第五章

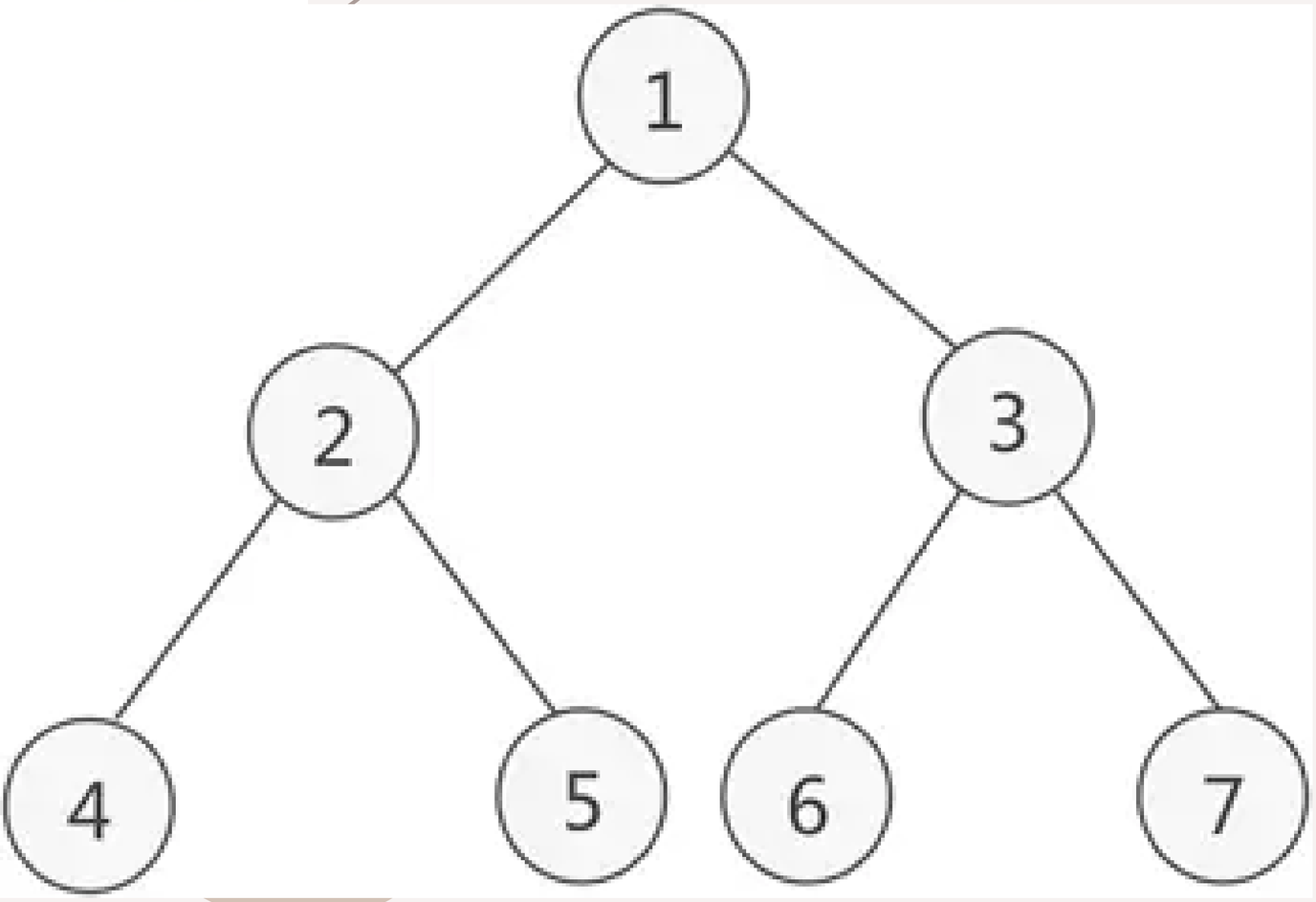
二叉树的遍历和树的深度

授课人：万全　　时间：2024/7/17

遍历二叉树的思路有 4 种，分别是：

- 前序遍历二叉树，有递归和非递归两种方式；
- 中序遍历二叉树，有递归和非递归两种方式；
- 后序遍历二叉树，有递归和非递归两种方式；
- 层次遍历二叉树，有递归和非递归两种方式。

遍历二叉树可以算作是对树存储结构做的最多的操作，既是重点，也是难点。



前序遍历：1 2 4 5 3 6 7

中序遍历：4 2 5 1 6 3 7

后序遍历：4 5 2 6 7 3 1

1) 先序遍历二叉树

所谓先序遍历二叉树，指的是从根结点出发，按照以下步骤访问二叉树的每个结点：

1. 访问当前结点；
2. 进入当前结点的左子树，以同样的步骤遍历左子树中的结点；
3. 遍历完当前结点的左子树后，再进入它的右子树，以同样的步骤遍历右子树中的结点；

举个简单的例子，下图是一棵二叉树：

先序遍历这棵二叉树的过程是：

访问根节点 1；

进入 1 的左子树，执行同样的步骤：

访问结点 2；

进入 2 的左子树，执行同样的步骤：

访问结点 4；

结点 4 没有左子树；

结点 4 没有右子树；

进入 2 的右子树，执行同样的步骤：

访问结点 5；

结点 5 没有左子树；

结点 5 没有右子树；

进入 1 的右子树，执行同样的步骤：

访问结点 3；

进入 3 的左子树，执行同样的步骤：

访问结点 6；

结点 6 没有左子树；

结点 6 没有右子树；

进入 3 的右子树，执行同样的步骤：

访问结点 7；

结点 7 没有左子树；

结点 7 没有右子树；

最终访问的次序是：1 2 4 5 3 6 7

前序遍历（中-左-右）

```
void qianxu(int p){//中-左-右
    if(!T[p])return ; //没有以p为根的子树
    printf("%d ",T[p]);//访问当前结点
    if(2*p<=n){
        qianxu(2*p);
    }//递归遍历左子树
    if(2*p+1<=n){
        qianxu(2*p+1);
    }//递归遍历右子树
}
```

中序遍历（左-中-右）

```
void zhongxu(int p){//左-中-右
    if(!T[p])return ; //没有以p为根的子树
    if(2*p<=n){
        zhongxu(2*p);
    }//递归遍历左子树
    printf("%d ",T[p]);//访问当前结点
    if(2*p+1<=n){
        zhongxu(2*p+1);
    }//递归遍历右子树
}
```

后序遍历（左-右-中）

```
void houxu(int p){//左-右-中
    if(!T[p])return ; //没有以p为根的子树
    if(2*p<=n){
        houxu(2*p);
    }//递归遍历左子树
    if(2*p+1<=n){
        houxu(2*p+1);
    }//递归遍历右子树
    printf("%d ",T[p]);//访问当前结点
}
```