数据结构与算法
Lecture 7-8 树

李星硕

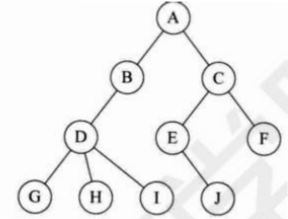
南京师范大学

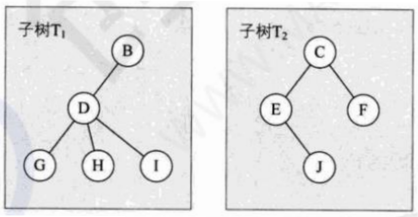
目录

目录

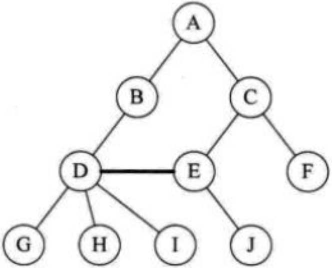
树的定义

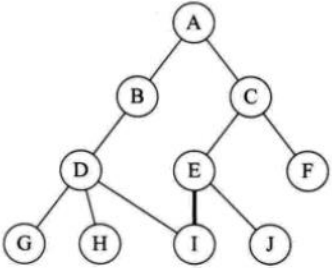
树（Tree）是 n（n≥0）个结点的有限集。n=0 时称为空树。在任意一棵非空树中：（1）有且仅有一个特定的称为根（Root）的结点；（2）当 n>1 时，其余结点可分为 m（m>0）个互不相交的有限集 T_1、T_2、……、T_m，其中每一个集合本身又是一棵树，并且称为根的子树（SubTree），如图 6-2-1 所示。





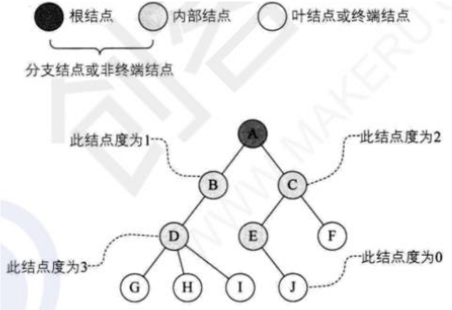
树的定义



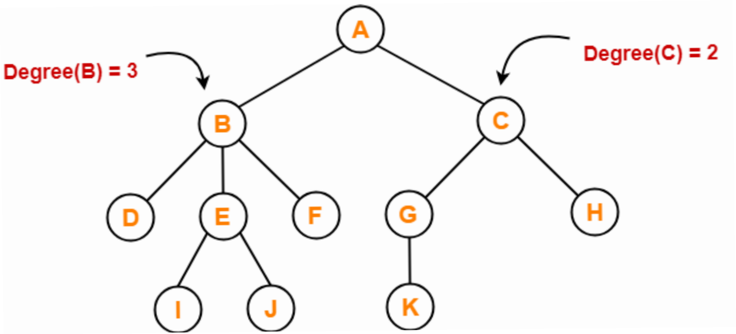


树的结点

树的结点包含一个数据元素及若干指向其子树的分支。结点拥有的子树数称为结点的度（Degree）。度为 0 的结点称为叶结点（Leaf）或终端结点；度不为 0 的结点称为非终端结点或分支结点。除根结点之外，分支结点也称为内部结点。树的度是树内各结点的度的最大值。

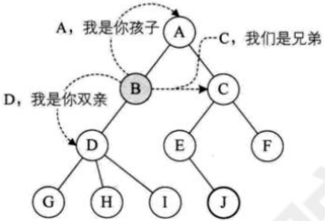


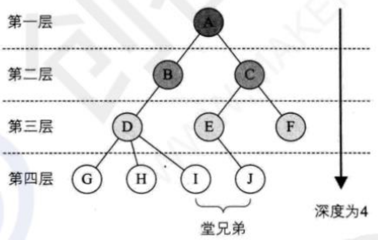
树的结点练习



结点间的关系

- 结点子树的根称为该结点的孩子（Child），相应的该结点称为孩子的双亲（Parent），同一个双亲之间的孩子之间互称为兄弟（Sibling）。
- 树中结点的最大层次称为树的深度（Depth）或者高度。

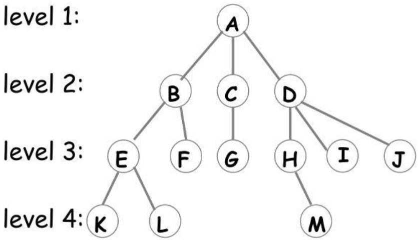




树的练习

Degree of Node and Degree of Tree

- The degree of a node is the number of subtrees of the node.
- The degree of a tree is the maximum degree of the nodes in the tree.



Node	degree
A	3
B	2
C	1
D	3
…	
M	0

Degree of tree = 3

树的存储结构：双亲表示法

1 概述
- 树的概念
- 树的存储结构：双亲表示法
- 树的存储结构：孩子表示法
- 树的存储结构：孩子兄弟表示法

2 二叉树

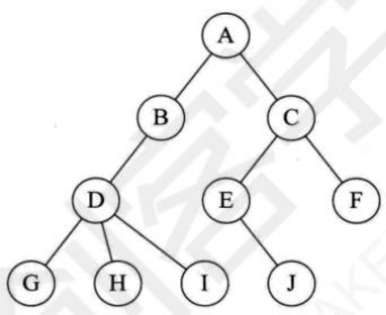
3 遍历二叉树

4 树、森林与二叉树的转换

双亲表示法：代码

```
/* 树的双亲表示法结点结构定义 */
#define MAX_TREE_SIZE 100
typedef int TElemType;    /* 树结点的数据类型，目前暂定为整型 */
typedef  struct PTNode    /* 结点结构 */
{
    TElemType data;        /* 结点数据 */
    int parent;            /* 双亲位置 */
} PTNode;
typedef  struct            /* 树结构 */
{
    PTNode nodes[MAX_TREE_SIZE];/* 结点数组 */
    int r,n;               /* 根的位置和结点数 */
} PTree;
```

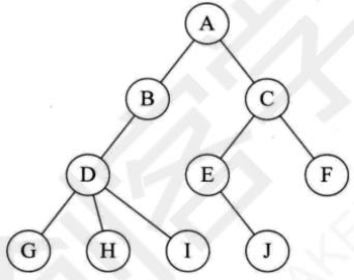
双亲表示法：双亲位置



下标	data	parent
0	A	-1
1	B	0
2	C	0
3	D	1
4	E	2
5	F	2
6	G	3
7	H	3
8	I	3
9	J	4

双亲表示法：长子位置

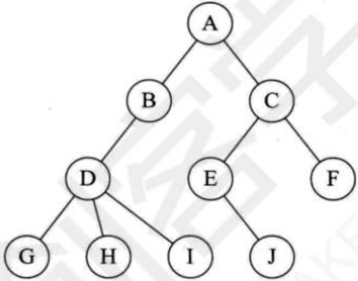
结点最左边孩子的域，不妨叫它长子域，



下标	data	parent	firstchild
0	A	-1	1
1	B	0	3
2	C	0	4
3	D	1	6
4	E	2	9
5	F	2	-1
6	G	3	-1
7	H	3	-1
8	I	3	-1
9	J	4	-1

双亲表示法：兄弟位置

·结点如果它存在右兄弟，则记录下右兄弟的下标



下标	data	parent	rightsib
0	A	-1	-1
1	B	0	2
2	C	0	-1
3	D	1	-1
4	E	2	5
5	F	2	-1
6	G	3	7
7	H	3	8
8	I	3	-1
9	J	4	-1

树的存储结构：孩子表示法

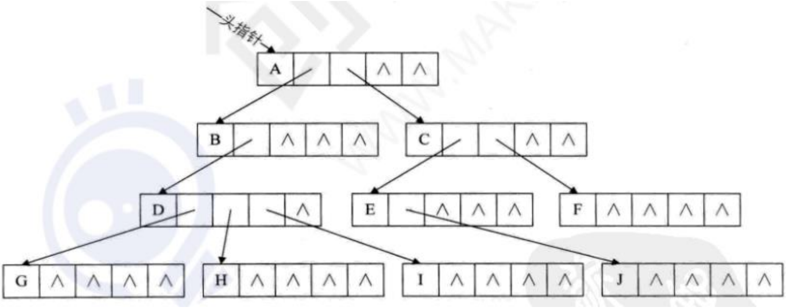
1 概述
- 树的概念
- 树的存储结构：双亲表示法
- 树的存储结构：孩子表示法
- 树的存储结构：孩子兄弟表示法

2 二叉树

3 遍历二叉树

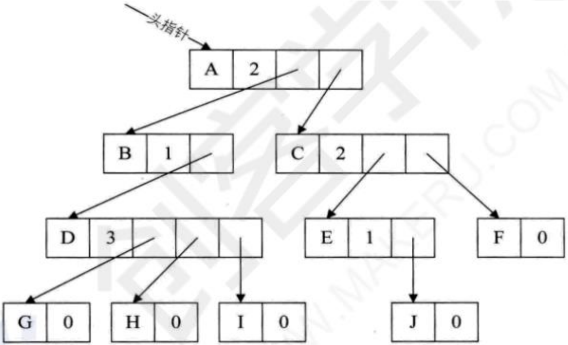
4 树、森林与二叉树的转换

孩子表示法：结点多指针域



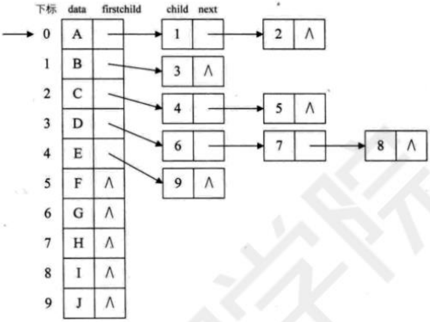
孩子表示法：增加结点的度（degree）

data	degree	child1	child2	······	childd



孩子表示法：顺序存储

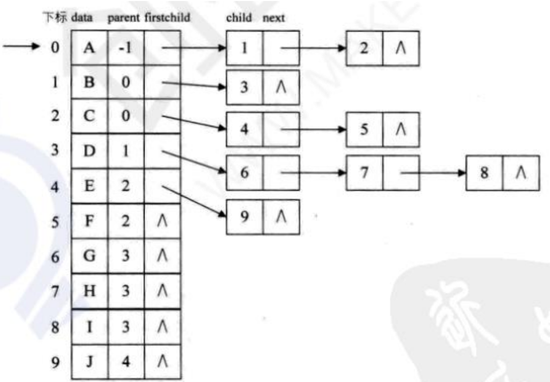
- 把每个结点的孩子结点排列起来，以单列表作为存储结构，则 n 个结点则有 n 个孩子链表，如果是叶接点则该单链表为空。然后 n 个头指针又组成一个线性表，采用顺序存储结构，存放进一个一维数组中。



孩子表示法：代码

```
/* 树的孩子表示法结构定义 */
#define MAX_TREE_SIZE 100
typedef struct CTNode/* 孩子结点 */
{
    int child;
    struct CTNode *next;
} *ChildPtr;
typedef struct          /* 表头结构 */
{
    TElemType data;
    ChildPtr firstchild;
} CTBox;
typedef struct          /* 树结构 */
{
    CTBox nodes[MAX_TREE_SIZE]; /* 结点数组 */
    int r,n;            /* 根的位置和结点数 */
} CTree;
```

孩子表示法：改进

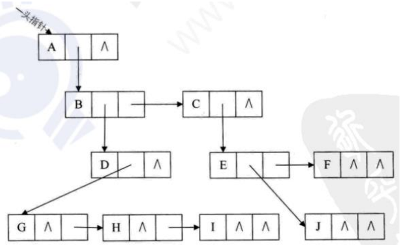

下标 data parent firstchild
child next
0 A -1
1 B 0
2 C 0
3 D 1
4 E 2
5 F 2 Λ
6 G 3 Λ
7 H 3 Λ
8 I 3 Λ
9 J 4 Λ
1
2 Λ
3 Λ
4
5 Λ
6
7
8 Λ
9 Λ

树的存储结构：孩子兄弟表示法

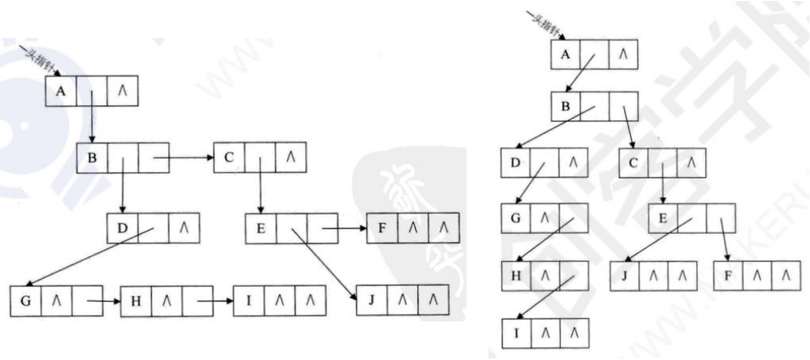
孩子兄弟表示法

data	firstchild	rightsib

```
/* 树的孩子兄弟表示法结构定义 */
typedef struct CSNode
{
   TElemType data;
   struct CSNode *firstchild,*rightsib;
} CSNode,*CSTree;
```



孩子兄弟表示法



目录

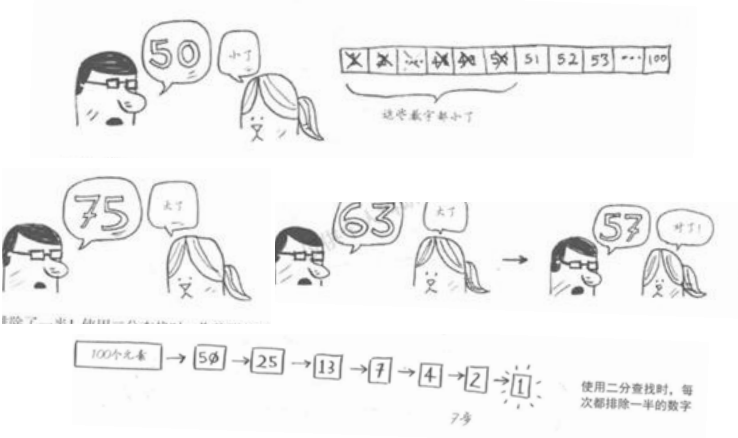
猜数字

- 我随便想一个从 1-100 的正整数，你来猜。你每猜一次，我都会告诉你小了、大了或对了。你最少几次能猜到该数字？

猜数字：简单查找

一种糟糕的猜数法

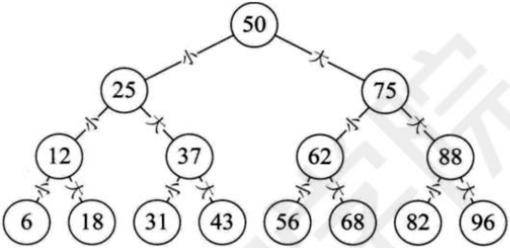
猜数字：二分查找


50
51
52
53
...
100
75
63
57
100个元素
50
25
13
7
4
2
1
7步
使用二分查找时，每次都排除一半的数字

为什么用二分查找


240K
120K
60K
30K
15K
7.5K
3750
1875
938
469
235
118
59
30
15
8
4
2
1
18步
sina 新浪科技 新浪科技 > 滚动新闻 > 正文
快看 | 微博2020年第一季度财报：月活跃用户达5.5亿
2020年05月19日 19:20 界面新闻
新浪财经APP

二分查找与二叉树

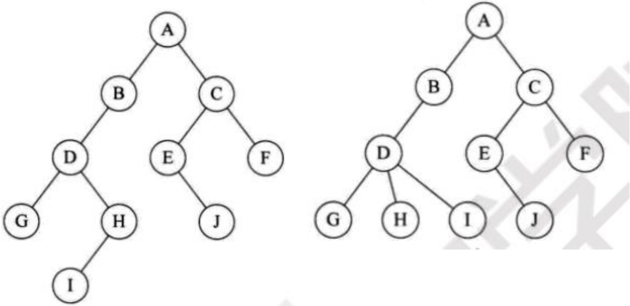


被猜数字	第一次	第二次	第三次	第四次	第五次	第六次	第七次
39	50	25	37	43	40	38	39
82	50	75	88	82			
99	50	75	88	96	98	99	
1	50	25	12	6	3	2	1

二叉树的概念

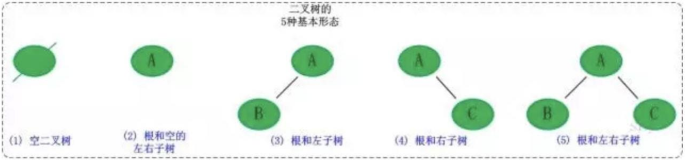
二叉树的概念

二叉树（Binary Tree）是 n（n≥0）个结点的有限集合，该集合或者为空集（称为空二叉树），或者由一个根结点和两棵互不相交的、分别称为根结点的左子树和右子树的二叉树组成。



二叉树的特点

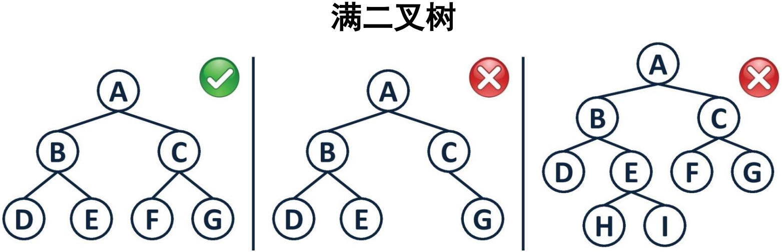
- 每个结点最多有两棵子树，所以二叉树中不存在度大于 2 的结点。注意不是只有两棵子树，而是最多有。没有子树或者有一棵子树都是可以的。
- 左子树和右子树是有顺序的，次序不能任意颠倒。就像人是双手、双脚，但显然左手、左脚和右手、右脚是不一样的，右手戴左手套、右脚穿左鞋都会极其别扭和难受。
- 即使树中某结点只有一棵子树，也要区分它是左子树还是右子树。



二叉树的种类

满二叉树

- 一棵树深度为 k、2^{k-1} 个节点的树是满二叉树

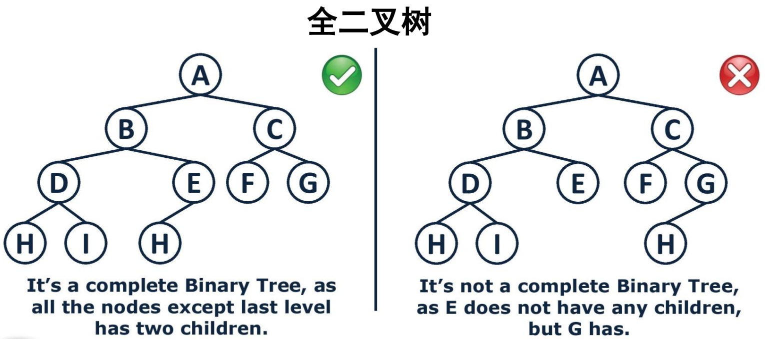


满二叉树的特点

- 所有内部节点都有两个子节点，最底一层是叶子节点
- 如果一颗树深度为 h，最大层数为 k，且深度与最大层数相同，即 k=h
- 第 k 层的结点数是 2^{k-1}
- 总结点数是：2^{k}-1
- 总节点数一定是奇数

全二叉树

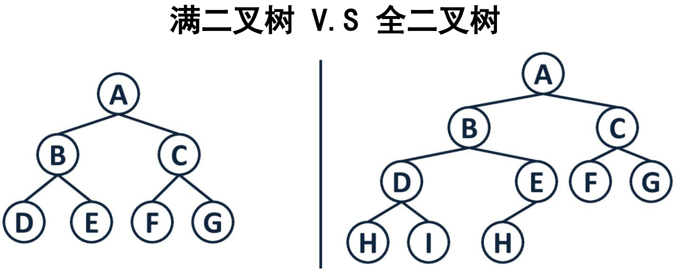
- 若设二叉树的深度为 h，除第 h 层外，其它各层 (1 ~ h-1) 的结点数都达到最大个数，第 h 层所有的结点都连续集中在最左边，这就是完全二叉树。



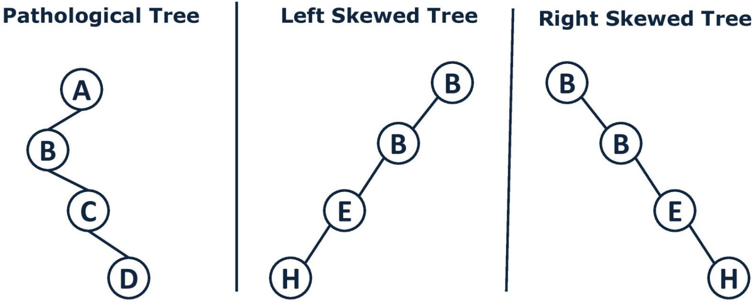
全二叉树的特点

- 深度为 k 的完全二叉树，至少有 2^{k-1} 个节点，至多有 2^{k}-1 个节点。
- 满二叉树一定是完全二叉树，完全二叉树不一定是满二叉树

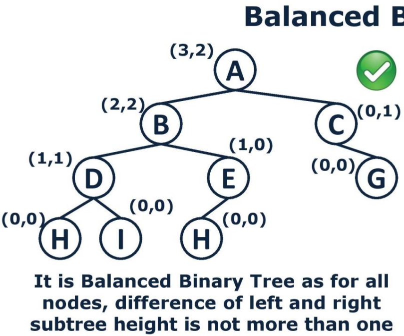
满二叉树与全二叉树



其他二叉树：斜二叉树

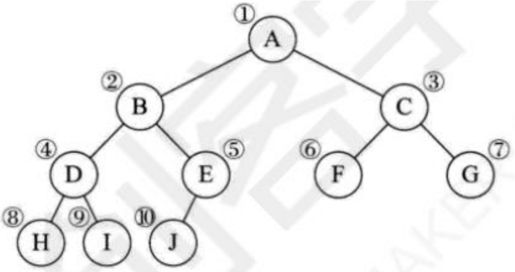

Pathological Tree
A
B
C
D
Left Skewed Tree
B
B
E
H
Right Skewed Tree
B
B
E
H

其他二叉树：平衡二叉树


Balanced Binary Tree
(3,2) A
(2,2) B
(0,1) C
(1,1) D
(1,0) E
(0,0) G
(0,0) H
(0,0) I
H (0,0)
It is Balanced Binary Tree as for all nodes, difference of left and right subtree height is not more than one
(3,3) A
(2,1) B
(0,2) C
(1,1) D
(0,0) E
(0,1) G
(0,0) H
(0,0) I
(0,0) F
Node C violates the property of Balanced Binary Tree.

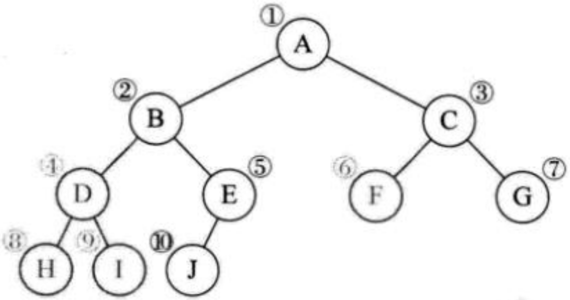
二叉树的存储结构

顺序结构



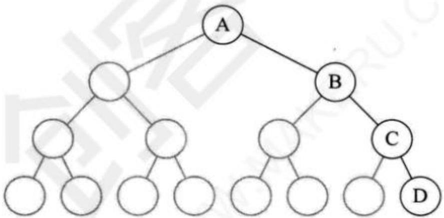
下标:	1	2	3	4	5	6	7	8	9	10
	A	B	C	D	E	F	G	H	I	J

顺序结构：浅色表示不存在



下标:	1	2	3	4	5	6	7	8	9	10
	A	B	C	∧	E	∧	G	∧	∧	J

顺序结构

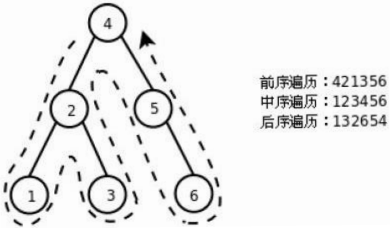


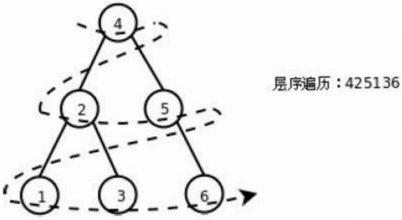
下标：	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	A	∧	B	∧	∧	∧	C	∧	∧	∧	∧	∧	∧	∧	D

目录

二叉树遍历原理

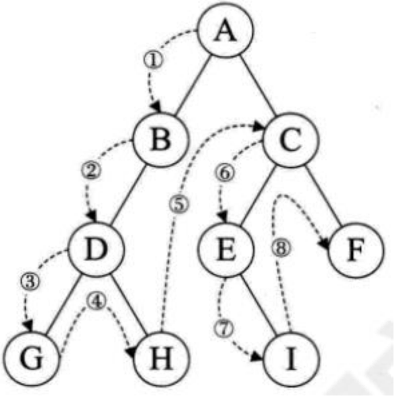
二叉树的遍历（traversing binary tree）是指从根结点出发，按照某种次序依次访问二叉树中所有结点，使得每个结点被访问一次且仅被访问一次。





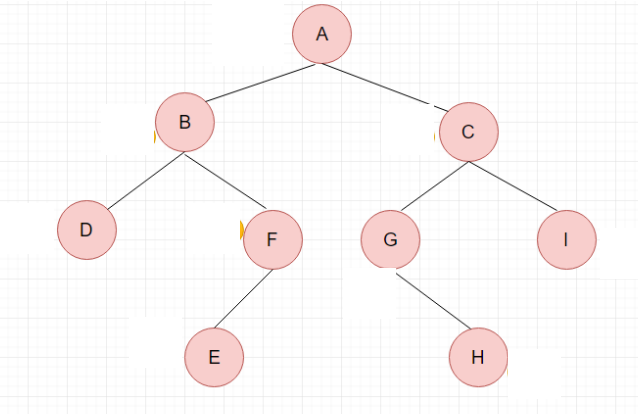
- 前、中、后序遍历也叫深度优先
- 层序遍历也叫广度优先

二叉树前序遍历

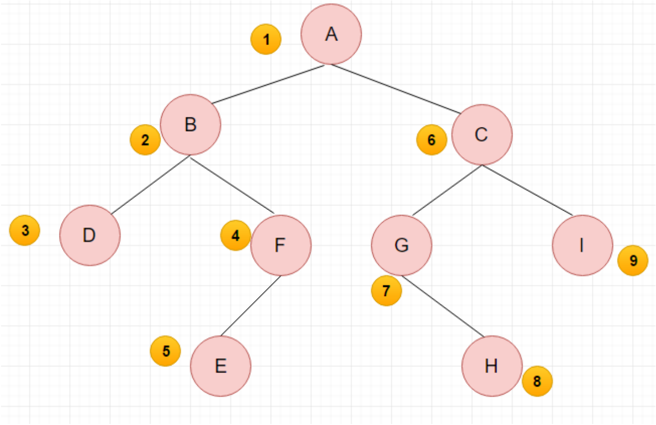


- 先访问根结点
- 然后先序遍历左子树
- 最后先序遍历右子树
- A B D G H C E I F

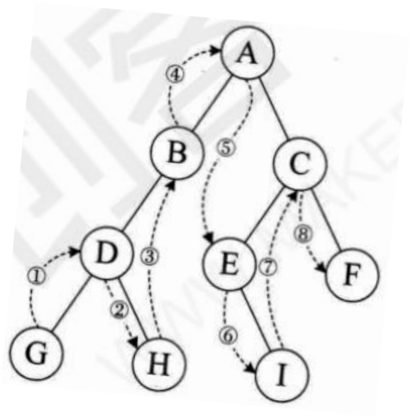
二叉树前序遍历练习



二叉树前序遍历练习答案

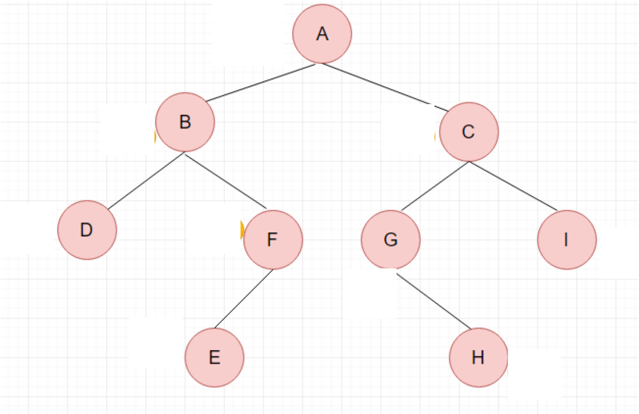


二叉树中序遍历

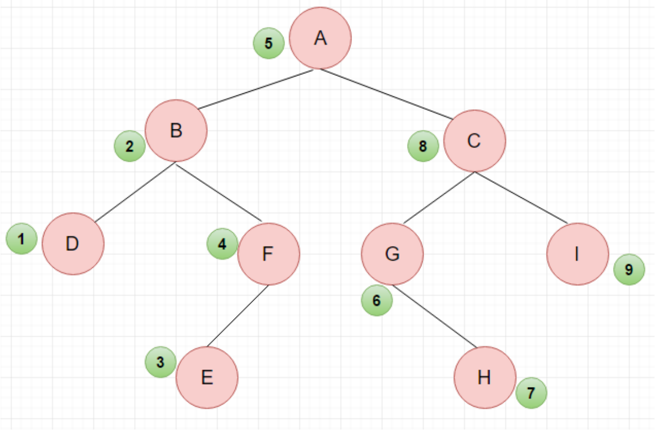


- 先中序遍历左子树
- 然后是根结点
- 最后中序遍历右子树
- G D H B A E I C F

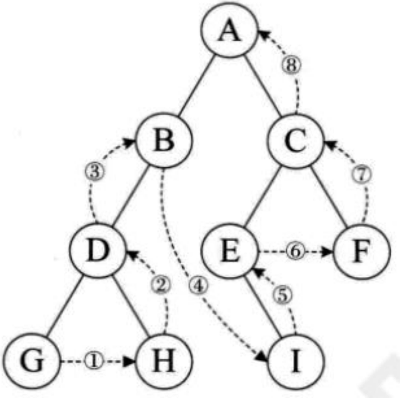
二叉树中序遍历练习



二叉树中序遍历练习答案

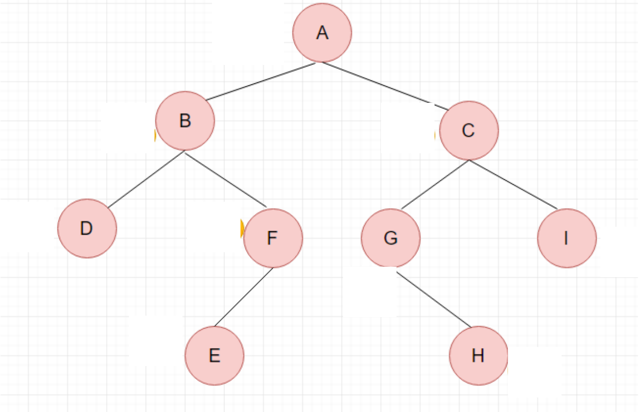


二叉树后序遍历

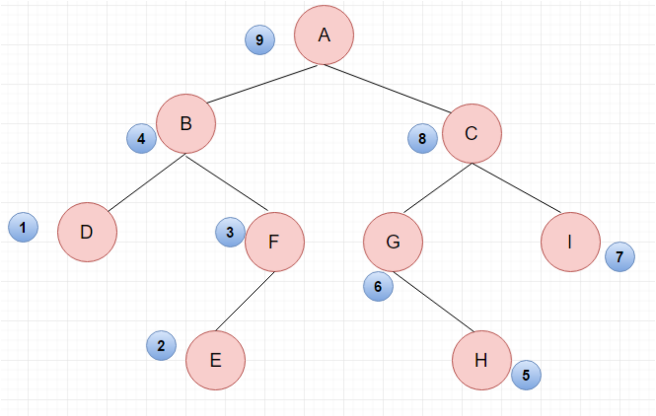


- 后序遍历左子树
- 后序遍历右子树
- 最后访问根节点
- G H D B I E F C A

二叉树后序遍历练习



二叉树后序遍历练习答案



二叉树遍历代码

二叉树前序遍历代码

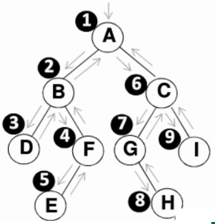
（1）先序遍历

遍历过程为：
① 访问根结点；
② 先序遍历其左子树；
③ 先序遍历其右子树。

A（B D F E）（C G H I）

先序遍历=> A B D F E C G H I

```
/* 二叉树的前序遍历递归算法 */
void PreOrderTraverse(BiTree T)
{
    if(T==NULL)
        return;
    printf("%c",T->data);/* 显示结点数据，可以更改为其他对结点操作 */
    PreOrderTraverse(T->lchild); /* 再先序遍历左子树 */
    PreOrderTraverse(T->rchild); /* 最后先序遍历右子树 */
}
```



二叉树中序遍历代码

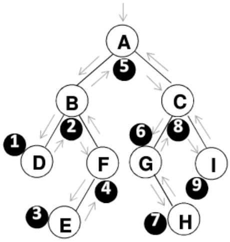
（2）中序遍历

遍历过程为：
① 中序遍历其左子树；
② 访问根结点；
③ 中序遍历其右子树。

(D B E F) A (G H C I)

中序遍历=> D B E F A G H C I

```
/* 二叉树的中序遍历递归算法 */
void InOrderTraverse(BiTree T)
{
    if(T==NULL)
        return;
    InOrderTraverse(T->lchild); /* 中序遍历左子树 */
    printf("%c",T->data);/* 显示结点数据，可以更改为其他对结点操作 */
    InOrderTraverse(T->rchild); /* 最后中序遍历右子树 */
}
```



二叉树后序遍历代码

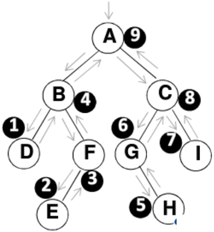
（3）后序遍历

遍历过程为：
① 后序遍历其左子树；
② 后序遍历其右子树；
③ 访问根结点。

（D E F B ） （H G I C） A

后序遍历=> D E F B H G I C A

```
/* 二叉树的后序遍历递归算法 */
void PostOrderTraverse(BiTree T)
{
    if(T==NULL)
        return;
    PostOrderTraverse(T->lchild); /* 先后序遍历左子树 */
    PostOrderTraverse(T->rchild); /* 再后序遍历右子树 */
    printf("%c",T->data);/* 显示结点数据，可以更改为其他对结点操作 */
}
```



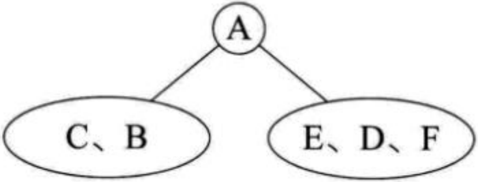
二叉树遍历推导

二叉树遍历推导

- 已知一颗二叉树的前序遍历为 ABCDEF；
- 中序遍历为 CBAEDF，
- 请问这颗二叉树的后序遍历是多少？

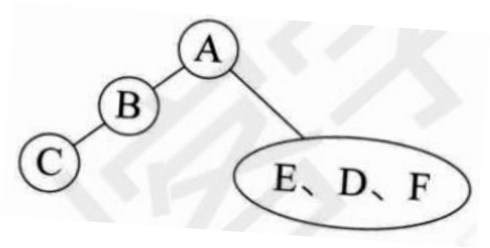
二叉树遍历推导 1

- 前序遍历从根节点开始，先打印再递归左和右，A 是根节点：ABCDEF。
- 中序遍历为 CBAEDF，可以知道 CB 是 A 的左边，EDF 在 A 的右边。



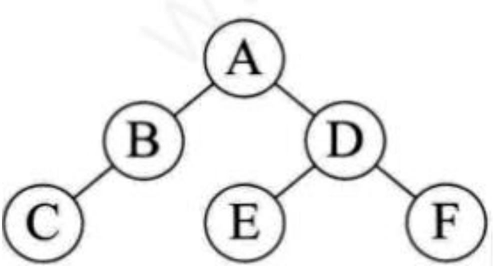
二叉树遍历推导 2

- 根据 ABCDEF 可知，是先打印 B 再打印 C，所有 B 是 A 的左孩子，而 C 只能是 B 的孩子，但是不能确定是左还是右。
- 根据中序遍历 CBAEDF，可以知道 C 是在 B 前面打印，所以 C 是 B 的左孩子。

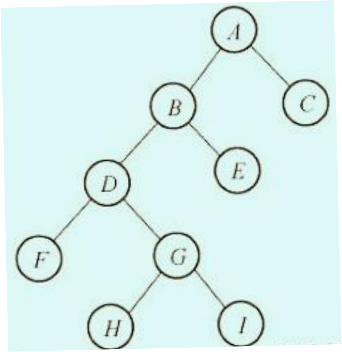


二叉树遍历推导 3

- 再看前序中 E、D、F：ABCDEF，那就意味着 D 是 A 结点的右孩子，E、F 是 D 的子孙。
- 再看中序遍历为 CBAEDF，由于 E 在 D 的左侧，而 F 在 D 的右侧，所以 E 是 D 的左孩子，F 是 D 的右孩子。



二叉树遍历推导练习 1

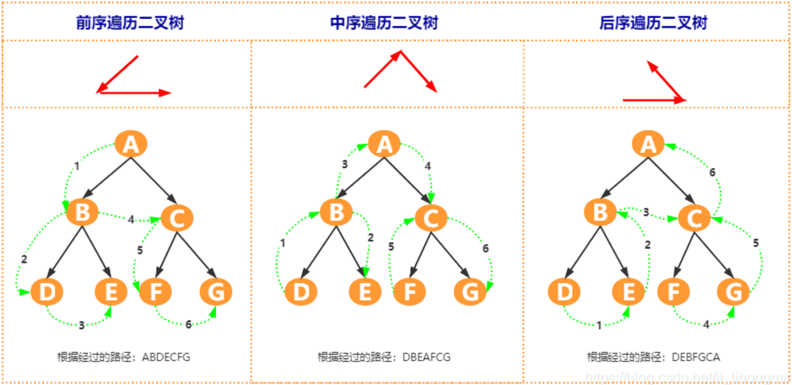


前序顺序A-B-D-F-G-H-I-E-C

中序顺序F-D-H-G-I-B-E-A-C

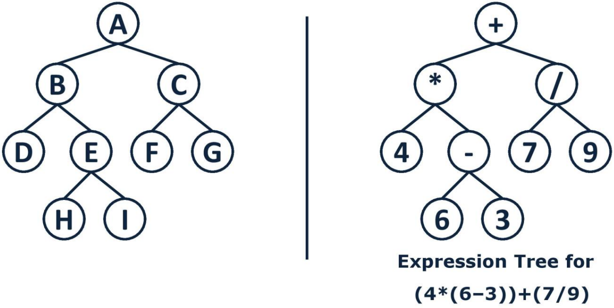
二叉树遍历小结

二叉树遍历小结


前序遍历二叉树
中序遍历二叉树
后序遍历二叉树
根据经过的路径：ABDECFG
根据经过的路径：DBEAFCG
根据经过的路径：DEBFGCA
https://blog.csdn.net/lt_lihongmin

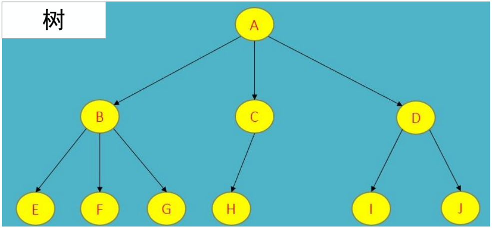
二叉树遍历作用

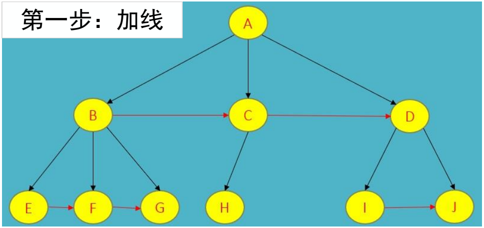
二叉树遍历举例



目录

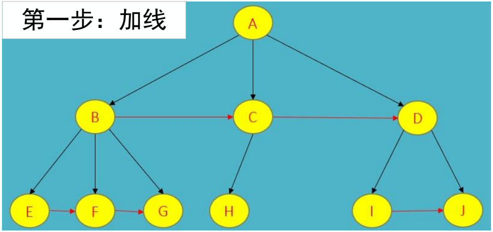
树转换为二叉树：加线

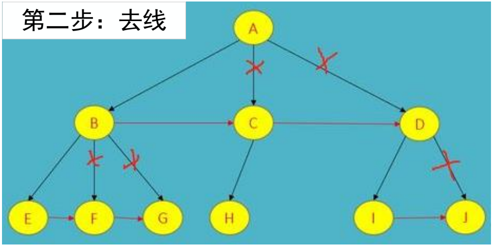




- 第一步，给所有的兄弟节点加线，即兄弟相连

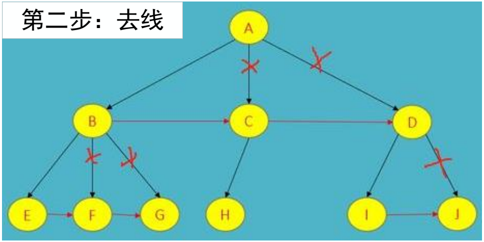
树转换为二叉树：去线

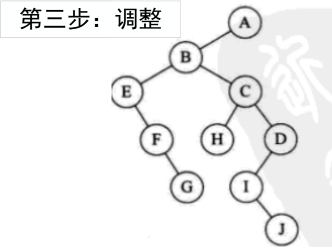




- 第二步，把除了长子以外的线全部删除

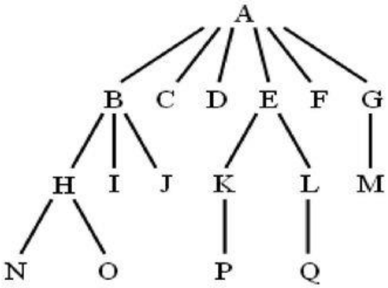
树转换为二叉树：调整



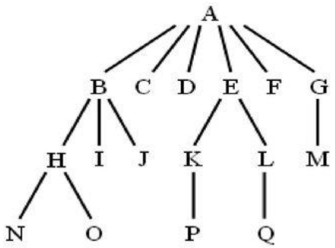


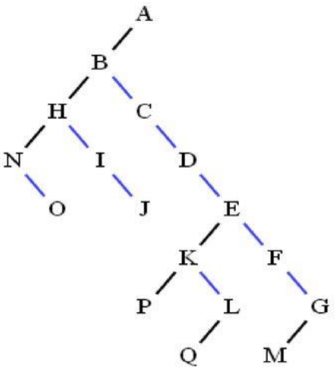
- 最后调整下层次关系，注意第一个孩子（即长子）是二叉树的左子树，之前的兄弟节点是长子的右孩子

树转换为二叉树：练习 1

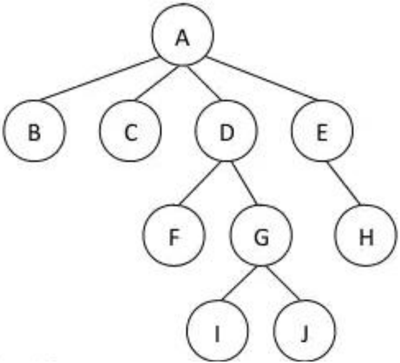


树转换为二叉树：答案 1



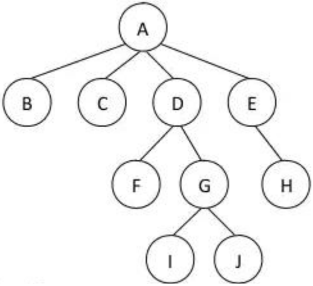


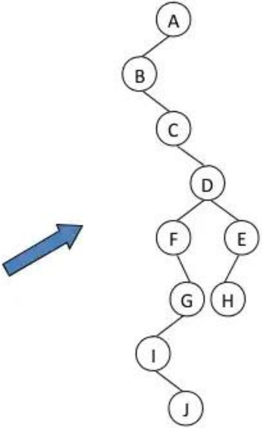
树转换为二叉树：练习 2



树转换为二叉树：答案 2

- Binary tree left child = leftmost child
- Binary tree right child = right sibling

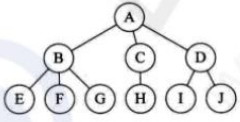


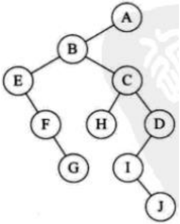


树转换为二叉树：代码

https://blog.csdn.net/weixin_42545675/article/details/97647275

```
//树转换成二叉树
BTNode* ExchangeToBTree(CSTree ct)
{
    if (ct == NULL)
        return NULL;
    else
    {
        BTNode * bt = (BTNode*)malloc(sizeof(BTNode));
        bt->data = ct->data;
        bt->lchild = ExchangeToBTree(ct->firstchild);
        bt->rchild = ExchangeToBTree(ct->nextsibling);
        return bt;
    }
}
```

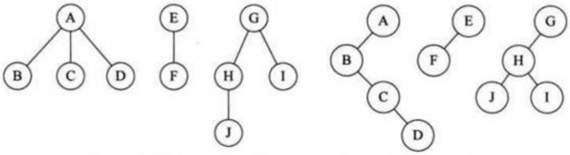




森林转换为二叉树

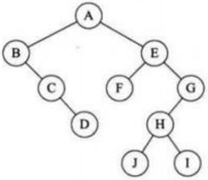
森林转换为二叉树

- 第一棵二叉树不动，从第二棵二叉树开始，一次把后一棵二叉树的根结点作为前一棵二叉树的根结点的右孩子，用线连接起来。



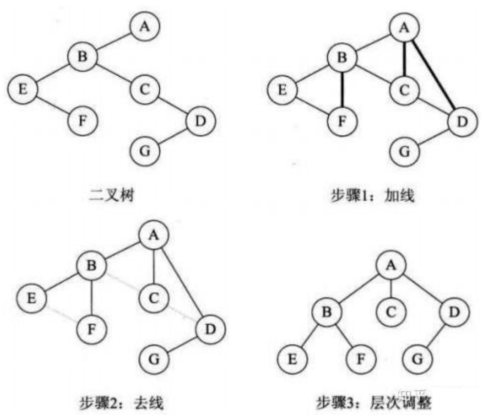
拥有三棵树的森林

步骤1:森林中每棵树转换为二叉树



步骤2：将所有二叉树转换为一棵二叉树

二叉树转化为树



- 加线。在所有的兄弟结点之间加一条线。
- 去线。树中的每个结点，只保留它与第一个孩子结点的连线，删除其他孩子结点之间的连线。
- 调整。以树的根结点为轴心，将整个树调节一下（第一个孩子是结点的左孩子，兄弟转过来的孩子是结点的右孩子）

二叉树转化为森林

- 从根节点开始，若右孩子存在，则把与右孩子结点的连线删除。再查看分离后的二叉树，若其根节点的右孩子存在，则连续删除。直到所有这些根结点与右孩子的连线都删除为止。