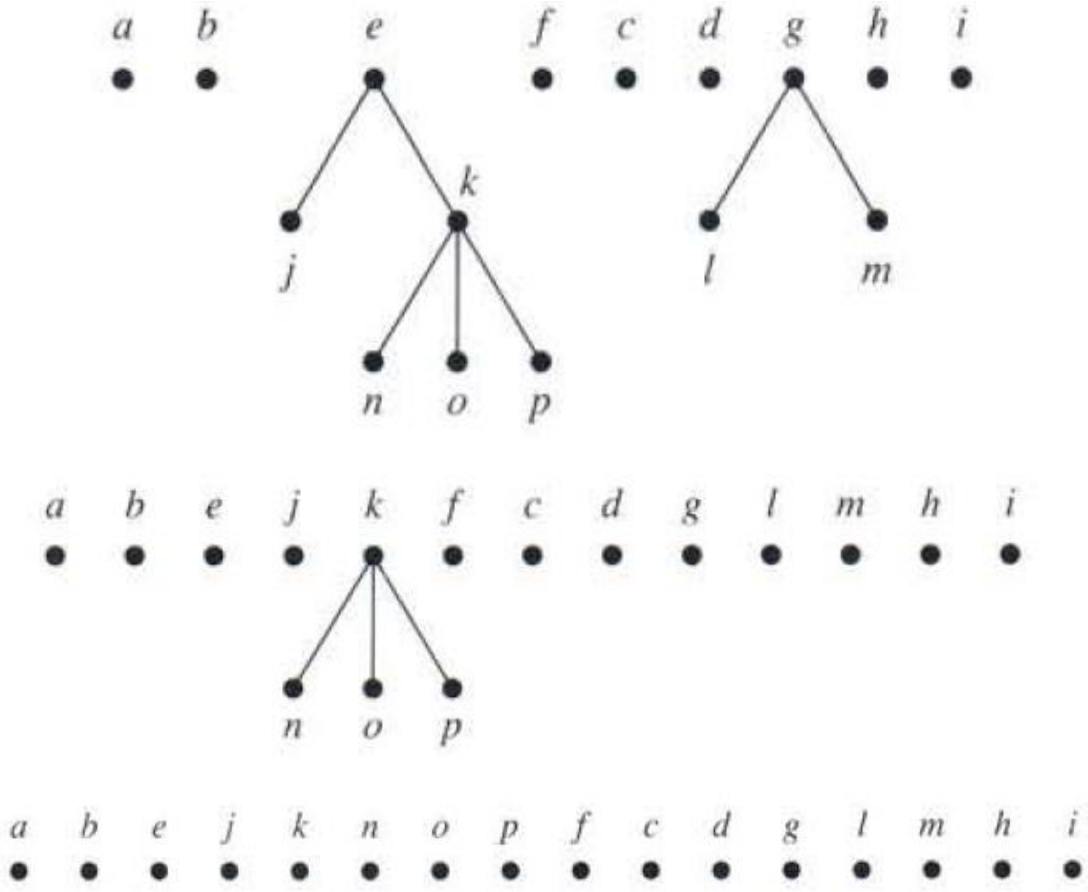
O percurso em pré-ordem de uma arvore enraizada ordenada fornece a mesma ordem dos vértices que a ordem obtida usando um sistema de endereçamento global.

Exemplo_a: Dado o grafo abaixo determinar em qual ordem um percurso em pré-ordem visita os vértices na árvore enraizada.



Resposta:





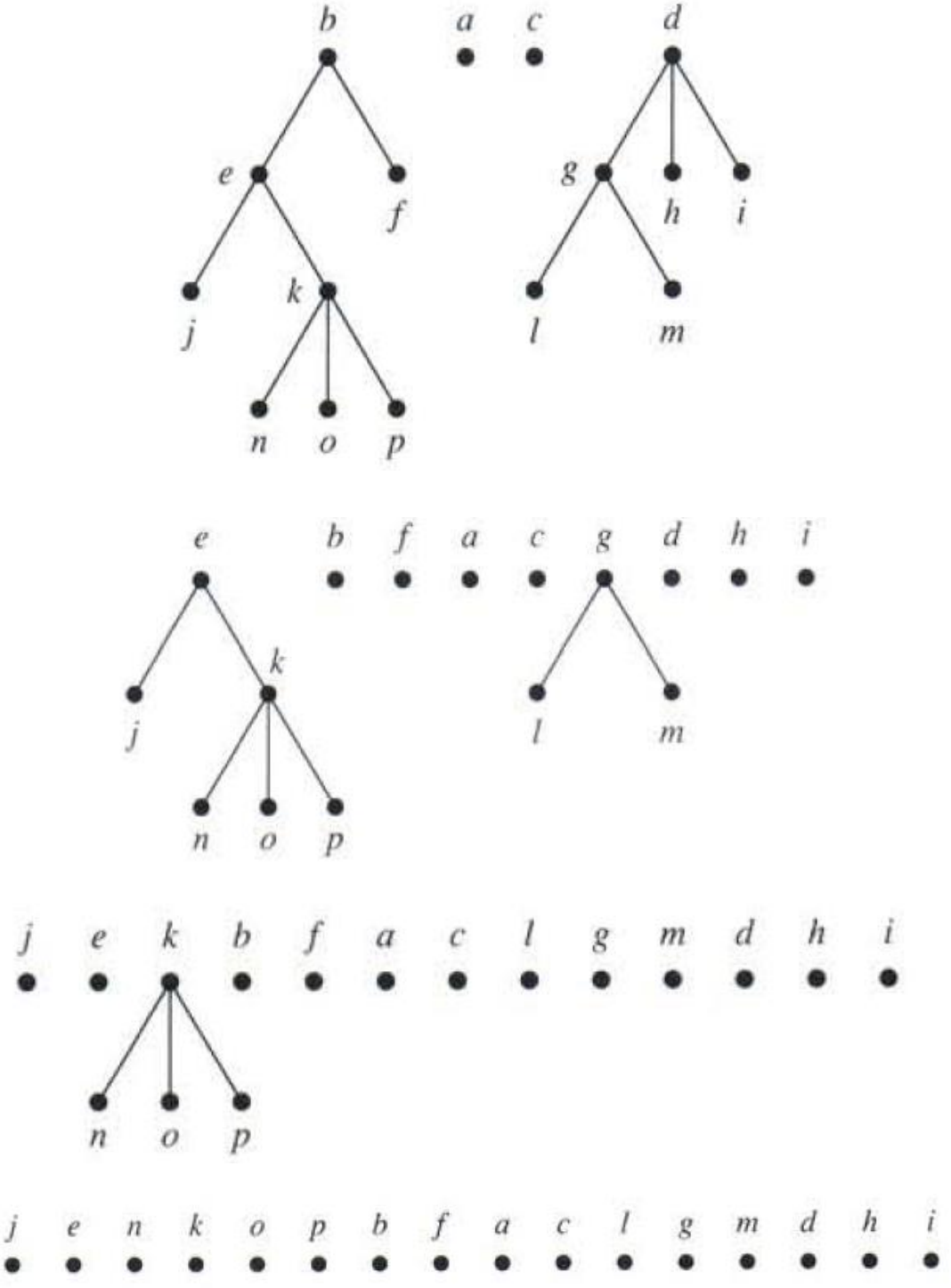
. **Percursos em ordem simétrica**: Seja T uma árvore enraizada ordenada com raíz r. Se T consistir apenas em r, então r é o percurso em ordem simétrica de T. Caso contrário, suponha que T1, T2,...,Tn sejam sub árvores em r da esquerda para a direita.

O percurso em ordem simétrica começa percorrendo T1 em ordem simétrica, visitando a seguir r. Continua percorrendo T2 em ordem simétrica, e depois T3 e final Tn em ordem simétrica.

Ou seja, visita a sub árvore mais a esquerda em ordem simétrica, depois a raiz é visitada e depois outras sub árvores são visitadas da esquerda para a direita, sempre em ordem simétrica.

Se a árvore for binária, a raiz é visitada entre as duas sub árvores.

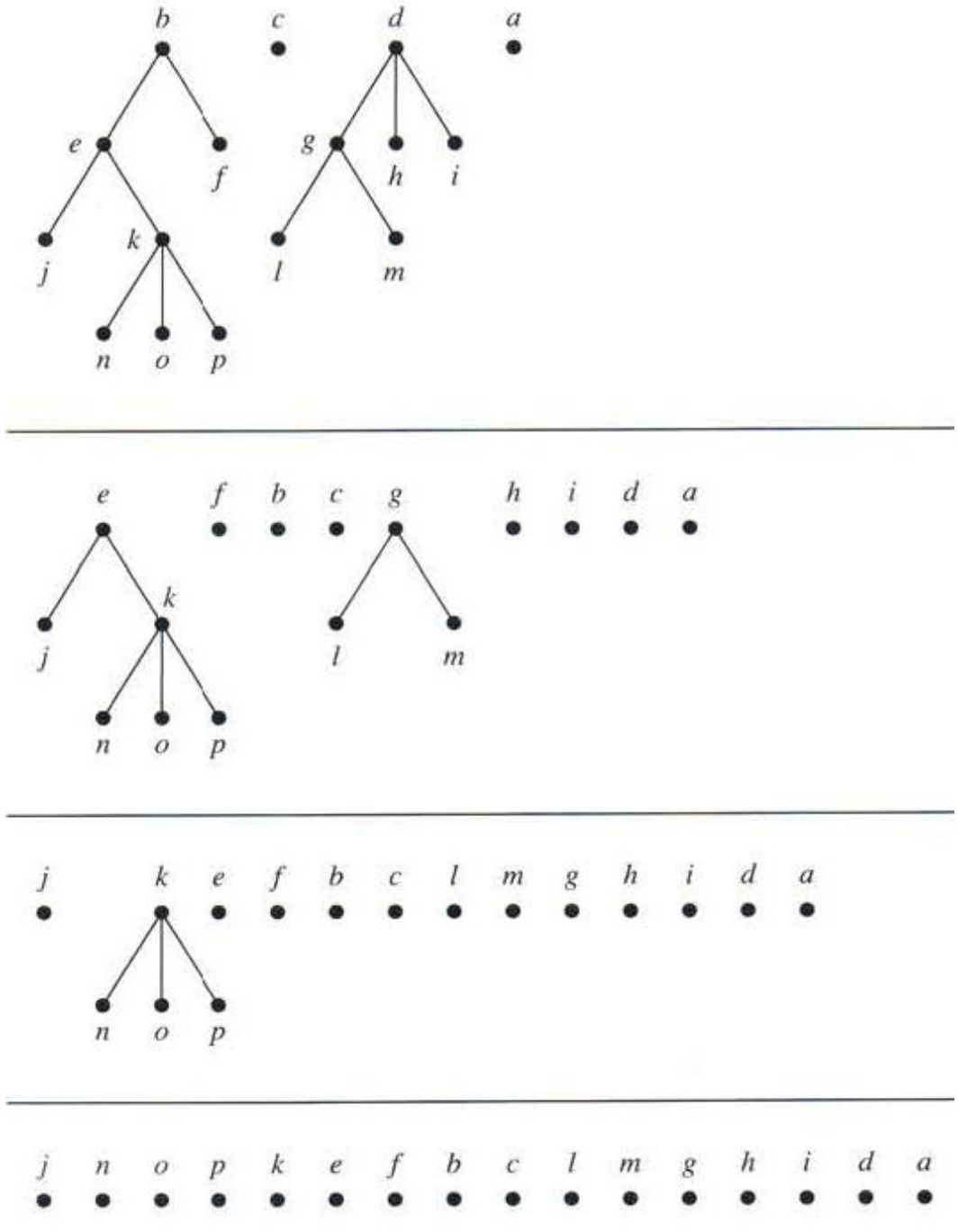
Considerando o grafo da figura anterior, o percurso em ordem simétrica é dado por:



. **Percursos em pós-ordem**: Seja T uma árvore enraizada ordenada com raíz r. Se T consistir apenas em r, então r é o

percurso em pós-ordem de T. Caso contrário, suponha que T1, T2,...,Tn sejam sub árvores em r da esquerda para a direita. O percurso em pós-ordem começa percorrendo T1 em pós-ordem, a seguir T2 em pós-ordem, a seguir Tn em pós-ordem, e termina visitando r.

A figura abaixo exemplifica como o circuito em pós-ordem é realizado.



Exemplo_b: Dado a figura 4.17 os resultados de busca aos nós da árvores variam de acordo com o algoritmo que está sendo utilizado.

Percurso pré-ordem: a,b,d,e,c,f,h,i,g.

Percurso ordem simétrica: d,b,e,a,h,f,i,c,g.

Percurso pós-ordem: d,e,b,h,i,f,g,c,a.

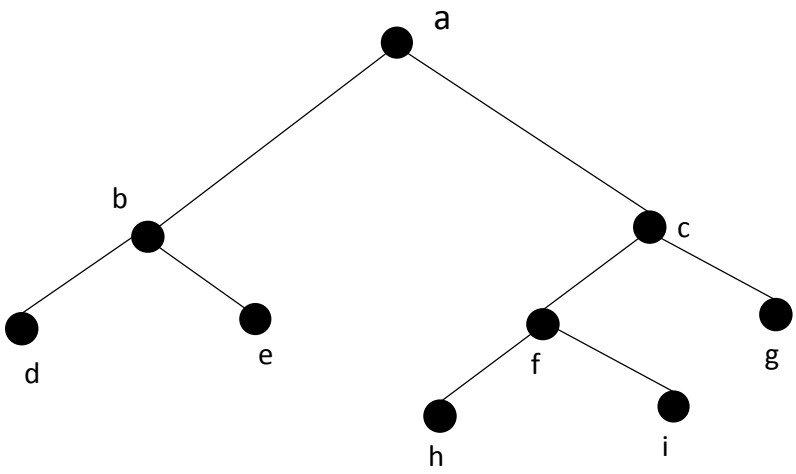


Fig. 4.17 – algoritmos de percurso em árvore binária.

Exemplo_c: Considere a figura 4.18 de uma árvore não binária.

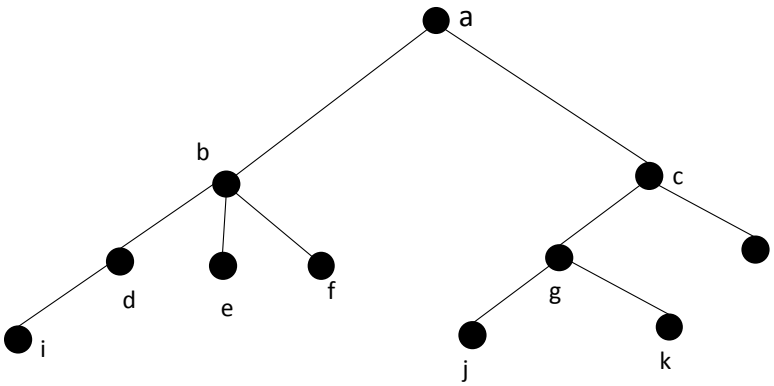


Fig. 4.18 – algoritmos de percurso em árvore.

Percurso pré-ordem: a,b,d,i,e,f,c,g,j,k,h.

Percurso ordem simétrica: i,d,b,e,f,a,j,g,k,c,h.

Percurso pós-ordem: i,d,e,f,b,j,k,g,h,c,a.

Exemplo_d: Considere a figura 4.19. Pode-se representar equações algébricas usando árvores ordenadas enraizadas.

Dada uma determinada expressão algébrica, a árvore binária que representa tal expressão pode ser construída de baixo para cima.

Os vértices internos representam operações e as folhas representam as variáveis ou números. Cada operação opera em suas sub árvores esquerda e direita.

O resultado da equação algébrica obtido com o tipo de algoritmo de busca utilizado.

Percurso pré-ordem: * + 2x4 (**forma prefixa** ou notaçãopolonesa), temos: (2 +x) * 4

Percurso ordem simétrica: (2 + x) * 4 (**notação infixa**). Esta notação precisa de parêntesis para garantir a ordem de precedência.

Percurso pós-ordem: 2x + 4* (**notação pós-fixada**),

temos: (2 + x)* 4

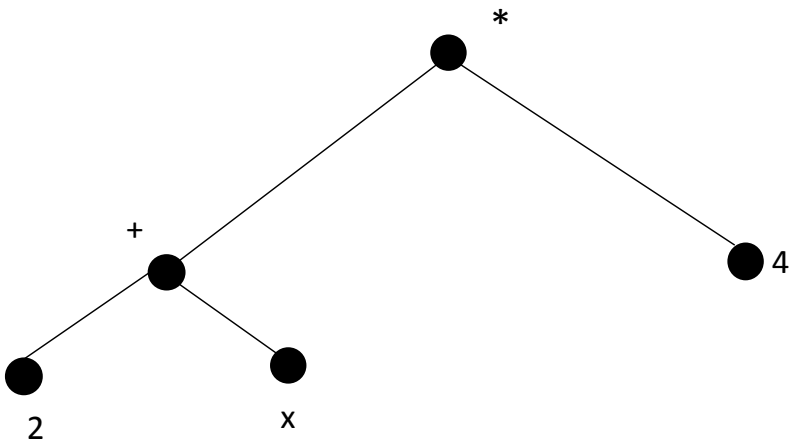
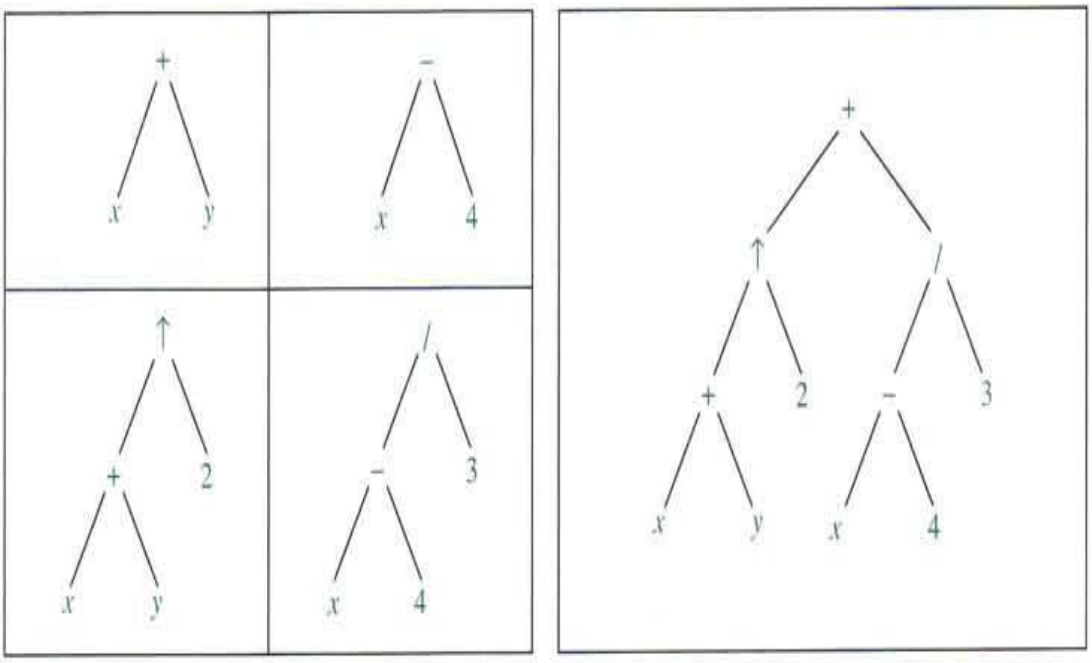


Fig. 4.19 – Representação da equação algébrica

Um percurso em **ordem simétrica** da árvore binária que representa uma expressão **produz a expressão original** com os elementos e operações na mesma ordem que eles ocorreram originalmente, exceto pelas operações unárias, que em vez disso imediatamente seguem seus operandos.

Por exemplo, a figura abaixo representa a expressão:

((X+Y)^2 + ((X-4)/3)

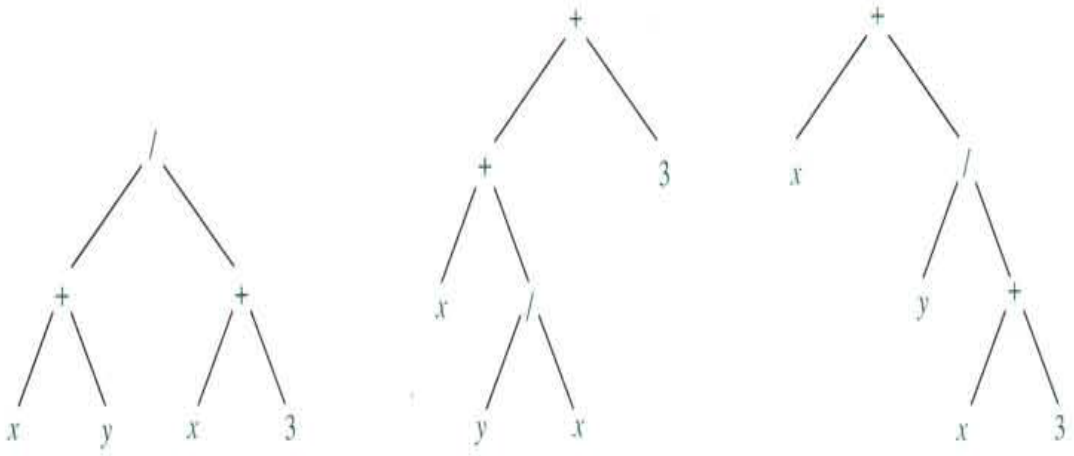


As figuras abaixo representam as expressões dadas por:

(x+y)/(x+3)

(x +(y/x)) + 3

X + (y/(x+3))



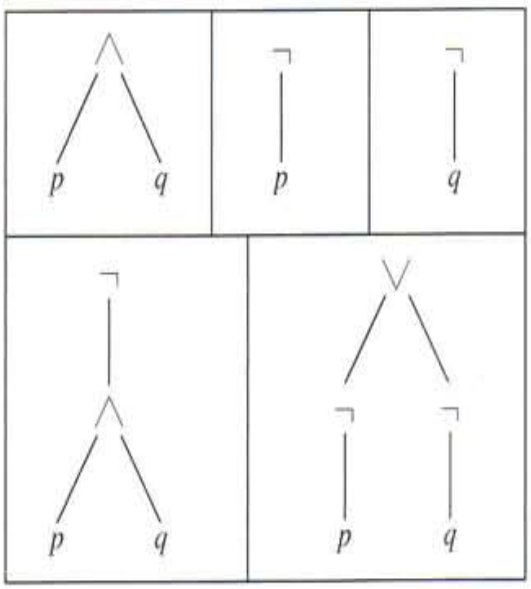
É necessário incluir parênteses n o percurso em ordem simétrica sempre que encontramos uma operação.

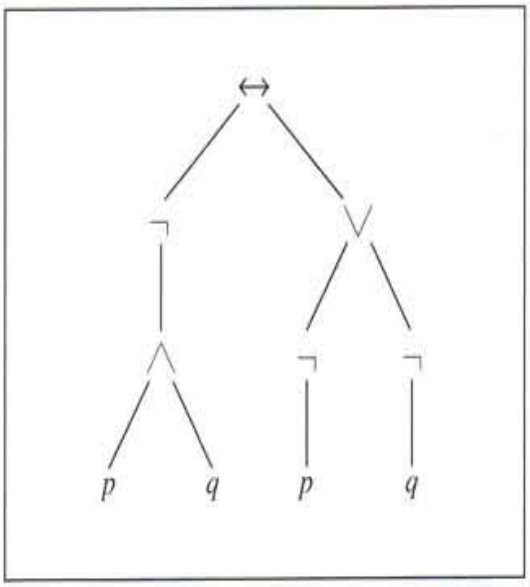
Exemplo_e: As árvores enraizadas podem também ser usadas para representar outros tipos de expressão, tal como as que representam **proposições compostas e combinações de conjuntos.**

Nestes exemplos, ocorrem operadores unários, tal como a negação de uma proposição. Para representar tais operadores e seus operandos são usados um vértice representando o operador e um filho deste vértice representando o operando.

Por exemplo, encontre a árvore enraizada que representa a proposição composta dada por:

$$(\neg(p \wedge q)) \leftrightarrow (\neg p \vee \neg q)$$





Exercício:

Abaixo são apresentados os três pseudocódigos dos algoritmos percurso em árvore. Implemente cada um deles**.**

ALGORITHM 1 Preorder Traversal.

```
procedure preorder(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
   T(c) := subtree with c as its root
   preorder(T(c))
end
```

ALGORITHM 2 Inorder Traversal.

```
procedure inorder(T: ordered rooted tree)
r := root of T
if r is a leaf then list r
else
begin
   l := first child of r from left to right
   T(l) := subtree with l as its root
   inorder(T(l))
   list r
   for each child c of r except for l from left to right
      T(c) := subtree with c as its root
      inorder(T(c))
end
```

ALGORITHM 3 Postorder Traversal.

procedure *postorder*(T: ordered rooted tree)
r := root of T
for each child c of r from left to right
begin
  $T(c)$:= subtree with c as its root
  postorder($T(c)$)
end
list r

Observação: Existe uma maneira mais fácil de listar os vértices de uma árvore enraizada ordenada.

Dado uma árvore G, primeiro desenhe uma curva em torno da árvore enraizada ordenada começando na raiz, movendo-se ao longo das arestas.

Pré-ordem: listar cada vértice na primeira vez que a curva passa por ele;

Ordem Simétrica: listar uma folha na primeira vez que a curva passa por ela, e listar cada vértice interno na segunda vez que a curva passa por ele.

Pós-ordem: listar o vértice na última vez que ele é visitado no caminho de volta para o seu pai.

Na figura abaixo é apresenta uma árvore enraizada com a curva desenhada em torno da árvore.

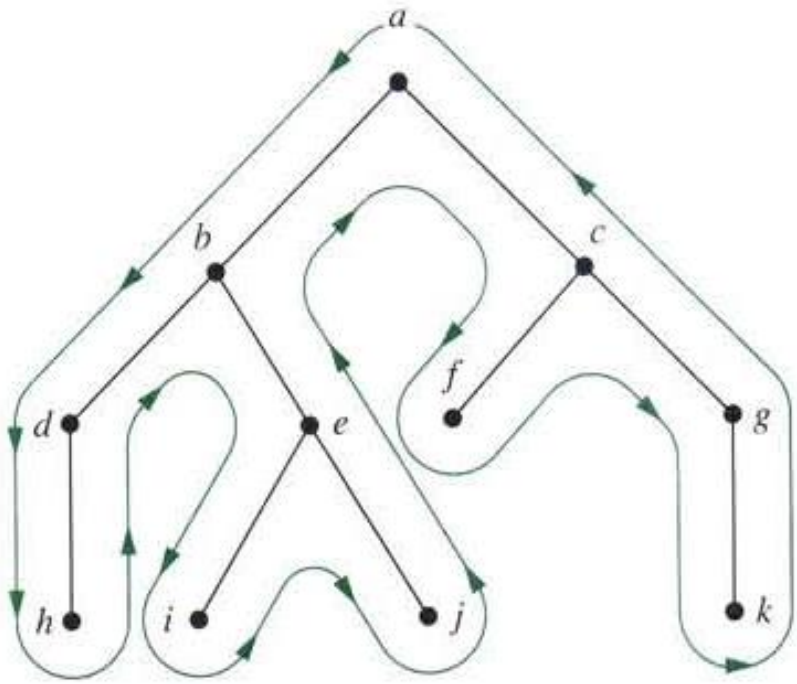


Fig. 4.19 – algoritmos de percurso em árvore.

Segue abaixo os percursos:

Pré-ordem: a, b, d, h, e, i, j, c, f, g, k;

Simétrica: h, d, b, i, e, j, a, f, c, k, g;

Pós-ordem: h, d, i, j, e, b, f, k, g, c, a.

Árvores de decisão: é uma árvore na qual os nós internos representam ações, os arcos representam os resultados de uma ação e as folhas representam os resultados finais.

As Árvores de decisão são utilizadas para modelar problemas nos quais uma série de decisões leva a uma solução.

As soluções possíveis de problema correspondem aos caminhos para as folhas desta árvore enraizada.

Diferentemente das estruturas das árvores apresentadas anteriormente, os nós da árvore **não possuem valores de dados associados**.

As **árvores de decisão** podem ser utilizadas para através de um procedimento de busca encontrar um elemento desejado x em uma lista de elementos, ou verificar que x não pertence à lista.

Esse algoritmo compara x sucessivamente com os elementos na lista. Os nós internos representam as ações de comparar x com os demais elementos na lista.

A altura da árvore representa o número de comparações, no pior caso, entre todos os casos possíveis.

. **Exemplo_a**: a figura 4.20 representa as diversas possibilidades de cinco jogadas de moedas com a restrição de que duas caras consecutivas não podem ocorrer. Cada nó interno da árvore representa uma ação (uma jogada de moeda) e os arcos representam o resultado da ação, que pode ser cara (C) ou coroa (K). As folhas das árvores representam os resultados finais.

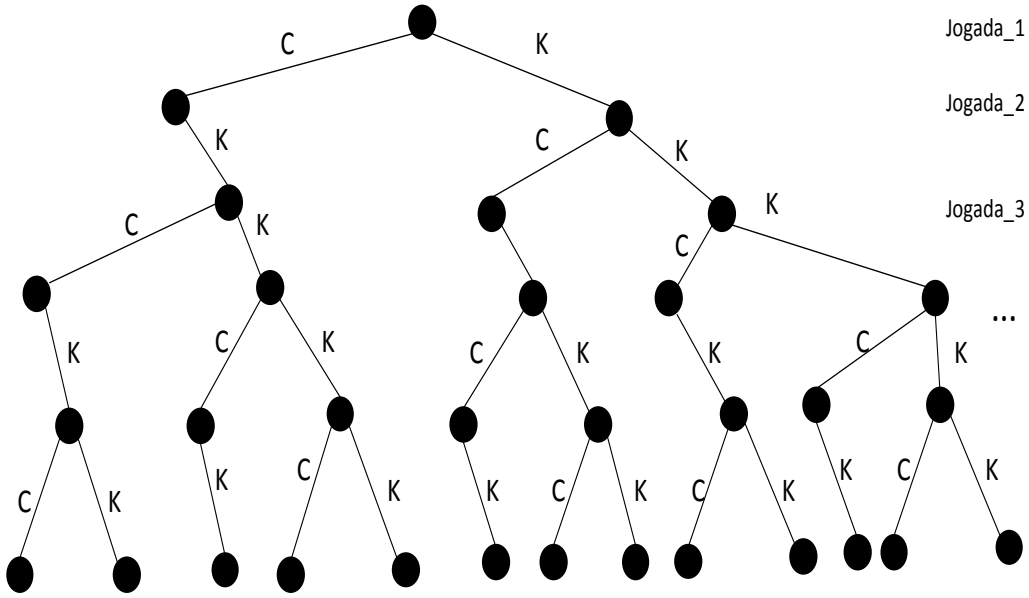


Fig. 4.20 – Representação jogadas de uma moeda

Exemplo_b: As árvores de decisão podem modelar algoritmos que ordenam uma lista de itens através de uma sequência de comparações entre dois itens da lista. Os nós internos são rotulados de uma árvore para indicar a comparação do item i da lista com o item j.

A figura 4.21 apresenta uma árvore de decisão que ordena os elementos da lista A,B e C.

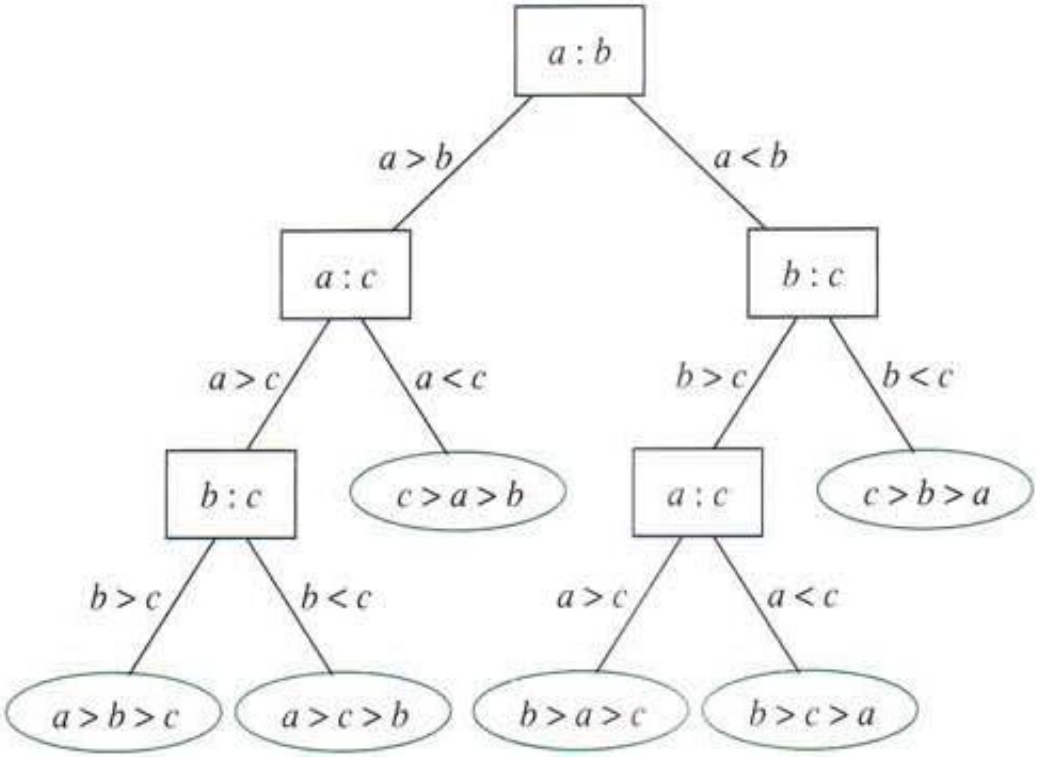


Fig. 4.21 – Uma árvore de decisão para ordenar três elementos distintos.

Exemplo_c: A figura 4.22 apresenta uma árvore de decisão para o algoritmo de busca sequencial agindo em uma lista ordenada com cinco elementos. O algoritmo sequencial tem apenas dois resultados possíveis para uma comparação entre x e L(i).

Se x = L(i), o algoritmo termina, já que x foi encontrado na lista. Se x ≠ L(i), a próxima comparação a ser feita é x:L(i+1), independente de se x ser maior ou menor do que L(i). As folhas da árvore correspondem aos resultados finais.

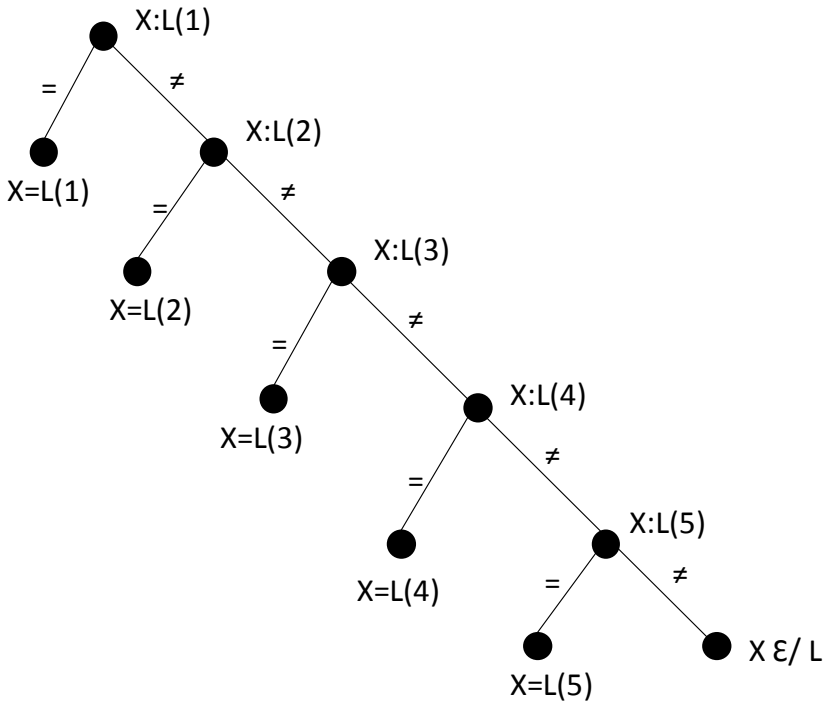


Fig. 4.22 – árvore de busca sequencial

Exemplo_d: A figura 4.23 apresenta uma árvore de decisão para o algoritmo de busca binária. A busca binária age em uma lista ordenada e possui 3 resultados possíveis para cada comparação:

. x = L (i) : o algoritmo termina, a lista foi encontrada;

. x < L(i) : o algoritmo vai para a metade esquerda da lista;

. x > L(i) : o algoritmo vai para a metade direita da lista.

As folhas da árvore binária representam todos os resultados possíveis quando x não pertence a lista.

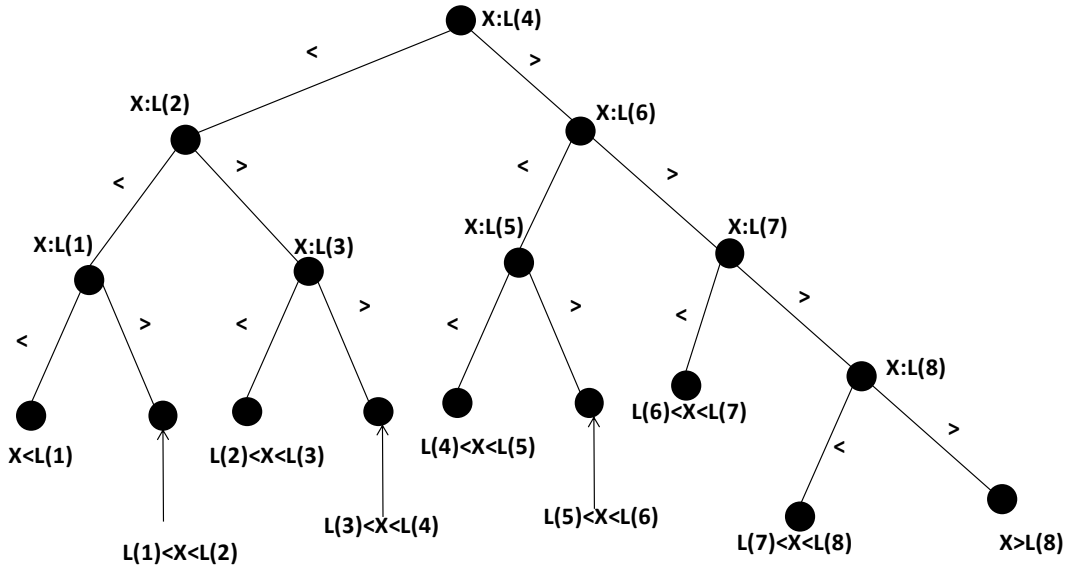


Fig. 4.23 – árvore de decisão binária

Observação: O algoritmo de busca binária necessita que os dados estejam ordenados.

Uma árvore binária de busca tem a propriedade de que o valor em cada nó é maior do que todos os valores em sua subárvore da esquerda e menor do que todos os valores em sua subárvore da direita.

Uma árvore binária de busca não é única para um determinado conjunto de dados. **A árvore depende da ordem na qual os dados são colocados na árvore**.

Exemplo_e: as duas sequências possuem o mesmo conjunto de números, porém ao serem ordenados geram árvores diferentes:

9,12,10,5,8,2,14 (figura 4.24.a)

5,8,2,12,10,14,9 (figura 4.24.b)

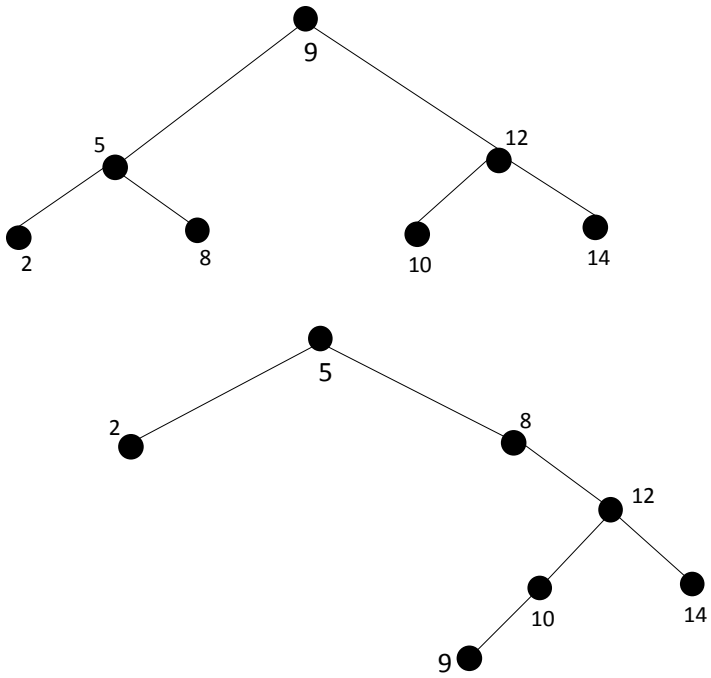


Fig. 4.24 – árvores binárias de busca

Exemplo_f: **Árvore de jogo**. As árvores tem sido utilizadas para analisar certos tipos de jogos. Em cada um desses jogos, dois jogadores se alteram nos movimentos.

A raiz representa a posição inicial do jogo e as folhas das árvores representam as posições finais de um jogo. Ou seja, pode-se definir recursivamente o valor de todos os vértices em uma árvore de jogo de maneira que nos permita determinar o resultado do jogo.

A figura 4.25 apresenta a árvore de jogo para o jogo da velha

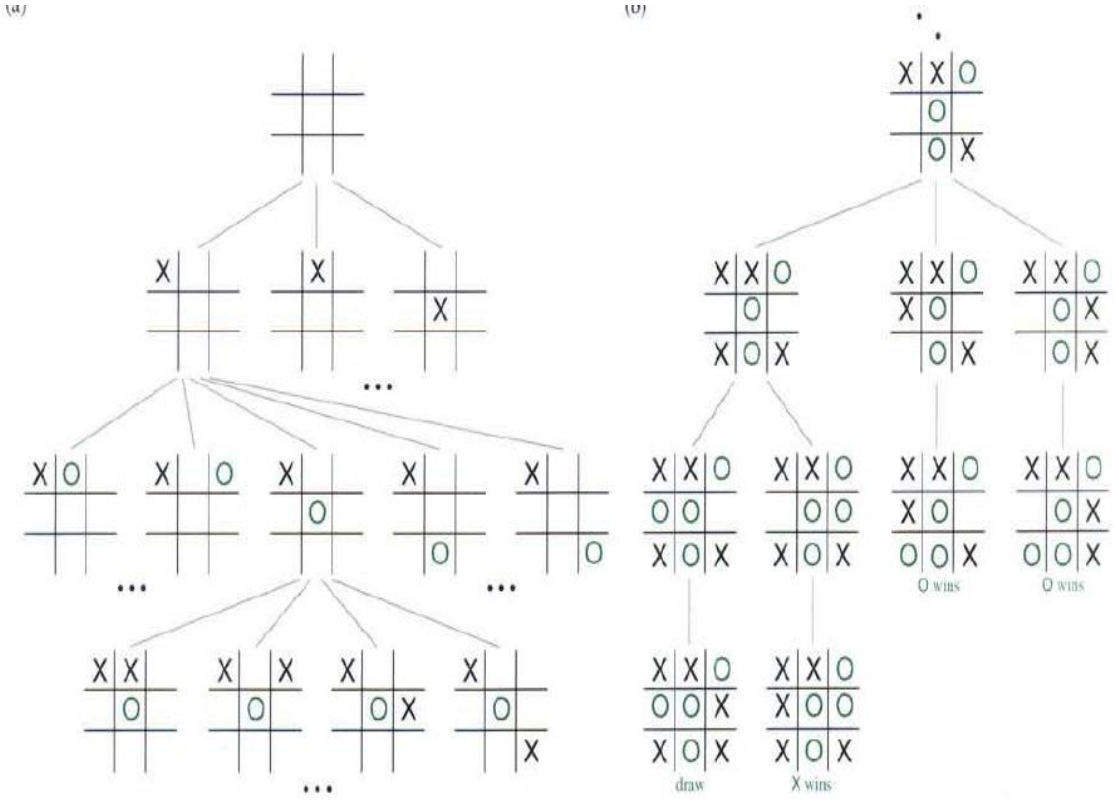


Fig. 4.25 – árvores para o jogo da velha

Exemplo_g: **Códigos de prefixo**. Considere o uso de sequencias de bits de comprimento diferentes para codificar letras. Quando letras são codificadas usando números variáveis de bits, deve ser usado algum método para determinar onde os bits para cada caractere começam e terminam.

Uma maneira de garantir que, nenhuma sequencia de bits corresponda a mais de uma sequencia de letras, é codificar as letras de modo que a sequencia de bits para uma letra nunca ocorra como a primeira parte de uma sequencia de bits para uma outra letra.

Um **código de prefixo** pode ser representado usando uma árvore binária, em que os caracteres são os rótulos das folhas das árvores.

As arestas são rotuladas de modo que a uma aresta que leva a um filho esquerdo seja associado um 0 e a uma aresta que leve a um filho da direita seja associado um 1.

A sequencia de bits usada para codificar um caractere é a sequencia de rótulo das arestas do único caminho da raiz à folha tem este caractere como rótulo.

A árvore pode ser utilizada para decodificar uma sequencia de bits.

A árvore da figura 4.26 representa a codificação da letra E por 0; A por 10; T por 110; N por 1110; e S por 111. Dada a sequencia de bits **11111011100** e usando a árvore codificada da figura 4.26.

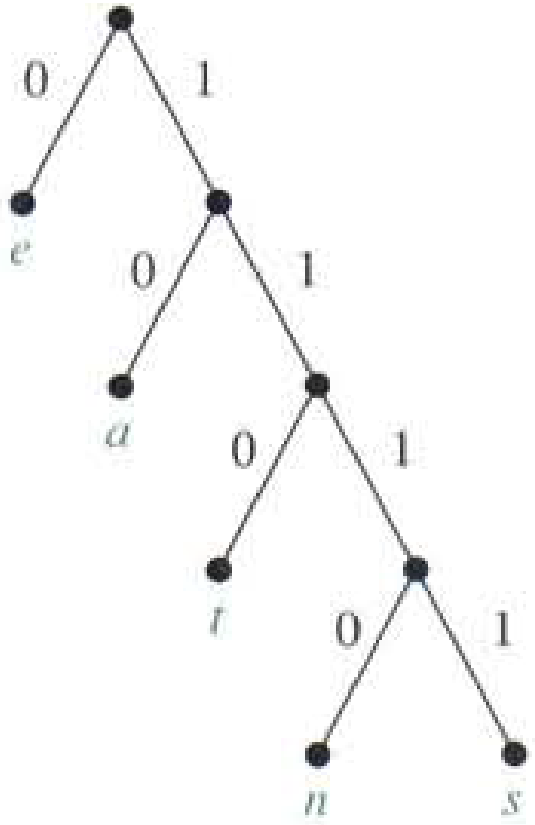


Fig. 4.26 – árvore binária com um código de prefixo

Esta sequencia de bits pode ser decodificada começando na raiz, usando a sequencia de bits para formar um caminho que termina quando uma folha é atingida.

Visitando as folhas da árvore decodifica-se a sequencia de bits, identificando a palavra: "***sane***".

Exemplo_h: **Códificação de Huffman**. Consiste em um algoritmo que tem como entrada de dados a frequência (ou probabilidade de ocorrência) com que determinados símbolos ocorrem dentro de uma sequencia de caracteres (strings), e determina um código de prefixo que codifica o conjunto mínimo de bits.

Dado a seguinte sequencia de símbolos e suas respectivas frequências de ocorrência dentro de um determinado string: A:0,8 ; B:0,10 ; C: 0,12 ; D: 0,15 ; E: 0,20 ; F: 0,35.

Utilize o codificação de Huffman para codificar os respectivos símbolos. Qual o número médio de bits que serão necessários para codificar um símbolo utilizando esta codificação.

A figura abaixo ilustra a execução do algoritmo passo-a-passo. A codificação produziu os seguintes códigos associados aos caracteres dados acima:

A: 111

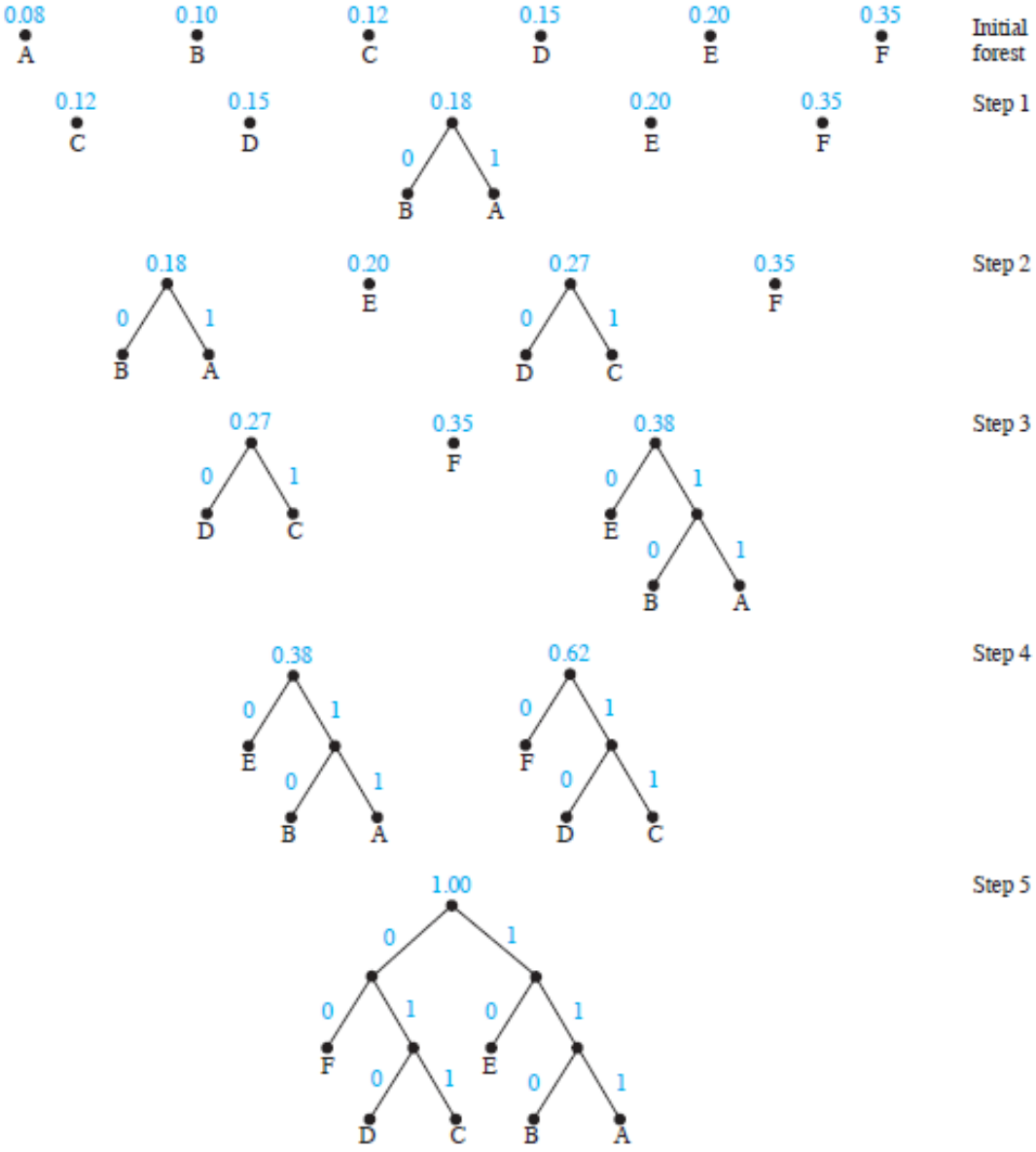
B: 110

C: 011

D: 010

E: 10

F: 00


0.08
A
0.10
B
0.12
C
0.15
D
0.20
E
0.35
F
Initial forest
0.12
C
0.15
D
0.18
0
1
B
A
0.20
E
0.35
F
Step 1
0.18
0
1
B
A
0.20
E
0.27
0
1
D
C
0.35
F
Step 2
0.27
0
1
D
C
0.35
F
0.38
0
1
E
0
1
B
A
Step 3
0.38
0
1
E
0
1
B
A
0.62
0
1
F
0
1
D
C
Step 4
1.00
0
1
0
1
F
0
1
0
1
E
D
C
B
A
Step 5

Exercício: Dado uma árvore enraizada (relação de vértices e arestas) binária encontre para um determinado vértice da árvore, quais são os seus ancestrais, filhos e descendentes. Encontre também o nível deste vértice. Utilize como dados de entrada a árvore da figura 4.19

Exercício: Dado uma árvore enraizada binária desenvolva um algoritmo que realize a inserção de um vértice (interno) nesta árvore.

Exercício: Dado o exemplo H implemente a codificação de Huffman a fim de encontrar o código para os respectivos símbolos.

Caminhamento em Grafos

Um algoritmo para resolver um problema utilizando grafos necessita que o problema esteja representado de forma correta. A especificação dos dados de um problema corresponde ao conjunto de vértices e de arestas de um grafo.

A técnica mais elementar que se pode aplicar para se examinar um grafo consiste na análise das listas de adjacências de seus vértices.

Na **técnica de adjacência**, em geral existe uma propriedade P(v) definida de modo apropriado, para vértices v Ɛ V de um grafo G(V,E).

A ideia consiste em se examinar listas de adjacências A(v), para v Ɛ V, de modo que se possa distinguir se w Ɛ A(v) satisfaz ou não a propriedade P(v).

O caminhamento ou busca em um grafo G dá-se, de tal forma que, uma aresta ou vértice ainda não foram visitados são marcados como não visitados.

Portanto, inicialmente, todos os vértices e arestas são marcados como não visitados e após terem sido visitados, os mesmos são marcados como visitados.

No final da busca ou percurso todos os vértices e arestas são marcados como visitados. A figura 4.26 apresenta **um algoritmo de busca genérica em grafos**.

```
Ler G = (N,M)
Escolher e marcar um vértice i
Enquanto existir j ∈ N marcado com uma aresta (j,k) não explorada Fazer
        Escolher o vértice j e explorar a aresta (j,k)
        // condição variável em conformidade com o tipo de busca //
        Se k é não marcado então marcar k
Fim_do_Enquanto
```

Fig. 4.26 – Algorítmo de busca genérica

Busca da saída em um labirinto: trata-se de uma aplicação clássica de busca em grafos !

Considera-se que a circulação no labirinto seja realizada margeando as paredes. Os pontos de mudança de direção são apresentados na figura 4.27.b.

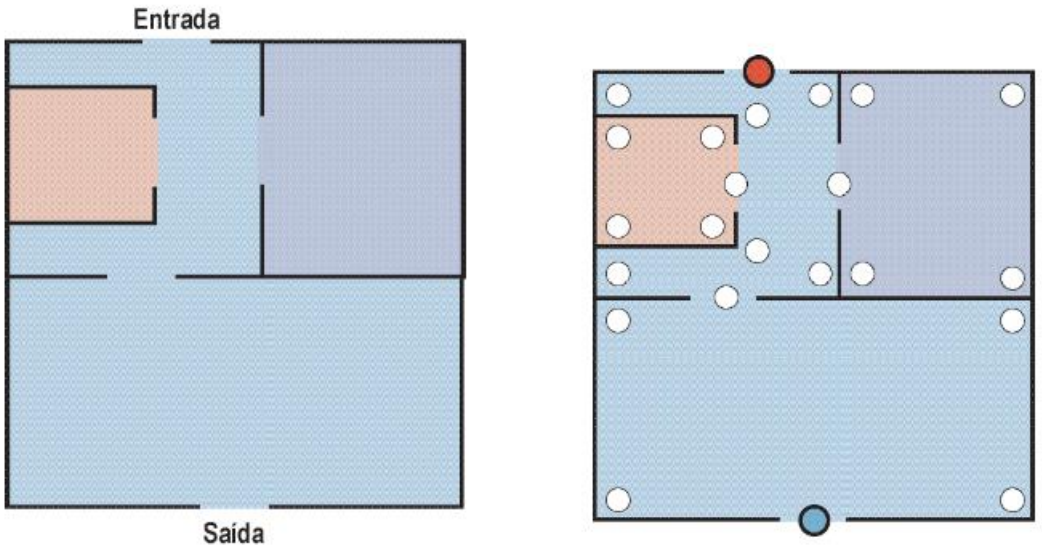


Fig. 4.27 – representação do labirinto

A figura 4.28 apresenta a construção do grafo a partir do formato do labirinto. As arestas do grafo apresentam alguma forma de

movimentação no labirinto entre os pontos de mudança de direção.

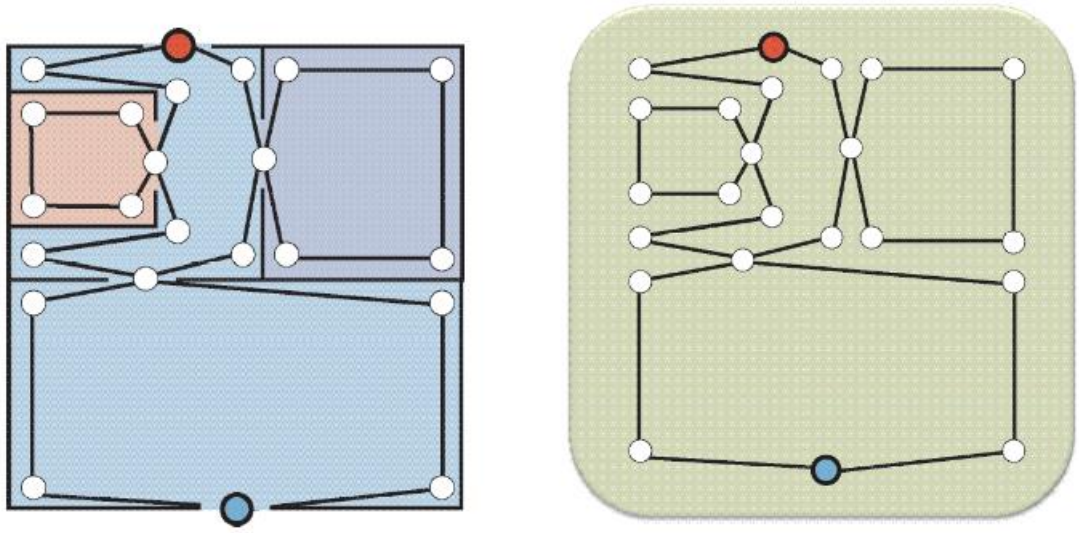


Fig. 4.28 – construção do grafo a partir do labirinto

Busca em profundidade em um grafo G (DFS) não direcionado

Inicia-se em um nó arbitrário **f** de um grafo G, marca-se este nó com tendo sido visitado e na sequencia escreve-se este nó.

Percorre-se então um caminho saindo de **f**, visitando e escrevendo os nós não marcados, indo tão longe quanto possível até não existirem mais nós que não foram visitados nesse caminho.

A **ordem** em que os nós e arestas de um grafo G(V,E) são visitados depende: **do nó inicial e da ordem em que os nós e as arestas aparecem na estrutura de dados.**

Volta-se então pelo caminho explorado. Em cada nó, no retorno, visita-se quaisquer caminhos laterais, até voltar finalmente para f.

Depois explora-se quaisquer novos caminhos restantes a partir de f.

Exemplo_a: A figura 4.26 apresenta um grafo após a visita dos primeiros nós usando busca em profundidade.

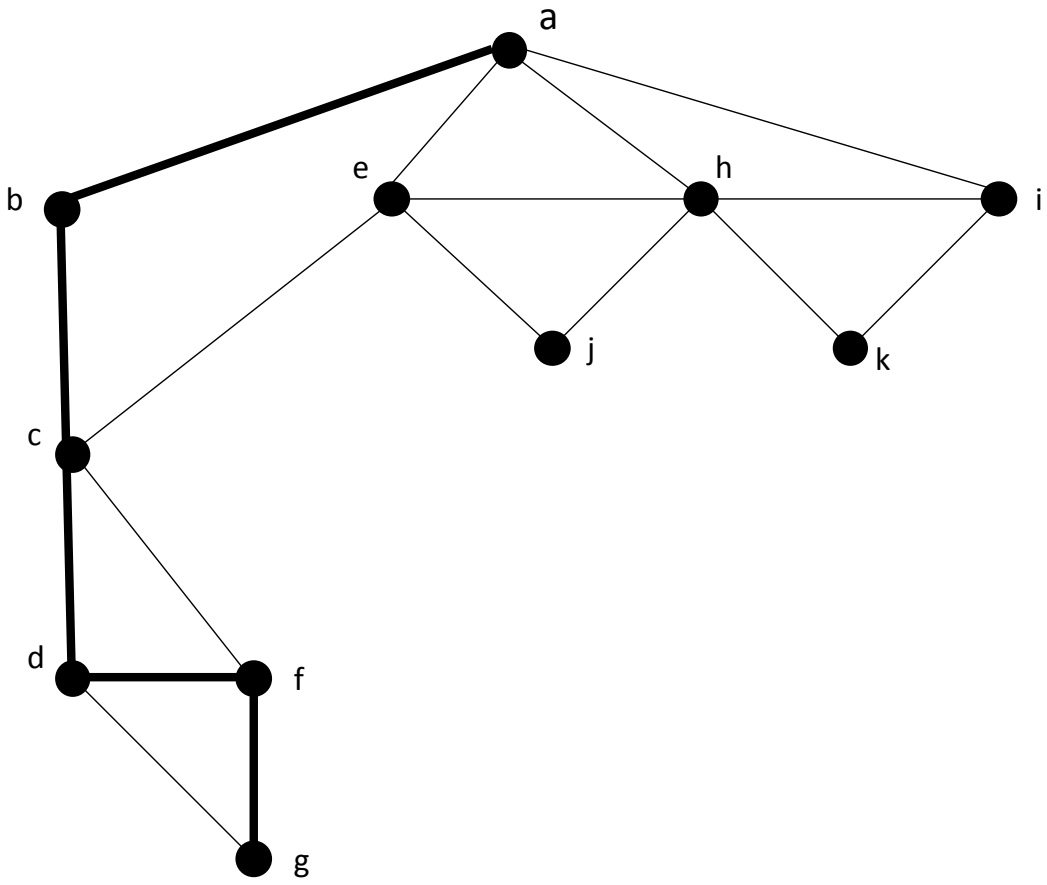


Fig. 4.26 – Exemplo de busca em profundidade em um grafo

A lista completa dos nós, na ordem em que foram escritos é: ***a,b,c,d,f,g,e,h,i,k,j.***

Exemplo_b: A figura 4.27 apresenta um segundo exemplo, de busca em profundidade em um grafo G a partir do nó 1. O algoritmo apresenta como resultado uma árvore.

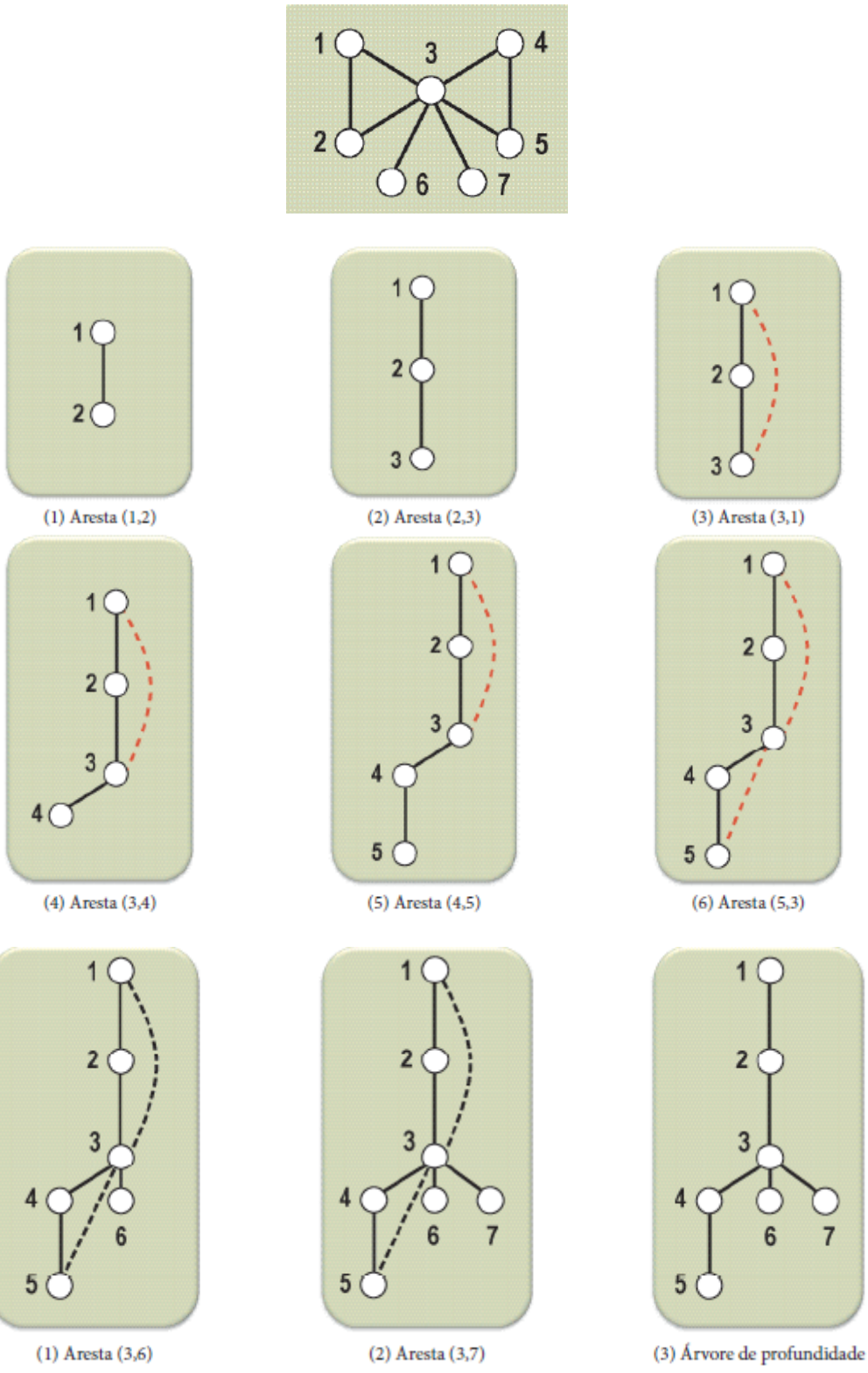


Fig. 4.27 – Exemplo de busca em profundidade em um grafo

Se o grafo for desconexo então o procedimento de busca em profundidade deve ser chamado novamente enquanto existir vértice não marcado no grafo. Neste caso, em vez de uma árvore em profundidade, o algoritmo construíra uma floresta em profundidade.

Exemplo: analise com detalhes o exemplo a seguir do algoritmo DFS.

Busca em profundidade em um grafo G (DFS) direcionado: A aplicação do algoritmo de busca em profundidade em grafos direcionados é essencialmente a mesma que nos grafos não direcionados. Uma diferença é que, mesmo o grafo sendo conexo, o algoritmo de busca pode gerar uma floresta em profundidade.

No algoritmo de busca em profundidade para grafos direcionados, as arestas do grafo são qualificadas em 4 conjuntos:

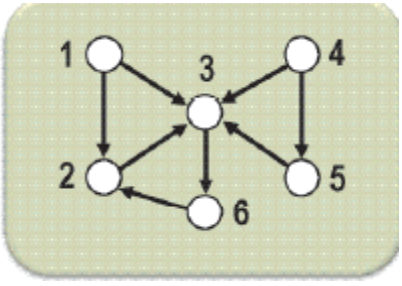
. arestas onde o vértice destino ainda não foi marcado;

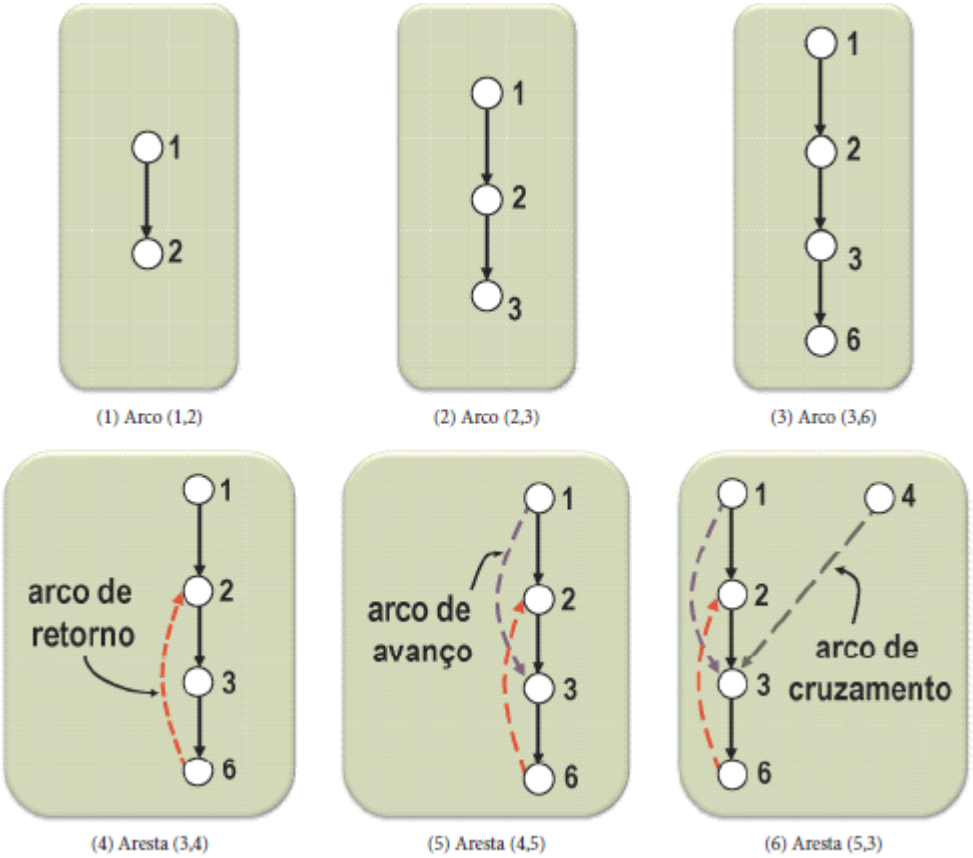
. arestas tipo avanço: se w é descendente de v na floresta;

. arestas tipo retorno: se v é descendente de w na floresta;

. arestas tipo cruzamento: se nem v é descendente de w, nem w é descendente de v na floresta.

Exemplo_c: A figura 4.28 apresenta um exemplo de busca em profundidade em um grafo direcionado.





A figura 4.28 apresenta o algoritmo que implementa a busca em profundidade em um grafo G. Os dados de entrada correspondem a um grafo simples e conexo G e um nó especificado a.

A saída é uma lista de todos os nós de G em ordem de profundidade de a. O algoritmo utiliza a recorrência.

O fato de utilizar recorrência pode empilhar várias chamadas de funções ! A pilha pode estourar !

Exemplo: A função GRAPHdfs() abaixo faz uma busca em profundidade num grafo G. A busca poderia começar por qualquer vértice, mas é natural começá-la pelo vértice 0. A função visita todos os vértices e todos os arcos do grafo G. A função atribui um número de ordem pre[x] a cada vértice x: o k-ésimo vértice descoberto recebe número de ordem k. A numeração dos vértices é registrada em um vetor pre[] indexado pelos vértices.

```
static int cnt;
int pre[1000];
void GRAPHdfs( Graph G)
{
  vertex v;
  cnt = 0;
  for (v = 0; v < G->V; ++v)
    pre[v] = -1;
  for (v = 0; v < G->V; ++v)
    if (pre[v] == -1)
      dfsR( G, v); // começa nova etapa
}
```

Seja X o conjunto dos vértices x que estão ao alcance de v e têm pre[x] == -1. Para cada vértice x em X, a função dfsR() atribui um valor positivo a pre[x] de modo que o k-ésimo vértice descoberto receba valor cnt+k. O código supõe que G é representado por uma matriz de adjacências.

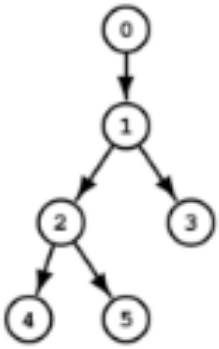
```
static void dfsR( Graph G, vertex v)
{
  vertex w;
  pre[v] = cnt++;
  for (w = 0; w < G->V; ++w)
    if (G->adj[v][w] != 0 && pre[w] == -1)
      dfsR( G, w);
}
```

Cada etapa da busca resume se em:

(a) escolha um vértice não descoberto v;

(b) visita todos os vértices que estão ao alcance de v mas ainda não foram descobertos.

Dado o grafo G, na sequencia é apresentado o rastreamento de uma execução



Cada linha da tabela abaixo registra o momento em que um arco é percorrido (ou seja, o momento em que a função dfsR() vai até a ponta final do arco ao examinar os vizinhos da ponta inicial) e a correspondente invocação de dfsR(). A execução de cada nova encarnação de dfsR() é indicada por uma indentação apropriada das linhas:

```
                              pre
0 dfsR(G,0)                    0
0-1 dfsR(G,1)                  1
. 1-2 dfsR(G,2)                2
. . 2-4 dfsR(G,4)              3
. . . 4 termina                .
. . 2-5 dfsR(G,5)              4
. . . 5 termina                .
. . 2 termina                  .
. 1-3 dfsR(G,3)                5
. . 3 termina                  .
. 1 termina                    .
0 termina                      .
```

Algorítmo EmProfundidade

```
EmProfundidade(grafo G; nó a)
// Escreve os nós do grafo G em ordem de profundidade a partir do nó a.
Marque a como tendo sido visitado
Escreva(a)
Para [cada nó n adjacente a (a)] faça
        Se [nó n não tiver sido visitado] então
                EmProfundidade(G; n)      // recursividade
        Fim_se
Fim_Para
Fim_EmProfundidade
```

Fig. 4.28 – Algorítmo de busca em profundidade.

Pode-se identificar uma **árvore geradora** para um grafo simples e conexo usando a **busca em profundidade**.

Para isto, escolhe-se um vértice do grafo arbitrariamente como raiz e forma-se um caminho começando neste vértice adicionando sucessivamente vértices e arestas, em que cada nova aresta é incidente ao último vértice do caminho e a um vértice que ainda não esteja no caminho.

Se o caminho passar por todos os vértices do grafo, a árvore que consiste neste caminho é uma **árvore geradora**. Se o caminho não passar por todos os vértices, mais vértices e arestas devem ser adicionados ao processo de busca.

Retorne para o penúltimo vértice no caminho e, se possível, forme um novo caminho começando neste vértice, passando por vértices que ainda não tenham sido visitados.

Se isto não puder ser realizado, volte mais um vértice no caminho, e tente novamente.

O algoritmo que identifica **árvores geradoras** a partir da remoção de arestas de ciclos simples (visto anteriormente) é ineficiente, pois, exige que ciclos simples sejam identificados.

Exemplo_d: Utilize a busca em profundidade para encontrar uma **árvore geradora** para o grafo G mostrado abaixo:

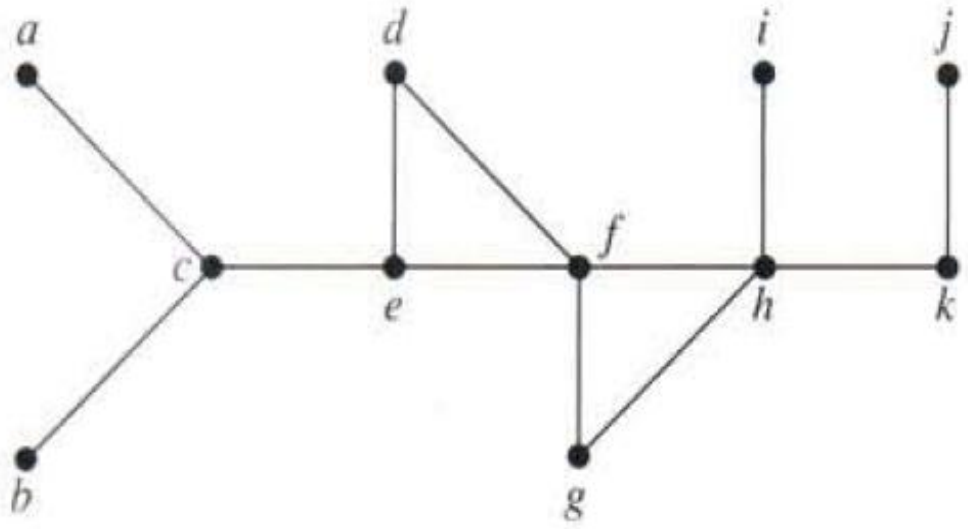
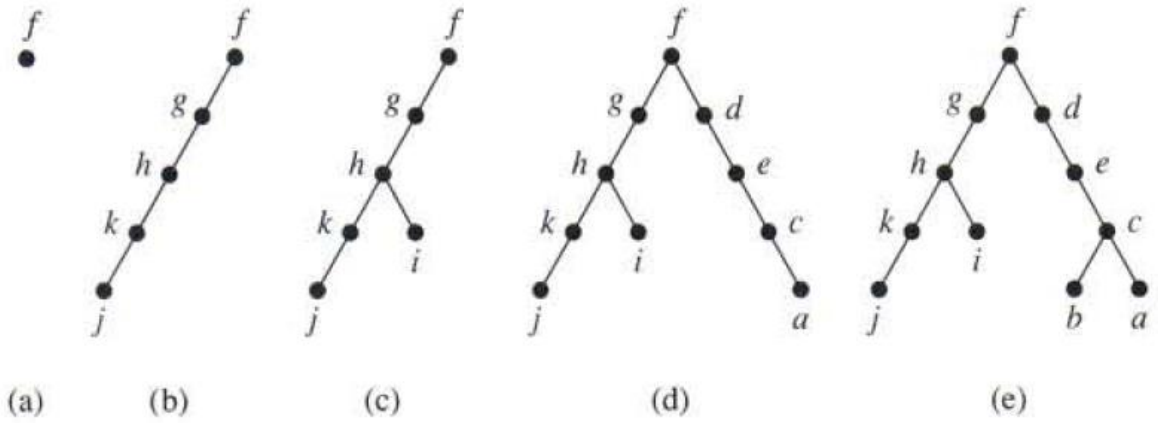


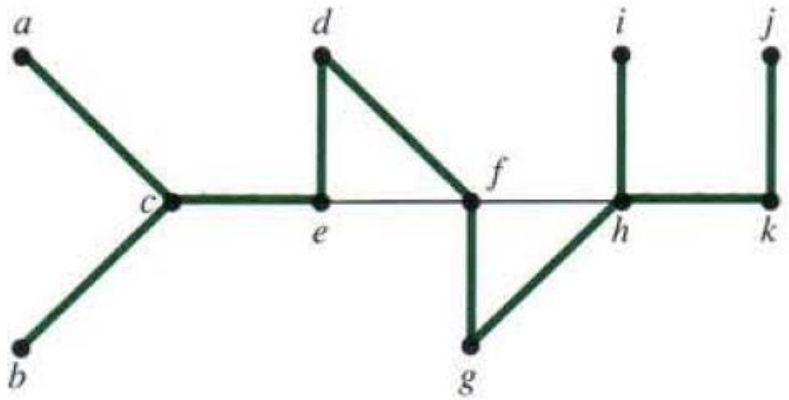
Fig. 4.28 – grafo exemplo_b

Resposta:

Escolhe-se arbitrariamente o vértice f do grafo G, e realiza-se a busca a partir de f.



Após a execução do algoritmo a árvore geradora identificada a partir do grafo G é apresentada na figura abaixo:



Algorítmo Busca_profundidade_Arvore_geradora

Procedure DFS (G: grafo conexo com vértices v1,v2,...vn)

T:= árvore que consiste apenas no vértice V1

Visit(v1)

Procedure visit (v: vértice de G)

FOR **cada vértice w adjacente a v e ainda não em T**

Begin

    Adicione vértice w e aresta {u,w} em T

    Visit(w)

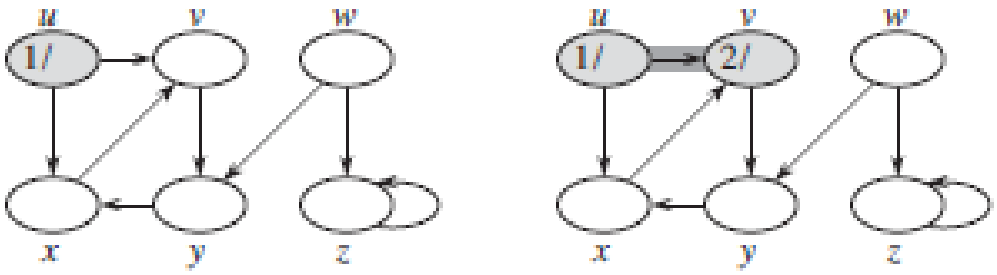
End

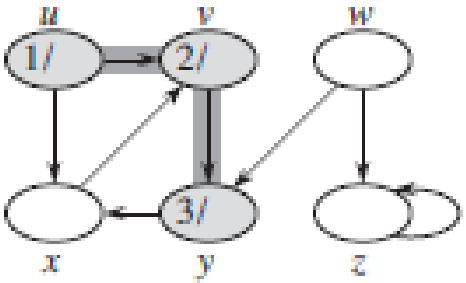
Fig. 4.29 – identificação de uma árvore geradora a partir do algoritmo de busca em profundidade

Exemplo_c: No algoritmo DFS os vértices de um grafo podem ser coloridos durante o processo de busca a fim de indicar os seus estados. Neste exemplo, os vértices são inicialmente brancos, e são transformados na cor cinza quando são visitados (descobertos) pela primeira vez na busca.

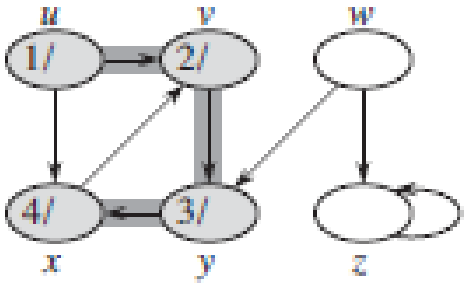
Os vértices que estão no estado da cor cinza são transformados na cor preta quando é finalizada a busca, ou seja, quando a respectiva lista de adjacência tiver sido examinada completamente.

A figura abaixo ilustra passo-a-passo o processo de busca em profundidade alterando as cores dos vértices do grafo, de acordo com a especificação acima.

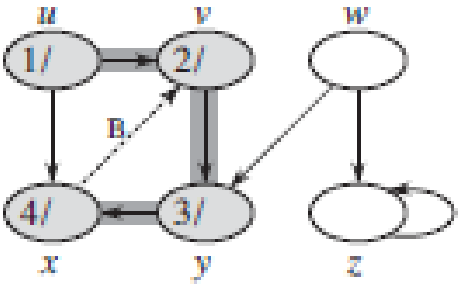



u
v
w
1/
2/
3/
x
y
z

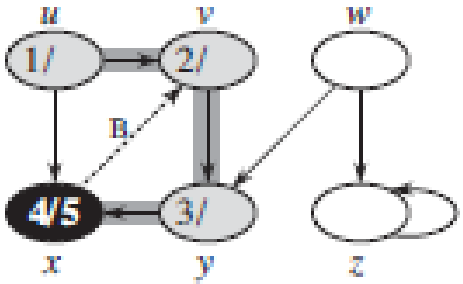
(c)


u
v
w
1/
2/
4/
3/
x
y
z

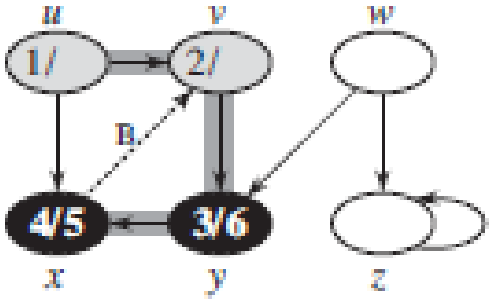
(d)


u
v
w
1/
2/
B
4/
3/
x
y
z

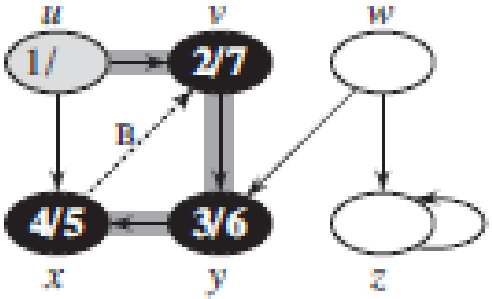
(e)


u
v
w
1/
2/
B
4/5
3/
x
y
z

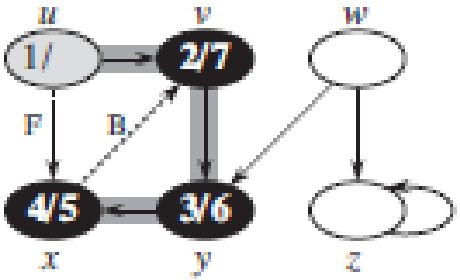
(f)


u
v
w
1/
2/
B
4/5
3/6
x
y
z

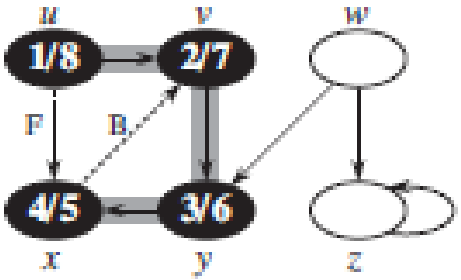
(g)


u
v
w
1/
2/7
B
4/5
3/6
x
y
z

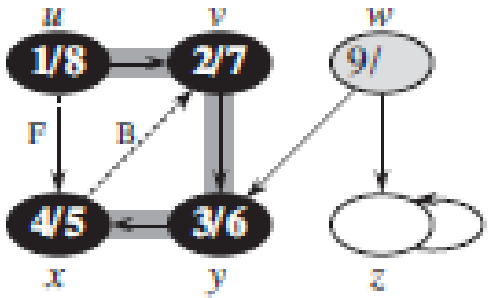
(h)


u
v
w
1/
2/7
F
B
4/5
3/6
x
y
z

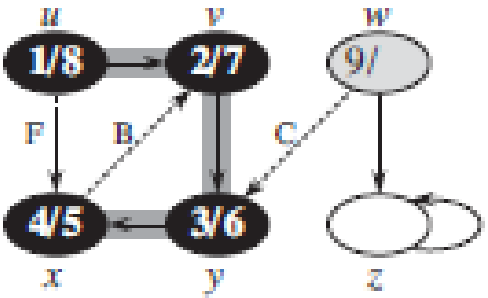
(i)


u
v
w
1/8
2/7
F
B
4/5
3/6
x
y
z

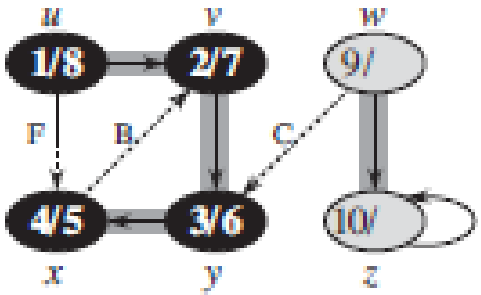
(j)



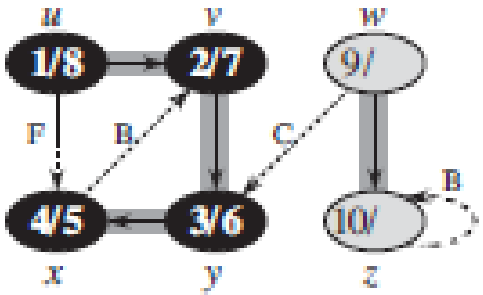
(k)



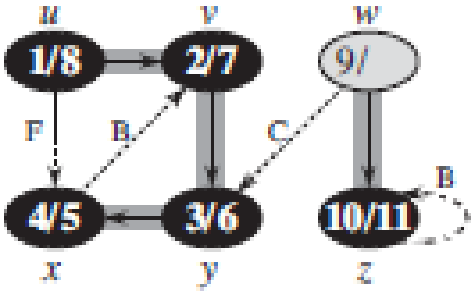
(l)



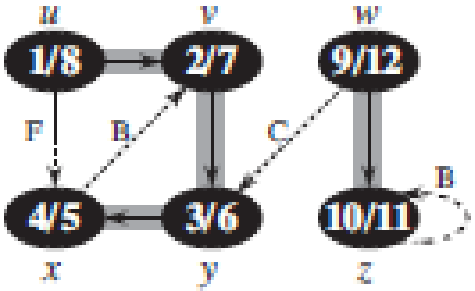
(m)



(n)



(o)



(p)

Exemplo_e: Utilize a busca em profundidade no grafo G abaixo, a partir do nó 4, para encontrar uma **árvore geradora.**

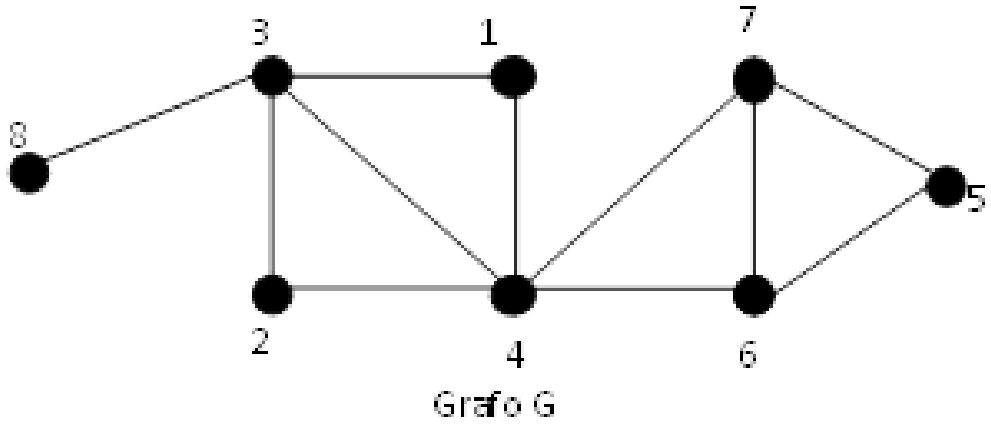
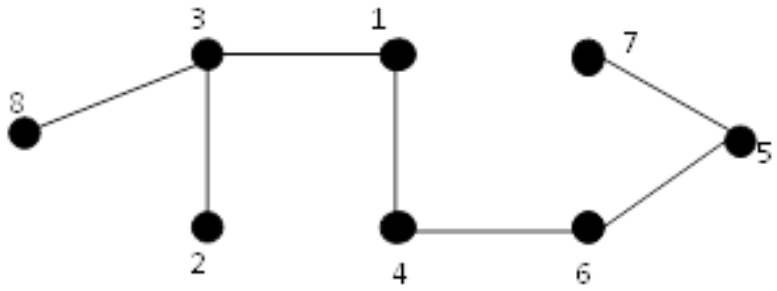


Fig. 4.30 – grafo exemplo_c

Resposta:



A busca em profundidade pode ser usada como base para algoritmos que resolvem muitos **problemas diferentes**:

. Encontrar caminhos e ciclos em um grafo;

. Determinar as componentes conexas de um grafo;

. Determinar vértices de corte de um grafo conexo.

Busca em nível ou largura (BFS): inicia-se também em um nó arbitrário a, e a partir deste nó, procura-se por todos os nós adjacentes. Na sequencia procura-se pelos nós adjacentes a esses, e assim por diante. Ou seja, enquanto for possível, o algoritmo visita todos os nós à mesma distância do nó inicial. Quando não for mais possível o algoritmo caminha no grafo (aprofundamento)

A figura 4.31 apresenta os primeiros nós visitados do mesmo grafo da figura 4.26

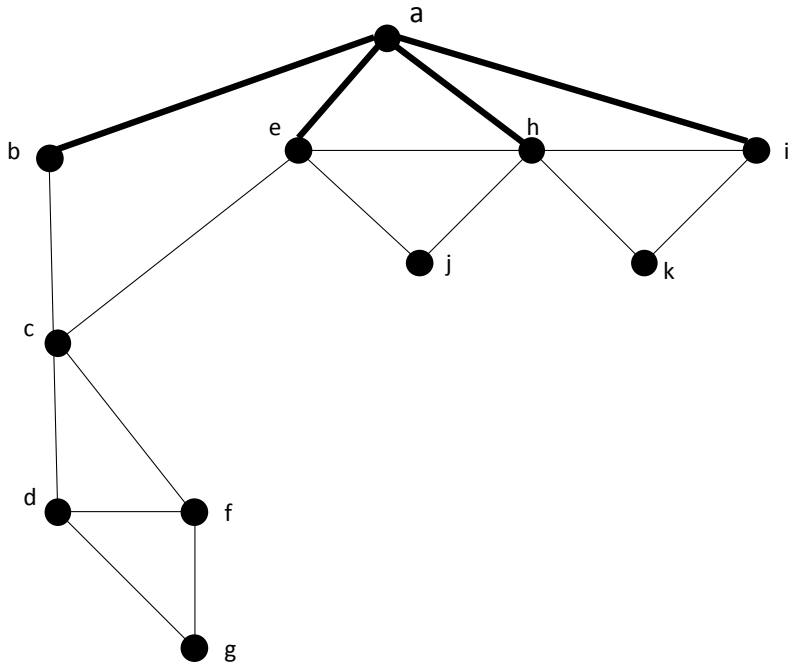


Fig. 4.31 – busca em largura em um grafo

A figura 4.32 apresenta o algoritmo que realiza a busca em nível.

De acordo com este algoritmo, na primeira iteração remove-se o nó a da frente da fila e escreve-se na fila os nós: ***b,e,h,i***

Na próxima iteração do laço **enquanto**, b é o elemento na frente da fila e o laço de **para** procura nós ainda não visitados adjacentes à b. O único que não foi visitado ainda é c, que é

escrito e incluído na fila. Após a remoção de b, a fila contêm: ***e,h,i,c.***

Executando o laço do **enquanto** novamente, o nó e está a frente da fila. A busca aos nós adjacentes a e produz um nó novo, j. Após a retirada de e, a fila contem: ***h,i,c,j.***

Ao procurar os nós adjacentes à h, encontra-se um nó novo que é o k. A fila fica assim: ***i,c,j,k***.

Ao procurar nós adjacentes a i, não é inserido nenhum nó novo na fila.

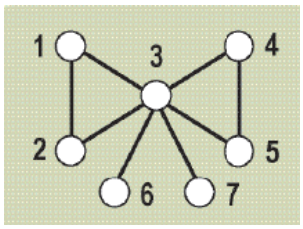
Quando c é o primeiro elemento da fila, uma busca de nós adjacentes a c produz dois novos nós, d e f. Na próxima iteração a fila fica assim: ***j,k,d,f***

Procurando nós adjacentes a j e depois a k não é inserido nenhum nó novo na fila. Quando d está a frente da fila, encontramos um nó novo g e a fila após a remoção de d fica assim: ***f,g.***

Processando f e depois g, não encontramos nenhum nó novo. Após a retirada de g da fila, a fila fica vazia, e portanto, o laço do enquanto termina.

Assim, a lista de nós em ordem de nível a partir de a: ***a,b,e,h,i,c,j,k,d,f,g***.

Exemplo_f: Dado o grafo G da figura abaixo, aplicar o algoritmo de busca em largura.



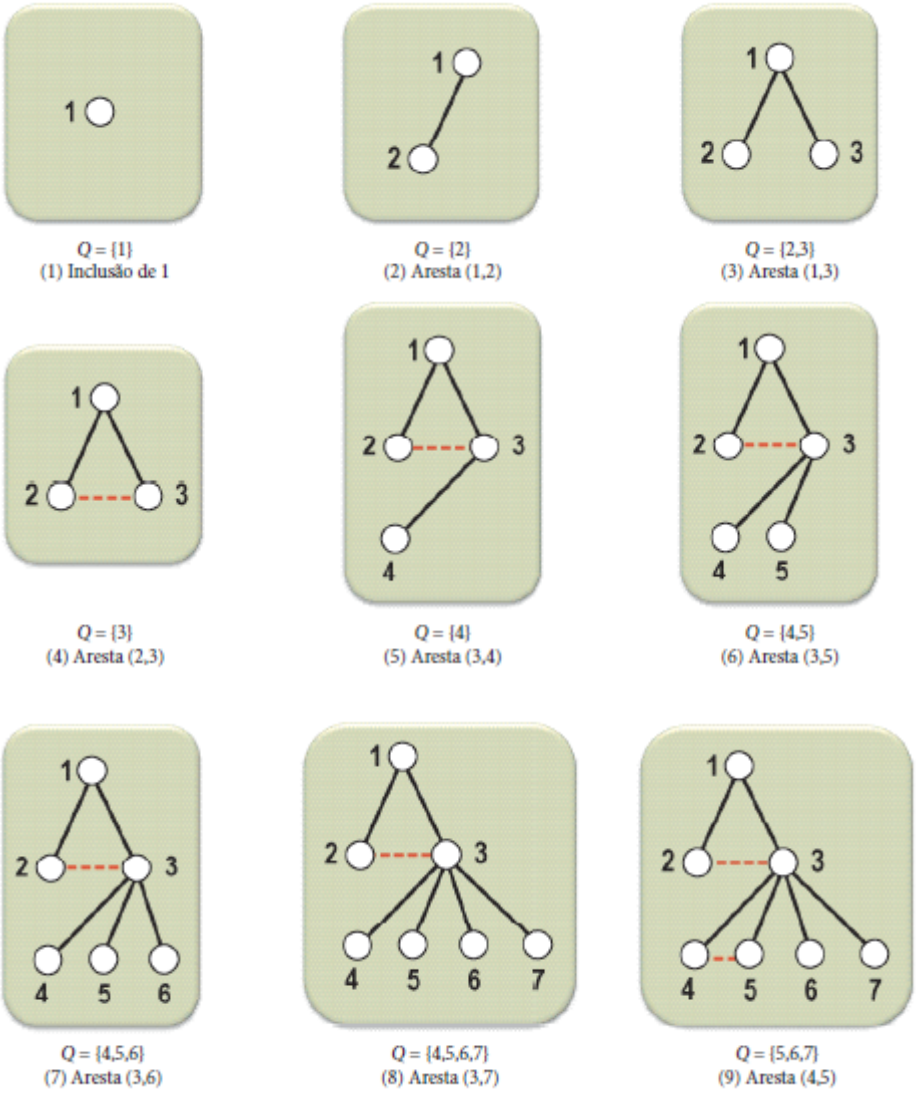


Fig. 4.31 – busca em largura em um grafo

ALGORITMO EmNivel

EmNivel *(grafo G; nó a)*
// Escreve os nós do grafo G em ordem de nível a partir do nó a
// utiliza a estrutura de fila

Variáveis locais*:*
Fila de nós F

Inicialize *F como sendo vazio*
Marque *a como tendo sido visitado*
Escreva*(a)*
Insira(*a,F)*
Enquanto *<F não é vazio>* ***faça***
    Para *<cada nó n adjacente a frente(F)>* ***faça***
        Se *< n não foi visitado>* ***então***
            Marque n como tendo sido visitado
            Escreva n
            Insira (n,F)
        Fim_Se
    Fim_Para
    Retire(F)
Fim_Enquanto
Fim_EmNivel

Fig. 4.32 – Algorítmo para busca em nível

Dado que:
Ni : corresponde ao conjunto de nós pertencentes a camada i=0, 1, 2, ...;
N0 : corresponde ao nó origem;
Ni+1 : corresponde ao conjunto de nós que não fazem parte de uma camada anterior (i) e que possuem uma aresta com algum nó da camada Ni

A busca em nível permite calcular a distância entre o nó N0 e os nós que estão localizados na camada Ni. A distância corresponde ao comprimento do menor caminho simples entre dois nós.

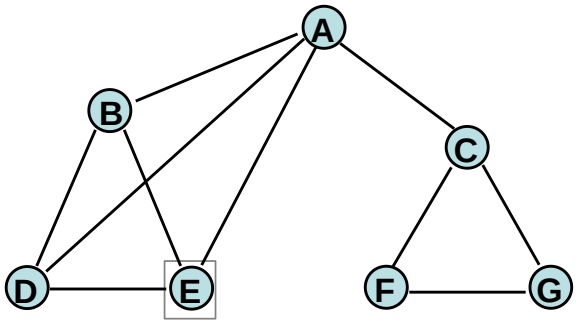
Pode-se também produzir uma **árvore geradora** de um grafo simples pelo uso da busca em nível (largura). Para isto cria-se uma árvore enraizada, e o grafo subjacente a esta árvore enraizada é a árvore geradora.

A figura 4.33 apresenta o algoritmo para criação da arvore geradora a partir da realização de uma busca em nível em uma árvore enraizada. Neste algoritmo, supomos que os vértices do grafo G estão ordenados como: v1,v2,...,vn.

```
Procedure BFS (G: grafo conexo com vértices v1,v2,...,vn)
T:= árvore que consiste apenas no vértice v1;
L:= lista vazia;
Coloque v1 na lista dos vértices  não processados
While L não estiver vazia do
Begin
      Remova o primeiro vértice, v, de L
      For cada vizinho w de v
            If w não estiver em L e não estiver em T then
            Begin
                  Adicione w ao final da lista L
                  Adicione w e a aresta {u,w} a T
            End
End
```

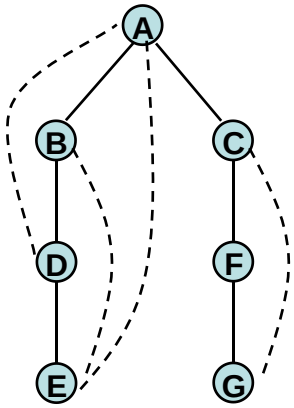
Fig. 4.33 – Algorítmo para geração árvore geradora com busca em nível

Exemplo_a: Dado o grafo G abaixo, determine a árvore de busca em profundidade e em nível, a partir do nó A.

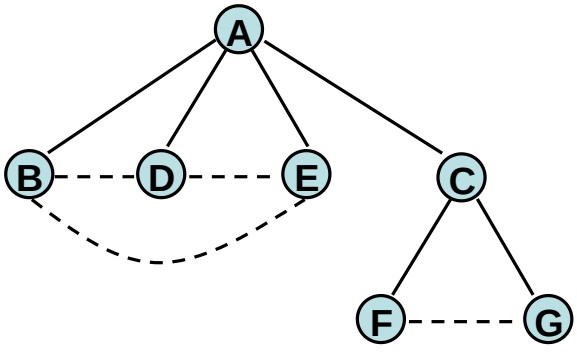


Resposta:

Busca em profundidade:



Busca em nível:



Exemplo_b: Dado o grafo G abaixo use a busca em nível para encontrar uma árvore geradora.

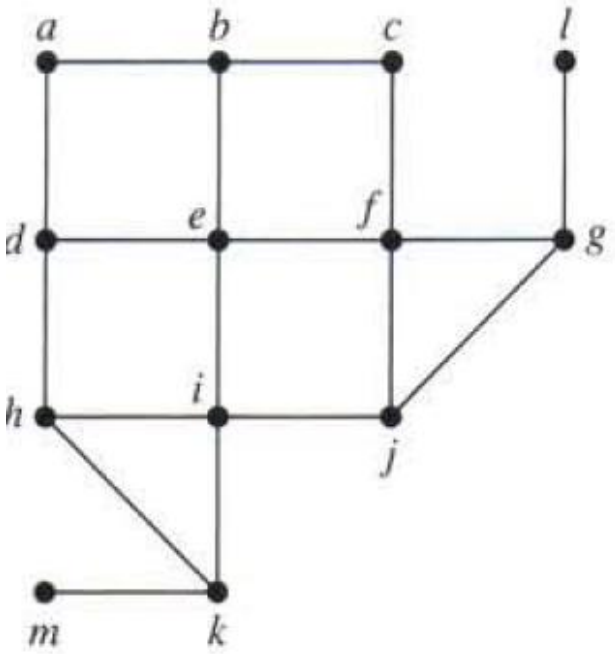


Fig. 4.34 – Grafo exemplo_a

Resposta: escolhe-se o vértice e para ser a raíz. A árvore geradora é apresentada na figura abaixo.

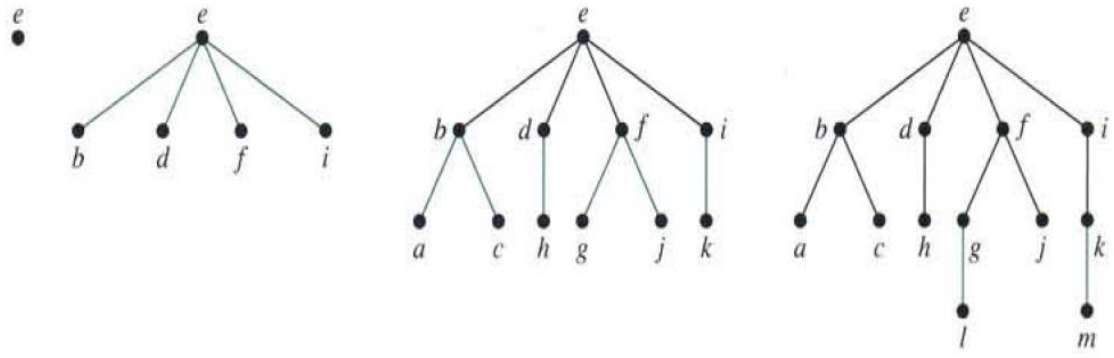
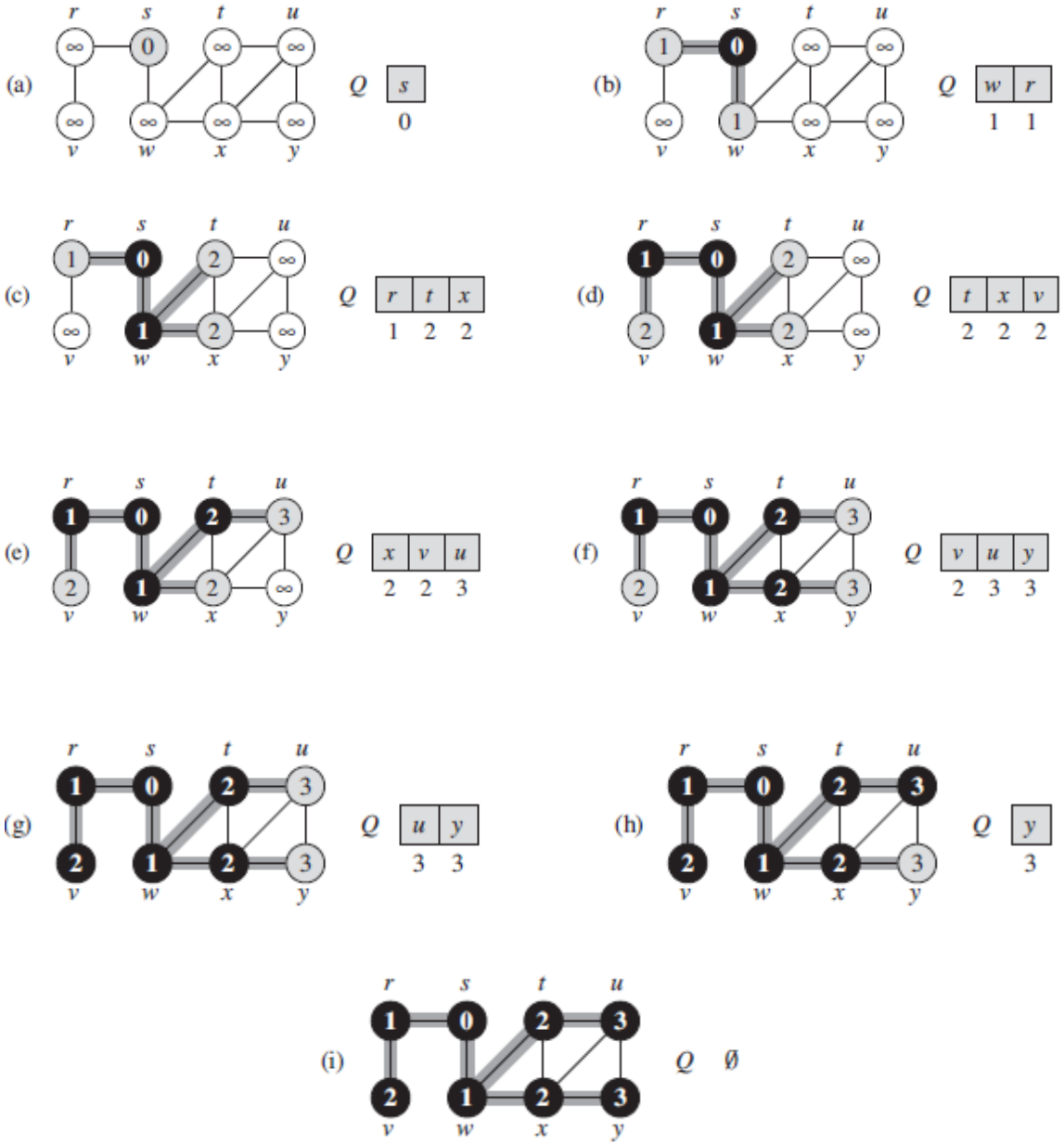


Fig. 4.35 – resposta exemplo_a

Tal como no algoritmo DFS, também pode-se atribuir cores aos vértices de um grafo quando se aplica o algoritmo BFS. A seguência ilustra passo-a-passo o caminhamento no grafo G utilizando o algoritmo BFS.


(a)
r s t u
∞ 0 ∞ ∞
∞ ∞ ∞ ∞
v w x y
Q
s
0
(b)
r s t u
1 0 ∞ ∞
∞ 1 ∞ ∞
v w x y
Q
w r
1 1
(c)
r s t u
1 0 2 ∞
∞ 1 2 ∞
v w x y
Q
r t x
1 2 2
(d)
r s t u
1 0 2 ∞
2 1 2 ∞
v w x y
Q
t x v
2 2 2
(e)
r s t u
1 0 2 3
2 1 2 ∞
v w x y
Q
x v u
2 2 3
(f)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q
v u y
2 3 3
(g)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q
u y
3 3
(h)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q
y
3
(i)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q ∅

Exemplo_c: Uma maneira de procurar sistematicamente (exaustivamente) por uma solução consiste em usar uma árvore de decisão, em que cada vértice interno representa uma decisão e cada folha uma solução possível. A sequencia de decisões pode ser representada por um caminho na árvore de decisão.

Como exemplo de aplicação, pode-se utilizar a busca para decidir se um grafo pode ser colorido usando **n cores**. Considere a figura abaixo e o objetivo consiste em colorir o grafo com n cores, e identificar qual o valor de n.

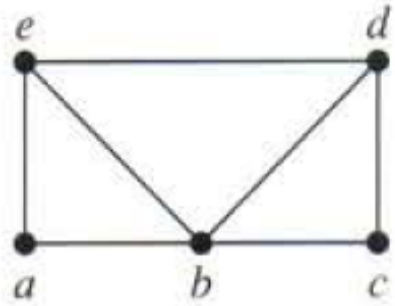


Fig. 4.36 – grafo exemplo_b

Resposta: Inicia-se o algoritmo utilizando o vértice a e associando a ele a cor vermelho. A árvore da figura abaixo ilustra, a partir da raiz a, a associação de uma cor a um vértice. Neste procedimento, o vermelho será usado primeiro, depois o azul e, finalmente, o verde.

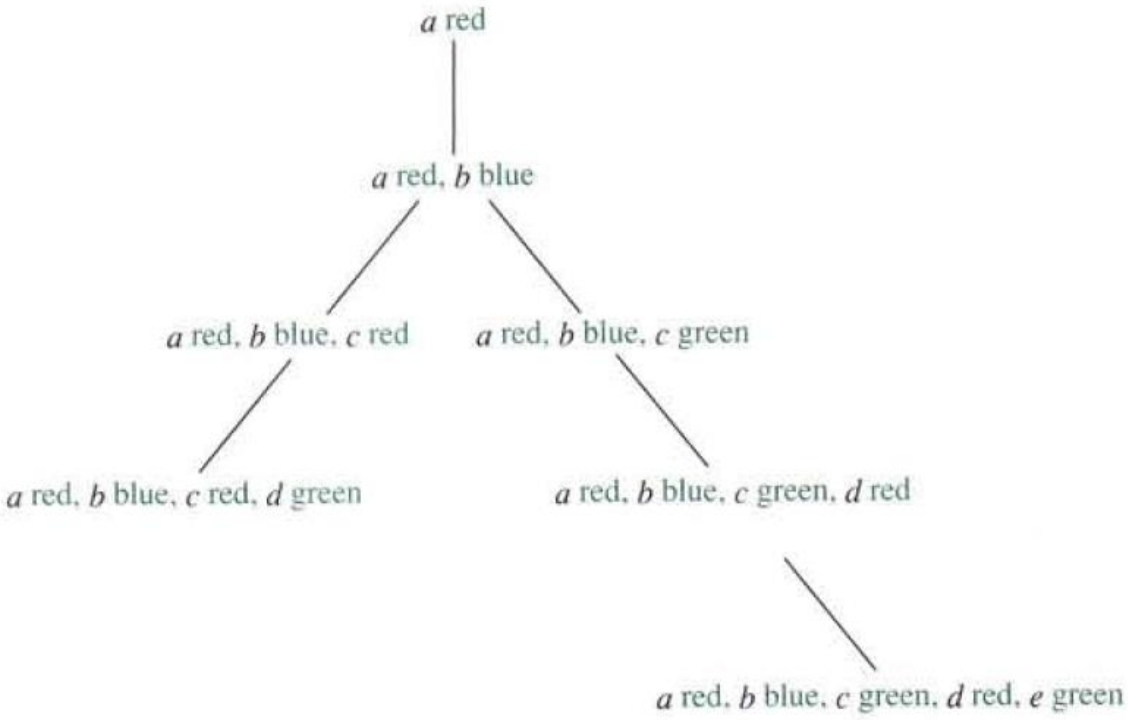


Fig. 4.37 – resposta exemplo_b

Exemplo_d: **Problema das n rainhas**. Este problema pergunta como n rainhas podem ser colocadas em um tabuleiro de xadrez NxN, de modo que nenhum par de rainhas possa atacar uma a outra. Como a busca sistemática pode ser utilizada para resolver este problema ?

Resposta: Deve-se encontrar N posições no tabuleiro NxN, de modo que duas destas posições nunca estejam na mesma linha, na mesma coluna, ou na mesma diagonal.

. Inicia-se com o tabuleiro vazio;

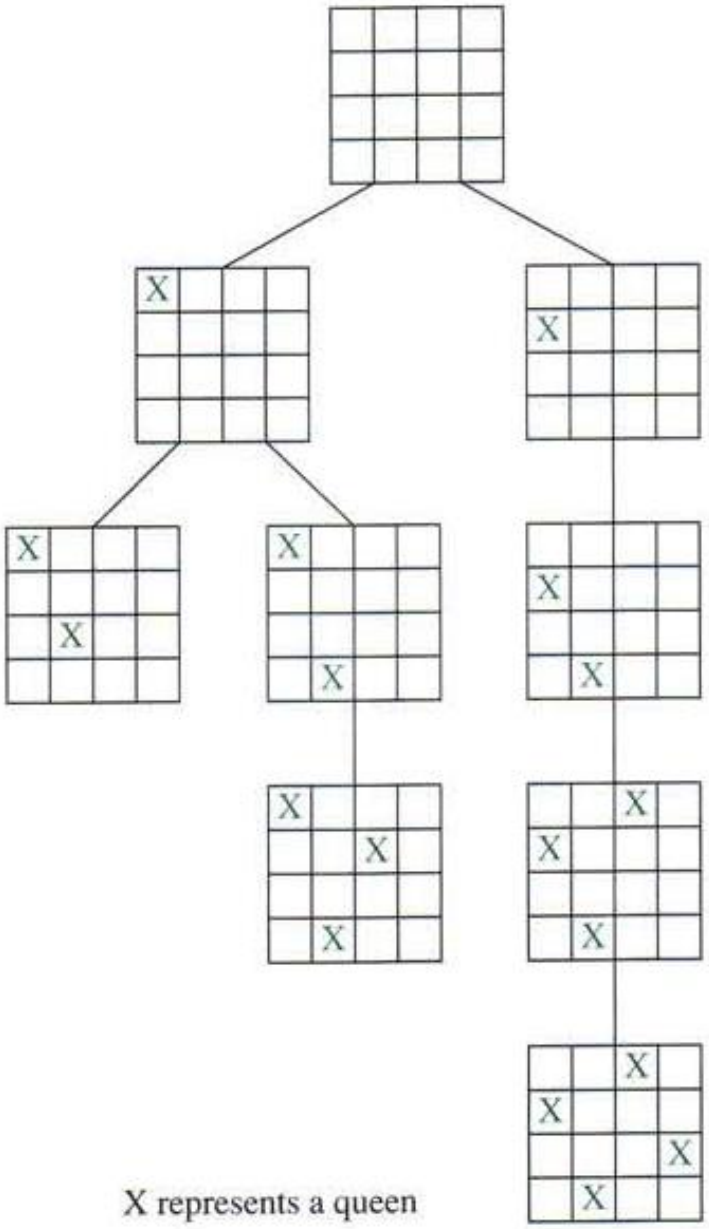
. Preenche-se a primeira posição (coluna K) da matriz (1,1);

. Na próxima etapa busca-se por uma posição (coluna K+1) que não conflite com a(s) posições já ocupada(s) por uma ou mais rainhas no tabuleiro;

. Se for impossível encontrar uma posição para colocar a rainha na coluna (K+1), retroceder para a colocação da rainha na coluna K e coloque esta rainha na próxima linha permitida nesta coluna, se tal linha existir. Se não existir tal linha, retroceda mais.

Apresentar o algoritmo para uma matriz 6x6.

A figura abaixo apresenta uma busca sistemática, em uma árvore enraizada, para o problema particular com 4 rainhas. Verifique que não encontra-se nenhuma solução quando uma rainha é colocada na primeira linha da primeira coluna. Assim, retrocede-se para o tabuleiro vazio e coloca-se uma rainha na segunda linha da primeira coluna. Assim, encontra-se a solução para 4 rainhas.



X represents a queen

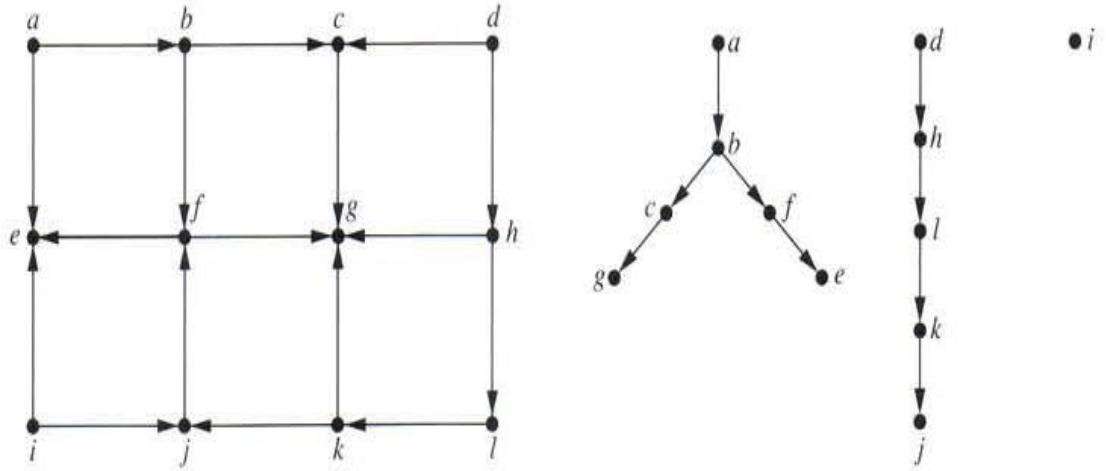
Busca em profundidade em grafos orientados

Na busca em profundidade em grafos direcionados não necessariamente obtêm-se como saída uma árvore geradora, mas em vez disso, uma floresta geradora.

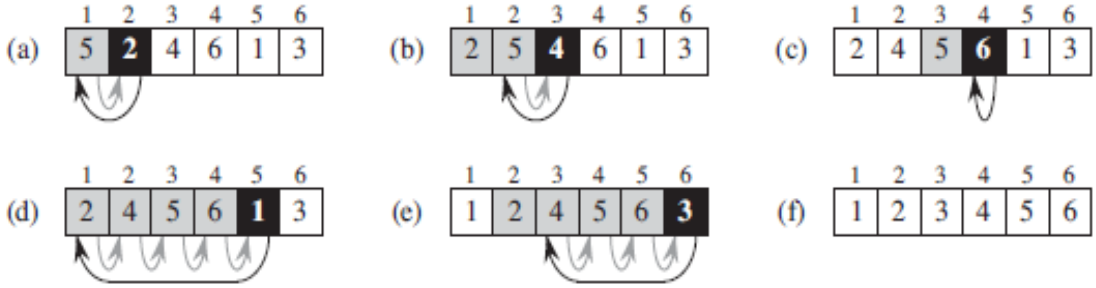
Na busca em profundidade em grafos orientados adiciona-se uma aresta apenas quando ela estiver orientada para longe do vértice que está sendo visitado, e para um vértice que ainda não foi adicionado.

Se em um determinado estágio da execução do algoritmo descobre-se que não existe nenhuma aresta começando em um vértice já adicionado para um que ainda não tenha sido adicionado, o próximo vértice adicionado pelo algoritmo se torna a raiz de uma nova árvore na floresta geradora.

Exemplo_a: Dada o grafo abaixo, iniciando a busca pelo nó a, obtêm-se a floresta geradora ao lado direito do grafo.



Análise de Algorítmos: **Dada sequencia de números qualquer, colocá-los em ordem crescente fazendo o sort (iterativo).**



INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$

Tempo de execução: **Função quadrática**

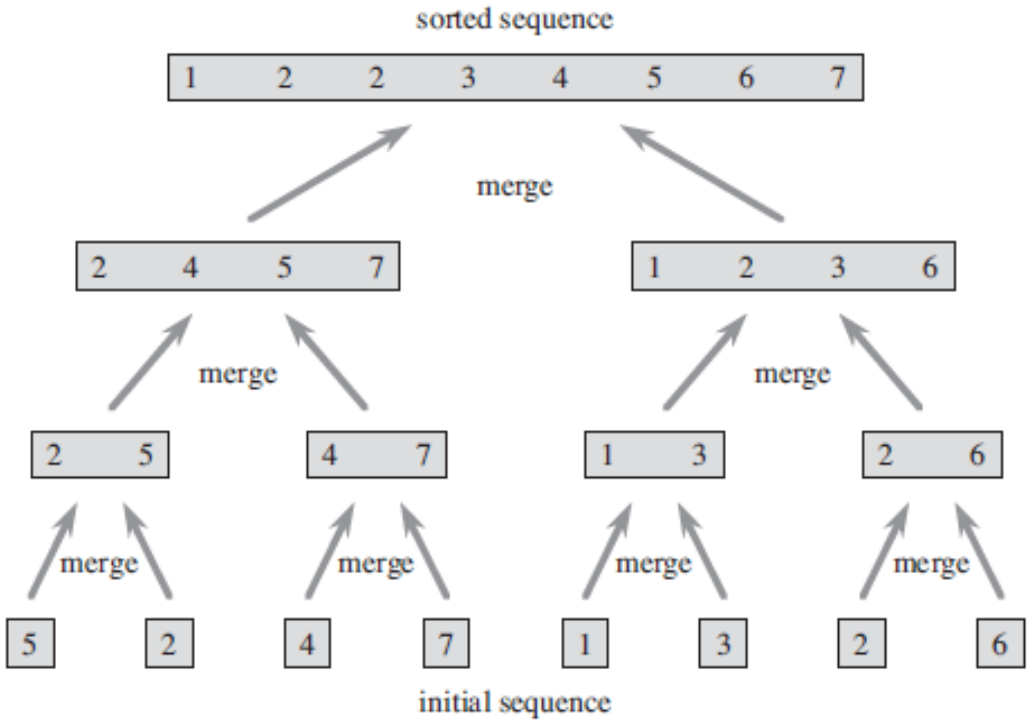
$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)\,.$$

$$an^2 + bn + c$$

Taxa de crescimento: relacionada à eficiência do algorítmo

$$\Theta(n^2)$$

Análise de Algorítmos: Dada uma sequencia de números qualquer, colocá-los em ordem crescente através do algorítmo "divide-and-conquer" (recursividade).




```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 . \end{cases}$$

$$\therefore T(n) = \Theta(n \lg n).$$