Trees（树）

Zhang Yanmei

ymzhang@bupt.edu.cn

College of Computer Science & Technology

Beijing University of Posts & Telecommunications

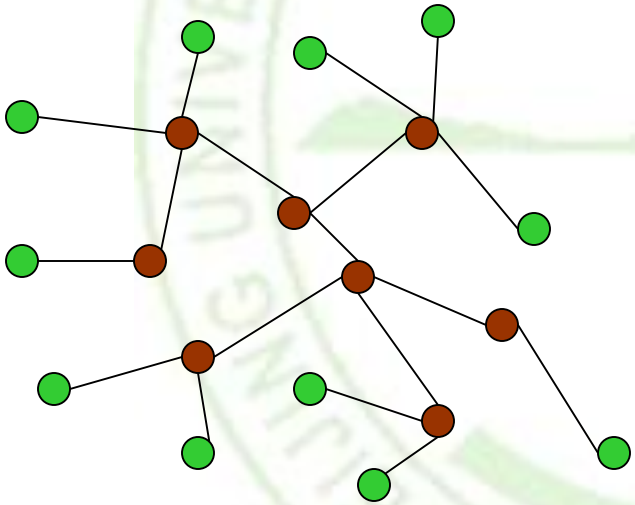
§ 11.1: Introduction to Trees

- A *tree* is a connected undirected graph with no simple circuits.
  - **Theorem:** There is a unique simple path between any two of its vertices.
- A (not-necessarily-connected) undirected graph without simple circuits is called a *forest*.
  - You can think of it as a set of trees having disjoint sets of nodes.
- A *leaf* node in a tree or forest is any pendant or isolated vertex. An *internal* node is any non-leaf vertex (thus it has degree ≥ ___).

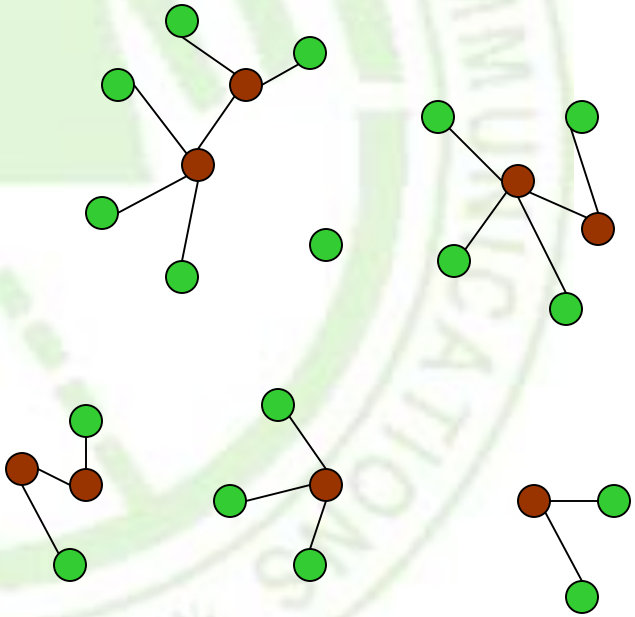
Tree and Forest Examples

Leaves in green, internal nodes in brown.

- A Tree:



- A Forest:



Rooted Trees有根树

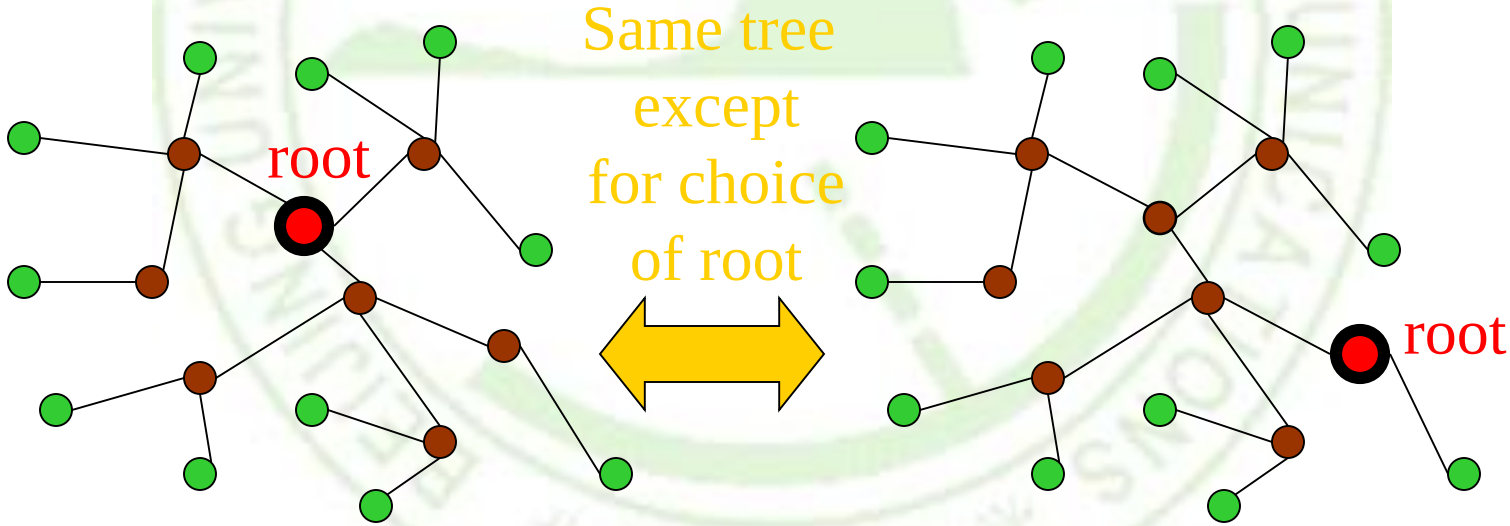
- A *rooted tree* is a tree in which one node has been designated the *root*.
  - Every edge is (implicitly or explicitly) directed away from the root.
- You should know the following terms about rooted trees:
  - Parent, child, siblings兄弟, ancestors祖先, descendents子孙, leaf树叶, internal node内点, subtree子树.

Rooted Trees

- If *v* is a vertex in *T* other than the root, the ***parent of v*** is the unique vertex ***u*** such that there is a directed edge from *u* to *v*
- When *u* is the parent of *v*, *v* is called a ***child*** of ***u.***
- Vertices with the same parent are called **siblings**.
- The **ancestors** of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root.
- The **descendants** of a vertex *v* are those vertices that have *v* as an ancestor.

Rooted Tree Examples

- Note that a given unrooted tree with *n* nodes yields *n* different rooted trees.



N-ARY TREES

- A rooted tree is called *n*-ary if every vertex has no more than *n* children.
  - It is called *full* if every internal (non-leaf) vertex has *exactly n* children.
- A 2-ary tree is called a *binary tree*.
  - These are handy for describing sequences of yes-no decisions.
    - **Example:** Comparisons in binary search algorithm.

Ordered Rooted Tree有序根树

- This is just a rooted tree in which the children of each internal node are ordered.
- In ordered binary trees, we can define:
  - left child, right child
  - left subtree, right subtree
- For *n*-ary trees with $n>2$, can use terms like "leftmost"最左边, "rightmost"最右边 *etc.*

Some Tree Theorems

- Any tree with n nodes has $e = n-1$ edges.
  - **Proof:** Consider removing leaves.
- A full m-ary tree with i internal nodes has $n=mi+1$ nodes, and $\ell=(m-1)i+1$ leaves.
  - **Proof:** There are mi children of internal nodes, plus the root. And, $\ell = n-i = (m-1)i+1$.
- Thus, when m is known and the tree is full, we can compute all four of the values e, i, n, and ℓ, given any one of them.
  - $i=(n-1)/m$, $l=[(m-1)n+1]/m$
  - $n=mi+1$, $l=(m-1)i+1$
  - $n=(ml-1)/m-1$, $i=(l-1)/(m-1)$

Example 9

- Suppose that someone starts a chain letter. Each person who receives the letter is asked to send it on to four other people. Some people do this, but others do not send any letters.
  - How many people have seen the letter, including the first person, if no one receives more than one letter and if the chain letter ends after there have been 100 people who read it but did not send it out?
  - How many people sent out the letter?

Some More Tree Theorems

- **Definition:** The *level* (层)of a node is the length of the simple path from the root to the node.
  - The *height*(*高度*) of a tree is maximum node level.

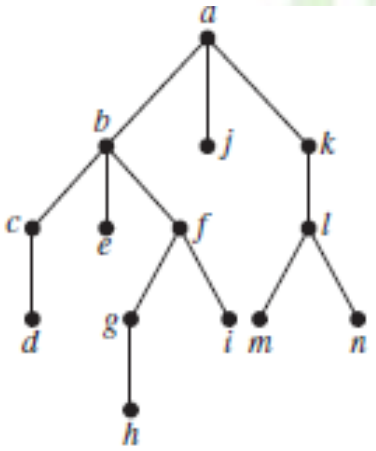


FIGURE 13 A Rooted Tree.

Some More Tree Theorems

- A rooted *m*-ary tree with height *h* is called *balanced* 平衡树if all leaves are at levels *h* or *h*−1.
  - Example 11

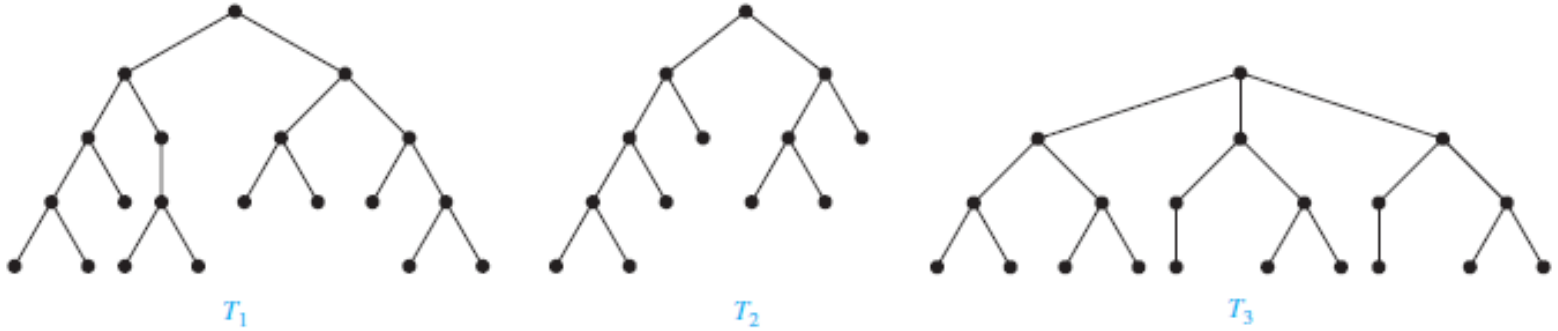


FIGURE 14 Some Rooted Trees.

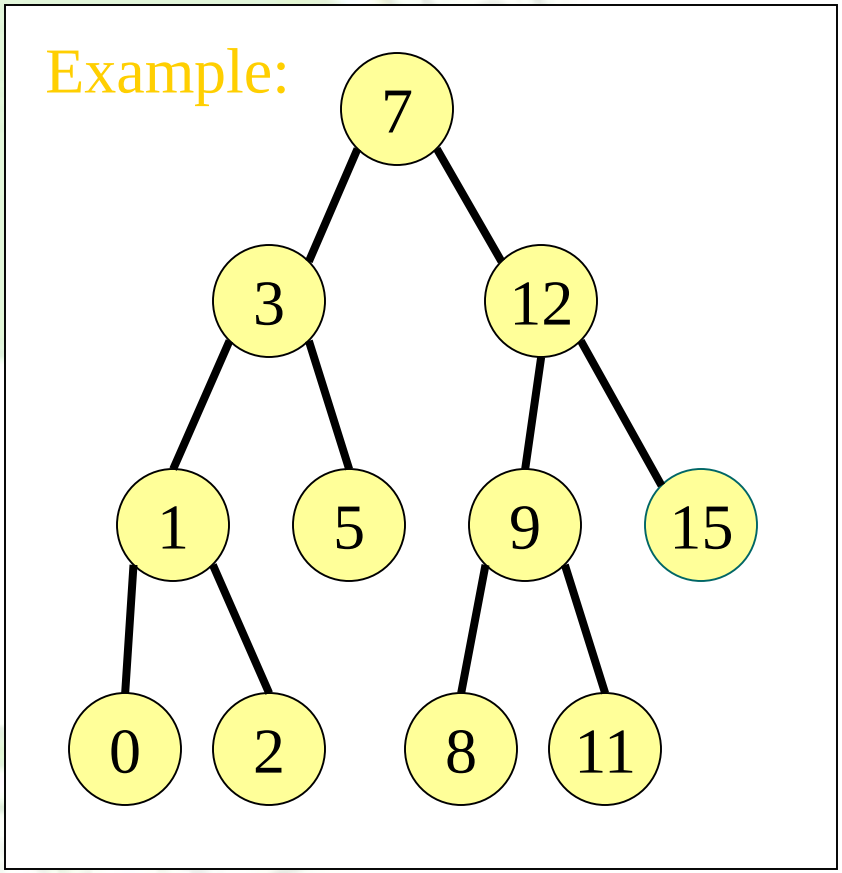
- **Theorem:** There are at most m^h leaves in an m-ary tree of height h.
  - **Corollary:** An m-ary tree with ℓ leaves has height $h \geq \lceil \log_m \ell \rceil$. If m is full and balanced then $h = \lceil \log_m \ell \rceil$.

§ 11.2: Applications of Trees

- Binary search trees
  - A simple data structure for sorted lists
- Decision trees
  - Minimum comparisons in sorting algorithms
- Prefix codes
  - Huffman coding
- Game trees

Binary Search Tree Format

- Items are stored at individual tree nodes.
- We arrange for the tree to always obey this invariant:
  - For every item *x*,
    - Every node in *x*'s left subtree is less than *x*.
    - Every node in *x*'s right subtree is greater than *x*.



Recursive Binary Tree Insert

```
procedure insert(T: binary tree, x: item)
    v := root[T]
    if v = null then begin
        root[T] := x; return "Done" end
    else if v = x return "Already present"
    else if x < v then
        return insert(leftSubtree[T], x)
    else {must be x > v}
        return insert(rightSubtree[T], x)
```

Decision Trees

- A decision tree represents a *decision-making process*.
  - Each possible "decision point" or situation is represented by a node.
  - Each possible choice that could be made at that decision point is represented by an edge to a child node.
- In the extended decision trees used in *decision analysis*, we also include nodes that represent random events and their outcomes.

Coin-Weighing Problem

- Imagine you have 8 coins, one of which is a lighter counterfeit, and a free-beam balance.
  - No scale of weight markings is required for this problem!
- How many weighings are needed to guarantee that the counterfeit coin will be found?

Applying the Tree Height Theorem

- The decision tree must have at least 8 leaf nodes, since there are 8 possible outcomes.
  - In terms of which coin is the counterfeit one.
- Recall the tree-height theorem, $h \geq \lceil \log_m \ell \rceil$.
  - Thus the decision tree must have height $h \geq \lceil \log_3 8 \rceil = \lceil 1.893\ldots \rceil = 2$.
- Let's see if we solve the problem with *only* 2 weighings…

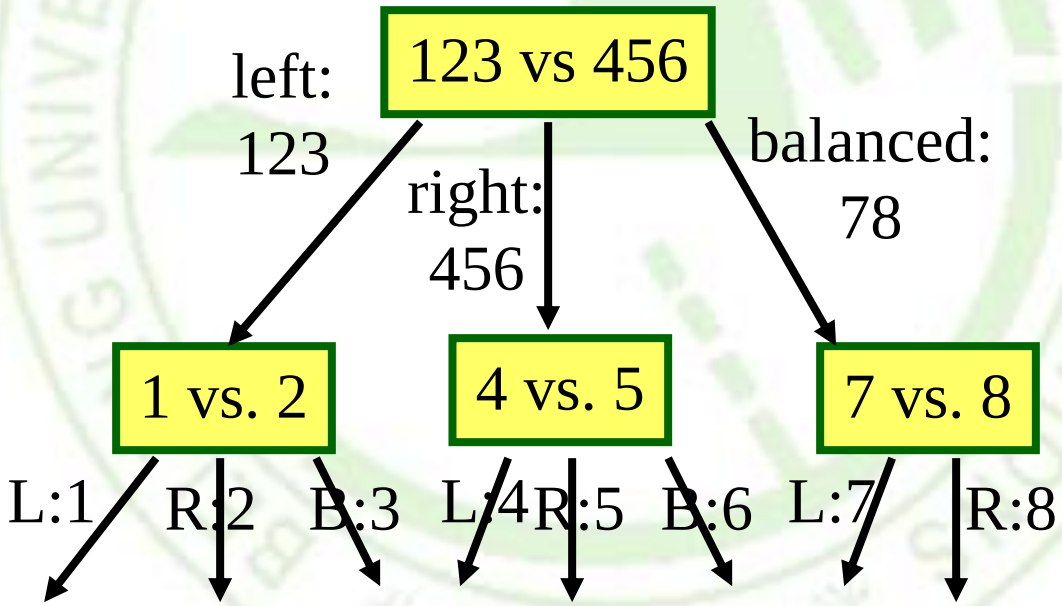
General Balance Strategy

- On each step, put $\lceil n/3 \rceil$ of the n coins to be searched on each side of the scale.
  - If the scale tips to the left, then:
    - The lightweight fake is in the right set of $\lceil n/3 \rceil \approx n/3$ coins.
  - If the scale tips to the right, then:
    - The lightweight fake is in the left set of $\lceil n/3 \rceil \approx n/3$ coins.
  - If the scale stays balanced, then:
    - The fake is in the remaining set of $n - 2\lceil n/3 \rceil \approx n/3$ coins that were not weighed!
- Except if $n \bmod 3 = 1$ then we can do a little better by weighing $\lfloor n/3 \rfloor$ of the coins on each side.

You can prove that this strategy always leads to a balanced 3-ary tree.

Coin Balancing Decision Tree

- Here's what the tree looks like in our case:



Game Trees

- the vertices of these trees represent the positions that a game can be in as it progresses;
- the edges represent legal moves between these positions.
- The leaves of a game tree represent the final positions of a game. We assign a value to each leaf indicating the payoff to the first player.

Game Trees

- Nim

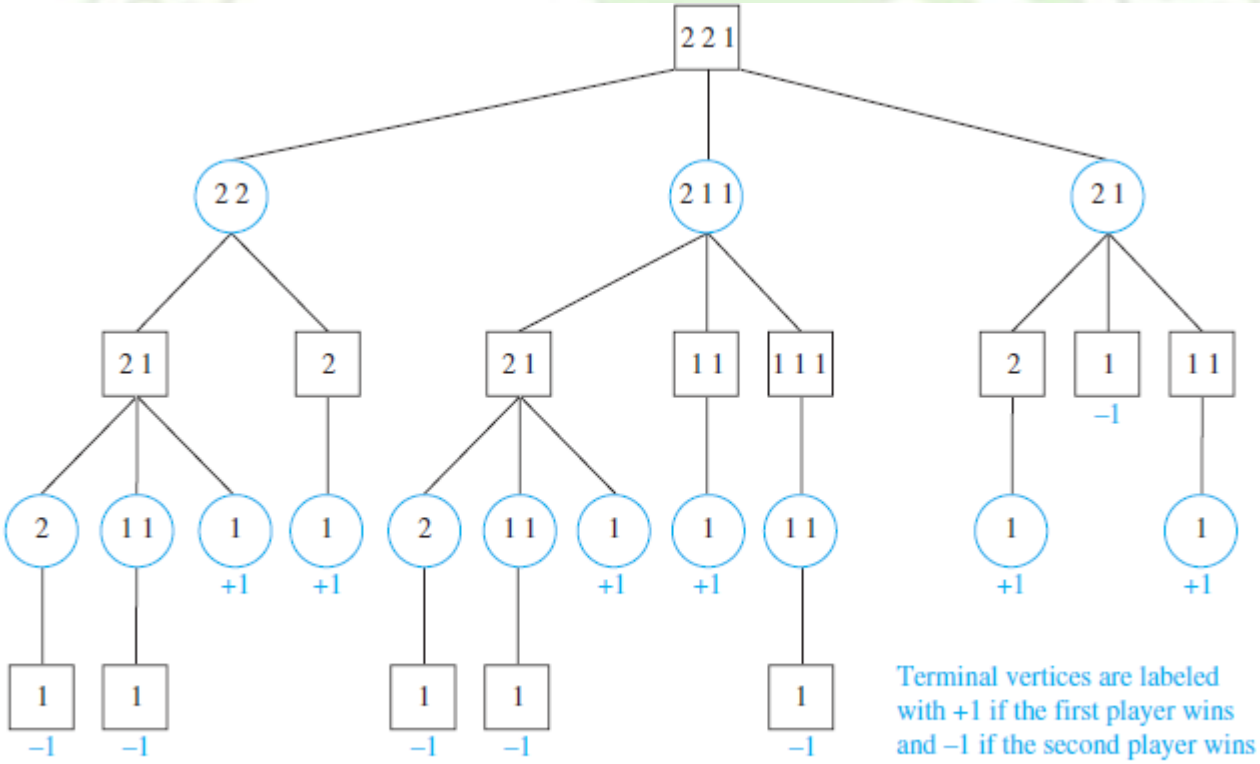


FIGURE 7 The Game Tree for a Game of Nim.

Game Trees

- The strategy where the first player moves to a position represented by a child with maximum value and the second player moves to a position of a child with minimum value is called the **minmax strategy**.
- We can determine who will win the game when both players follow the minmax strategy by calculating the value of the root of the tree.
- The minmax strategy is the optimal strategy for both players.

Homework

- § 11.1
  - 4, 20,28
- § 11.2
  - 6, 30

Tree Traversal

Zhang Yanmei

ymzhang@bupt.edu.cn

College of Computer Science & Technology

Beijing University of Posts & Telecommunications

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*,…*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*,…*A.* k_v.

EXAMPLE 1

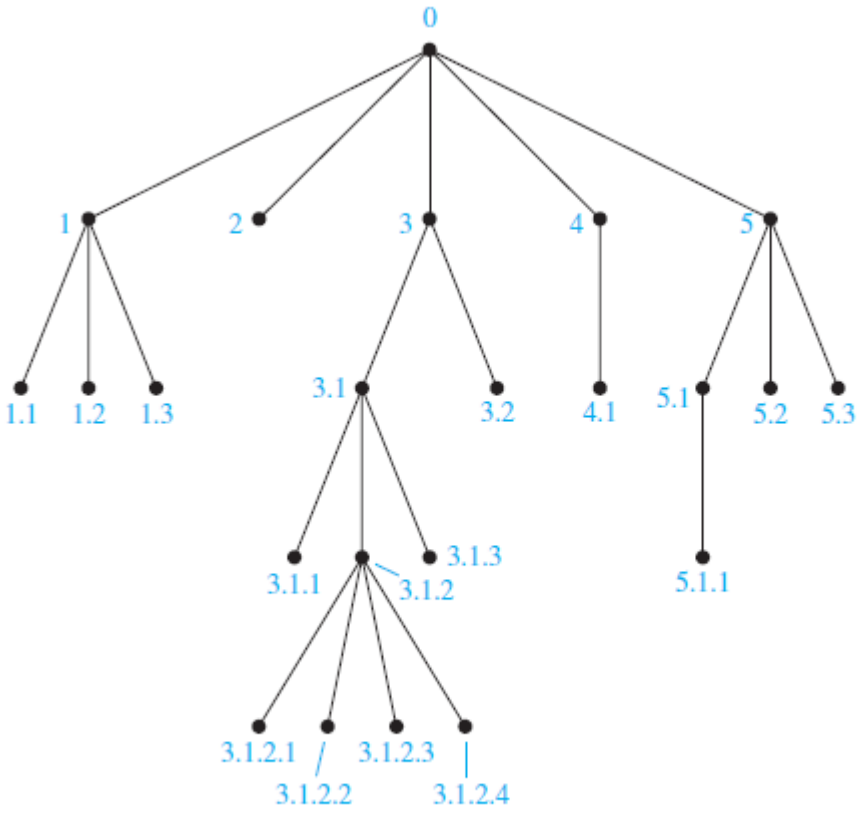


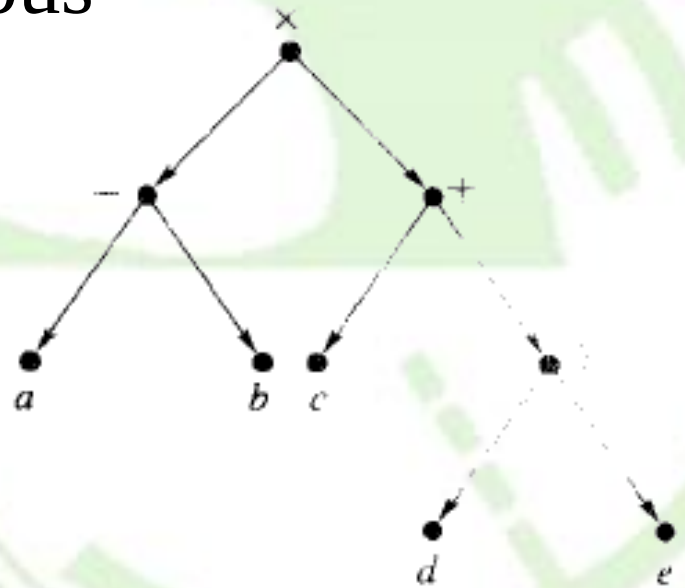
FIGURE 1 The Universal Address System of an Ordered Rooted Tree.

Traversal algorithms

- Procedures for systematically visiting every vertex of an ordered rooted tree are called traversal algorithms.
  - Preorder traversal
  - Inorder traversal
  - Postorder traversal

INFIX NOTATION（中缀表示法）

- ambiguous



(a)

Figure 7.14

PREFIX（前缀）

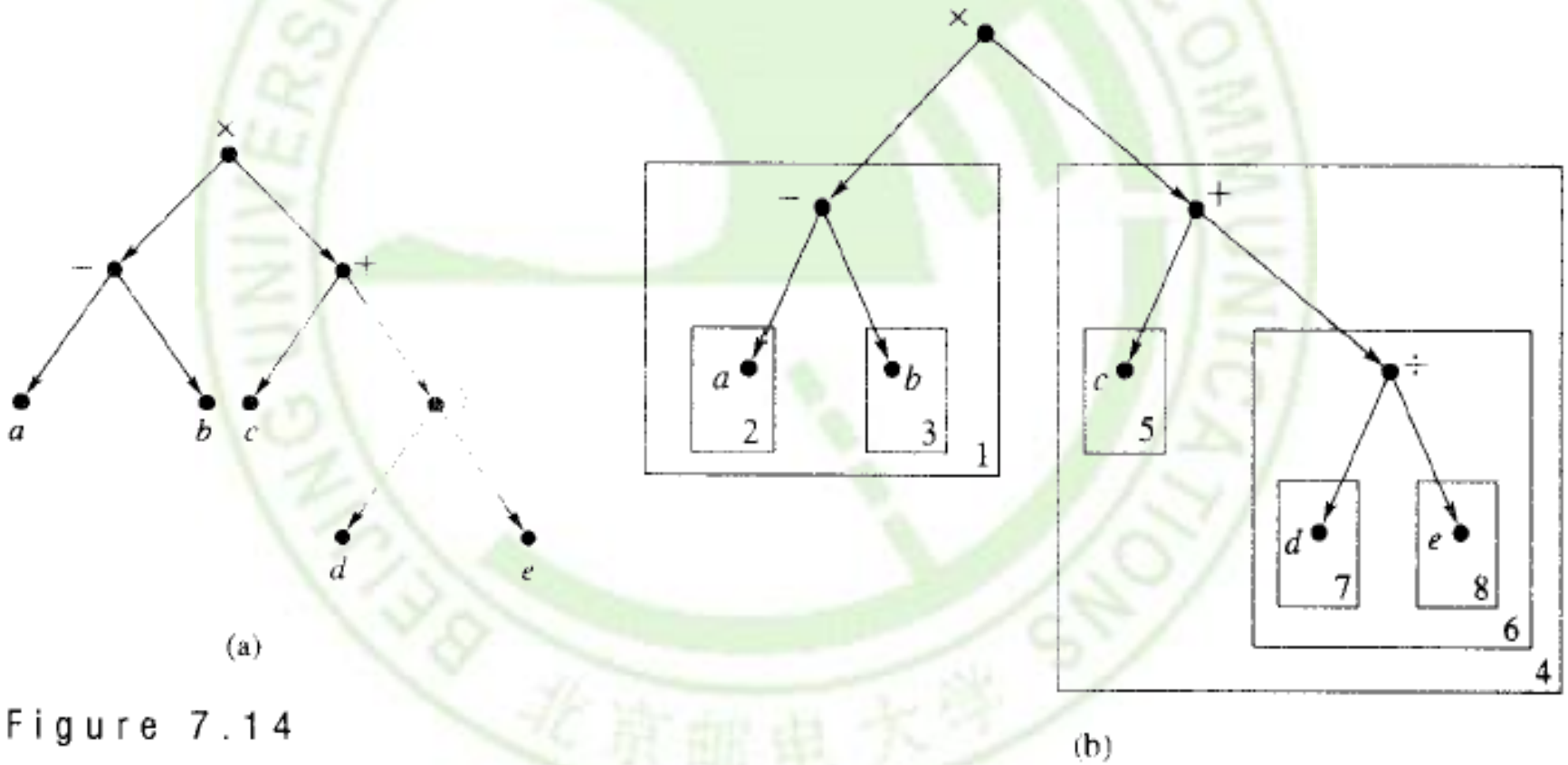


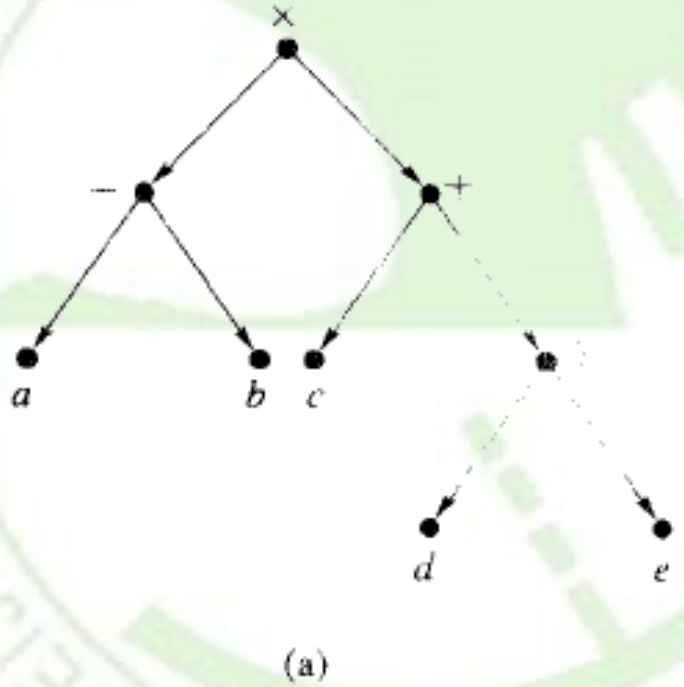
Figure 7.14

Prefix(Polish Form)（前缀表达式（波兰式））

- Unambiguously without parentheses
- from right to left　Fxy
- operation symbol precedes the argument
- Example 7

$$+ - * 235 / \uparrow 234$$

Postfix（后缀表达式（逆波兰式））



(a)

Figure 7.14

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

- Example 10

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

Spanning Trees

Zhang Yanmei

ymzhang@bupt.edu.cn

College of Computer Science & Technology

Beijing University of Posts & Telecommunications

Introduction

- Definition 1
- 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.

Example 4

- 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.**

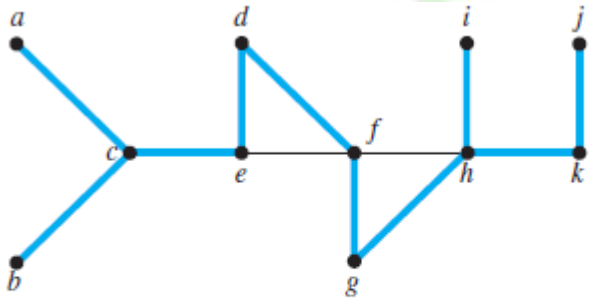


FIGURE 8 The Tree Edges and Back Edges of the Depth-First Search in Example 4.

Algorithm 1 Depth-first search

procedure DFS(*G*: connected graph with vertices $v_1, v_2, \ldots v_n$)
T:=tree consisting only of the vertex v_1
visit(v_1)
procedure *visit* (*v*: vertex of *G*)
for each vertex *w* adjacent to *v* not yet in *T*
begin
    add vertex *w* and edge {*v*, *w*} to *T*
    visit (*w*)
end

Breadth-First Search

procedure BFS(*G*: connected graph with vertices $v_1, v_2, \ldots v_n$)
T:=tree consisting only of the vertex v_1
L:=empty list
put v_1 in the list *L* of unprocessed vertices
while *L* is not empty
begin
remove the first vertex, *v*, from *L*
for each neighbor *w* of *v*
If *w* is not in *L* and not in *T* then
    begin
    add *w* to the end of the list *L*
    add *w* and edge {*v*, *w*} to *T*
    end
end

Backtracking applications

- Graph colorings

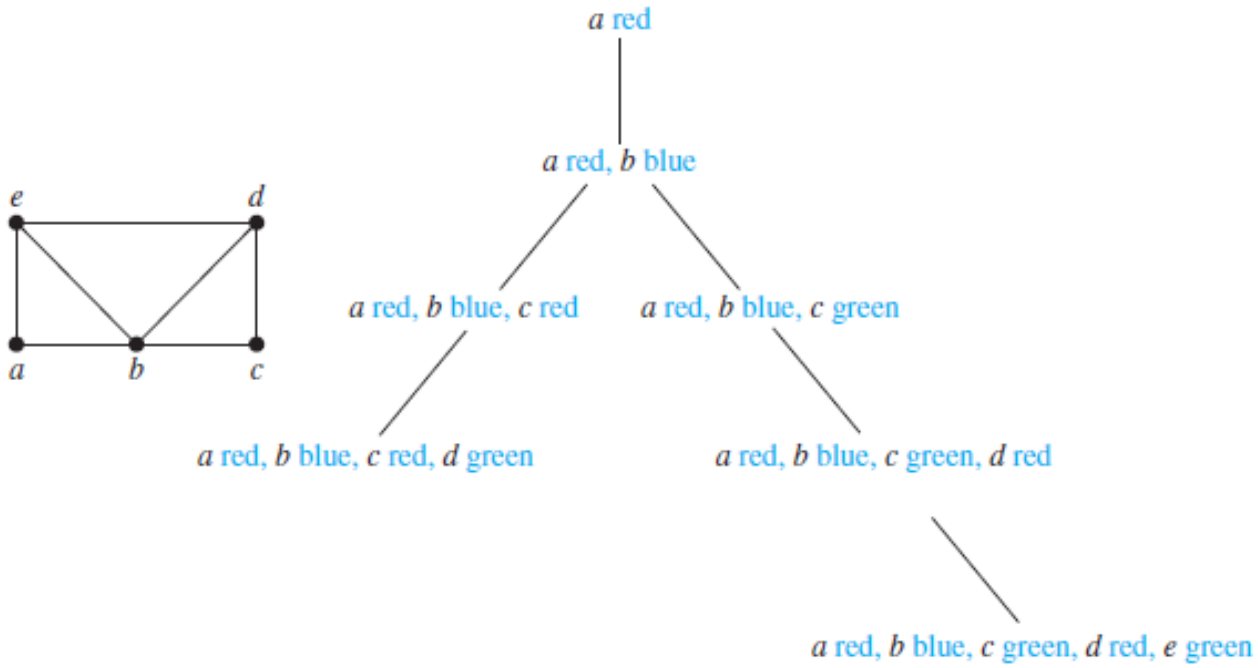


FIGURE 11 **Coloring a Graph Using Backtracking.**

Depth-first search in directed graphs

- Example 9

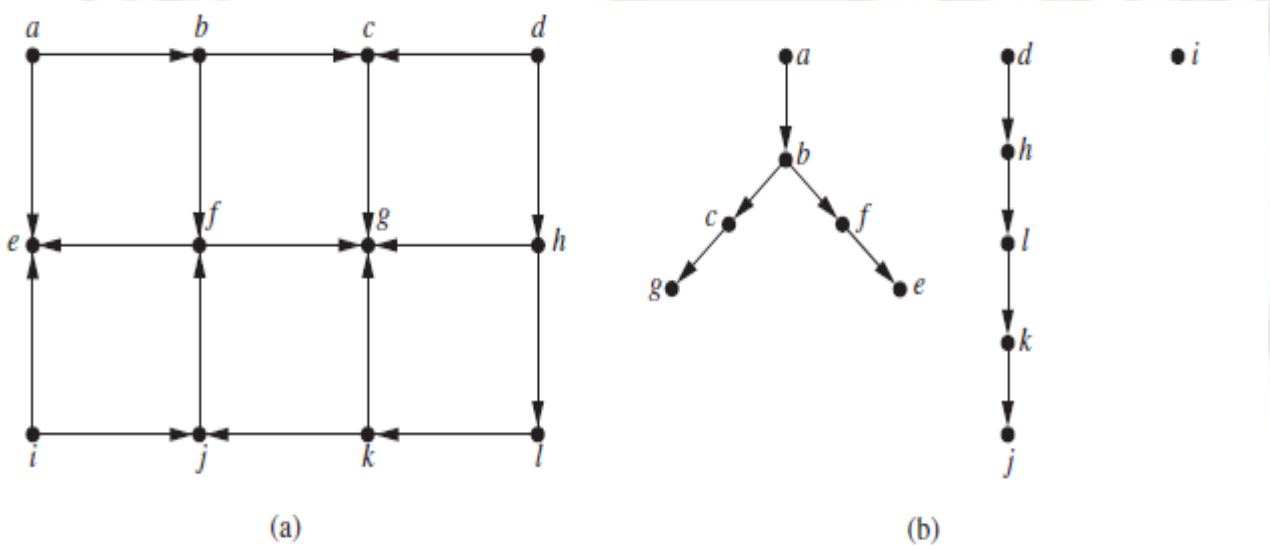


FIGURE 14 Depth-First Search of a Directed Graph.

- Web Spiders

Homework

- § 11.3
  - 6, 22
- § 11.4
  - 12, 14(be unique, because alphabetical order and deepth-first search)
  - 12: 区分无根树和有根树;
  - tree isomorphism:degree series, length of the longest simple path.

Minimum Spanning Trees（最小生成树）

Zhang Yanmei

ymzhang@bupt.edu.cn

College of Computer Science & Technology

Beijing University of Posts & Telecommunications

minum spanning trees

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

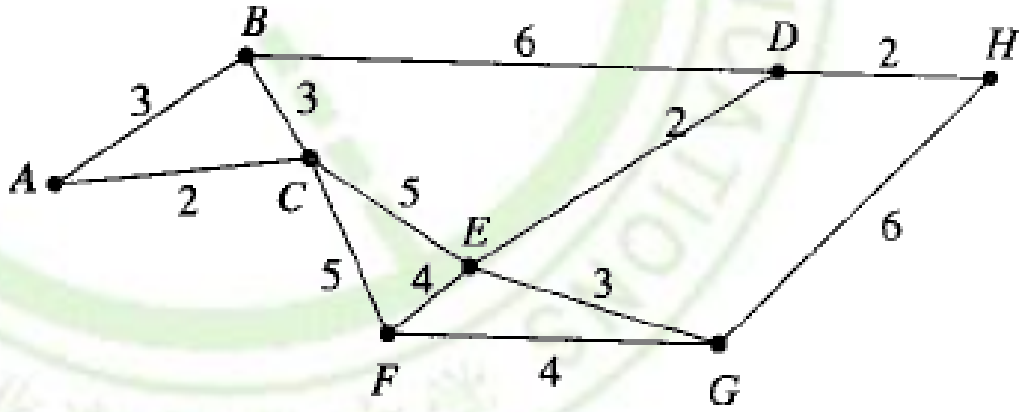


Figure 7.49

最近邻点

- 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.

点集的最近邻点

- 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

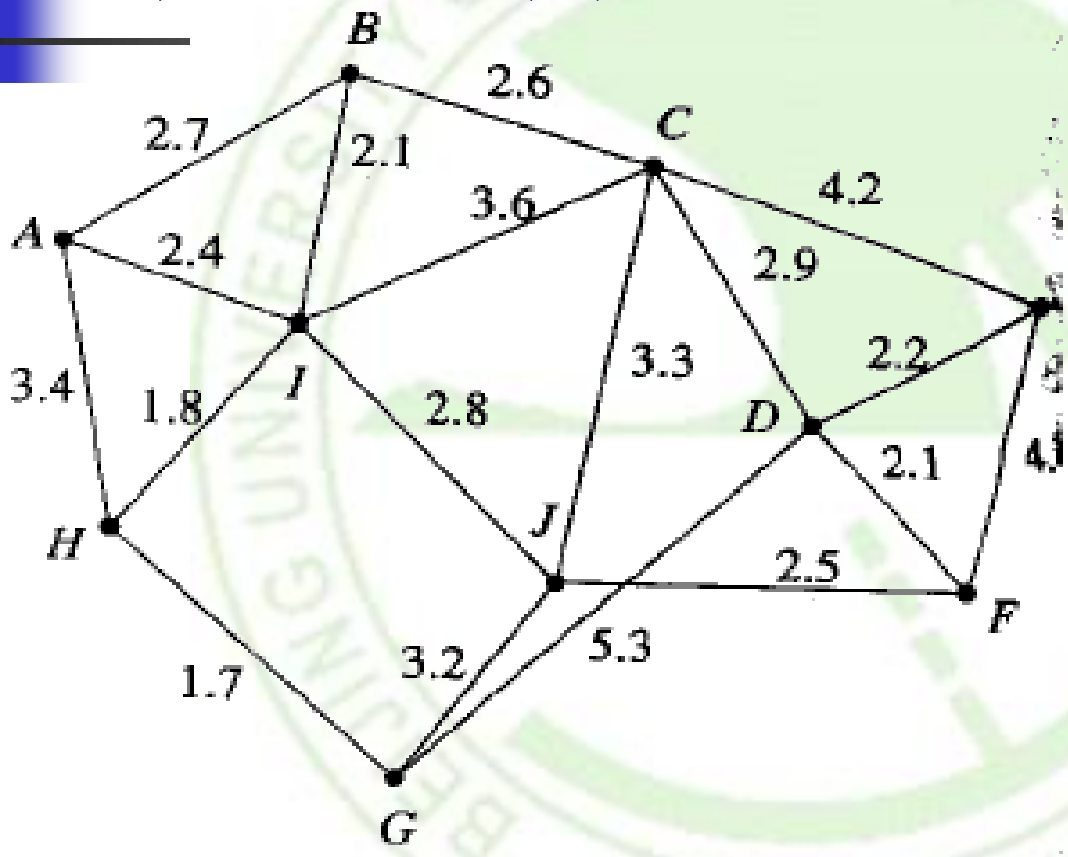


Figure 7.50

- V={C,E,J}
- D is the nearest neighbor of V.

Minimum spanning tree（最小生成树）

- Definition: a minimum spanning tree in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.
- 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 minimum spanning tree.

Prim's Algorithm(1930,1957)

Procedure *Prim* (*G*: weighted connected undirected graph with *n* vertices)

T:=a **minimum-weight 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* }

Theorem 1

- Prim's algorithm produces a minimal spanning tree for the graph G.

Proof:Let G have *n* vertex. Let T be the spanning tree for G produced by Prim's algorithm.

- Suppose that the edges of T are $t_1, t_2, \ldots, t_{n-1}$ in the order of they were selected.
- we define T_i (i=1 to n-1) to be the tree with $t_1, t_2, \ldots, t_i$ and $T_0=\{\}$Then $T_0 \subset T_1 \subset \ldots \subset T_{n-1}=T$.
- We now proved that each T_i is contained in a minimal spanning tree for G by mathematical induction.

- Basis step: Clearly P(0):$T_0=\{\}$ is contained in every minimal spanning tree for G
- Induction Step:
  - suppose P(k):T_k is contained in a minimal spanning tree T' for G.
  - By definition $\{t_1, t_2,\ldots,t_k\} \subset T'$. If t_{k+1}also belongs to T',then $T_{k+1} \subset T'$ and we have P(k+1) is true.
  - If t_{k+1}does not belongs to T',then $T' \cup \{t_{k+1}\}$must contain a cycle.
  - This cycle would be for some edges $s_1, s_2,\ldots,s_k$ in T'
  - The edges of this cycle cannot all be from T_k or T_{k+1} would contain this cycle.

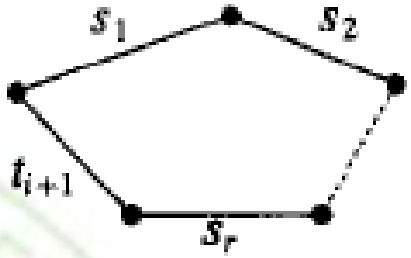


Figure 7.51

- Let s_l be the edge with smallest index l that is not in T_k.when t_{k+1} was chosen by Prim's algorithm, s_l was also available.thus the weight of s_l is at least as large as that of t_{k+1}.
- The spanning tree $(T'-\{s_l\})\cup\{t_{k+1}\}$contains T_{k+1}. The weight of this tree is less than or equal to the weight of T', so it is a minimal tree for G.
- P(k+1) is true. So T_{n-1}=T is contained in a minimal spanning tree and must in fact be that minimal spanning tree.

EXAMPLE

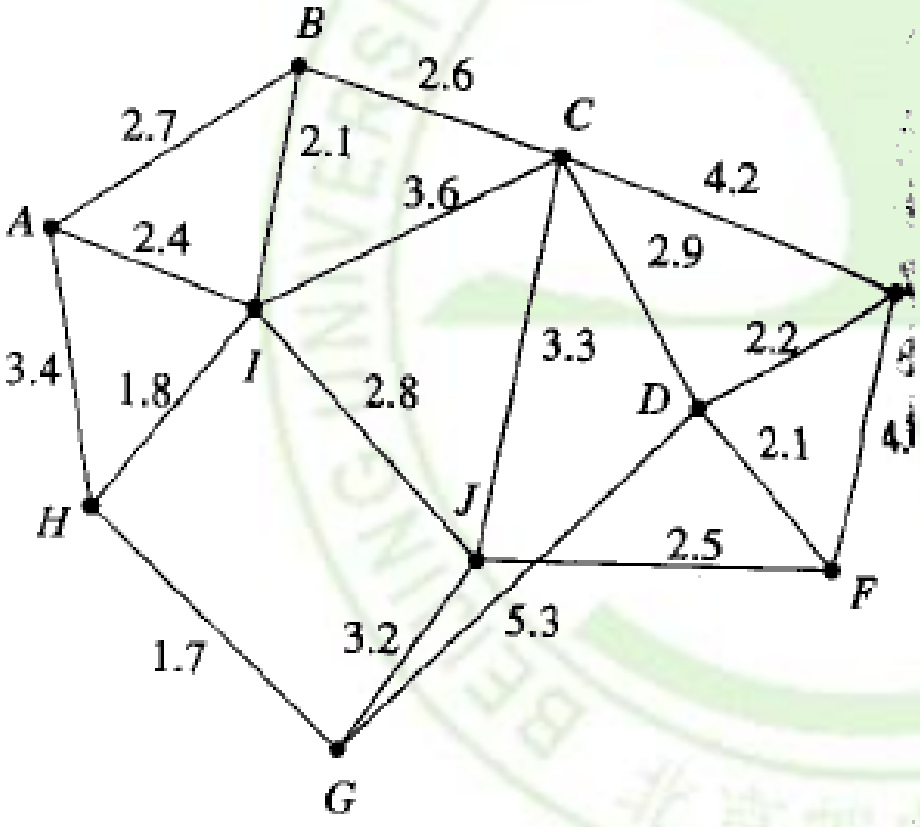


Figure 7.50

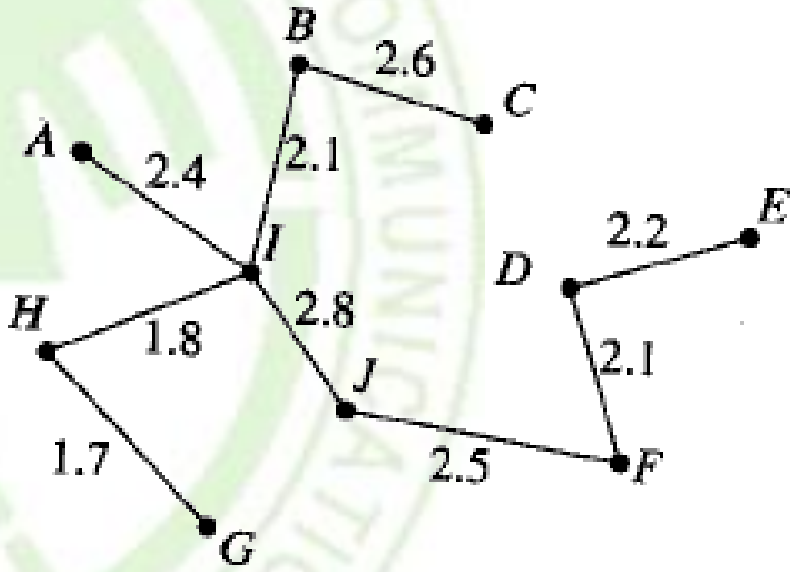


Figure 7.53

Kruskal's Algorithm(1956)

- Procedure Kruskal (G: weighted connected graph with n vertices)
- T:=empty graph
- for i:=1 to n-1
- begin
  - e:=**any edge in G with smallest weight** that does not form a simple circuit when added to T
  - T:= T with e added
- end {T is a minimal spanning tree of G}

EXAMPLE

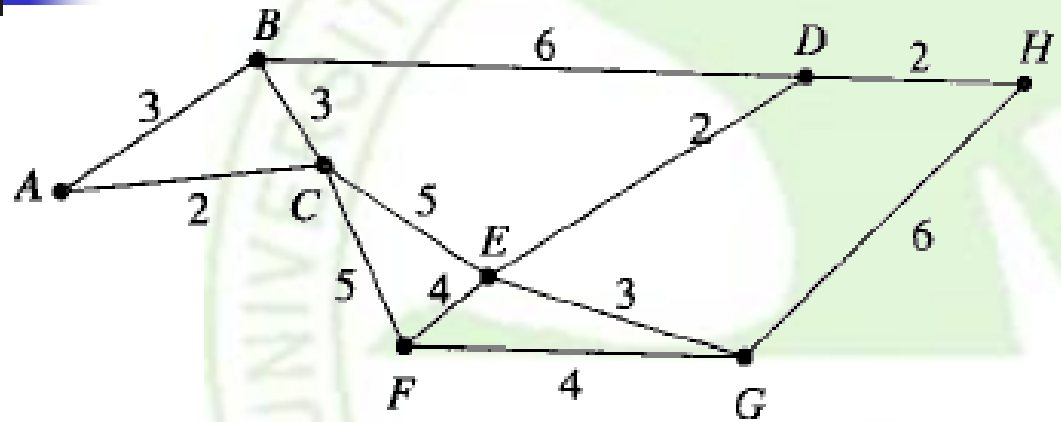


Figure 7.49

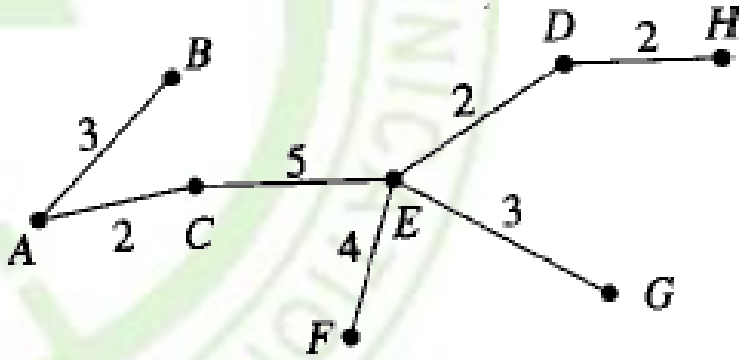


Figure 7.54

- $f(n) = af(n/b) + cn^d$ $O(n^d \log n)$?
- Kruskal's algorithm can be carried out using O(mlogm) operations.
- Prim's algorithm can be carried out using O(mlogn) operations.

- § 11.5
  - 4(Prim's), 8(Kruskal’s), 10.
  - 10: sanning forest of minimum weight.

    Kruskal’s algorithm, do until no such edges.

    Prim's algorithm, when no vertex adjacent to S, grow a new tree from a shortest edge not in S.