Data Structures

Ch5

树和二叉树

Trees & Binary Trees

2024 年 10 月 22 日

学而不厌 诲人不倦

Chapter 5　树和二叉树

5.5 二叉树的存储结构

5-5-1　二叉树的顺序存储结构

1. 二叉树的顺序存储结构

用一组连续的存储单元依次存储数据元素，由存储位置表示元素之间的逻辑关系

二叉树的顺序存储结构是用一维数组存储二叉树的结点，结点的存储位置（下标）应能体现结点之间的逻辑关系——父子关系

如何利用数组下标来反映结点之间的逻辑关系?

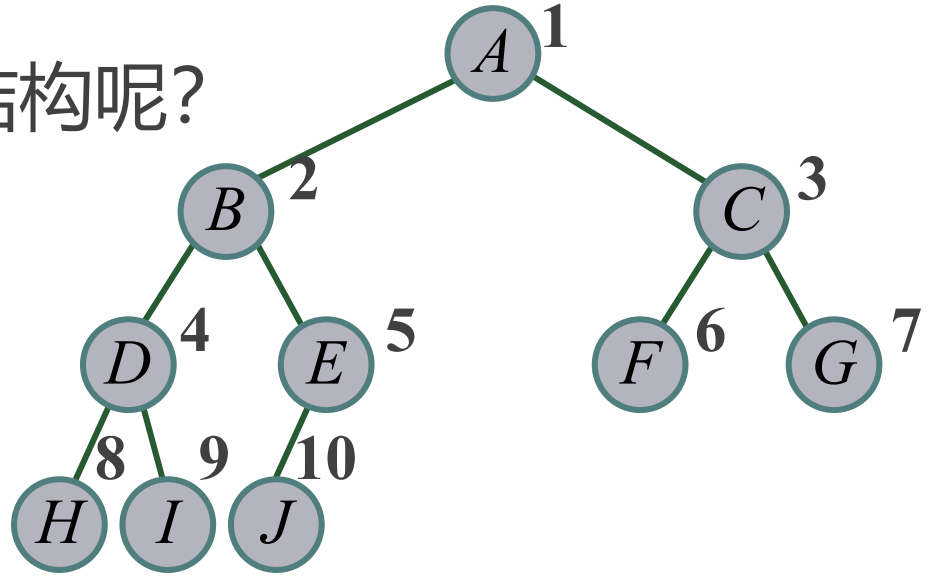
完全二叉树中结点的编号可以唯一地反映结点之间的逻辑关系

5.5 二叉树的存储结构

1. 二叉树的顺序存储结构

如何定义二叉树的顺序存储结构呢?

```
const MaxSize = 100;
template <typename DataType>
struct SeqBiTree
{
    DataType data[MaxSize];
    int biTreeNum;
};
```



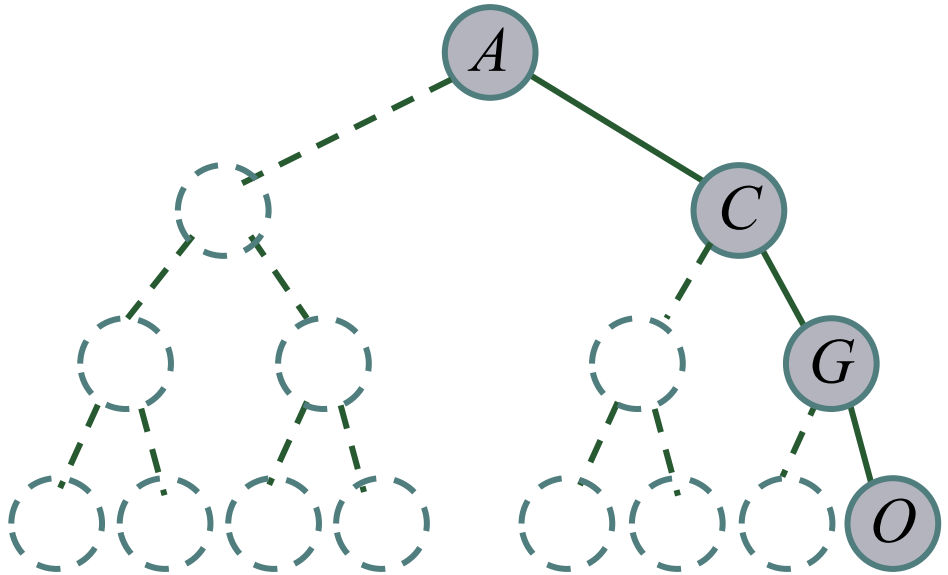
以编号为下标

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

1. 二叉树的顺序存储结构

顺序存储一棵右斜树会发生什么情况?

缺点：浪费存储空间



二叉树的顺序存储结构一般仅存储完全二叉树

1. 二叉树的顺序存储结构

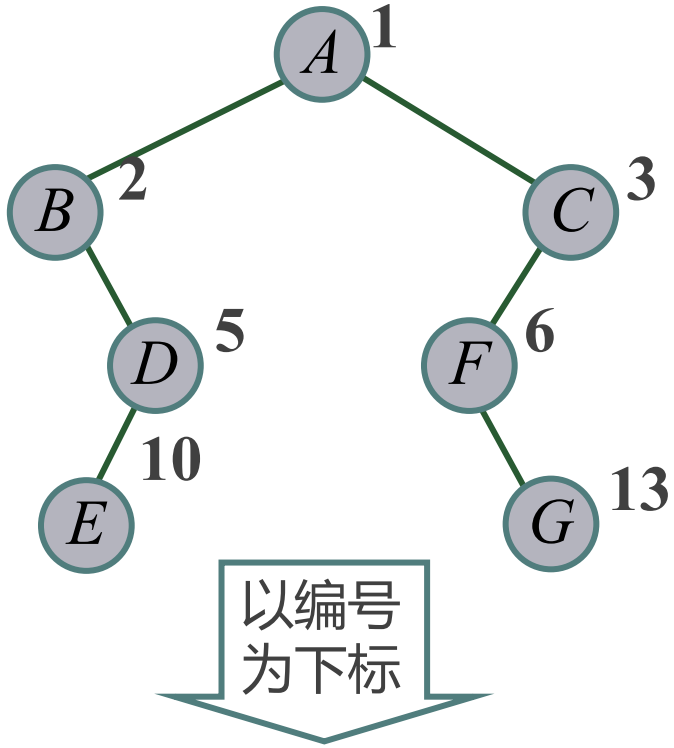
对于普通的二叉树，如何顺序存储呢？

将二叉树按完全二叉树编号：

（1）根结点的编号为 1

（2）若某结点 i 有左孩子，则其左孩子的编号为 $2i$

（3）若某结点 i 有右孩子，则其右孩子的编号为 $2i+1$



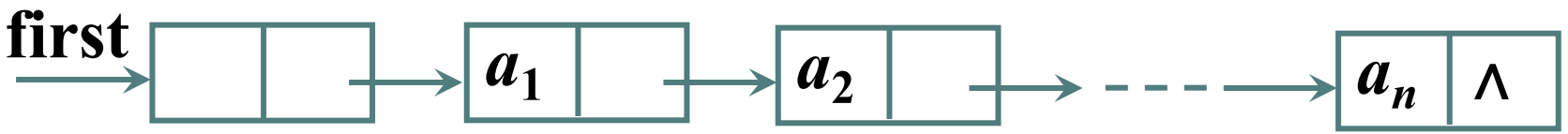
1	2	3	4	5	6	7	8	9	10	11	12	13
A	B	C	∧	D	F	∧	∧	∧	E	∧	∧	G

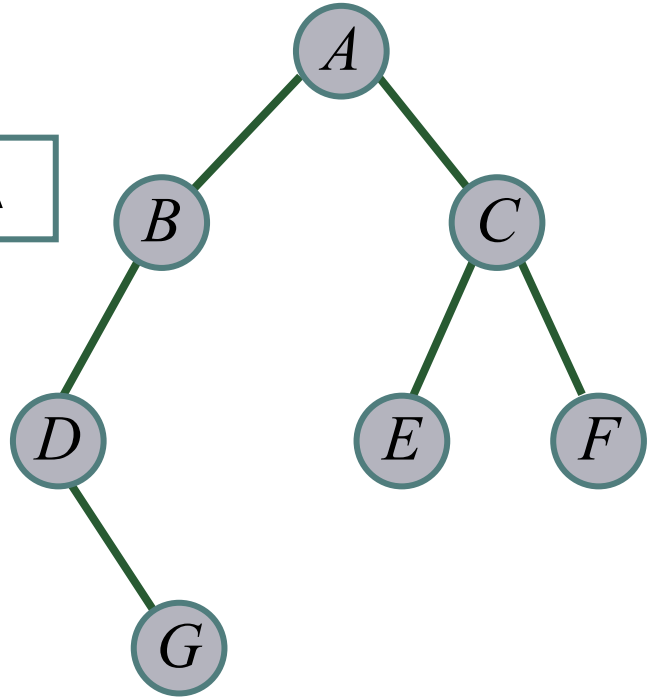
5.5 二叉树的存储结构

5-5-2　二叉链表

1. 二叉链表的存储方法

如何用链接存储方式存储二叉树呢?

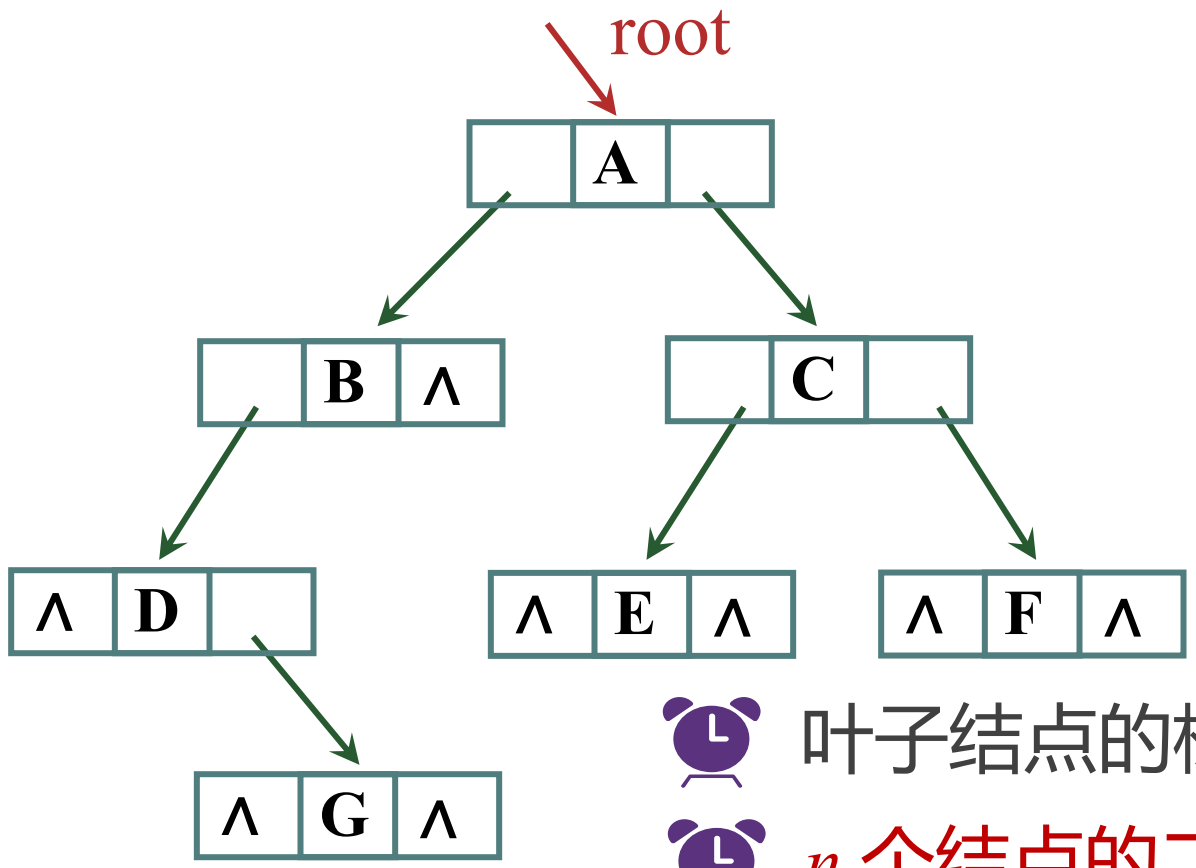


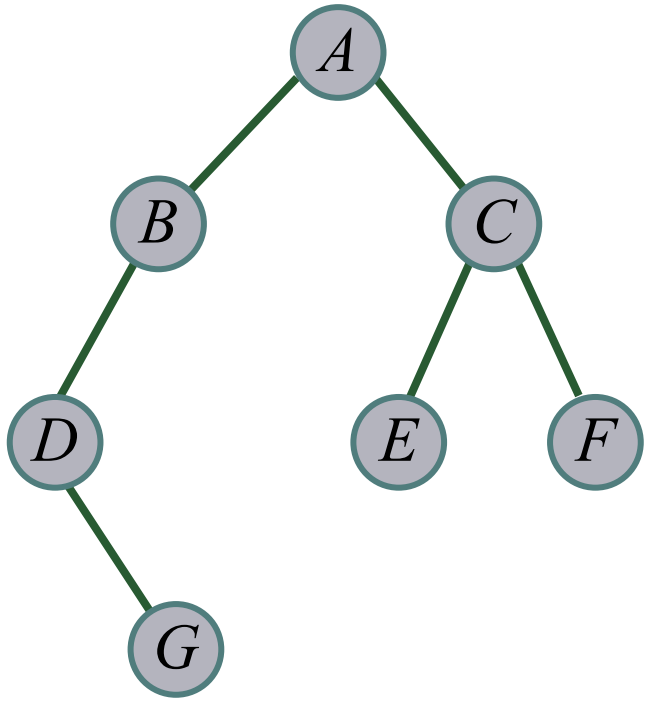


二叉链表：二叉树的每个结点对应一个链表结点，链表结点存放结点的数据信息和指示左右孩子的指针

lchild	data	rchild

1. 二叉链表的存储方法



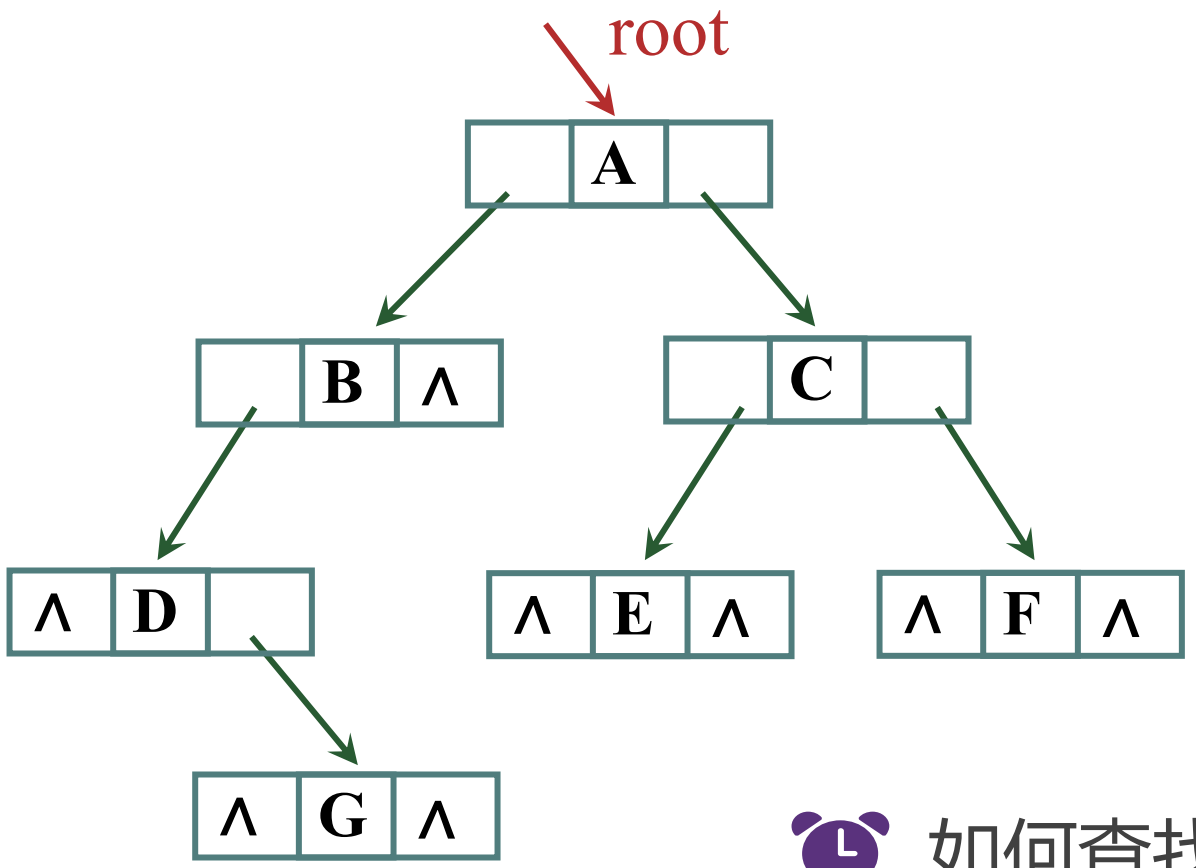


叶子结点的标志？ ⇨ 左右孩子指针均为空

n 个结点的二叉链表有多少个空指针？

$2n-(n-1)=n+1$ 个空指针

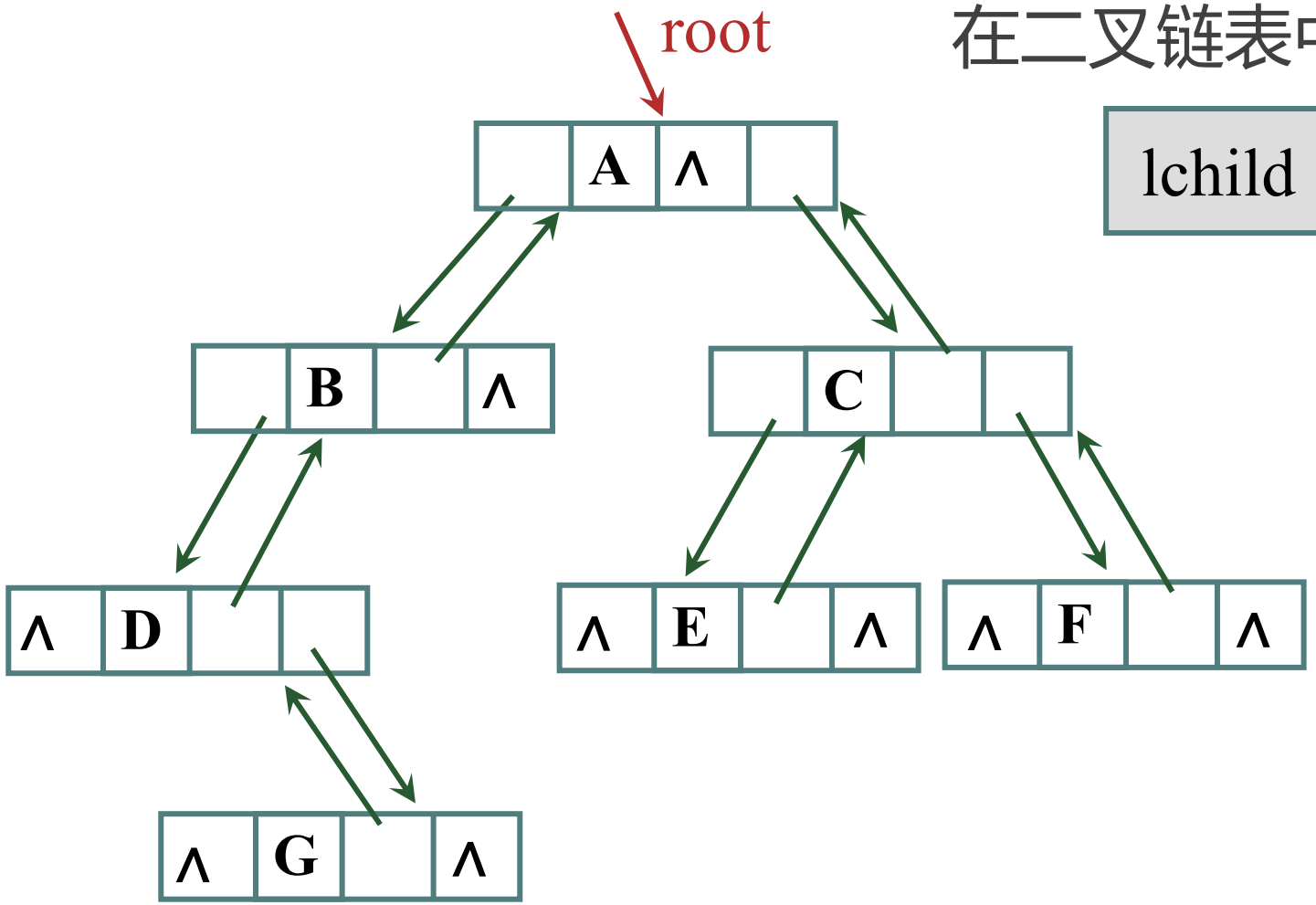
1. 二叉链表的存储方法




```
template <typename DataType>
struct BiNode
{
    DataType data;
    BiNode< DataType > *lchild, *rchild;
};
```

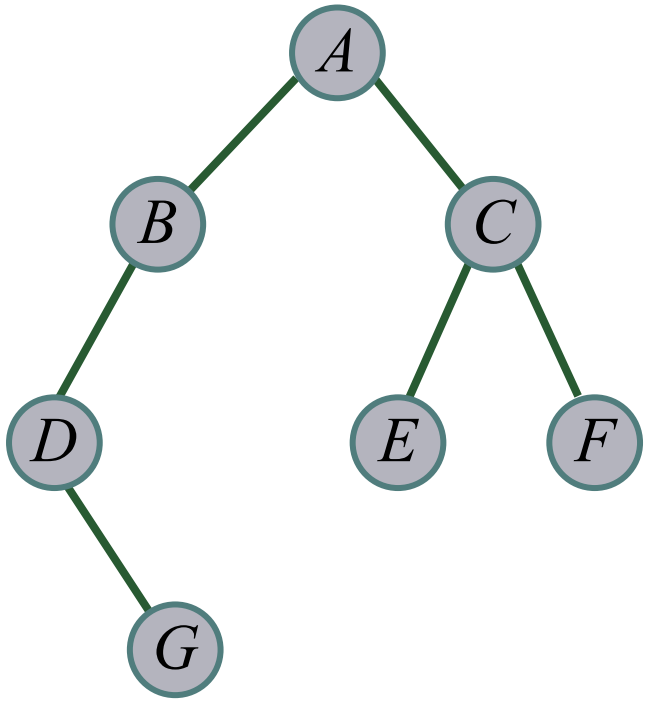
如何查找双亲？时间性能？ ⇨ $O(n)$

2. 三叉链表的存储方法



在二叉链表中增加一个指向双亲的指针域

lchild	data	parent	rchild



3. 二叉链表的类定义

二叉树的抽象数据类型定义？

InitBiTree：初始化一棵空的二叉树

CreatBiTree：建立一棵二叉树

DestroyBiTree：销毁一棵二叉树

PreOrder：前序遍历二叉树

InOrder：中序遍历二叉树

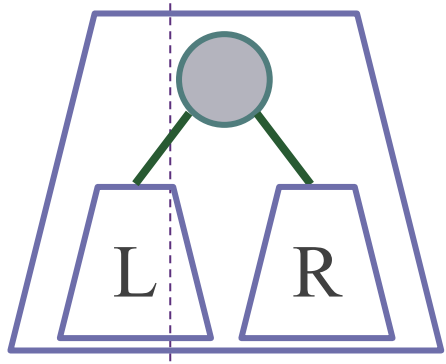
PostOrder：后序遍历二叉树

LevelOrder：层序遍历二叉树

```
template <typename DataType>
class BiTree
{
public:
    BiTree( ){root = Creat(root);}
    ~BiTree( ){Release(root);}
    void PreOrder( ){PreOrder(root);}
    void InOrder( ){InOrder(root);}
    void PostOrder( ){PostOrder(root);}
    void LevelOrder( );
private:
    BiNode<DataType> *Creat(BiNode<DataType> *bt);
    void Release(BiNode<DataType> *bt);
    void PreOrder(BiNode<DataType> *bt);
    void InOrder(BiNode<DataType> *bt);
    void PostOrder(BiNode<DataType> *bt);
    BiNode<DataType> *root;
};
```

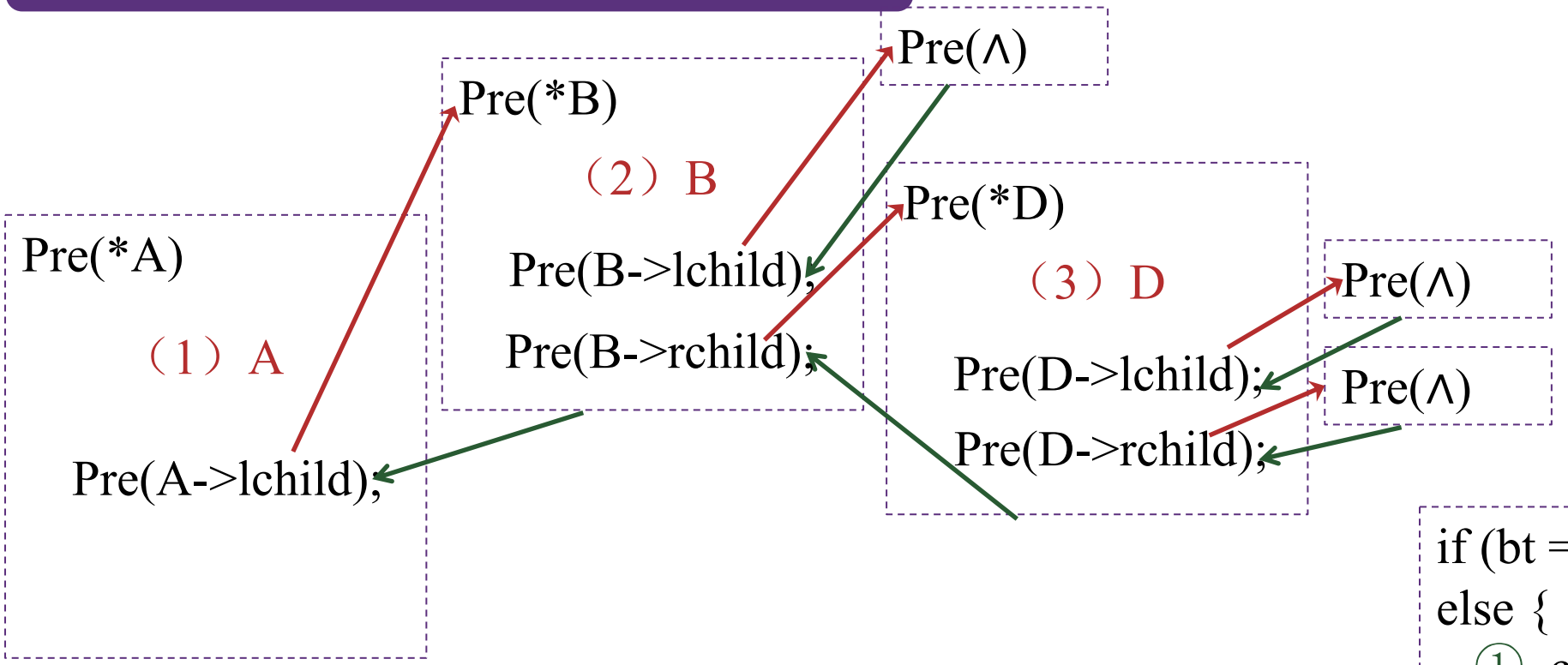
4. 二叉树的前序遍历

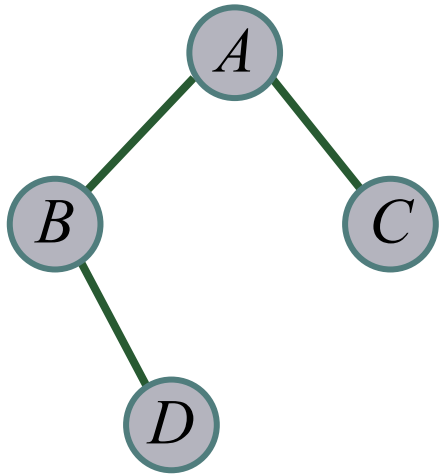
```
template <typename DataType>
void BiTree<DataType> :: PreOrder(BiNode<DataType> *bt)
{
    if (bt == nullptr) return;                    //递归调用的结束条件
    else {
        cout << bt->data;                         //访问根结点bt的数据域
        PreOrder(bt->lchild);                     //前序递归遍历bt的左子树
        PreOrder(bt->rchild);                     //前序递归遍历bt的右子树
    }
}
```



按照先左后右的方式扫描二叉树，区别仅在于访问结点的时机

4. 二叉树的前序遍历

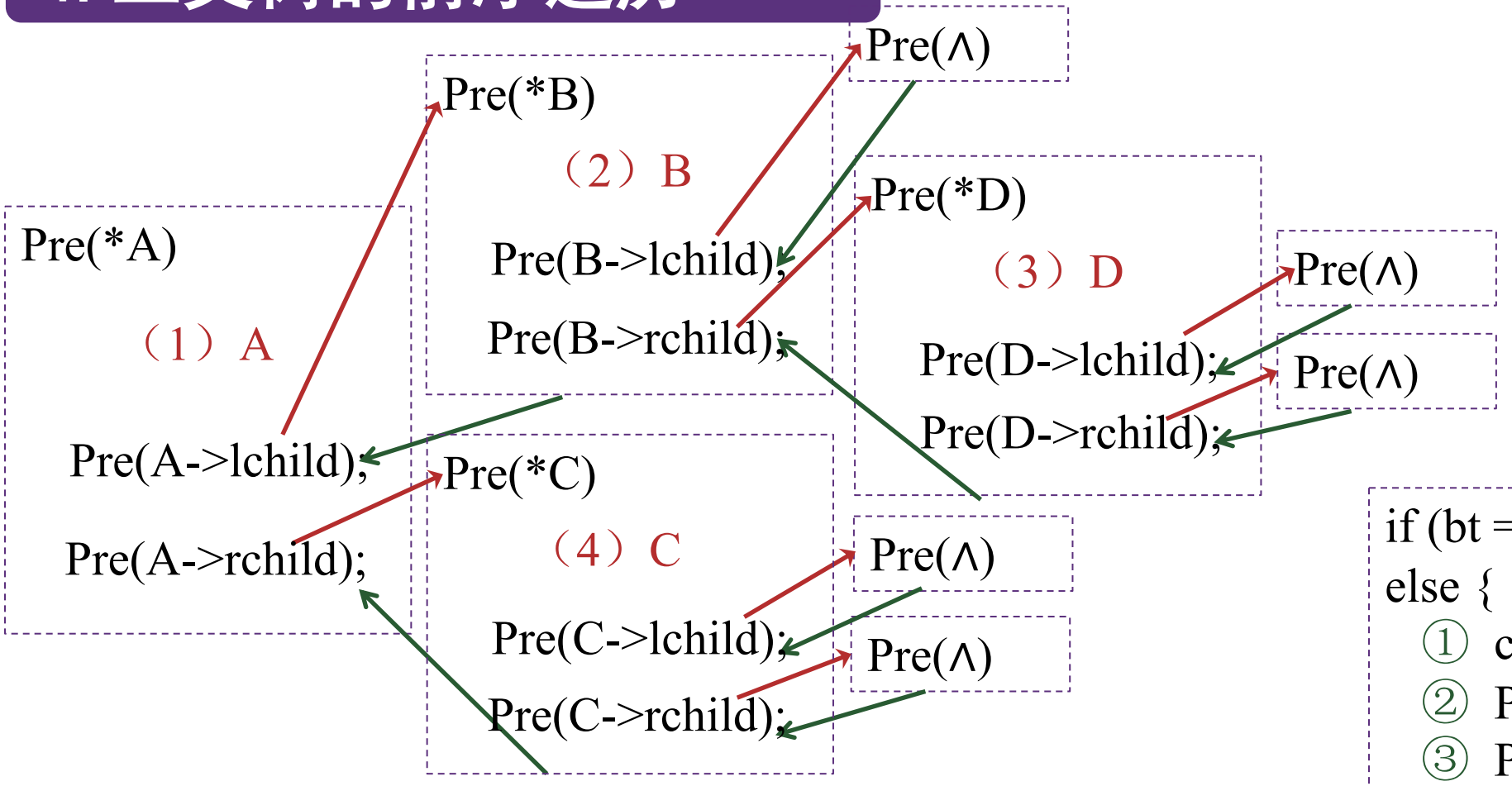


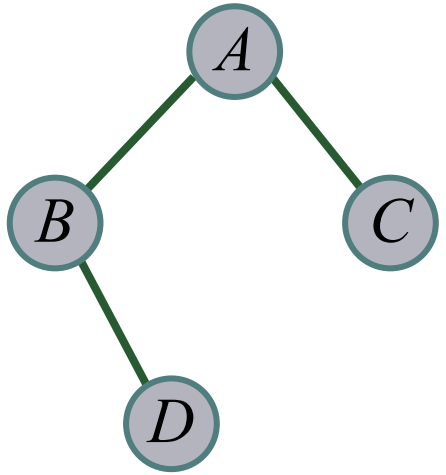



```
if (bt == nullptr)  return;
else {
   ①  cout << bt->data;
   ②  PreOrder(bt->lchild);
   ③  PreOrder(bt->rchild);
}
```

约定：*A 表示根指针指向结点 A

4. 二叉树的前序遍历



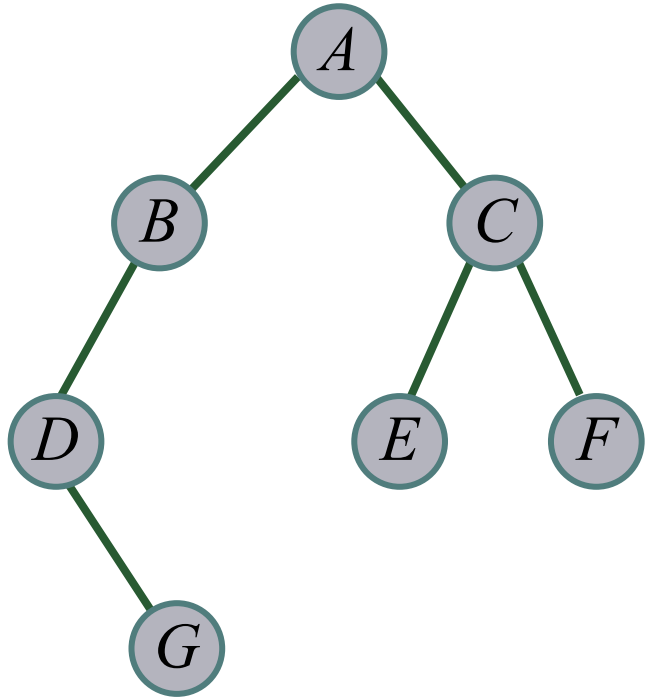



```
if (bt == nullptr)  return;
else {
   ①  cout << bt->data;
   ②  PreOrder(bt->lchild);
   ③  PreOrder(bt->rchild);
}
```

得到前序序列：A B D C

5. 二叉树的层序遍历

1. 队列 Q 初始化；
2. 如果二叉树非空，将根指针入队；
3. 循环直到队列 Q 为空
   - 3.1 q = 队列 Q 的队头元素出队；
   - 3.2 访问结点 q 的数据域；
   - 3.3 若结点 q 存在左孩子，则将左孩子指针入队；
   - 3.4 若结点 q 存在右孩子，则将右孩子指针入队；



出队 ← A B C D E F G ← 入队

遍历序列：A B C D E F G

5. 二叉树的层序遍历

```
template <typename DataType>
void BiTree<DataType> :: LevelOrder( )
{
    BiNode<DataType> *Q[100], *q = nullptr;
    int front = -1, rear = -1;
    if (root == nullptr) return; //空树
    Q[++rear] = root;   //存根结点
    while (front != rear)
    {
        q = Q[++front];      cout << q->data;
        if (q->lchild != nullptr)  Q[++rear] = q->lchild;
        if (q->rchild != nullptr)  Q[++rear] = q->rchild;
    }
}
```

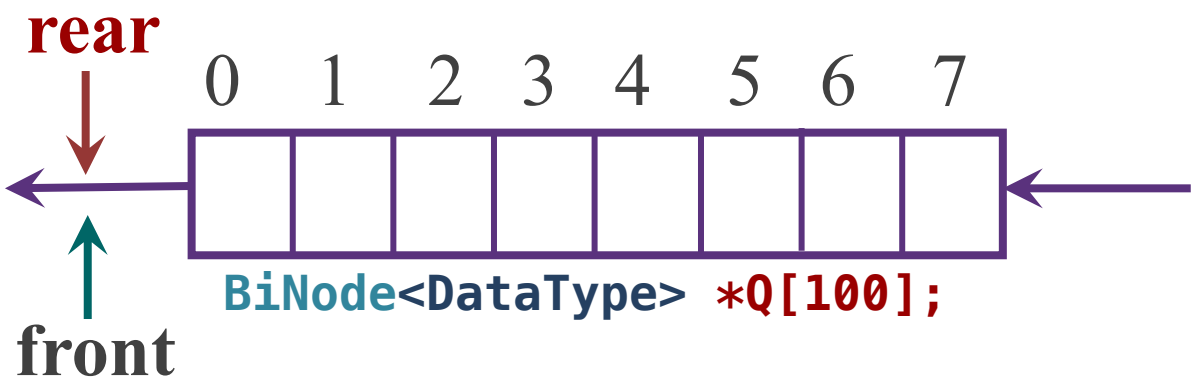
```
template <typename DataType>
struct BiNode
{
    DataType data;
    BiNode< DataType > *lchild, *rchild;
};
```

$O(n)$

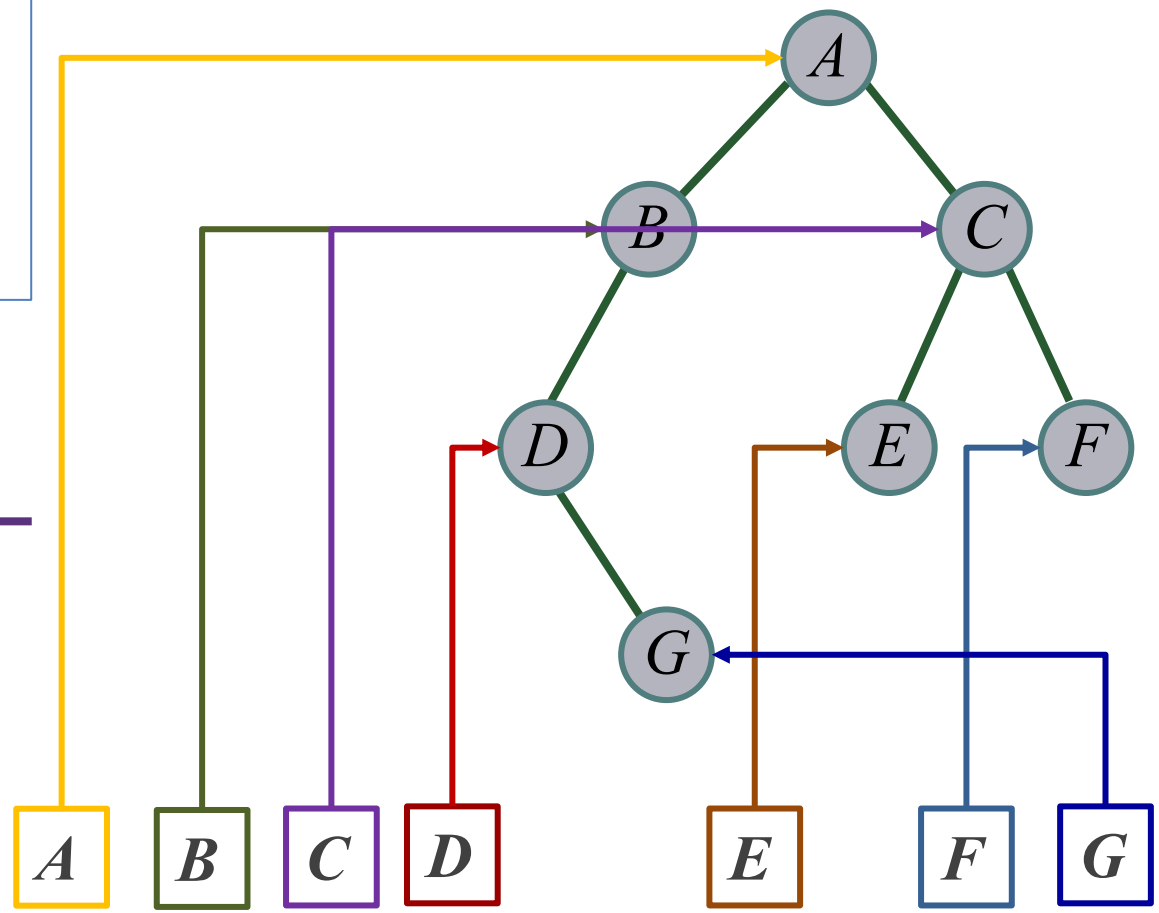
时间复杂度? ⇨ 每个结点进队出队一次 ⇨ $O(n)$

5. 二叉树的层序遍历

```
while (front != rear)
    {
        q = Q[++front];      cout << q->data;
        if (q->lchild != nullptr)  Q[++rear] = q->lchild;
        if (q->rchild != nullptr)  Q[++rear] = q->rchild;
    }
```




```
int front = -1, rear = -1; // 初始化
if (root == nullptr) return; //空树
Q[++rear] = root;
```

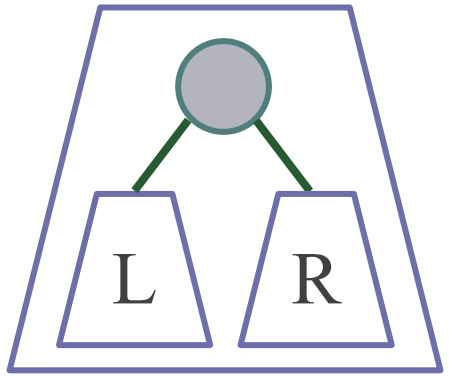


6. 二叉链表的建立

在内存中建立一棵二叉链表，如何输入二叉树的信息？

遍历将二叉树审视一遍，将非线性结构转换为线性结构

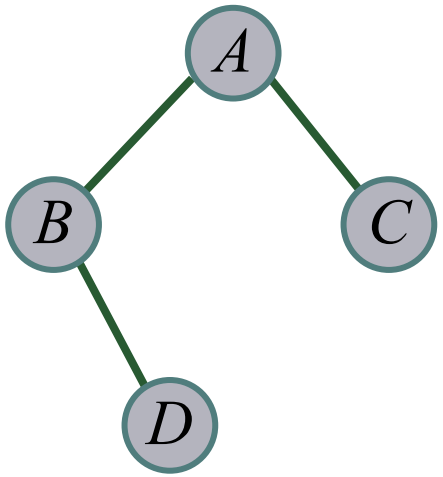
遍历是二叉树各种操作的基础，可以在遍历的过程中建立一棵二叉树

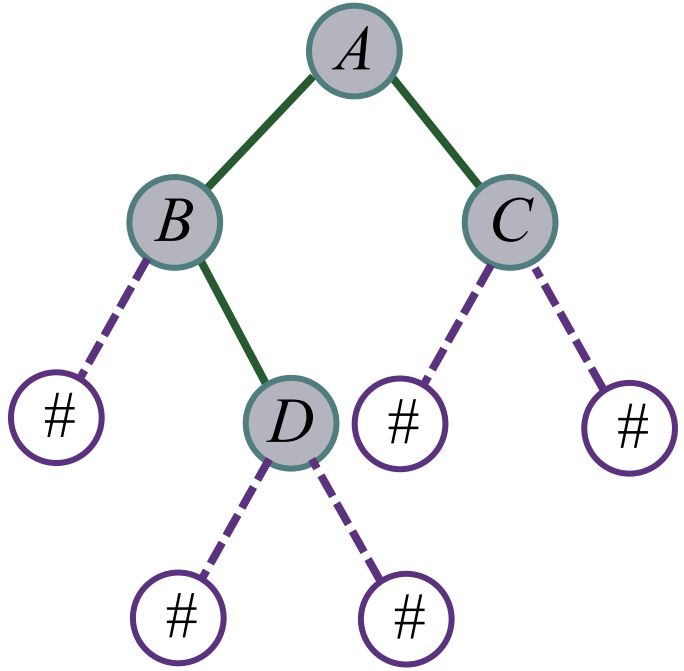


6. 二叉链表的建立

如何由一种遍历序列生成该二叉树?

扩展二叉树：将二叉树中每个结点的空指针引出一个虚结点，其值为一特定值如 '#'





扩展二叉树的前序遍历序列：*A B # D # # C # #*

6. 二叉链表的建立

```
template <typename DataType>
BiNode<DataType> *BiTree<DataType> :: Creat(BiNode<DataType> *bt)
{
    char ch;
    cin >> ch;                                  //输入结点的数据信息，假设为字符
    if (ch == '#') bt = nullptr;                //建立一棵空树
    else {
        bt = new BiNode<DataType>;  bt->data = ch;
        bt->lchild = Creat(bt->lchild);         //递归建立左子树
        bt->rchild = Creat(bt->rchild);         //递归建立右子树
    }
    return bt;
}
```

7. 二叉链表的销毁

为什么要销毁内存中的二叉链表?

二叉链表是动态存储分配，二叉链表的结点是在程序运行过程中动态申请的，在二叉链表变量退出作用域前，要释放二叉链表的存储空间

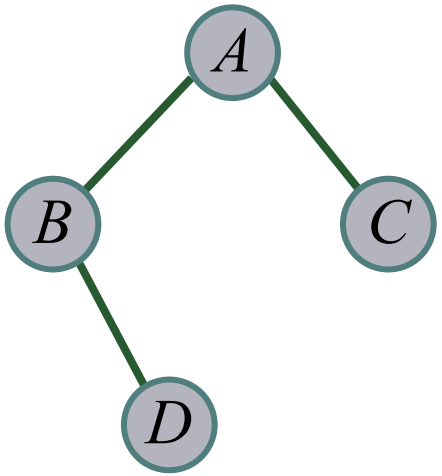
```
template <typename DataType>
void BiTree<DataType> :: Release(BiNode<DataType> *bt)
{
    if (bt == nullptr) return;
    else{
        Release(bt->lchild);          //释放左子树
        Release(bt->rchild);          //释放右子树
        delete bt;                    //释放根结点
    }
}
```

8. 二叉链表的使用

```
int main()
{
    cout <<"Please input BiTree:";
    BiTree<char> T;
    cout<<endl<<"PreOrder: ";
    T.PreOrder();
    cout<<endl<<"InOrder: ";
    T.InOrder();
    cout<<endl<<"PostOrder: ";
    T.PostOrder();
    cout<<endl<<"LevelOrder: ";
    T.LevelOrder();
    cout<<endl<<"The number of leafnodes:"<<T.BiTreeLeaf();
    cout<<endl;
    cout<<"The depth of the bitree:"<<T.BiDepth()<<endl;
    return 0;
}
```

```
Please input BiTree:AB#D##C##

PreOrder: ABDC
InOrder: BDAC
PostOrder: DBCA
LevelOrder: ABCD
The number of leafnodes:2
The depth of the bitree:3
```



小结

1. 理解二叉树的顺序存储结构及其特点
2. 熟练掌握二叉链表的定义及节点结构
3. 熟练掌握二叉树遍历的递归方法
4. 掌握二叉树的层序遍历方法
5. 掌握二叉树的建立和销毁方法

实验五、树和二叉树的实现与应用

一、实验目的

1．掌握二叉树的逻辑结构、存储结构及遍历方法。

2．掌握二叉链表的定义、节点结构与实现方法。

3．用C++语言实现相关算法，并上机调试。

二、实验内容

1. 实现二叉链表类，并完成前序、中序、后序和层序四种方式遍历。

2. 设计算法分别实现叶子结点个数统计和二叉树深度的计算。

3. 利用Huffman编码实现文本数据的压缩编码。 （扩展）

4. 给出测试过程和测试结果。

实验时间： 第10周周四晚 19:00-21:00
实验地点： 格物楼A216

作业

1. 编写一算法，求二叉树中叶子结点的个数，并通过函数值返回。

函数头：

```
int BiTree<DataType>::BiTreeLeaf(BiNode<DataType>*bt)
```

提示：树中叶子结点的个数等于其左、右子树中叶子结点的个数之和，递归算法。

2.编写一算法，求二叉树的深度，并通过函数值返回。

函数头：

```
int BiTree<DataType>::BiDepth(BiNode<DataType> *bt)
```

提示：树的深度等于其左、右子树中深度较大的一个加上1，递归算法。

Thank You !

Q & A