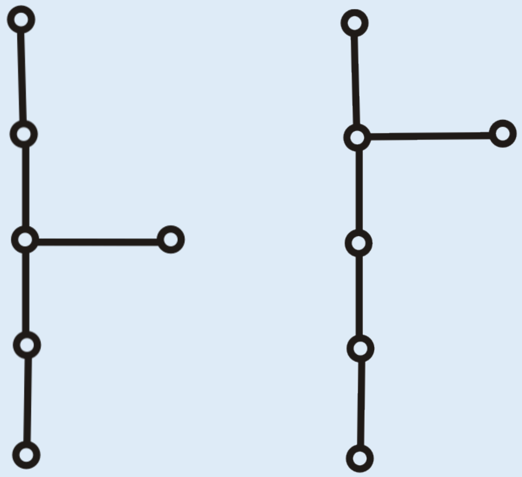
- **Example:** Given an undirected tree T with one vertex of degree **3**, two vertices of degree **2**, and all other vertices being leaves, determine the *number of leaves* and draw all non-isomorphic undirected trees that satisfy these conditions.

- **Solution :** Let the tree have x leaves 、 m edges and n (= 1+2+ x) vertices . The total degree is = $3*1+2*2+1*x$. Using the properties of trees $m=n-1$ and the Handshaking Lemma:

$$2\times m = 2\times(n-1) = 2\times(3+x-1) = 2\times(2+x)$$

$$2\times(2+x) = 1\times 3 + 2\times 2 + x$$



- **Answer:** The tree has **3 leaves**.

The degree sequence of T : **1, 1, 1, 2, 2, 3.**

- **Example:** Draw all non-isomorphic undirected trees of order 6.
- **Solution:** There are 5 edges, and the total degree is 10.

Possible degree sequences:

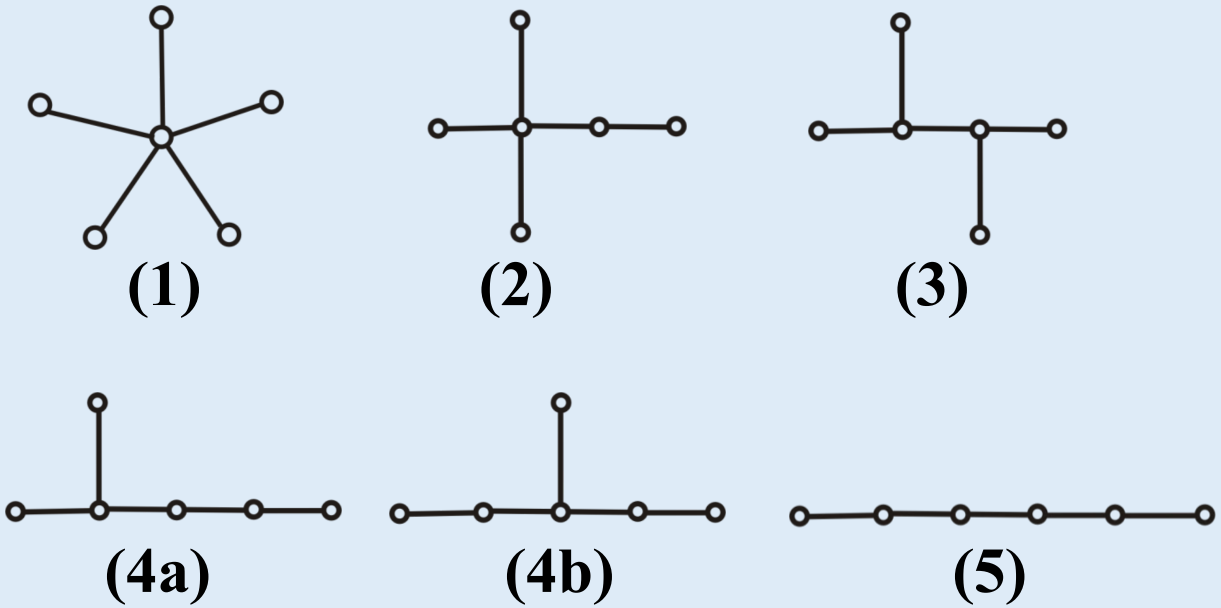
(1) 1,1,1,1,1,5

(2) 1,1,1,1,2,4

(3) 1,1,1,1,3,3

(4) 1,1,1,2,2,3

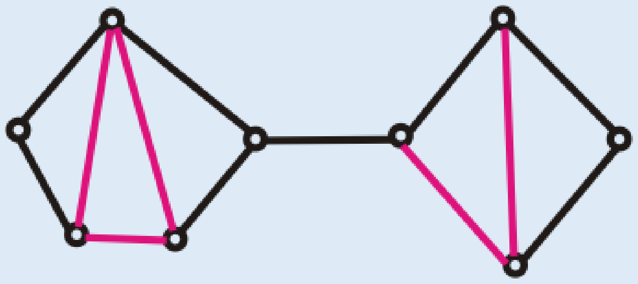
(5) 1,1,2,2,2,2



- 7.1.1 Definition and Properties of Undirected Trees
- 7.1.2 Spanning Trees

↳ Spanning Tree T and Its Co-Tree $\overline{T}$

- Let ***G*** be an undirected connected graph.
  - A ***spanning tree T*** of ***G***: a spanning subgraph of ***G*** that is also a tree.
  - A ***branch*** of the spanning tree ***T***: an edge of ***G*** that belongs to ***T***.
  - A ***chord*** of the spanning tree ***T***: an edge of ***G*** that does **not** belong to ***T***.
  - The ***co-tree*** of the spanning tree ***T***: the subgraph induced by the set of all chords.
- **Note:** The co-tree $\overline{T}$ is not necessarily connected, and it may contain cycles.
- **Example:** The black edges form a spanning tree, and the red edges form the co-tree.



↳ Spanning Tree Existence Theorem

- **Theorem 7.3 :**Every undirected connected graph has a spanning tree.
- **Proof:** Use the **cycle-breaking method**.

  ① If the graph contains no cycles, then the graph itself is a spanning tree.

  ② Otherwise, remove any edge from a cycle — this does not destroy connectivity.

  ③ Repeat this process until there are no cycles remaining. The resulting graph is a spanning tree of the original graph.

- **Corollary 1:** If an undirected connected graph has $\boldsymbol{n}$ vertices and $\boldsymbol{m}$ edges, then $\boldsymbol{m \geq n-1}$.
- **Corollary 2:** If an undirected connected graph has n vertices and m edges, then the co-tree of any spanning tree contains $\boldsymbol{m-n+1}$ edges.

- Each edge ***e*** of a graph ***G*** is assigned a real number ***w*(*e*)**, called the ***weight*** of edge ***e*.**
  - The graph ***G*** together with the weights on its edges is called a ***weighted graph***, denoted by ***G*=<*V*,*E*,*W*>.**
  - Let ***H*** be a subgraph of ***G***, The **weight of *H*** denoted ***W*(*H*)** is the sum of the weights of all edges in ***H*.**
- ***Minimum Spanning Tree*** **(MST):**

  A spanning tree of a weighted graph ***G*** that has the **minimum total weight** among all possible spanning trees.

同济经管 TONGJI SEM

- **Kruskal's Algorithm** — An algorithm for ***finding a minimum spanning tree***. Let G be an undirected connected weighted graph of order n.

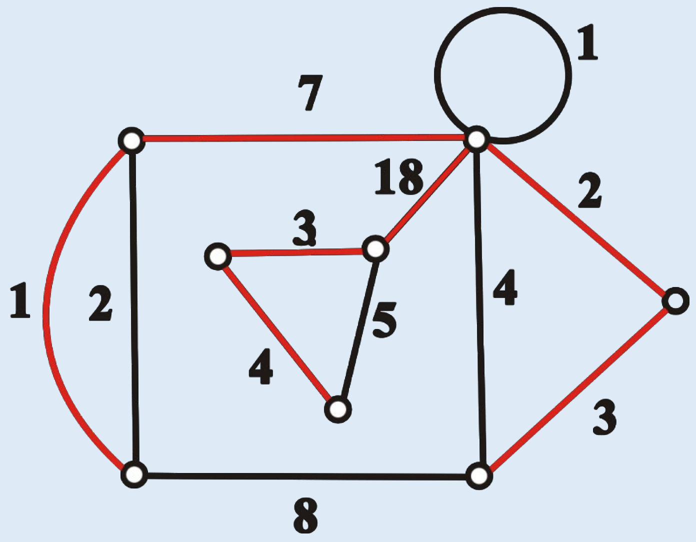
(1) Sort all edges in non-decreasing order of weight (excluding cycles), i.e. $W(e_1) \le W(e_2) \le \ldots \le W(e_m)$.

(2) Initialize $T \leftarrow \varnothing$, $i \leftarrow 1$, $k \leftarrow 0$.

(3) If e_i does not form a cycle with the edges already in T, then set $T \leftarrow T \cup \{e_i\}$, $k \leftarrow k+1$.

(4) If $k<n-1$, then set $i \leftarrow i+1$, and repeat step (3).

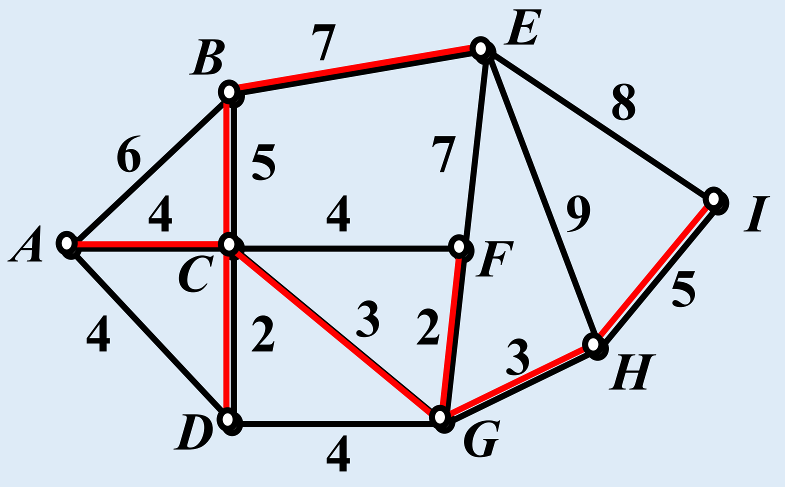
- **Example:** Find a minimum spanning tree of the graph.



$W(T)=38$

同济经管 TONGJI SEM

- **Example:** A telecommunications company has a *fiber-optic network* coverage requirement as shown in the figure. The possible cable routes between buildings and their lengths (in meters) are given in the diagram.
- **Question:** Under the condition that the entire network must be connected, how should the cable routes be selected to ensure that the *total cable length is minimized*?
- **Solution:**



Total Length：

2+2+3+3+4+5+5+7=31(meters)

CAMEA 中国高质量MBA教育认证 AACSB ACCREDITED EQUIS ACCREDITED

Objective :

Key Concepts :

Discrete Mathematics 2025 Spring

同济经管 TONGJI SEM

魏可佶 kejiwei@tongji.edu.cn

CAMEA 中国高质量MBA教育认证 AACSB ACCREDITED EQUIS ACCREDITED

- 7.1 Undirected Trees
- 7.2 Rooted Trees and Their Applications

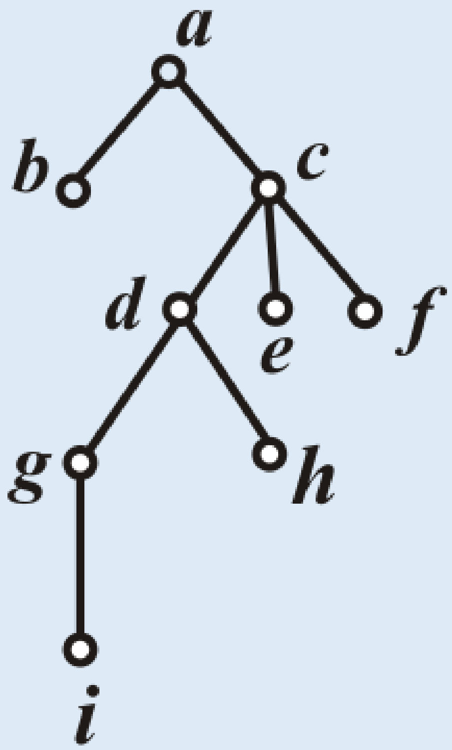
7.2 Rooted Trees and Their Applications

- 7.2.1 Rooted Trees and Their Classifications
- 7.2.2 Optimal Trees and the Huffman Algorithm
- 7.2.3 Optimal Prefix Codes
- 7.2.4 Traversals of Rooted Trees and Their Applications
  - Inorder Traversal, Preorder Traversal, and Postorder Traversal
  - Polish Notation and Reverse Polish Notation

- ***Directed Tree***: A directed graph that becomes an undirected tree when edge directions are ignored.
- ***Rooted Tree***: A nontrivial directed tree in which exactly one vertex has in-degree **0** (***the root***), and all other vertices have in-degree **1**.
- **Root:** The vertex with **in-degree 0** in a directed tree.
- **Leaf:** A vertex with in-degree **1** and ***out-degree 0*** in a directed tree.
- **Internal Vertex:** A vertex with in-degree **1** and ***out-degree greater than 0*** in a directed tree.
- **Branching Vertex:** A general term referring to both the ***root and internal vertices***.
- **Level of a Vertex *v*:** The *length of the path* from the root to vertex ***v***.
- **Height of the Tree:** The *maximum level* among all vertices in the directed tree.

↳ Drawing a Rooted Tree(e.g.)

- **Example**: (as shown in the diagram on the right):
  - ***a*** is the root.
  - ***b*,*e*,*f*,*h*,*I*** are the leaves.
  - ***c*,*d*,*g*** are internal vertices.
  - ***a*,*c*,*d*,*g*** are branching vertices.
  - ***a*** is at level 0; level 1 contains ***b*,*c***; level 2 contains ***d*,*e*,*f***;level 3 contains ***g*,*h***; level 4 contains ***i***.
  - The height of the tree is 4.



↳ Viewing a Rooted Tree as a Family Tree

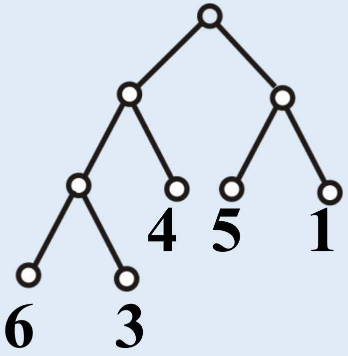
- If vertex a is adjacent to vertex b, then b is called the ***child*** of a, and a is the ***parent*** of b.
- If b and c are both children of the same vertex, they are called ***siblings***.
- If $a \neq b$ and a can reach b, then a is an ***ancestor*** of b, and b is a ***descendant*** of a.
- Let v be a vertex in the rooted tree that is not the root. The subgraph induced by v and all its descendants is called the ***subtree rooted*** at v.
- If a fixed order is assigned to the vertices at each level of a rooted tree, it is called an ***ordered tree***.

- Rooted trees can be *further classified* based on the number of child nodes per parent, the order of child nodes, the regularity of the structure, and the completeness of the tree.
  - ***r-ary Tree***: A rooted tree in which each branching vertex has at most *r* children.
  - ***r-ary Regular Tree***: A rooted tree in which each branching vertex has exactly *r* children.
  - ***Complete r-ary Regular Tree***: An r-ary regular tree in which all leaves are at the same level.
  - ***Ordered r-ary Tree***: An r-ary tree in which a specific order is assigned to the children of each vertex.
  - ***Ordered r-ary Regular Tree***: An ordered tree in which each branching vertex has exactly *r* ordered children.
  - ***Ordered Complete r-ary Regular Tree***: An ordered r-ary regular tree in which all leaves are at the same level.

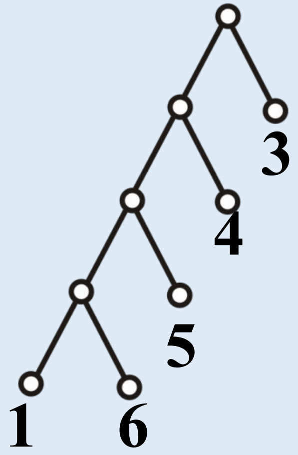
7.2 Rooted Trees and Their Applications

- 7.2.1 Rooted Trees and Their Classifications
- 7.2.2 Optimal Trees and the Huffman Algorithm
- 7.2.3 Optimal Prefix Codes
- 7.2.4 Traversals of Rooted Trees and Their Applications
  - Inorder Traversal, Preorder Traversal, and Postorder Traversal
  - Polish Notation and Reverse Polish Notation

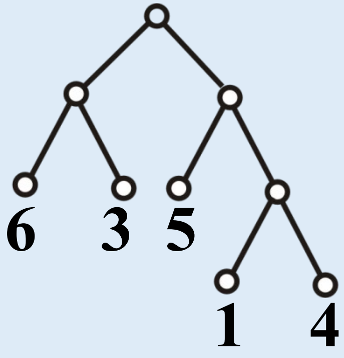
- **Definition7.1**：Let T be a binary tree with ***t leaves*** $v_1, v_2, \ldots, v_t$, and let the weights of the leaves be $w_1, w_2, \ldots, w_t$.
  - The **weight** of tree T is defined as: $W(t)=\sum_{i=1}^{t} w_i l(v_i)$, where $l(v_i)$ is the level (depth) of leaf v_i.
  - Among all binary trees with t leaves and weights $w_1, w_2, \ldots, w_t$, the one with the minimum total weight $W(T)$,) is called the *optimal binary tree*.
- **Example**：



$W(T_1)=47$



$W(T_2)=54$



$W(T_3)=43$

- **Huffman Algorithm:** Constructing the ***Optimal Binary (Huffman) Tree***.

  Given real numbers $w_1, w_2, \ldots, w_t$:

  ①Create t leaves, each assigned a weight of $w_1, w_2, \ldots, w_t$ respectively.

  ②Among all vertices with in-degree **0** (not necessarily leaves), select the two with the smallest weights. Create a new branching node with these two as its children. The weight of the new node is the sum of its two children's weights.

  ③Repeat step ② until there is only one vertex with in-degree **0** remaining.

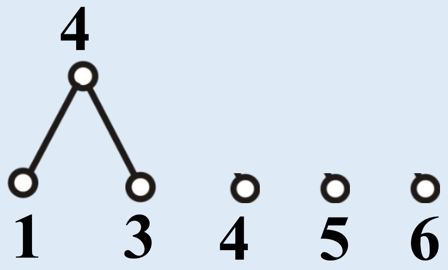
- The total weight $W(T)$ is the sum of the weights of all branching nodes. The resulting binary tree is the ***optimal binary tree***, having the ***minimum weighted path length***.

↳ Static Huffman Coding Algorithm(e.g.)

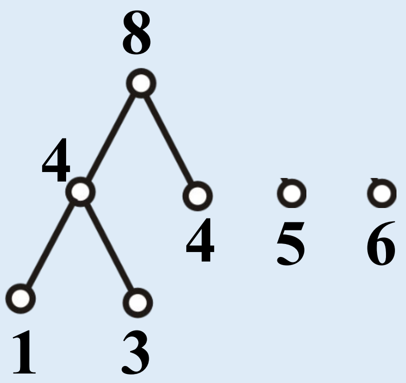
- **Example:** Find the optimal binary tree with weights 1, 3, 4, 5, and 6, and compute its total weight.
- **Solution:**

  Steps **(a)** through **(e)** show the computation process using the Huffman algorithm.

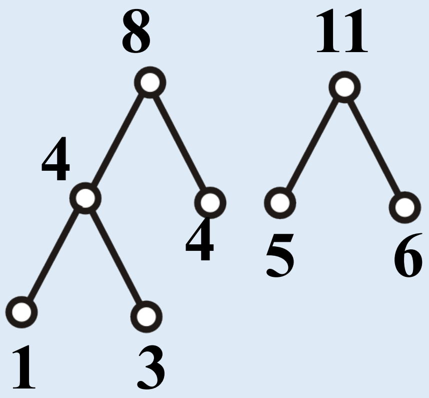
(a)



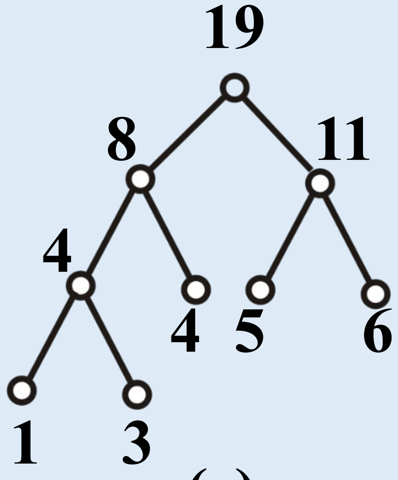
(b)



(c)



(d)



(e)

$W(T)$=4+8+1
1+19=42

- 7.2.1 Rooted Trees and Their Classifications
- 7.2.2 Optimal Trees and the Huffman Algorithm
- 7.2.3 Optimal Prefix Codes
- **7.2.4 Traversals of Rooted Trees and Their Applications**
  - Inorder Traversal, Preorder Traversal, and Postorder Traversal
  - Polish Notation and Reverse Polish Notation

Definition7.2:

- Let $\beta = \alpha_1\alpha_2 \ldots \alpha_{n-1}\alpha_n$ be a string of length n, and let $\alpha_1\alpha_2 \ldots \alpha_j$ （$1 \leq j \leq n$） be a *prefix* of β （with length j ）.
- If no two distinct strings β_i, β_j ($i \neq j$) in a non-empty set **B**= {β_1, β_2, …, β_m} are prefixes of one another, then B is called a ***prefix code***.
- A prefix code that uses only two symbols (e.g., 0 and 1) is called a ***binary prefix code***.

Examples :

- **{0,10,110, 1111}, {10,01,001,110}** are binary prefix codes.
- **{0,10,010, 1010}** is **not** a prefix code.

↳ Generating Binary Prefix Codes Using a Binary Tree

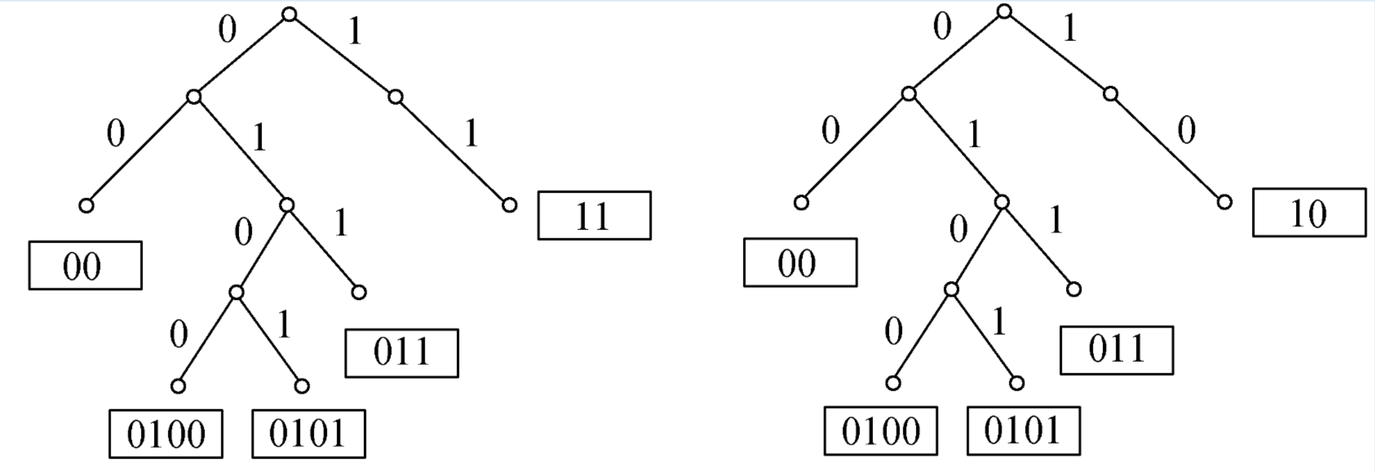
- A binary tree can generate a set of ***binary prefix codes***:

①For each branching node, if it has two associated edges, label the **left** edge with **0** and the **right** edge with **1**.

②If a branching node has only **one** associated edge, you may label it with either **0** (interpreted as left) or **1** (interpreted as right).

③For each leaf, record the sequence of digits along the path from the root to that leaf. The resulting strings form a ***binary prefix code***.

- **Examples：**



Prefix Code：

{00, 11, 011, 0100 , 0101},
{00, 10, 011, 0100 , 0101}

- **Example:** In communication, suppose the frequency (%) of octal digits is given as follows: 0: 30, 1: 20, 2: 15, 3: 10, 4: 10, 5: 5, 6: 5,7: 5.
  - Using binary prefix codes, find the binary prefix code that transmits the digits with the **fewest number of binary bits** (this is called the **optimal prefix code**).
  - Also, calculate the **total number of binary digits** needed to transmit 10,000 octal digits following the given frequency distribution using the optimal prefix code.
  - The **total number of binary digits** required if a **fixed-length code** (of length 3) is used for each octal digit instead.

↳ Examples of Binary Prefix Code Applications

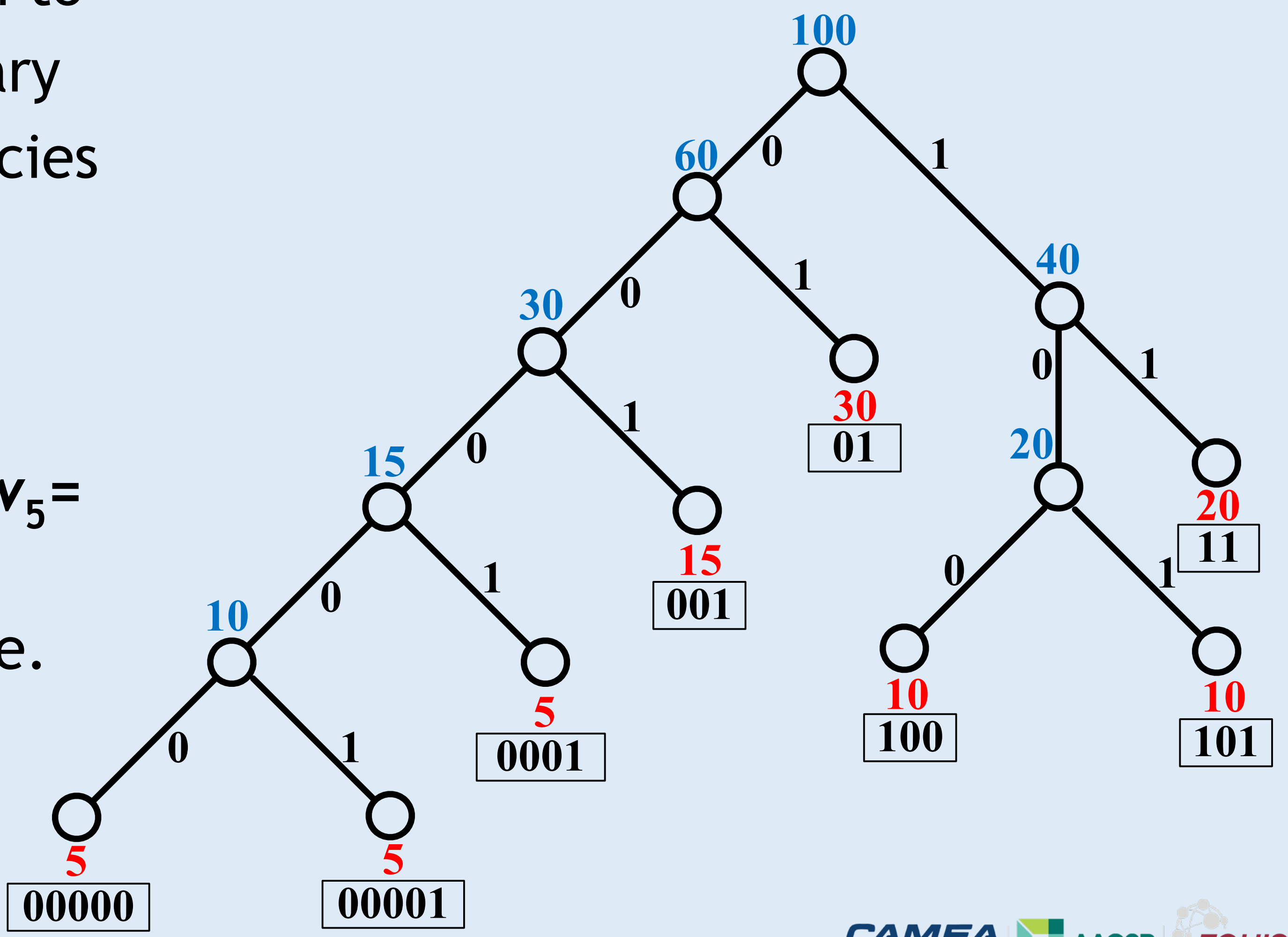
- **Solution:**

① Use the Huffman algorithm to construct the optimal binary tree, treating the frequencies (multiplied by 100) as the weights.

Let:

$w_1=5, w_2=5, w_3=5, w_4=10, w_5=10, w_6=15, w_7=20, w_8=30.$

② Construct the Huffman tree.



↳ Examples of Binary Prefix Code Applications

- **Solution:**

③ Assign codes. For characters with the same frequency, their codes or fixed-length codes can be interchanged without affecting the **validity** or **optimality** of the encoding.

Digit	0	1	2	3	4	5	6	7
Frequency	30%	20%	15%	10%	10%	5%	5%	5%
Huffman Code	01	11	001	100	101	0001	00001	00000
Code Length	2	2	3	3	3	4	5	5
Fixed-Length (Length 3) Code	000	001	010	011	100	101	110	111

- **Solution:**

④Calculate the number of binary digits required to ***transmit 10,000 octal digits*** according to the given frequency distribution:

- ***Huffman encoding*** **(optimal prefix code):**

  $W(T)=(3000+2000)*2+(1500+1000+1000)*3+500*4+(500+500)*5=27500.$

- **Fixed-length encoding (length 3):**

  $10000*3=30000.$

- ***Bits saved*** **by using the optimal prefix code:**

  30000-27500=2500 , approximately 8.33% .

- **Traversing (Touring) a *Rooted Tree*:**
  Visiting each vertex of a rooted tree exactly once.
- **Traversal methods for a *binary ordered tree*:**
  (1) *Inorder traversal*: Left subtree → **Root** → Right subtree
  (2) *Preorder traversal*: **Root** → Left subtree → Right subtree
  (3) *Postorder traversal*: Left subtree → Right subtree → **Root**
  - When the binary ordered tree is not a full (regular) tree, the left or right subtree may be absent.

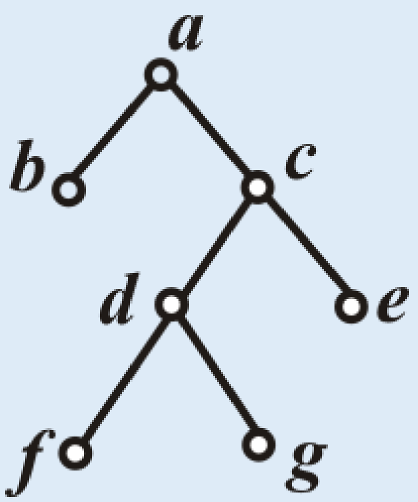
↳ Preorder, inorder, and postorder of an ordered rooted tree(e.g.)

- **Example**: The traversal results of the tree on the right are:

  (1) **Inorder traversal:** $b\ \underline{a}\ (f\ \underline{d}\ g)\ \underline{c}\ e$

  (2) **Preorder traversal:** $\underline{a}\ b\ (\underline{c}\ (\underline{d}\ f\ g)\ e)$

  (3) **Postorder traversal:** $b\ ((f\ g\ \underline{d})\ e\ \underline{c})\ \underline{a}$



- **Note:** Underlined nodes are (sub)tree roots, and each pair of parentheses encloses a subtree.

↳ Expression Tree Construction Rules

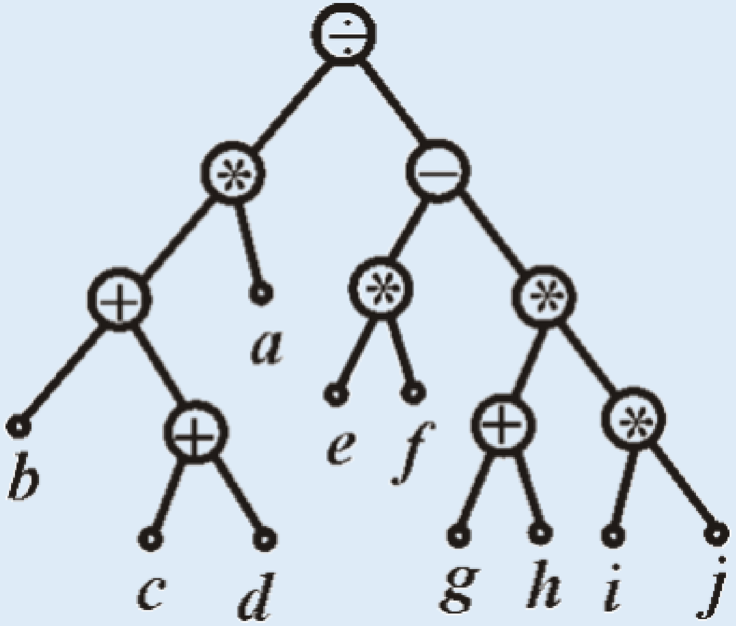
① Place an **operator** at each branching node.

② A **binary operator** node has two children. The operation applies to the subexpressions represented by the left and right subtrees. By convention, the **minuend** or **dividend** is placed in the **left subtree**.

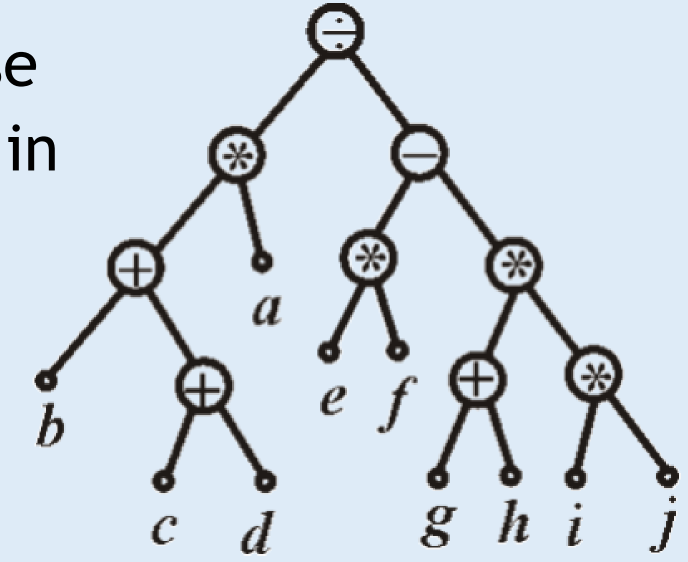
③ A **unary operator** node has only one child. The operation applies to the subexpression represented by the subtree rooted at that child.

④ **Numbers** and **variables** are placed at the **leaves** of the tree.

- **Example:**
  - The diagram on the right is a binary ordered tree representing the expression $((b+(c+d))*a)\div((e*f)-(g+h)*(i*j))$
  - The **inorder traversal** of this binary ordered tree yields the **original expression**.



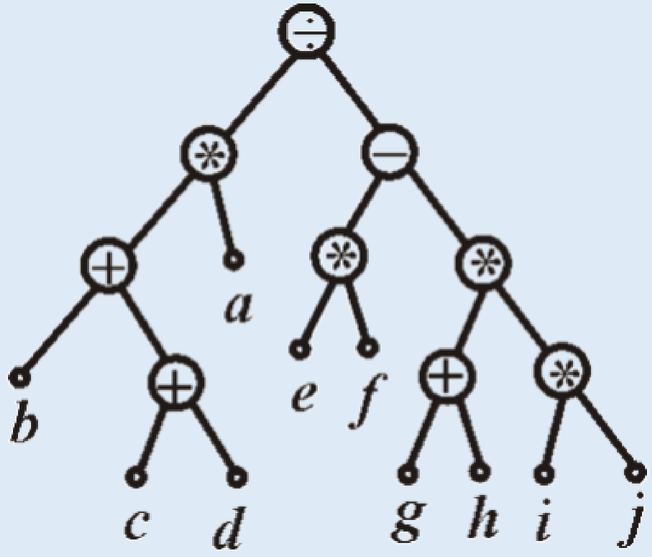
- ***Polish Notation* (Prefix Notation):**
  Traverse the binary ordered tree representing the expression using **preorder traversal**, and omit all parentheses.
- ***Reverse Polish Notation* (Postfix Notation):**
  Traverse the binary ordered tree using **postorder traversal**, and omit all parentheses.
- **Example:** The Polish notation and Reverse Polish notation for the expression shown in the diagram on the right are:
  - Polish Notation (Prefix):
    $\div * + b + c d a - * e f * + g h * i j$
  - Reverse Polish Notation (Postfix):
    $b c d + + a * e f * g h + i j * * - \div$



7.2.4 Traversals of Rooted Trees and Their Applications

↳ How computers process Polish and reverse Polish notation

- **Computer Processing Method:**
  - For an expression represented in **Polish notation**, computation starts from the **root node**—the operator is encountered first, followed by its operands.
  - For an expression in **Reverse Polish notation**, computation begins from the **leaf nodes** and proceeds **upward step by step** until reaching the root node.



- Polish Notation (Prefix):
  $\div * * + b + c\, d\, a - * e f * + g h * i j$
- Reverse Polish Notation (Postfix):
  $b\, c\, d + + a * e f * g h + i j * * - \div$

Objective :

Key Concepts :

- 7.1 Undirected Trees
- 7.2 Rooted Trees and Their Applications