树/Trees

11.1 树的概念/Introduction of Trees

11.2 树的应用/Applications of Trees

11.3 树的遍历/Tree Traversal

11.4 生成树Spanning Trees

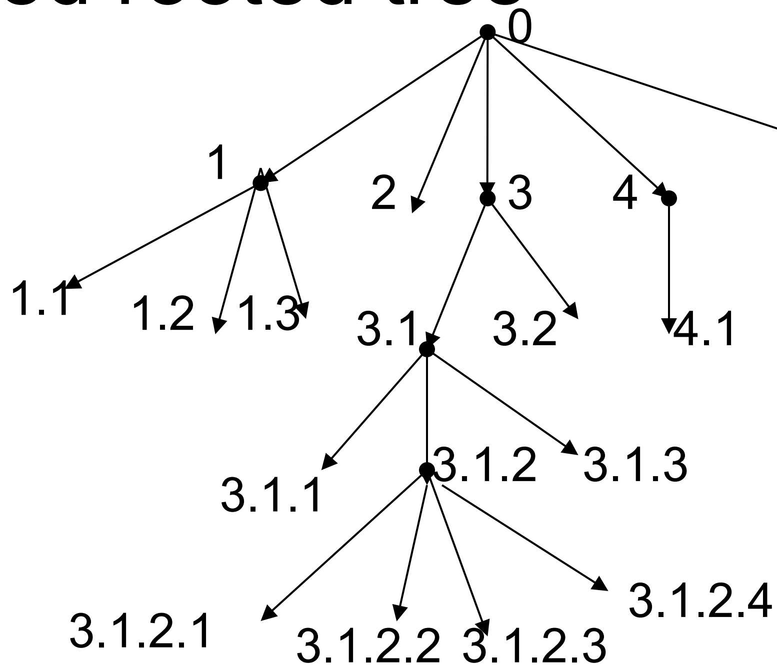
11.5 最小生成树 minimum Spanning Trees

11.3 Universal address systems

- Label all the vertices
  - Label the root with the integer 0. Then label its k children from left to right with $1, 2, \ldots k$.
  - For each vertex v at level n with label A, label its k_v children, as they are drawn from left to right, with $A.1, A.2, \ldots A.\ k_v$.

Tree Traversal

- Universal address systems
- ordered rooted tree



0<1<1.1<1.2<1.3<2<3<3.1<3.1.1<3.1.2<3.1.2.1<3.1.2.2<3.1.2.3<3.1.2.4<3.1.3<3.2<4<4.1<5<5.1<5.1.1<5.2<5.3

Tree Traversal
遍历算法

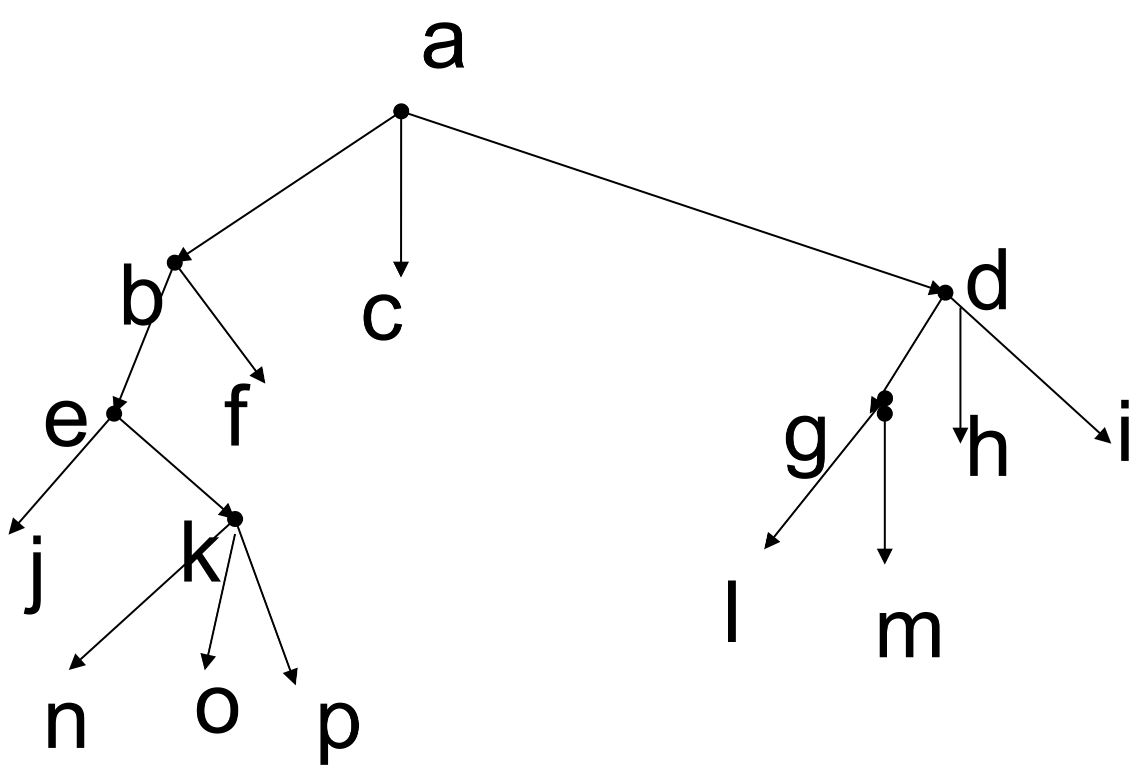
- Visiting:performing appropriate tasks at a vertex.

  （Searching or tree search :the process of visiting each vertex of a tree in some specific order.(walking or traversing（遍历））。
- Left subtree（左子树）:T(v_L)；Right subtree （右子树） :T(v_R)
- Procedures for systematically visiting every vertex of an ordered rooted tree are called traversal algorithms.
  - Preorder traversal
  - Inorder traversal
  - Postorder traversal

Traversal Algorithms
前序遍历

- Definition 1:Let T be an ordered rooted tree with root r. If T consists only of r, then r is the **preorder traversal** of T. Otherwise, suppose that $T_1, T_2, \ldots, T_n$ are subtrees at r from left to right in T. The preorder traversal begins by visiting r. It continues by traversing T_1 in preorder, then T_2 in preorder, and so on, until T_n is traversed in preorder.

preorder traversal

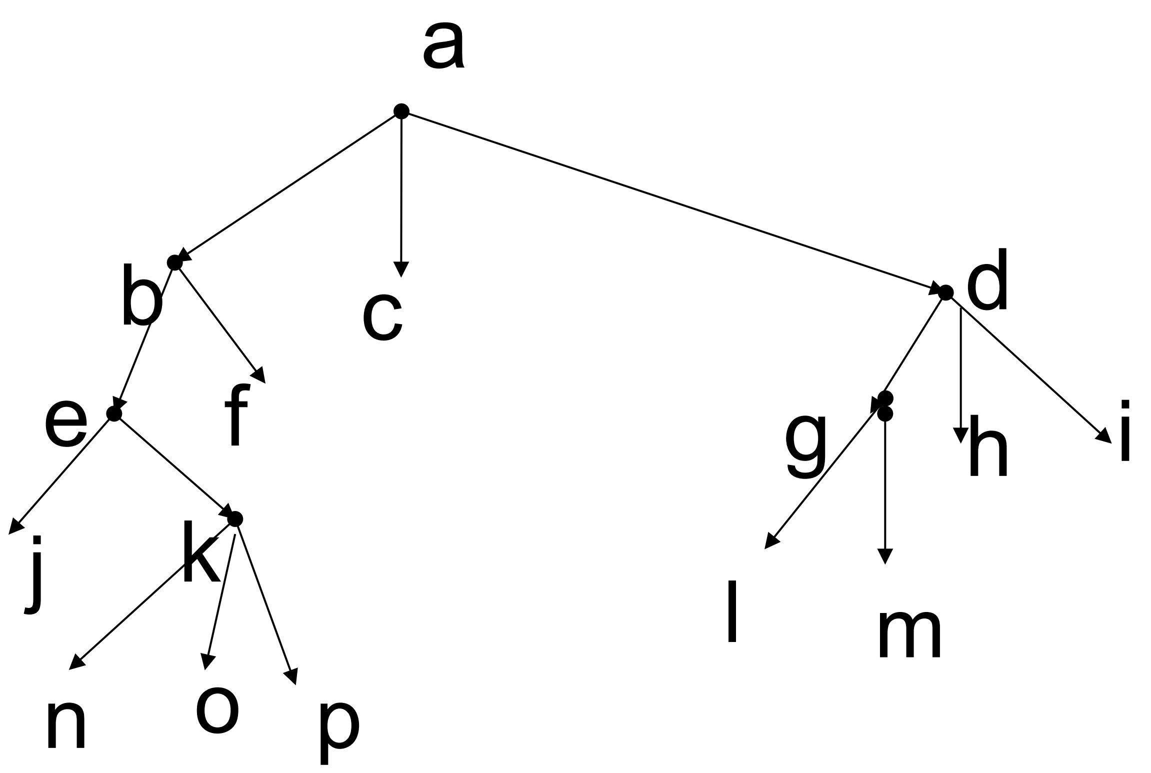


a）前序：a,b,e,j,k,n,o,p,f, c,d,g,l,m,h,i

Traversal Algorithms
中序遍历

- Definition 2:Let T be an ordered rooted tree with root r. If T consists only of r, then r is the **inorder traversal** of T. Otherwise, suppose that $T_1, T_2, \ldots, T_n$ are subtrees at r from left to right in T. The inorder traversal begins by traversing T_1 in inorder , then visiting r. It continues by traversing T_2 in inorder, and so on, until T_n is traversed in inorder.

inorder traversal

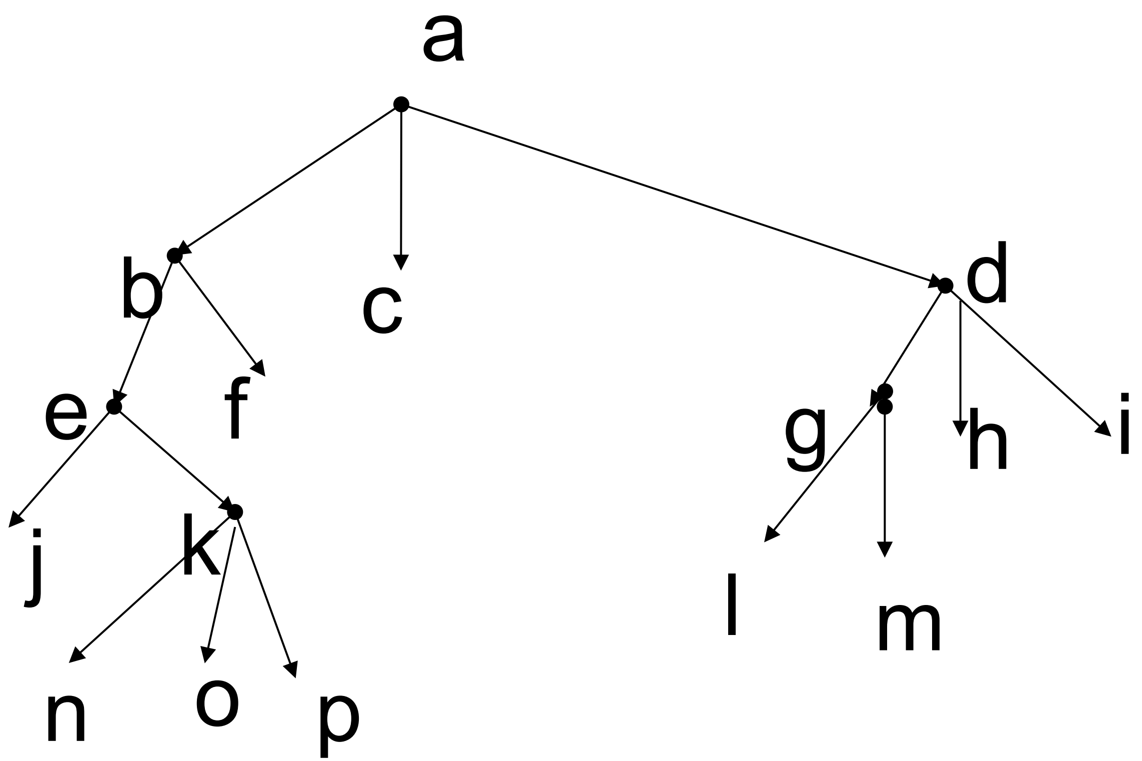


b）中序：j,e,n,k,o,p,b,f,a,c,l,g,m,d,h,i

Traversal Algorithms
后序遍历

- Definition 3:Let T be an ordered rooted tree with root r. If T consists only of r, then r is the **postorder traversal** of T. Otherwise, suppose that $T_1, T_2, \ldots, T_n$ are subtrees at r from left to right in T. The postorder traversal begins by traversing T_1 in postorder. then T_2 in postorder, and so on, then T_n is traversed in postorder,and ends by visiting r.

postorder traversal



c） 后序：j,n,o,p,k,e,f,b,c,l,m,g,h,i ,d,a

Preorder （前序）

- Algorithm preorder
  - Step 1 visit v.
  - Step 2 if v_L exists,then apply this algorithm to $(v_L, T(v_L))$.
  - Step 3 if v_R exists,then apply this algorithm to $(v_R, T(v_R))$.
  - end of algorithm

Inorder （中序）

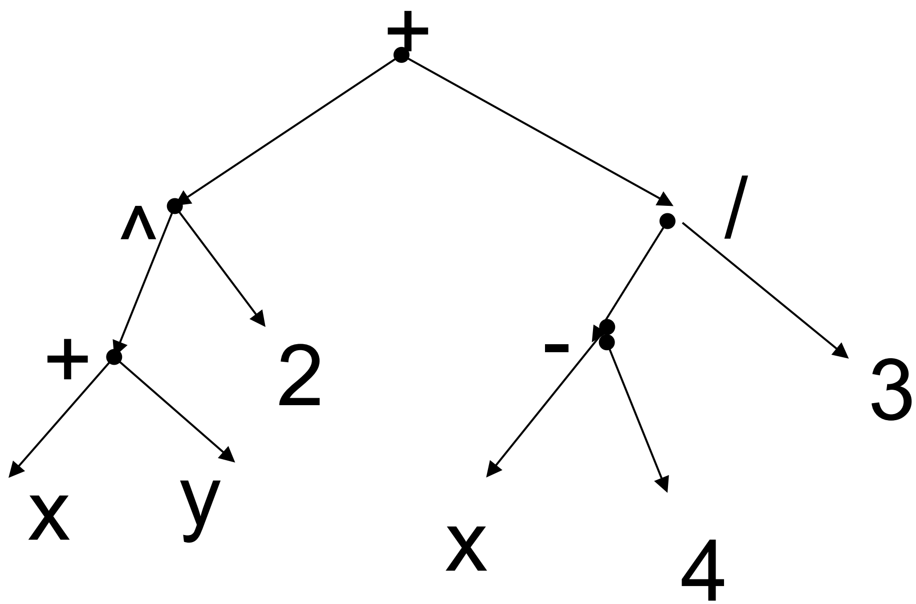
- Algorithm inorder
  - Step 1 search the left subtree($T(v_L)$, v_L), if it exists.
  - Step 2   visit the root v.
  - Step 3 search the right subtree($T(v_R)$, v_R), if it exists.
  - end of algorithm

Postorder（后序）

- Algorithm postorder
  - Step 1 search the left subtree($T(v_L)$, v_L), if it exists.
  - Step 2. search the right subtree($T(v_R)$, v_R), if it exists.
  - Step 3. visit the root v.
  - end of algorithm

Infix,Prefix,and Postfix Notation
中缀、前缀、后缀表示

A Binary Three Representing((x+y)^2+((x-4)/3)



a ）Prefix：+^+xy2/-x43

b ）Infix：x+y^2+x-4/3

c ）Postfix：xy+2^x4-3/+

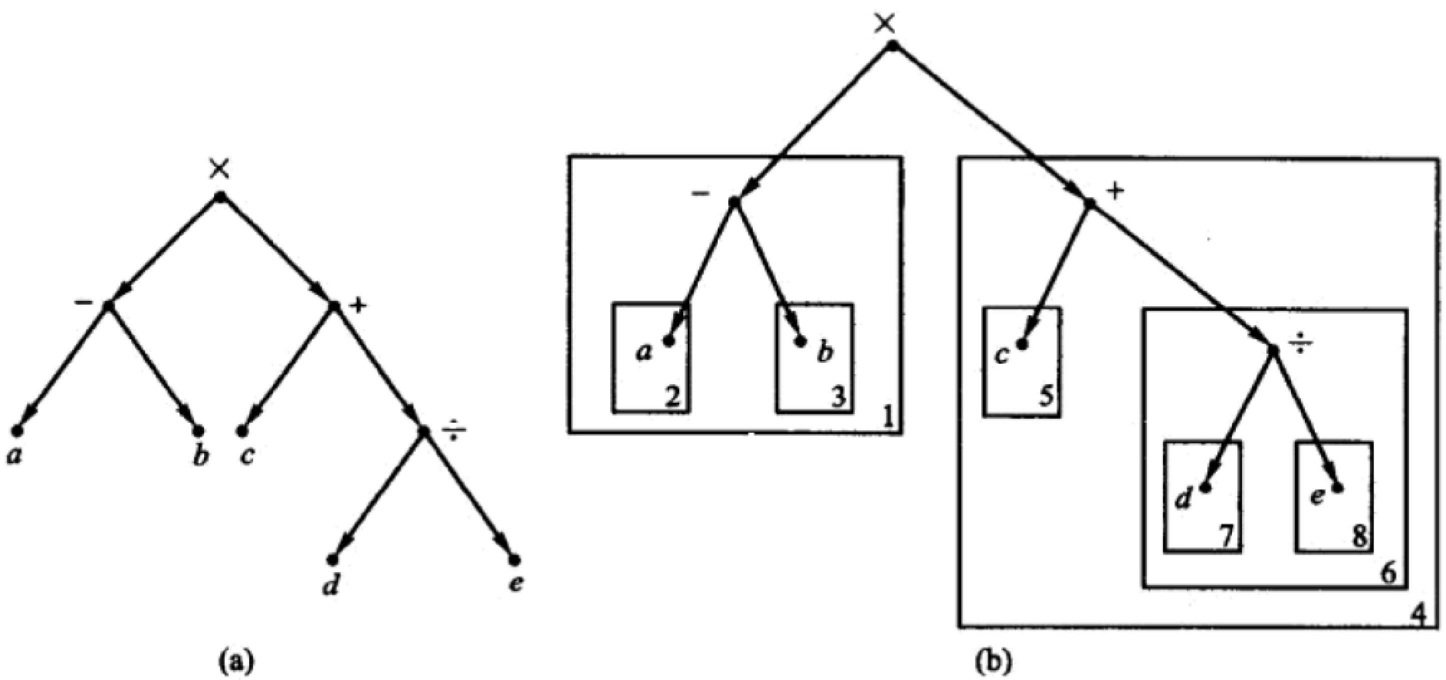
Prefix(Polish notation)
（前缀表达式（波兰式））

- Unambiguously without parentheses
- from left to right Fxy
- operation symbol precedes the argument

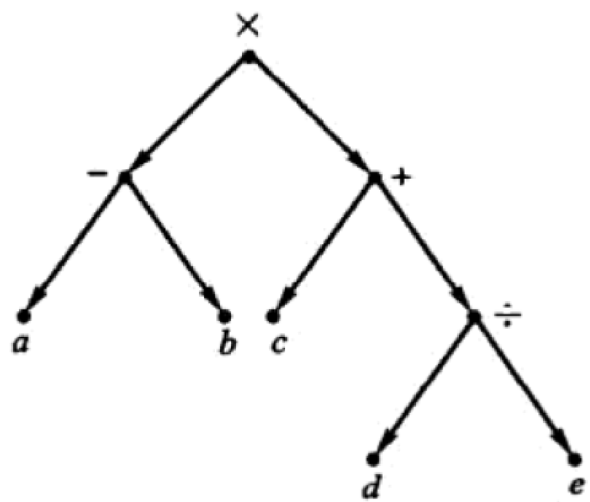
1. $\times \underline{-6\ 4} + 5 \div 2\ 2$
2. $\times 2 + 5\ \underline{\div 2\ 2}$
3. $\times 2\ \underline{+ 5\ 1}$
4. $\underline{\times 2\ 6}$
5. $\underline{1\ 2}$

Since the first string of the correct type is -64 and 6-4=2
Replacing $\div$ 22 by 2 $\div$2 or 1
Replacing+51 by 5 +1 or 6
Replacing×26 by 2 × 6

前缀表示法的执行顺序



Postfix（reverse-Polish notation）



- ab-cde/+*
- a=2,b=1,c=3
- d=4, e=2

1. 2 1–3 4 2÷+×
2. 1 3 4 2÷+×，用2–1或1代替2 1–。
3. 1 3 2+×，用4÷2或2代替4 2÷。
4. 1 5×，用3+2或5代替3 2+。
5. 5，用1×5或5代替1 5×。

Example 6

- 前缀和后缀计算对比

$$+\ -\ *\ 2\ 3\ 5\ /\ \underbrace{\uparrow\ 2\ 3}_{2\uparrow 3=8}\ 4$$

$$+\ -\ *\ 2\ 3\ 5\ \underbrace{/\ 8\ 4}_{8/4=2}$$

$$+\ -\ \underbrace{*\ 2\ 3}_{2*3=6}\ 5\ 2$$

$$+\ \underbrace{-\ 6\ 5}_{6-5=1}\ 2$$

$$\underbrace{+\ 1\ 2}_{1+2=3}$$

Value of expression: 3

FIGURE 12 Evaluating a Prefix Expression.

$$7\ \underbrace{2\ 3\ *}_{2*3=6}\ -\ 4\ \uparrow\ 9\ 3\ /\ +$$

$$\underbrace{7\ 6\ -}_{7-6=1}\ 4\ \uparrow\ 9\ 3\ /\ +$$

$$\underbrace{1\ 4\ \uparrow}_{1^4=1}\ 9\ 3\ /\ +$$

$$1\ \underbrace{9\ 3\ /}_{9/3=3}\ +$$

$$\underbrace{1\ 3\ +}_{1+3=4}$$

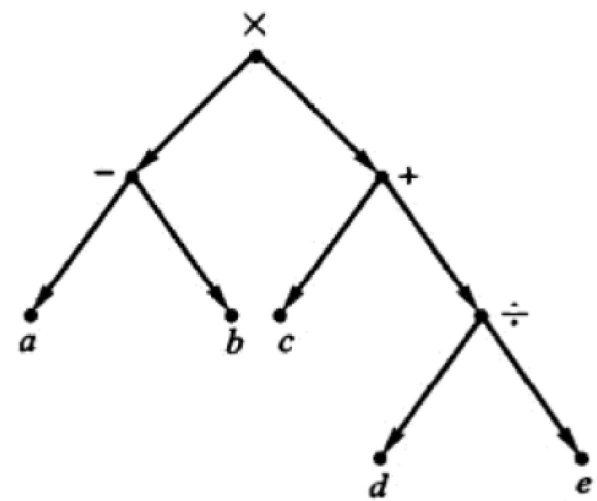
Value of expression: 4

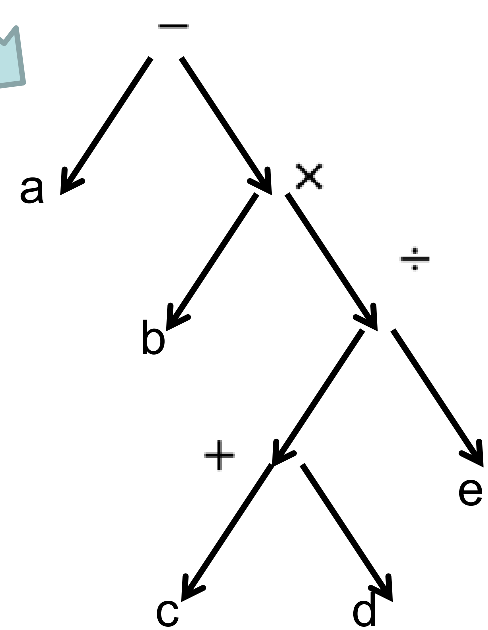
FIGURE 13 Evaluating a Postfix Expression.

Infix notation（中缀表示法）

- ambiguous

$$a - b \times c + d \div e$$





Example 10

Find the ordered rooted tree representing the compound proposition $(\neg(p \wedge q)) \leftrightarrow (\neg p \vee \neg q)$. Then use this rooted tree to find the prefix, postfix, and infix forms of this expression.

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

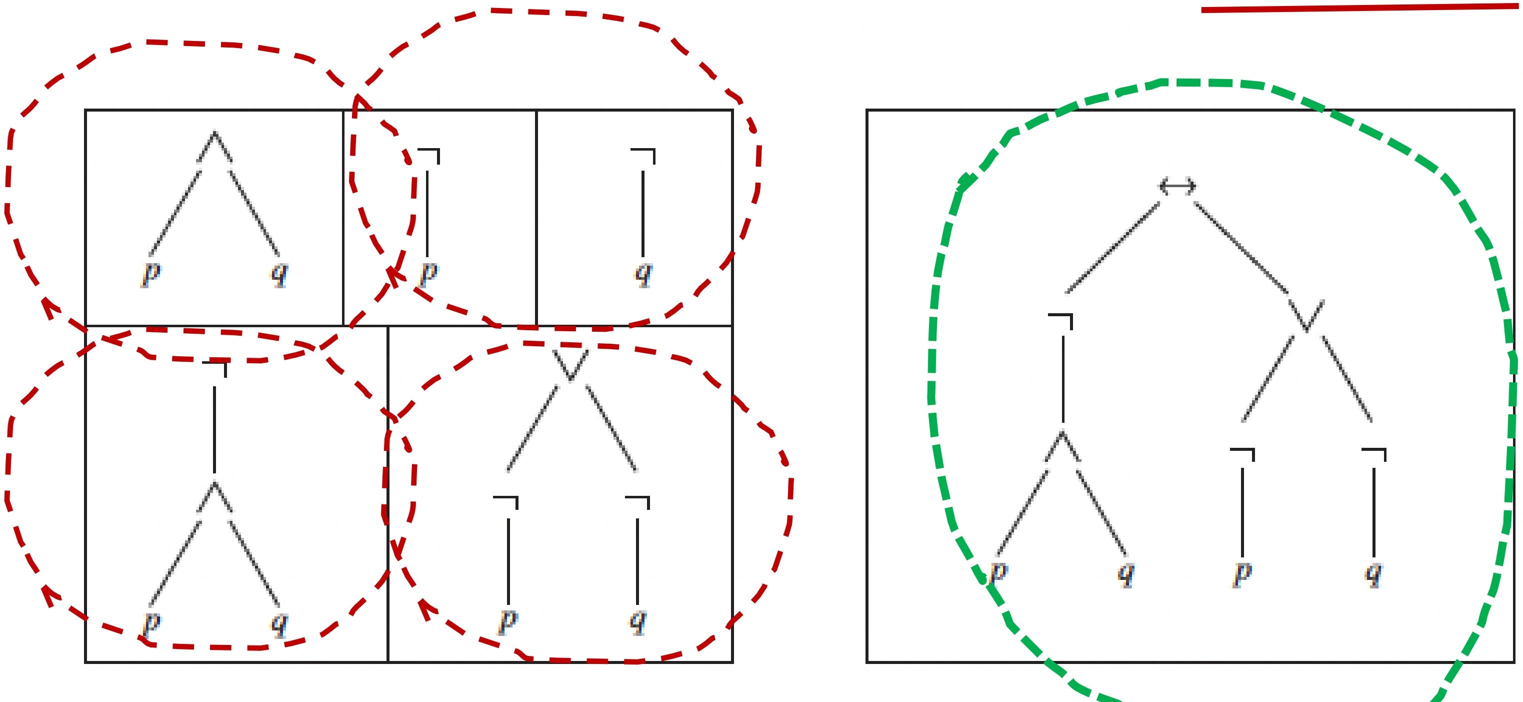


FIGURE 14 Constructing the rooted tree for a compound proposition.

$\leftrightarrow \neg \wedge pq \vee \neg p \neg q$, $pq \wedge \neg p \neg q \neg \vee \leftrightarrow$, and $(\neg(p \wedge q)) \leftrightarrow ((\neg p) \vee (\neg q))$,

Homework

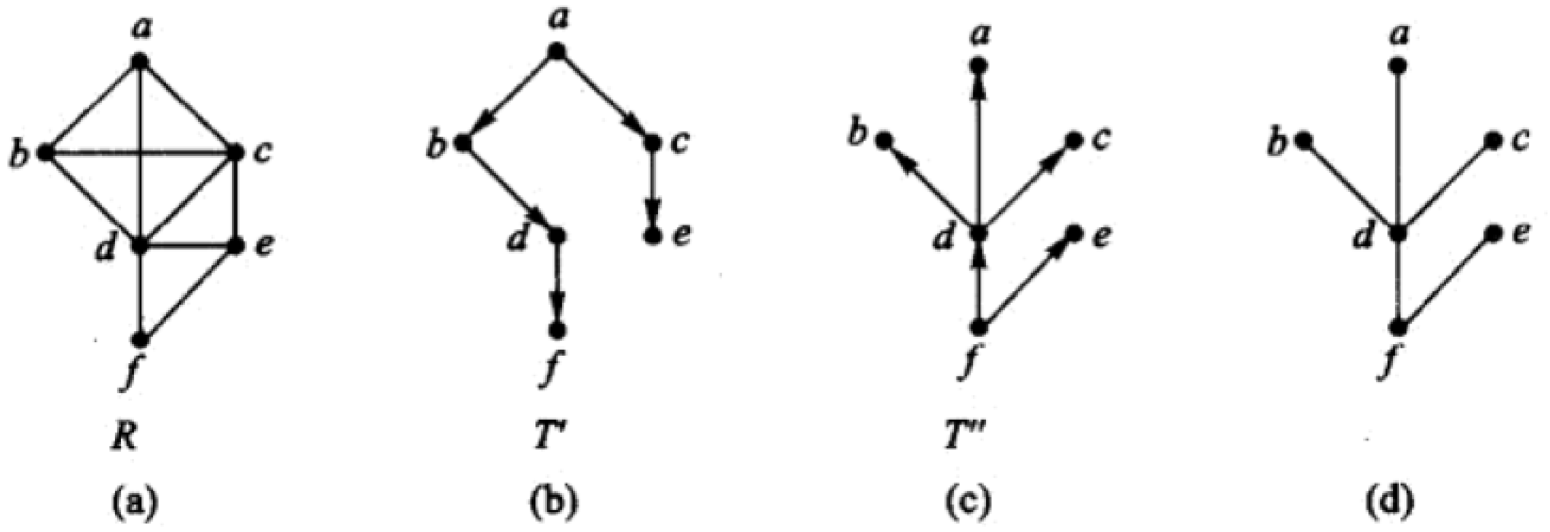
- §11.3
  - 6, 18, 22

11.4 Spanning Trees （生成树）

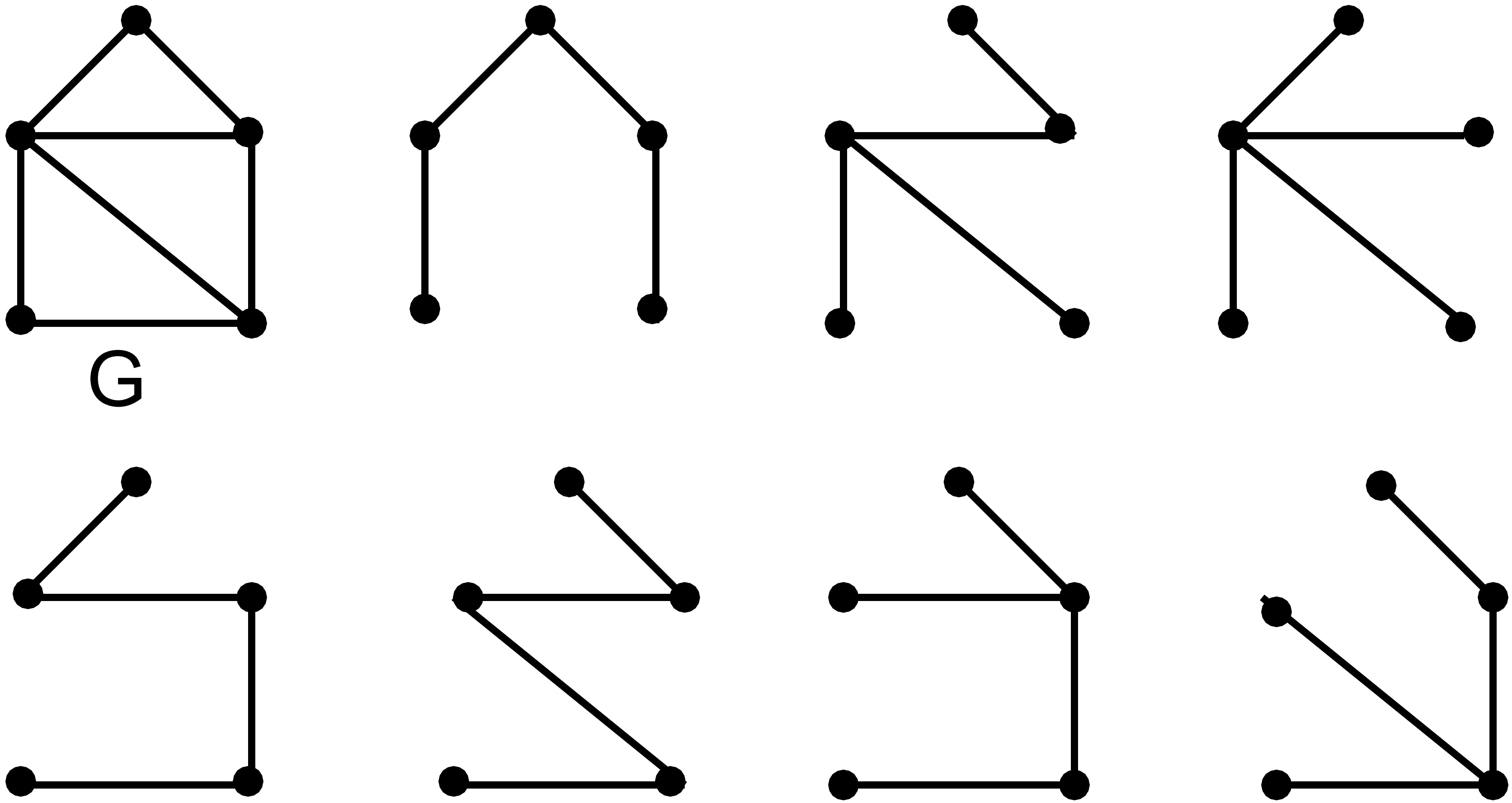
- Let G be a simple graph.A spanning tree of G is a subgraph of G that is a tree containing every vertex of G.
  - T is a tree with exactly the same vertices as G
  - T can be obtained from G by deleting some edges of G.

Example 3

- Spanning trees are not unique.



Spanning trees of G?



- **Theorem 1**：A simple graph is connected if and only if it has a spanning tree.

- Example：IP multicasting

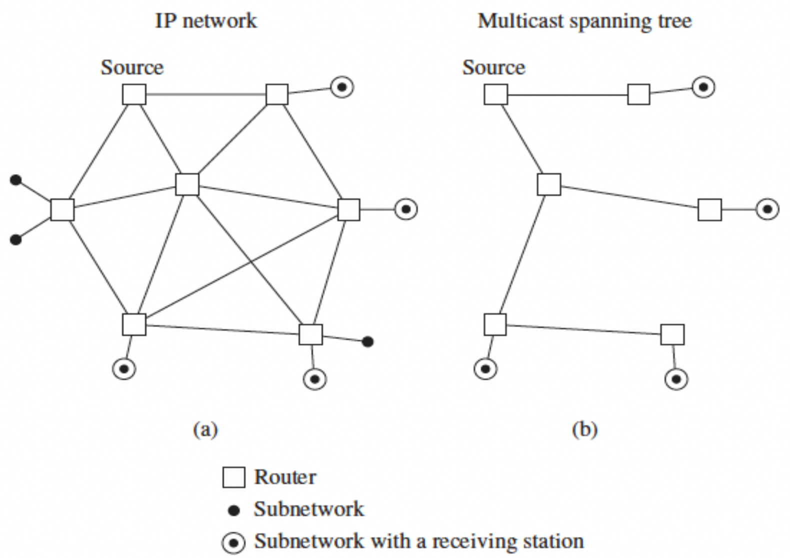
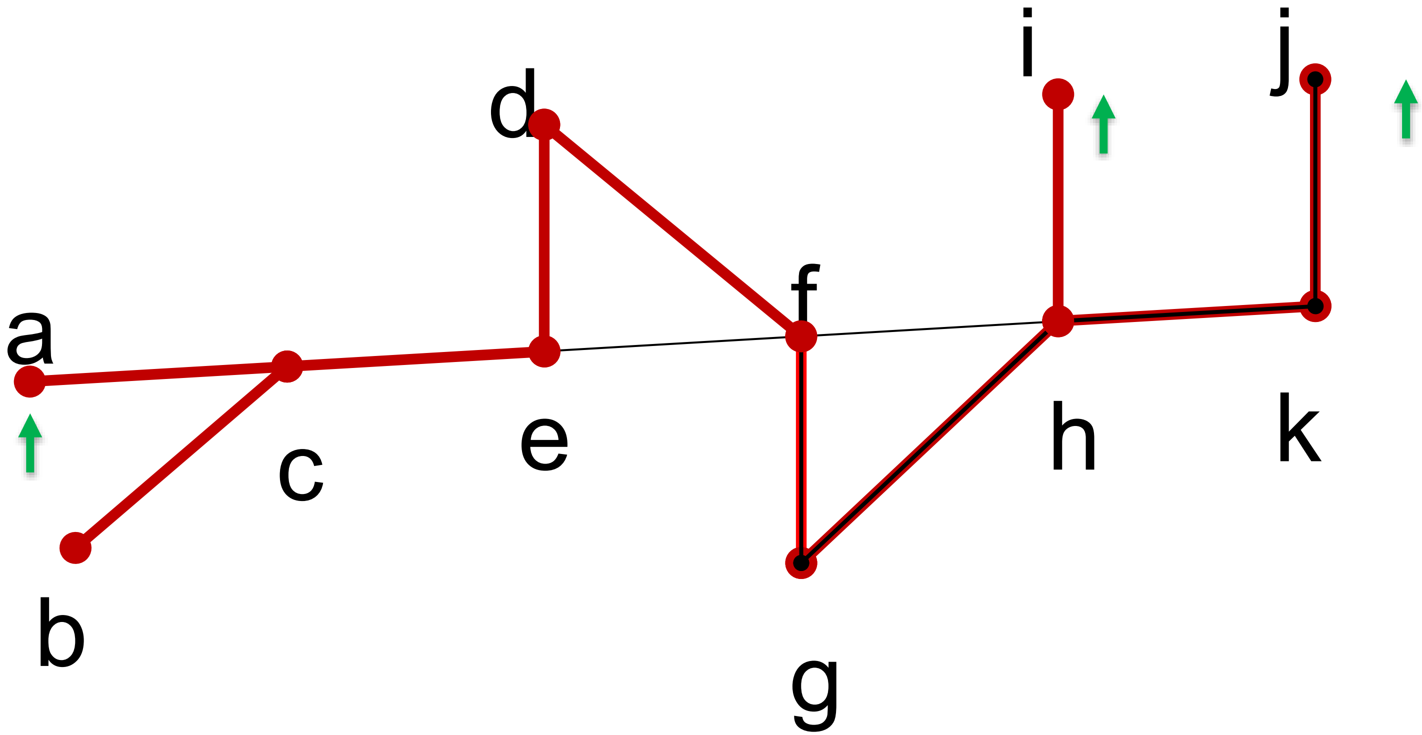


FIGURE 5 **A multicast spanning tree.**

Depth-first search
深度优先

- Build a spanning tree for a connected simple graph using a depth-first search.

Depth-first search（DFS）
深度优先



Tree edges（树边） back edges（背边）

The edges selected by depth-first search of a graph are called **tree edges.**
All other edges of the graph must connect a vertex to an ancestor or descendant of this vertex in the tree. These edges are called **back edges.**

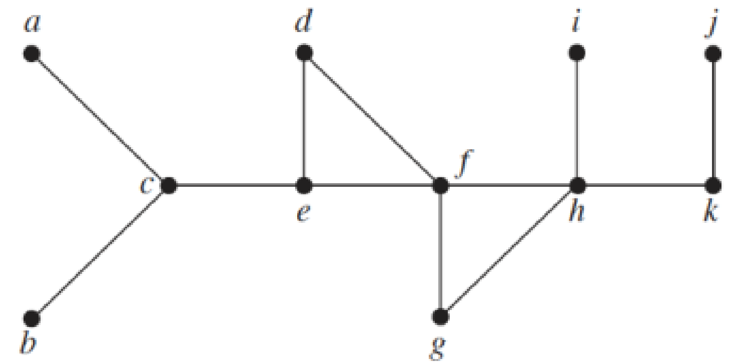


FIGURE 6 The Graph G.

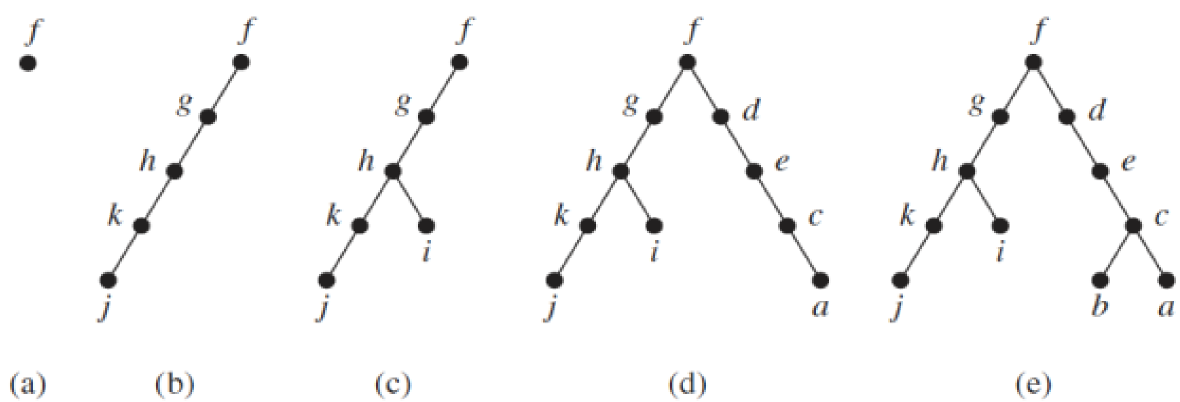


FIGURE 7 Depth-First Search of G.

Breadth-first search（BFS）
广度优先

- Produce a spanning tree of a simple graph by the use of a breadth-first search.

BFS：

- Arbitrarily choose a root
- Add all edges incident to to this vertex
- Follow the same procedure

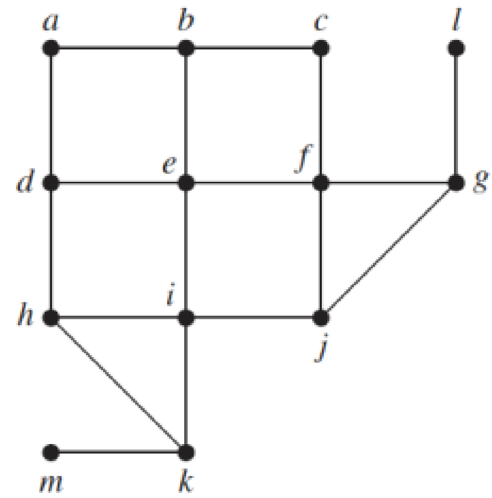


FIGURE 9 A Graph *G*.

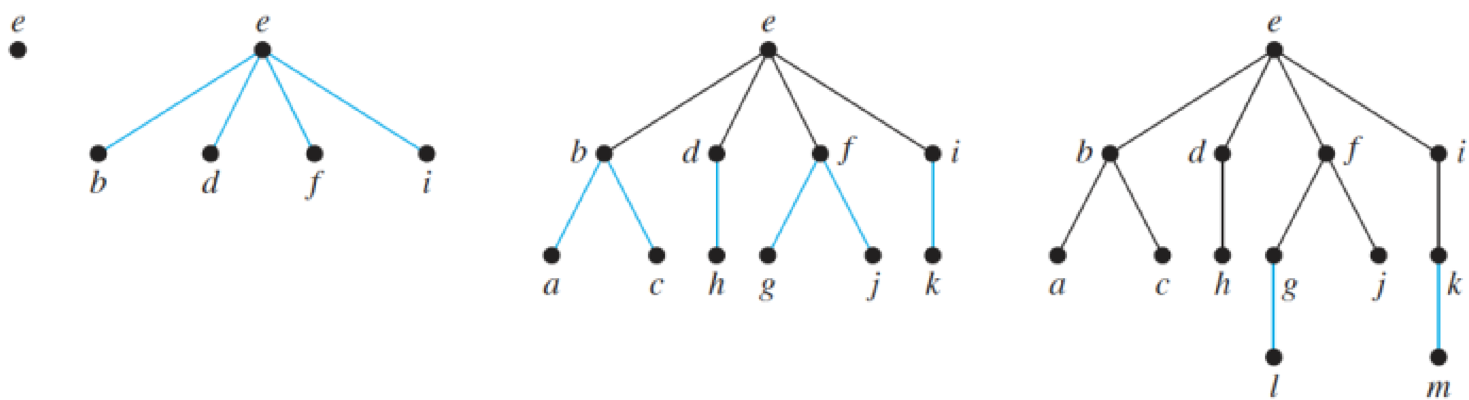
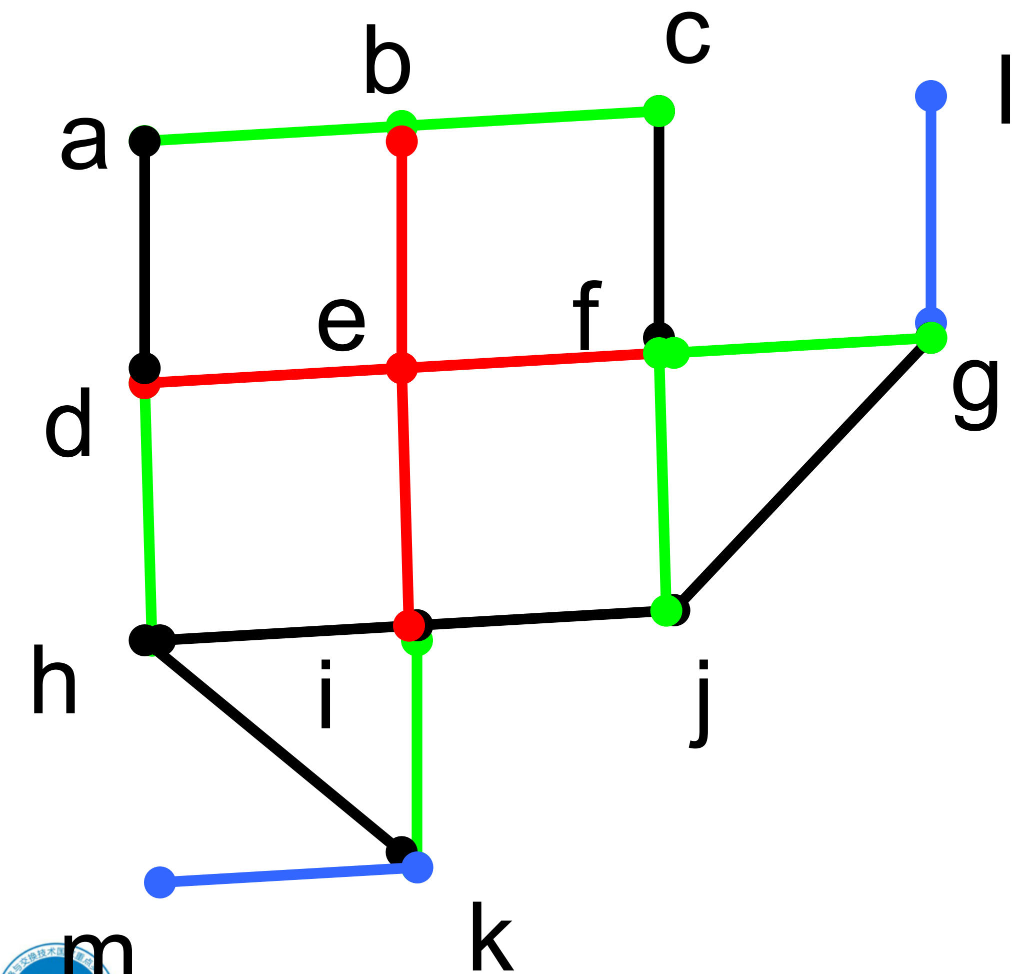


FIGURE 10 Breadth-First Search of *G*.

Breadth-first search广度优先



Backtracking 回溯法应用

- Graph Colorings
- The n-Queens Problem
- In directed graphs

Backtracking applications
回溯法图着色

- Graph colorings （图着色）

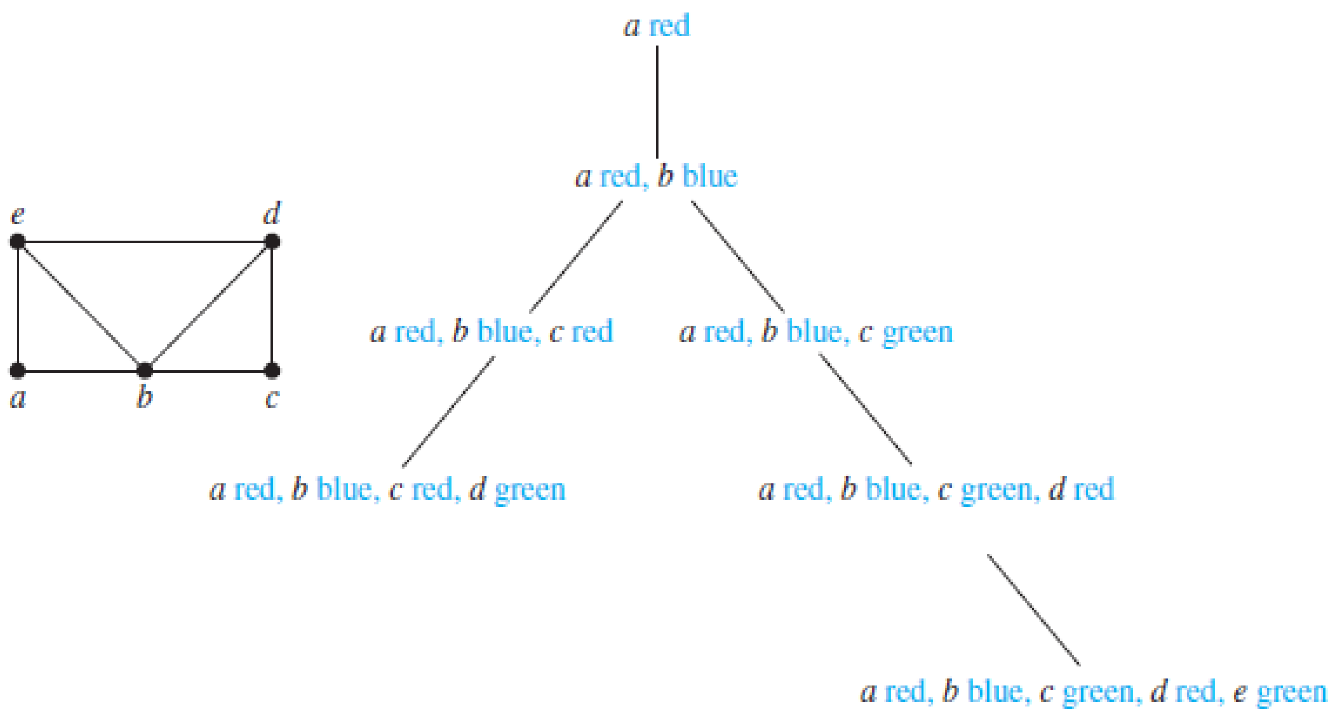


FIGURE 11 **Coloring a Graph Using Backtracking.**

Backtracking applications

- The n-Queens Problem

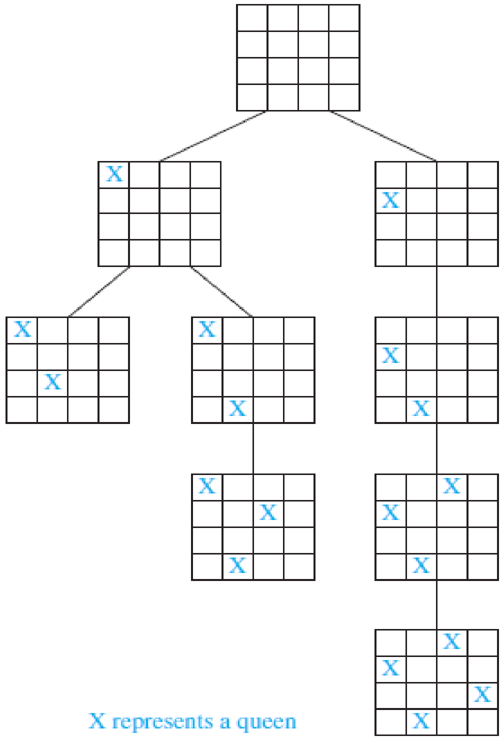


FIGURE 12 A Backtracking Solution of the Four-Queens Problem.

Backtracking applications

- Sums of Subsets {31,27,15,11,7,5}

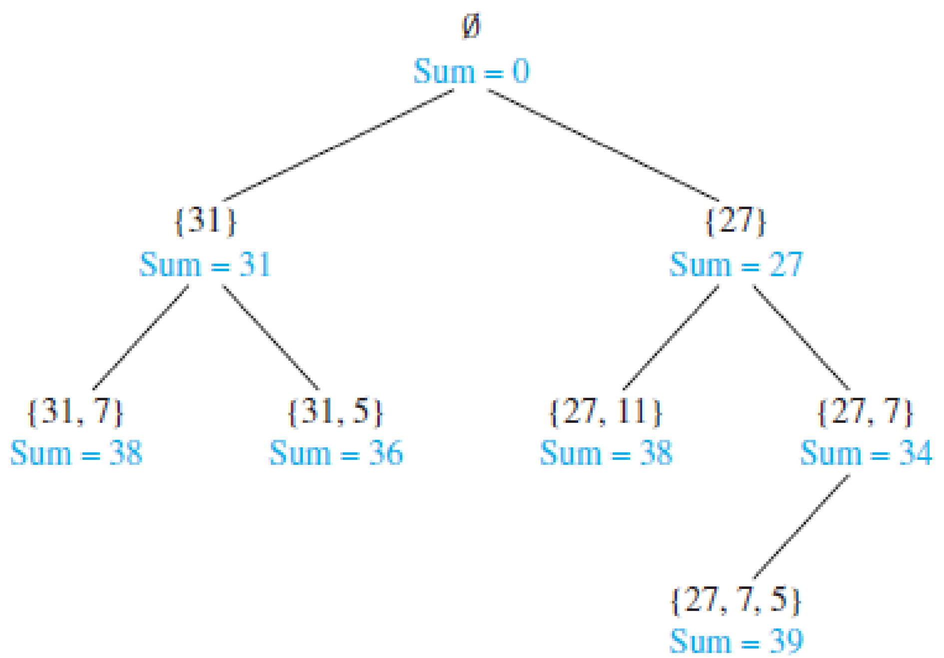


FIGURE 13 **Find a Sum Equal to 39 Using Backtracking.**

Depth-first search in directed graphs
有向图的深度优先搜索

- Example 9

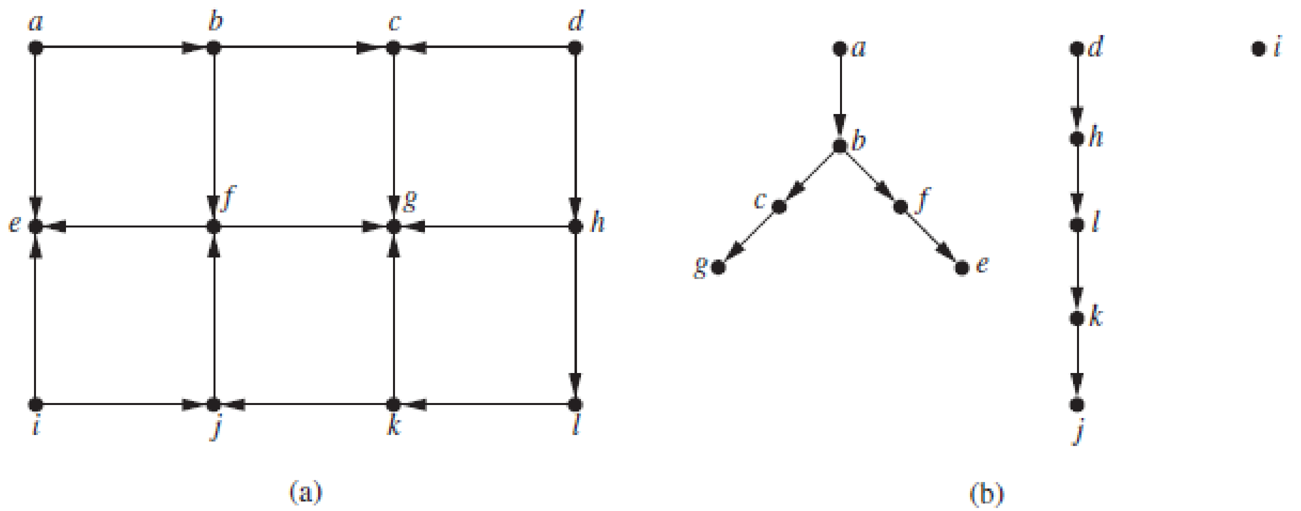
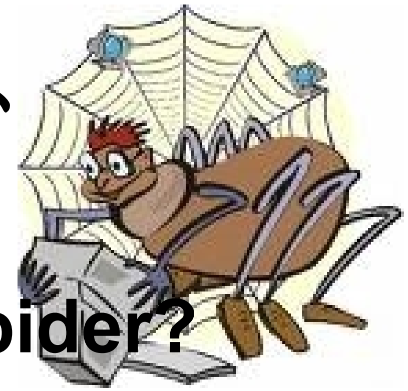


FIGURE 14 Depth-First Search of a Directed Graph.

Web spiders网络爬虫 BFS，DFS



- **What is a Spider or Search Engine Spider?**
- Description: A **Spider** or *Search Engine Spider* is a program that automatically traverses the Web and requests documents from URLs. A spider usually starts from a historical list of URLs and retrieves referenced documents. As it visits new Internet websites it checks to see if the site is already listed in its database. If the site is already listed it usually updates any changes it finds. Spiders are also commonly known as *Robots or Crawlers.* Other names sometimes used: *Webwalkers, Wanderers, or *Worms*

11.5 Minimal Spanning Trees （最小生成树）

Weighted graph（加权图）

- Definition:a **weighted graph** is a graph for which each edge is labeled with a numerical value called its **weight**（权值）.
- Example 1

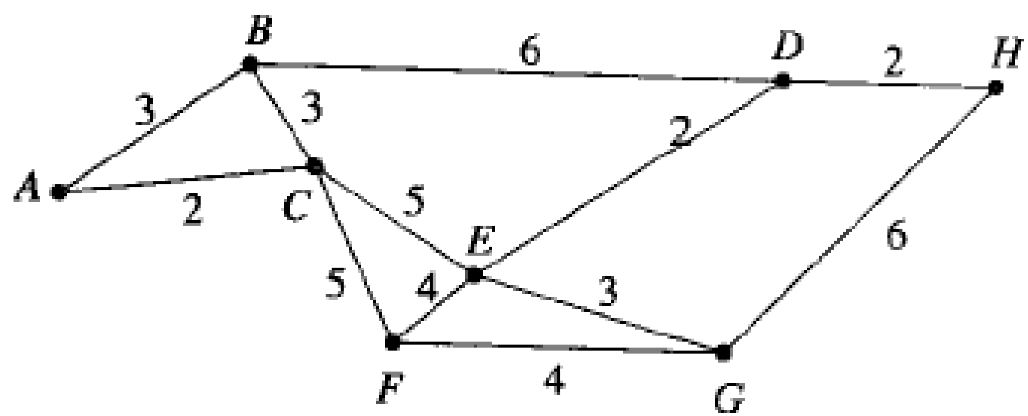


Figure 7.49

- The weight of an edge (v_i,v_j) is some times referred to as the *distance between vertice* v_i and v_j.
- A vertex u is a nearest neighbor of vertex v if u and v are adjacent and no other vertex is joined to v by an edge of lesser weight than(u,v).
- Note v may have more than one nearest neighbor.

Example 3

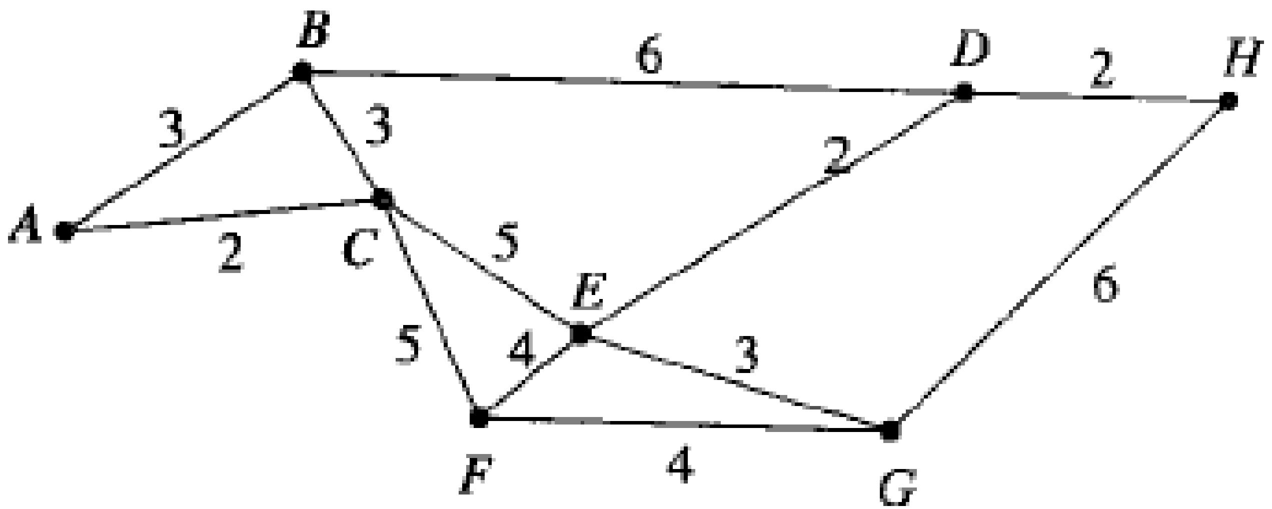


Figure 7.49

- A vertex v is a **nearest neighbor of a set of vertices** V={$v_1, v_2, \ldots v_k$} in a graph if v is adjacent to some member v_i of V and no other vertex adjacent to a member of V is joined by an edge of lesser weight than (v, v_i).
- This vertex v may belong to V.

Example 4

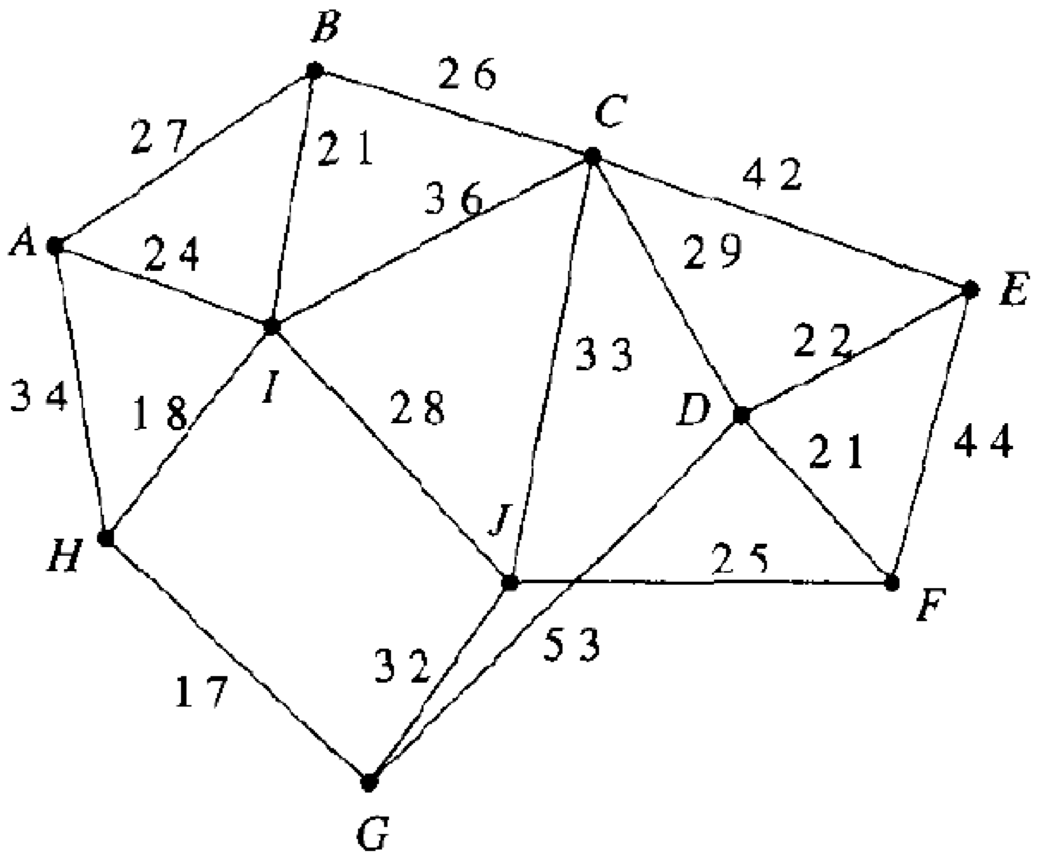


Figure 7 50

V={C,E,J}

D is the nearest neighbor of V.

Minimal spanning tree （最小生成树）

- Definition: an undirected spanning tree of a weighted graphs,for which the total weight of the edges in the tree is as small as possible. Such a spanning tree is called a minimal spanning tree.

Prim's algorithm

- **Procedure** Prim(*G*:weighted connected undirected graph with n vertices)
- *T*:= a minimum-weighted edge
- For *i*:=*1* to *n-2*
- **Begin**
- *e*:=an edge of minimum weight incident to a vertex in *T* and not forming a simple circuit in *T* if added to *T*
- T:= T with e added
- **End**{*T* is a minimum spanning tree of *G*}

Example 5

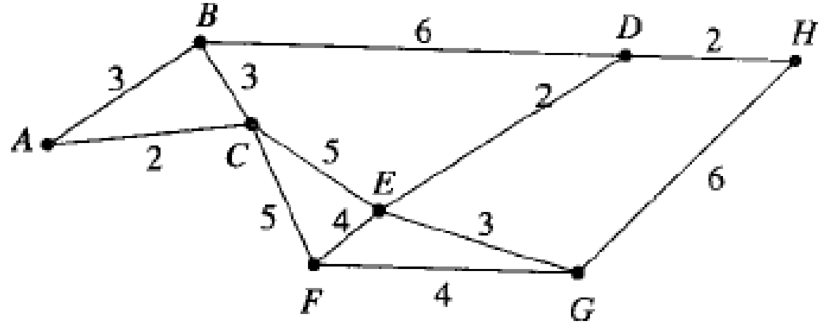


Figure 7.49

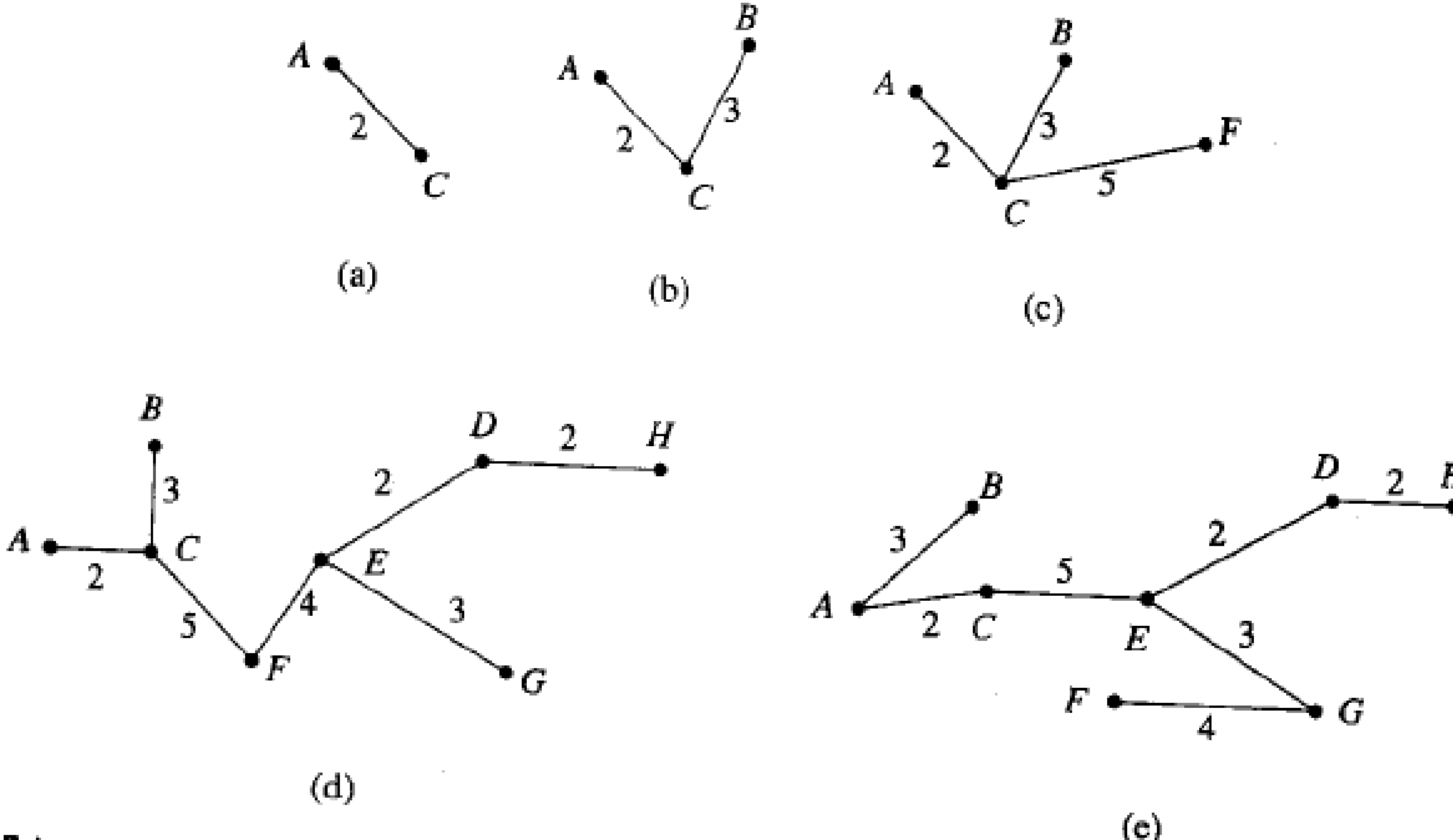


Figure 7.52

Example 6

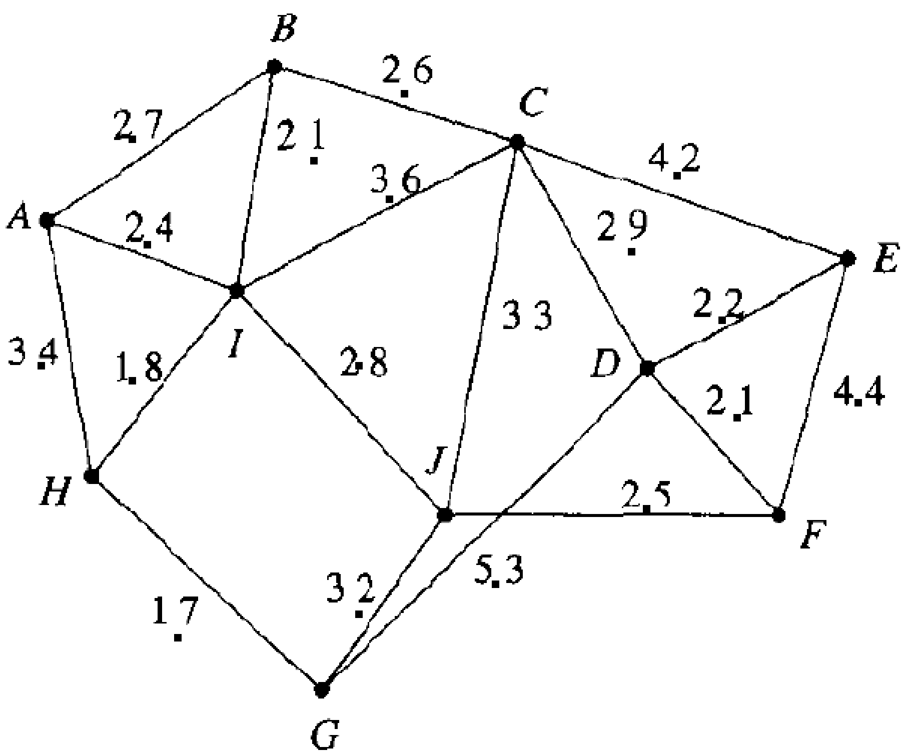


Figure 7.50

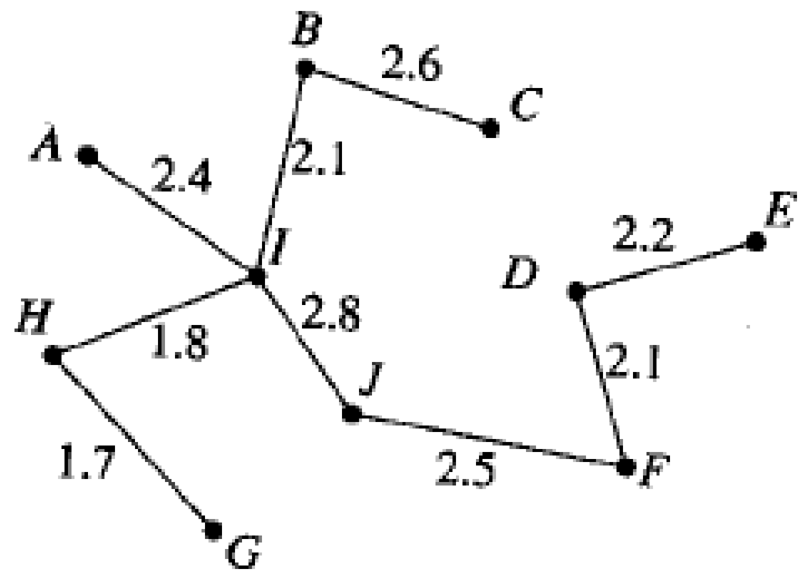
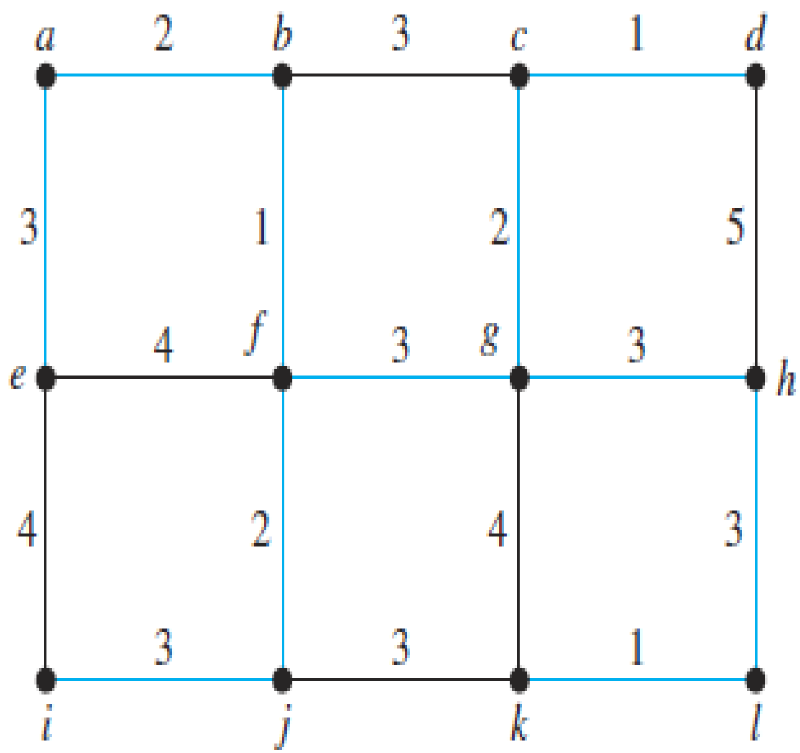


Figure 7.53



(a)

Choice	Edge	Weight
1	$\{b, f\}$	1
2	$\{a, b\}$	2
3	$\{f, j\}$	2
4	$\{a, e\}$	3
5	$\{i, j\}$	3
6	$\{f, g\}$	3
7	$\{c, g\}$	2
8	$\{c, d\}$	1
9	$\{g, h\}$	3
10	$\{h, l\}$	3
11	$\{k, l\}$	1
	Total:	24

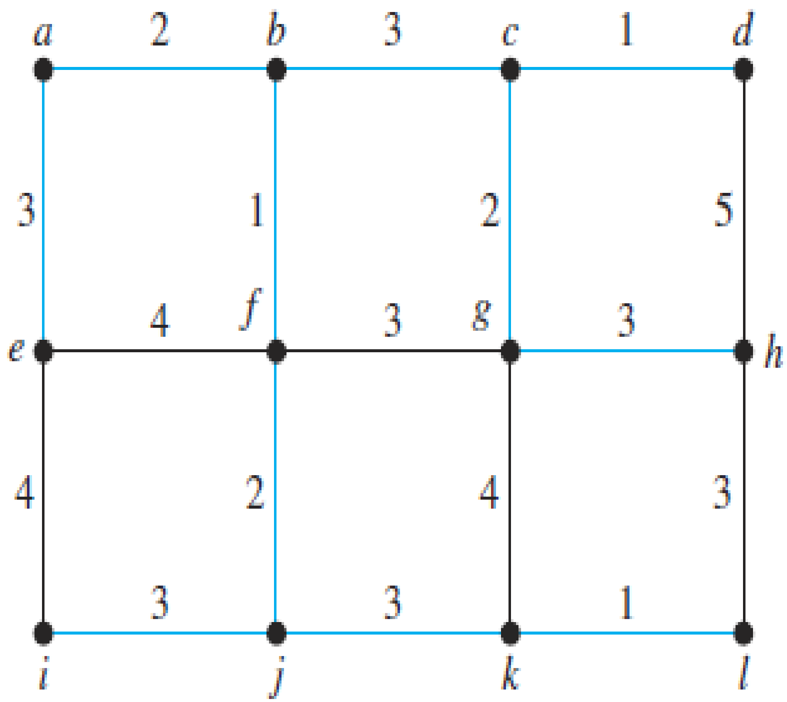
(b)

FIGURE 4 A Minimum Spanning Tree Produced Using Prim's Algorithm.

Kruskal's algorithm

- Let G be a weighted connected undirected graph with n vertices and let $S=\{e_1, e_2, \ldots, e_k\}$ be the set of weighted edges of G.
  - Step 1 Choose an edge e_1 in S of least weight.Let $E=\{e_1\}$ Replace S with $S-\{e_1\}$.
  - Step 2 Choose an edge e_i in S of least weight that will not make a cycle with members of E.Replace E with $E \cup \{e_i\}$and S with $S-\{e_i\}$.
  - Step 3 Repeat step 2 until $|E|=n-1$.
  - End of algorithm

时间复杂度 O（nlog(n)）



(a)

Choice	Edge	Weight
1	$\{c, d\}$	1
2	$\{k, l\}$	1
3	$\{b, f\}$	1
4	$\{c, g\}$	2
5	$\{a, b\}$	2
6	$\{f, j\}$	2
7	$\{b, c\}$	3
8	$\{j, k\}$	3
9	$\{g, h\}$	3
10	$\{i, j\}$	3
11	$\{a, e\}$	3
	Total:	24

(b)

FIGURE 5 **A Minimum Spanning Tree Produced by Kruskal's Algorithm.**

Example 7

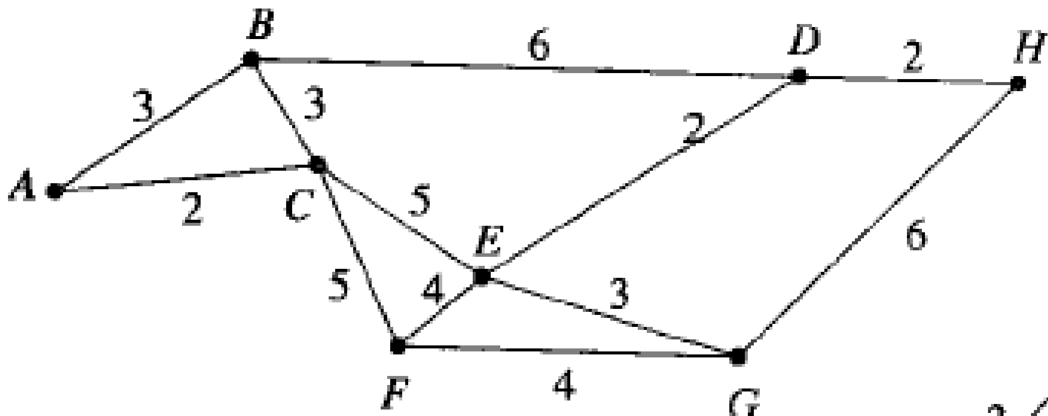


Figure 7.49

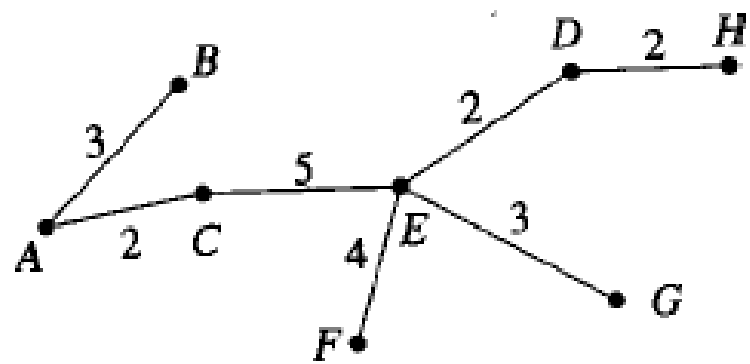


Figure 7.54

Example 8

- Use Kruskal's algorithm to find a minimal spanning tree for the graph

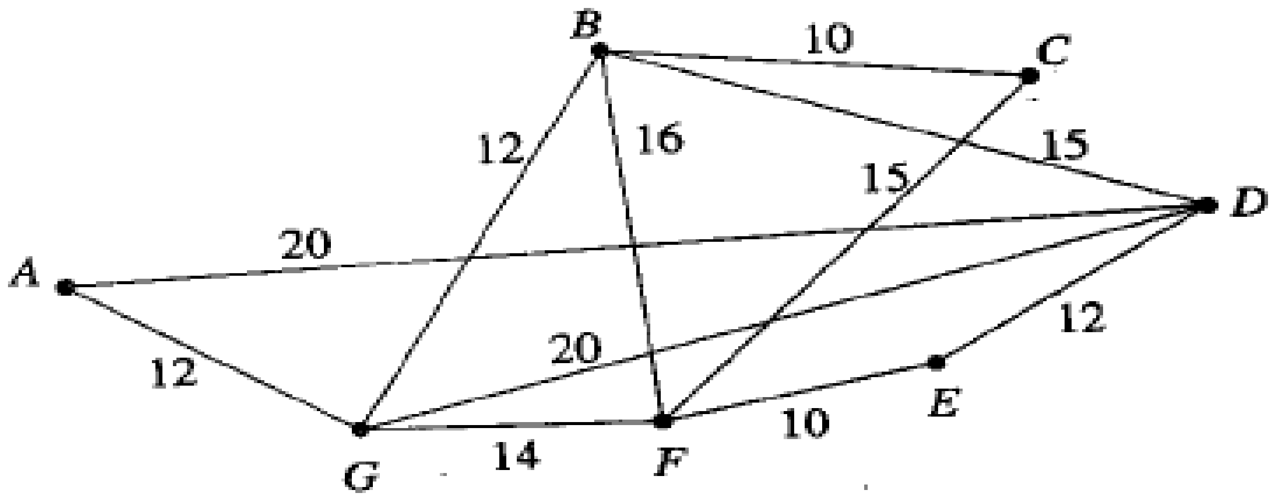
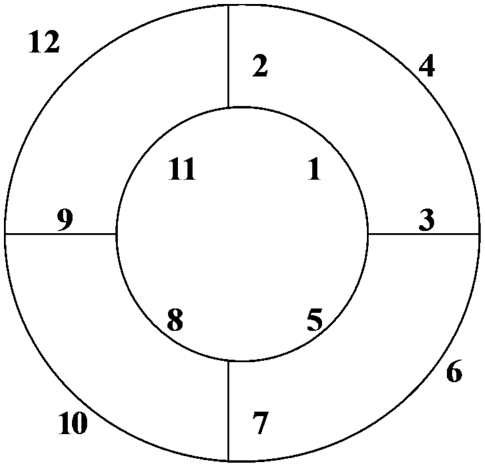


Figure 7.55

解题过程：



(1) 边排序:(v1,v2),(v2,v7),(v1,v6),(v6,v7)(v1,v4),(v5,v6),(v4,v5),(v3,v4),(v3,v8)(v5,v8),(v2,v3),(v7,v8)

(2) 选s1={e1} e1=(v1,v2)

(3) 选e2=(v2,v7)

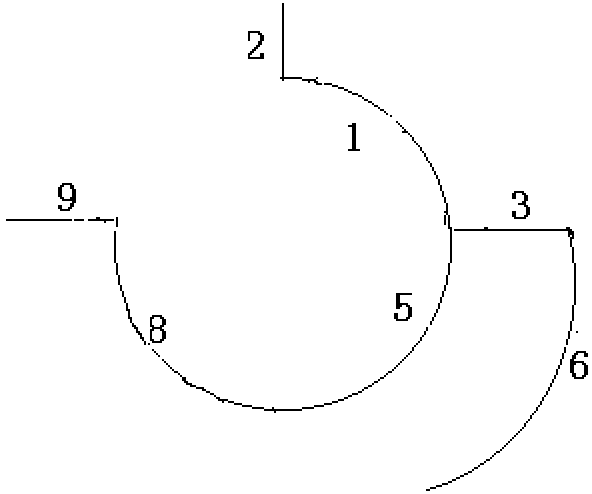
(4) e3=(v1,v6)

(5) e4=(v1,v4)=5

(6) e5=(v5,v6)=6

(7) e6=(v3,v4)=8

(8) e7=(v3,v8)



作业

- §11.3 – 6, 18, 22, 34
- §11.4 – 12, 14, 22, 24, 50, 54
- §11.5 – 4, 8, 10, 14, 18