Lecture4
Tree, Binary Tree, Heap

陆清怡

2022.11.4

991《数据结构与算法》考纲

5、树

（1）树的概念和性质。

（2）二叉树的概念、性质和实现。

（3）二叉树的顺序存储结构和链式存储结构。

（4）遍历二叉树。

（5）树和森林的存储结构、遍历。

（6）堆与优先队列。

（6）二叉排序树。

（7）平衡二叉树。

（8）哈夫曼(Huffman)树和哈夫曼编码。

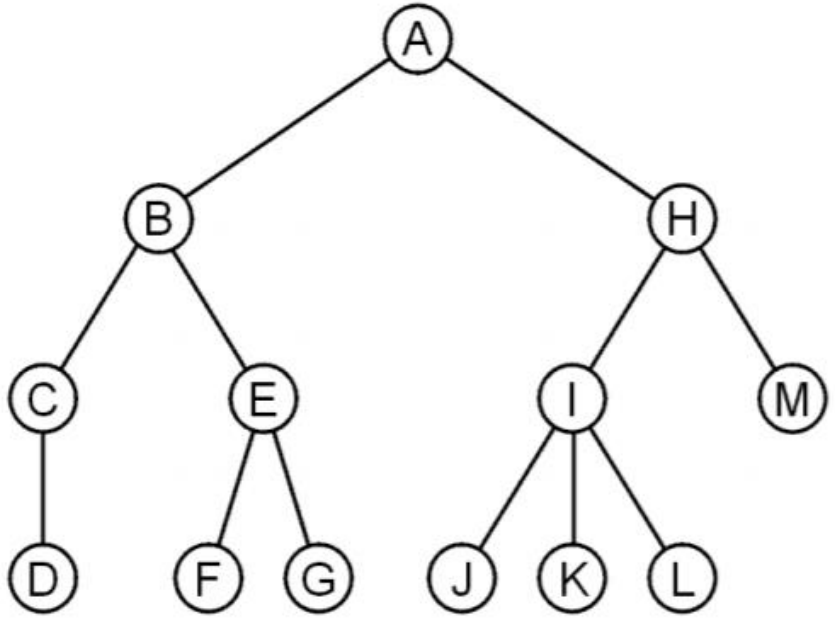
Tree

- Definition
- Implementation
- Operations
- Traversal
- Forest
- Binary Tree
- Perfect binary tree
- Complete binary tree

Definition

A recursive definition of a tree:

- A degree-0 node is a tree
- A node with degree n is a tree if it has n children and all of its children are disjoint trees (*i.e.*, with no intersecting nodes)



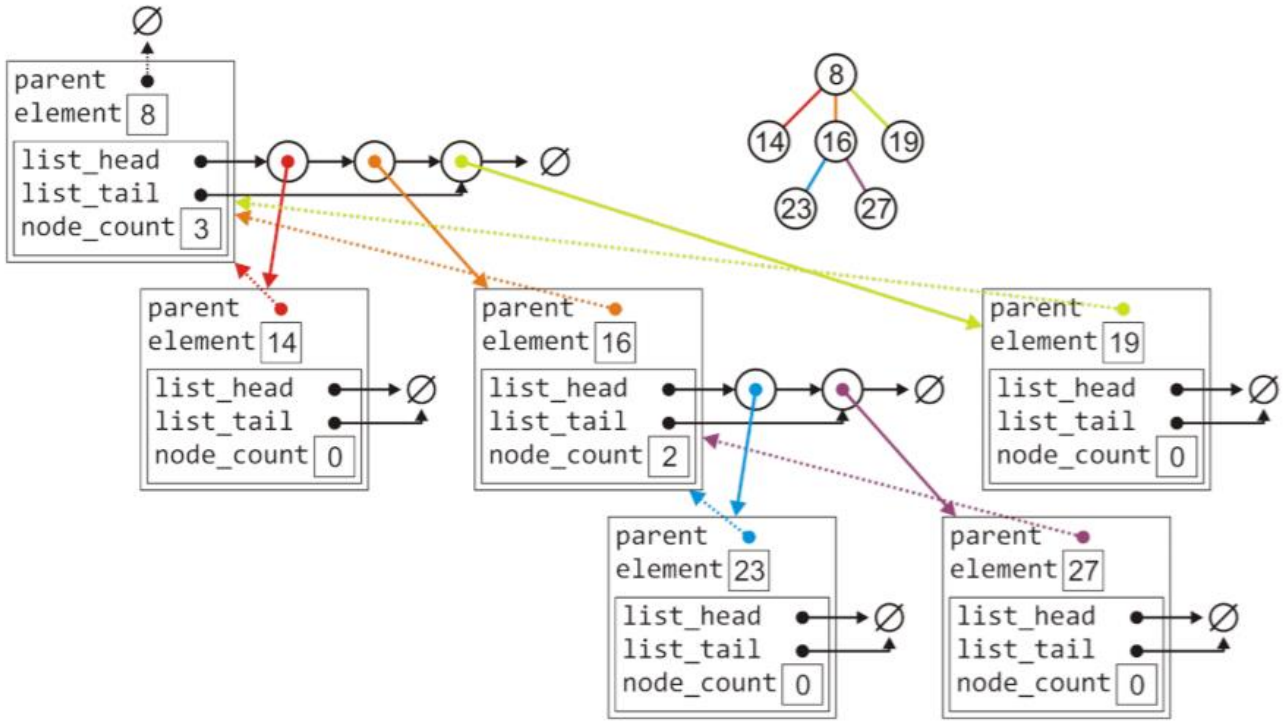
- Root
- Child
- Parent
- Degree
- Sibling
- Leaf node
- Internal node
- Path
- Depth
- Height
- Ancestor
- Descentor

Implementation-Child List Pointer Tree

We can implement a general tree by using a class which:

- Stores an element
- Stores the children in a list

The tree with six nodes would be stored as follows:



Implementation-Parent Pointer Tree

A *parent pointer tree* is a tree where each node only keeps a reference to its parent node

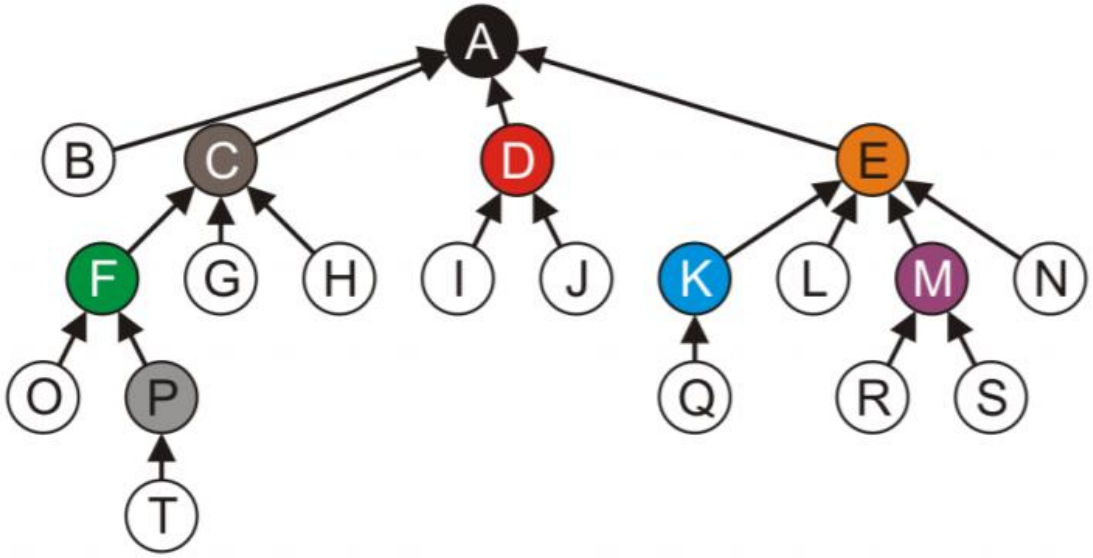
This requires significantly less memory than our general tree structure, as no data structure is required to track the children

Implementation-Parent Pointer Tree

Store the index of the parent in each node

- The root node, wherever it is, points to itself

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	2	2	2	3	3	4	4	4	4	5	5	10	12	12	15
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T



Operations

Operations on a tree include:

- Accessing the root:
- Given an object in the container:
  - Access the parent of the current object
  - Find the degree of the current object
  - Get a reference to a child,
  - Attach a new sub-tree to the current object
  - Detach this tree from its parent

Operations-size()

```
template <typename Type>
int Simple_tree<Type>::size() const {
    int s = 1;

    for (
        Single_node<Simple_tree *> *ptr = children.head();
        ptr != nullptr;
        ptr = ptr->next()
    ) {
        s += ptr->retrieve()->size();
    }

    return s;
}
```

Operations-height()

```
template <typename Type>
int Simple_tree<Type>::height() const {
    int h = 0;

    for (
        Single_node<Simple_tree *> *ptr = children.head();
        ptr != nullptr;
        ptr = ptr->next()
    ) {
        h = std::max( h, 1 + ptr->retrieve()->height() );
    }

    return h;
}
```

Traversal

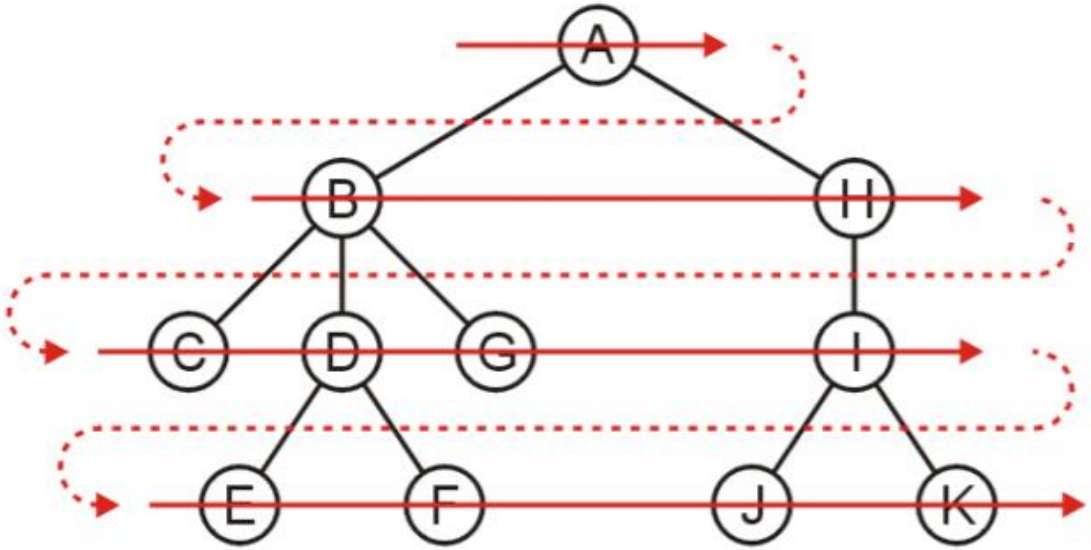
Two types of traversals

- Breadth-first traversal
- Depth-first traversal

Traversal-BFS

Breadth-first traversals visit all nodes at a given depth before descending a level

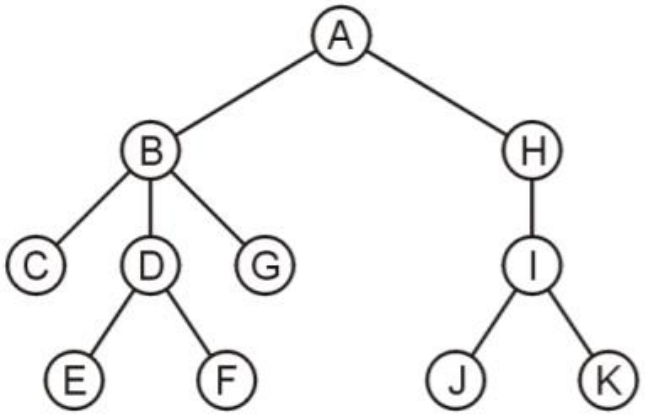
- Order: A B H C D G I E F J K



Traversal-BFS

The easiest implementation is to use a queue:

- Place the root node into a queue
- While the queue is not empty:
  - Pop the node at the front of the queue
  - Push all of its children into the queue



Computational complexity

- Run time is $\Theta(n)$
- Space: maximum nodes at a given depth, $O(n)$

Traversal-DFS

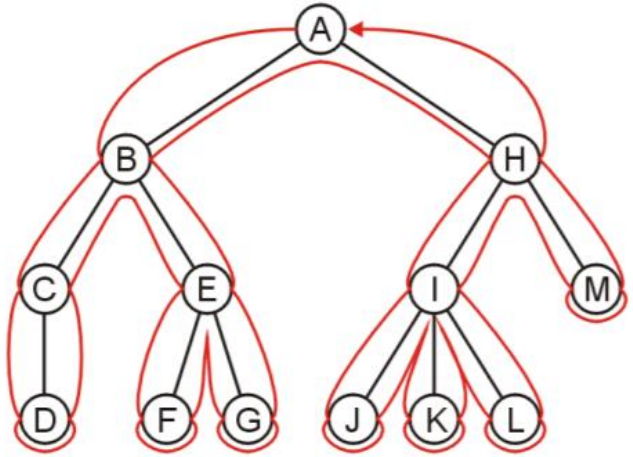
A backtracking algorithm for stepping through a tree:

- At any node, proceed to the first child that has not yet been visited
- If we have visited all the children (of which a leaf node is a special case), backtrack to the parent and repeat this process

We end once all the children of the root are visited

Each node is visited multiple times in such a scheme

- First time: before any children
- Last time: after all children, before backtracking



Alternatively, we can use a stack:

- Create a stack and push the root node onto the stack
- While the stack is not empty:
  - Pop the top node
  - Push all of the children of that node to the top of the stack in reverse order

Computational complexity of DFS using stack

- Run time is $\Theta(n)$
- The objects on the stack are all unvisited siblings from the root to the current node
  - If each node has a maximum of two children, the memory required is $\Theta(h)$: the height of the tree

Traversal-DFS-Pre visit v.s. Post visit

Implementation with recursion:

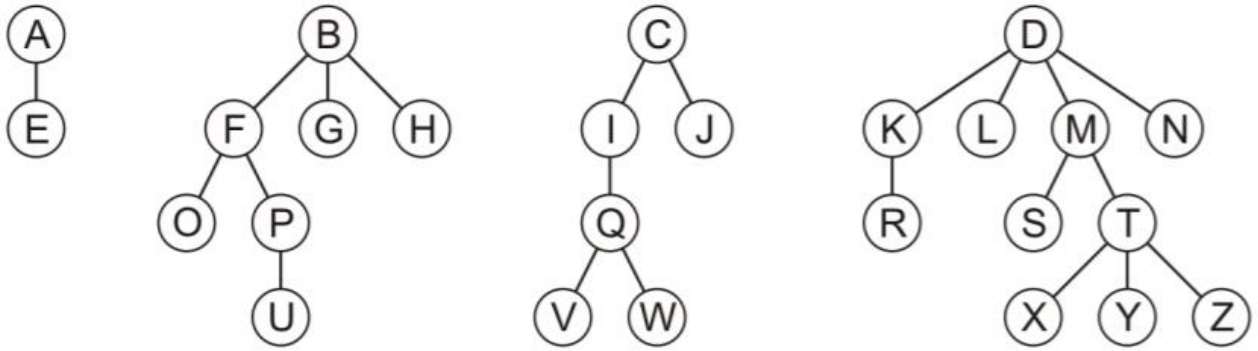
```
template <typename Type>
void Simple_tree<Type>::depth_first_traversal() const {
    // Perform pre-visit operations on the element
    std::cout << element << ' ';

    // Perform a depth-first traversal on each of the children
    for (
        Single_node<Simple_tree *> *ptr = children.head();
        ptr != 0; ptr = ptr->next()
    ) {
        ptr->retrieve()->depth_first_traversal();
    }

    // Perform post-visit operations on the element
    std::cout << element << ' ';
}
```

Forest

A rooted forest is a data structure that is a collection of disjoint rooted trees

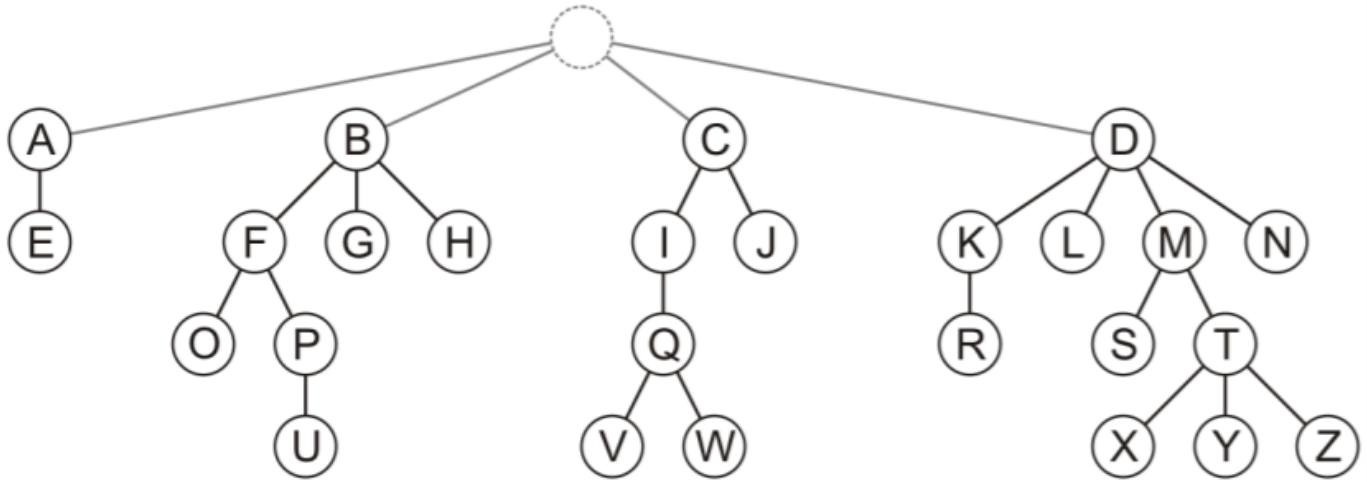
Note that:

- Any tree can be converted into a forest by removing the root node
- Any forest can be converted into a tree by adding a root node that has the roots of all the trees in the forest as children



Forest

Traversals on forests can be achieved by treating the roots as children of a notional root



Pre-order traversal: A E B O F P U G H C I Q V W J D K R L M S T X Y Z N

Post-order traversal: E A O U P F G H B V W Q I J C R K L S X Y Z T M N D

Breadth-first traversal: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Binary Tree-Definition

- Binary tree
- Full node v.s. Leaf node v.s. Neither
- Empty node v.s. Null sub-tree
- Full binary tree

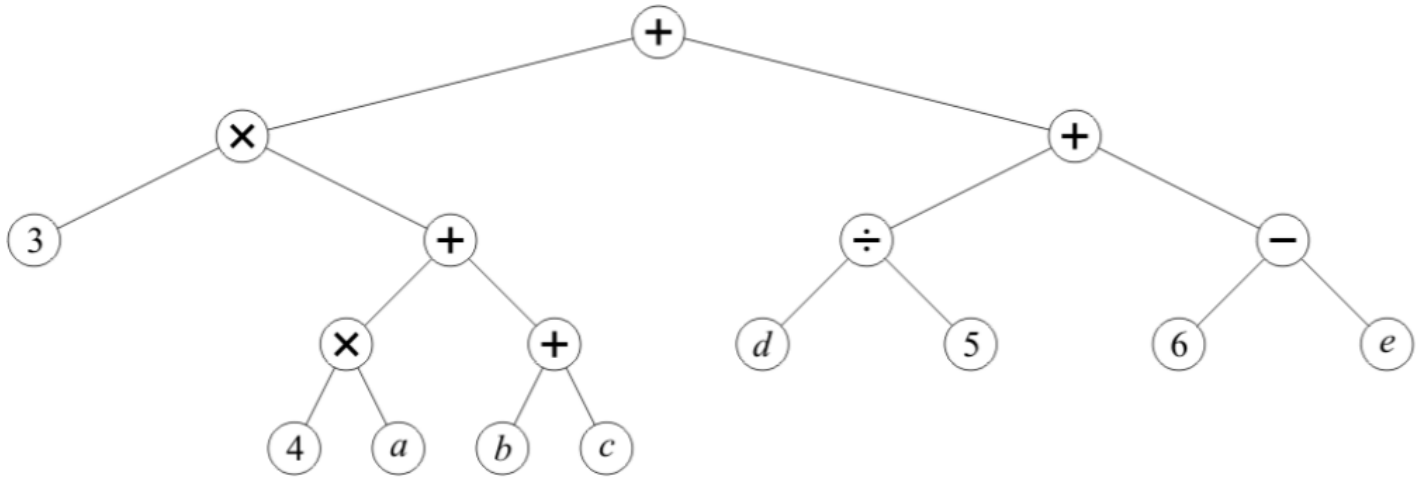
Binary tree-Implementation

The binary node class is similar to the single node class:

```
template <typename Type>
class Binary_node {
    protected:
        Type element;
        Binary_node *left_tree;
        Binary_node *right_tree;

    public:
        Binary_node( Type const & );

        Type retrieve() const;
        Binary_node *left() const;
        Binary_node *right() const;

        bool is_leaf() const;
        int size() const;
}
```

Binary tree-Three traversal ways

- Pre-order traversal
- Post-order traversal
- In-order traversal
  - Left sub-tree
  - Current node
  - Right sub-tree



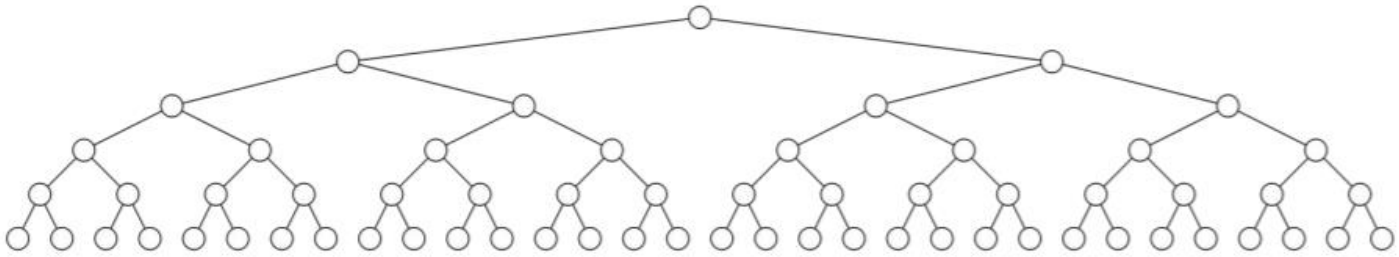
Binary tree-Perfect Binary Tree

Standard definition:

- A perfect binary tree of height h is a binary tree where
  - All leaf nodes have the same depth h
  - All other nodes are full

Recursive definition:

- A binary tree of height $h = 0$ is perfect
- A binary tree with height $h > 0$ is a perfect if both sub-trees are prefect binary trees of height $h - 1$



Binary tree-Perfect Binary Tree

Four theorems of perfect binary trees:

- A perfect binary tree of height h has $2^{h+1} - 1$ nodes
- The height is $\Theta(\ln(n))$
- There are 2^h leaf nodes
- The average depth of a node is $\Theta(\ln(n))$

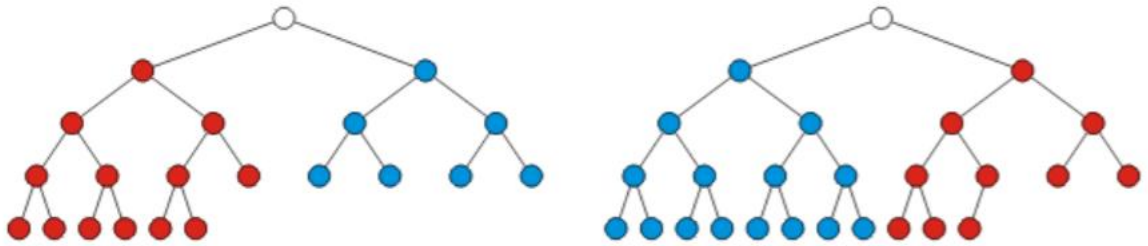
Binary tree-Complete Binary Tree

A complete binary tree filled at each depth from left to right

- Identical order to that of a breadth-first traversal

Recursive definition: a binary tree with a single node is a complete binary tree of height $h = 0$ and a complete binary tree of height h is a tree where either:

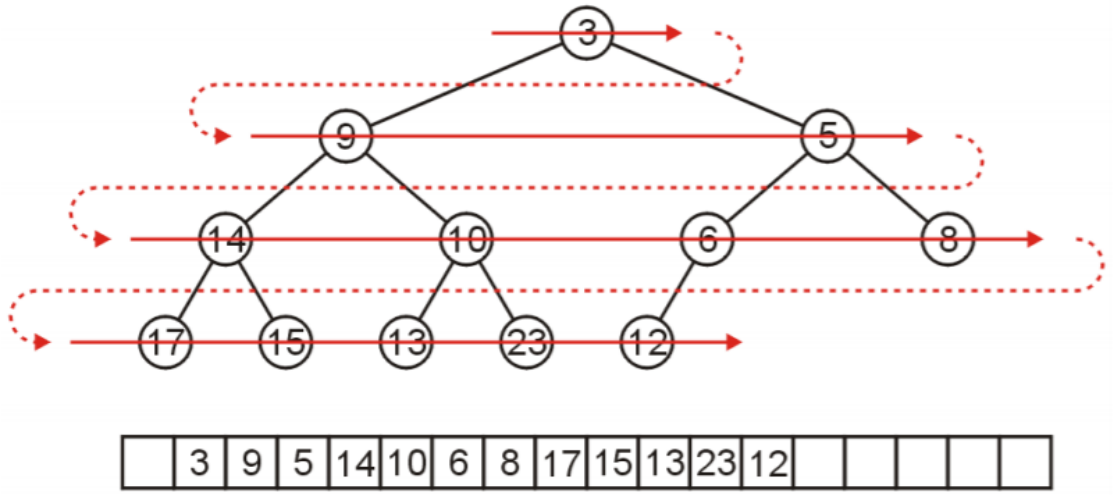
- The left sub-tree is a **complete tree** of height $h - 1$ and the right sub-tree is a **perfect tree** of height $h - 2$, or
- The left sub-tree is **perfect tree** with height $h - 1$ and the right sub-tree is **complete tree** with height $h - 1$



Binary tree-Complete Binary Tree

We are able to store a complete tree as an array

- Traverse the tree in breadth-first order, placing the entries into the array



Leaving the first entry blank yields a bonus:

- The children of the node with index k are in $2k$ and $2k + 1$
- The parent of node with index k is in $k \div 2$

历年真题

一棵完全二叉树（Complete binary tree）的节点数量为 n，则该树中节点的平均深度为 $\Theta(\log_2(n))$。（　　）

For a rooted binary tree, n_0 denotes the number of nodes with degree 0, n_1 denotes the number of nodes with degree 1 and n_2 denotes the number of nodes with degree 2. Then we have the property that $n_0 = n_2 + 1$. （ ）

在一棵有根二叉树中，n_0代表度为 0 的节点的个数，n_1代表度为 1 的节点的个数，n_2代表度为 2 的节点的个数，那么等式 $n_0 = n_2 + 1$ 成立。（ ）

The number of nodes in a tree can be more than twice the number of leaf nodes. （ ）

一棵树的节点个数有可能大于叶节点个数的两倍。（ ）

历年真题

节点数量为 n 的完美二叉树（Perfect binary tree）高度是（　　）。注意：此处定义二叉树的高度为 h，空树 h=-1，一个节点的树 h=0，两层树 h=1；　ln()为自然对数。

A. $\log_2(n+1)-1$　　B. $\log_2(n-1)+1$　　C. $\ln(n+1)-1$　　D. $\ln(n-1)+1$

设高度为 h 的二叉树上只有度（degree）为 0 和度为 2 的结点，则此类二叉树中所包含的结点数至少为（　　）。 注意：此处定义二叉树的高度为 h，空树 h=-1，一个节点的树 h=0，两层树 h=1。

A. 2h-1　　B. 2h+1　　C. h+2　　D. h+1

下面关于二叉树的叙述中，正确的是（　　）

A. 若有一个结点（Node）是二叉树中某个子树的中序遍历（In-order traversal）结果序列的最后一个结点，则它一定是该子树的前序遍历（Pre-order traversal）结果序列的最后一个结点。
B. 若有一个结点是二叉树中某个子树的前序遍历结果序列的最后一个结点，则它一定是该子树的中序遍历结果序列的最后一个结点。
C. 若有一个叶子结点（Leaf node）是二叉树中某个子树的中序遍历结果序列的最后一个结点，则它一定是该子树的前序遍历结果序列的最后一个结点。
D. 若有一个叶子结点是二叉树中某个子树的前序遍历结果序列的最后一个结点，则它一定是该子树的中序遍历结果序列的最后一个结点。

历年真题

Consider a 4-degree tree with 20 4-degree nodes, 10 3-degree nodes, 1 2-degree node and 10 1-degree nodes, then how many leaf nodes are there in the tree?
已知一个度为 4 的树中，有 20 个度为 4 的节点，10 个度为 3 的节点，1 个度为 2 的节点和 10 个度为 1 的节点。那么这棵树上有几个叶节点？

A. 41
B. 82
C. 113
D. 122

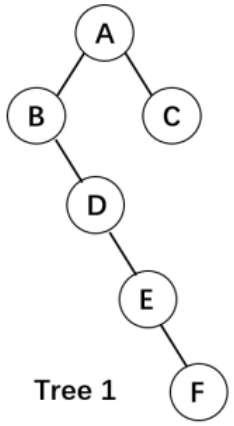
If a complete binary tree has a number of 1880 nodes, what is the height of the tree?
一个完全二叉树有 1880 个节点，那么这棵树的高度是：

A. 9
B. 10
C. 11
D. 12

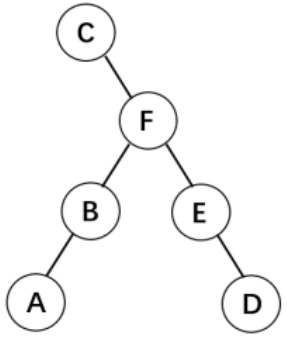
历年真题

对于图 1 中所示的两个二叉树 Tree 1 和 Tree2，下列哪两种深度优先遍历会产生一致的节点输出序列？

A. Tree 1 后序遍历，Tree 2 后序遍历
B. Tree 1 后序遍历，Tree 2 中序遍历
C. Tree 1 先序遍历，Tree 2 后序遍历
D. Tree 1 中序遍历，Tree 2 先序遍历



Tree 1



Tree 2

历年真题

满二叉树的所有中间节点都有两个孩子节点。一个有 500 个叶子节点的满二叉树有多少个中间节点？（ ）

A. 250

B. 499

C. 500

D. 501

E. 1,000

以下哪一个关于树的描述是错误的？（ ）

A. 非叶节点有可能没有孩子

B. 在一棵树中加一条边会形成一个环

C. 一棵树中并非每个节点都有父亲节点

D. 一棵包含一个节点的树高度为 0

历年真题

请简述为什么完全二叉树的高度为 $\lfloor \log_2(n) \rfloor$，其中 n 为节点数量，$\lfloor \cdot \rfloor$ 是取整符号。

历年真题

请简述为什么完全二叉树的高度为 $\lfloor \log_2(n) \rfloor$，其中 n 为节点数量，$\lfloor \cdot \rfloor$ 是取整符号。

Proof:

- Base case:
  - When $n = 1$ then $\lfloor \lg(1) \rfloor = 0$ and a tree with one node is a complete tree with height $h = 0$
- Inductive step:
  - Assume that a complete tree with n nodes has height $\lfloor \lg(n) \rfloor$
  - Must show that $\lfloor \lg(n + 1) \rfloor$ gives the height of a complete tree with $n + 1$ nodes
  - Two cases:
    - If the tree with n nodes is perfect, and
    - If the tree with n nodes is complete but not perfect

历年真题

请简述为什么完全二叉树的高度为 $\lfloor \log_2(n) \rfloor$，其中 n 为节点数量，$\lfloor \cdot \rfloor$ 是取整符号。

Case 1 (the tree with n nodes is perfect):

- If it is a perfect tree then
  - It had $n = 2^{h+1} - 1$ nodes
  - Adding one more node must increase the height
- So the tree with $n+1$ nodes has height $h+1$ and we have:

$$\lfloor \lg(n+1) \rfloor = \lfloor \lg(2^{h+1} - 1 + 1) \rfloor = \lfloor \lg(2^{h+1}) \rfloor = h+1$$

Case 2 (the tree with n nodes is complete but not perfect):

- If it is not a perfect tree then

$$2^h \le n < 2^{h+1} - 1$$

$$2^h + 1 \le n + 1 < 2^{h+1}$$

$$h < \lg(2^h + 1) \le \lg(n+1) < \lg(2^{h+1}) = h+1$$

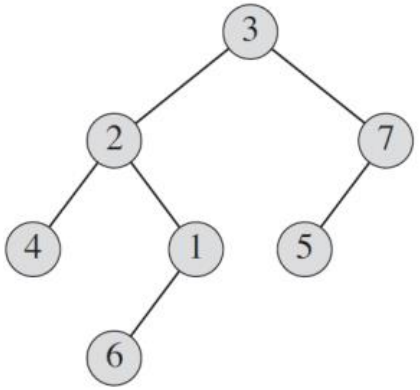
$$h \le \lfloor \lg(2^h + 1) \rfloor \le \lfloor \lg(n+1) \rfloor < h+1$$

- So the tree with $n+1$ nodes has height h and we have $\lfloor \lg(n+1) \rfloor = h$

历年真题

5. Binary Trees (25 points) 二叉树（25 分）

1) Write down the preorder, inorder, postorder, and breadth-first traversals of the following binary tree in the form of a sequence of nodes. （9 points）
请写出下面这棵二叉树的前序、中序、后序和广度优先遍历，以节点序列的形式给出。（9 分）



2) Draw the binary tree whose preorder is 1,3,2,7,6,5,4 and whose inorder is 1,2,3,4,5,6,7. （8 points）
一棵二叉树的前序遍历是 1,3,2,7,6,5,4，中序遍历是 1,2,3,4,5,6,7。请画出这棵树。（8 分）

3) Draw the binary tree whose postorder is 1,3,2,7,6,5,4 and whose inorder is 1,2,3,4,5,6,7. （8 points）
一棵二叉树的后序遍历是 1,3,2,7,6,5,4，中序遍历是 1,2,3,4,5,6,7。请画出这棵树。（8 分）

Q&A