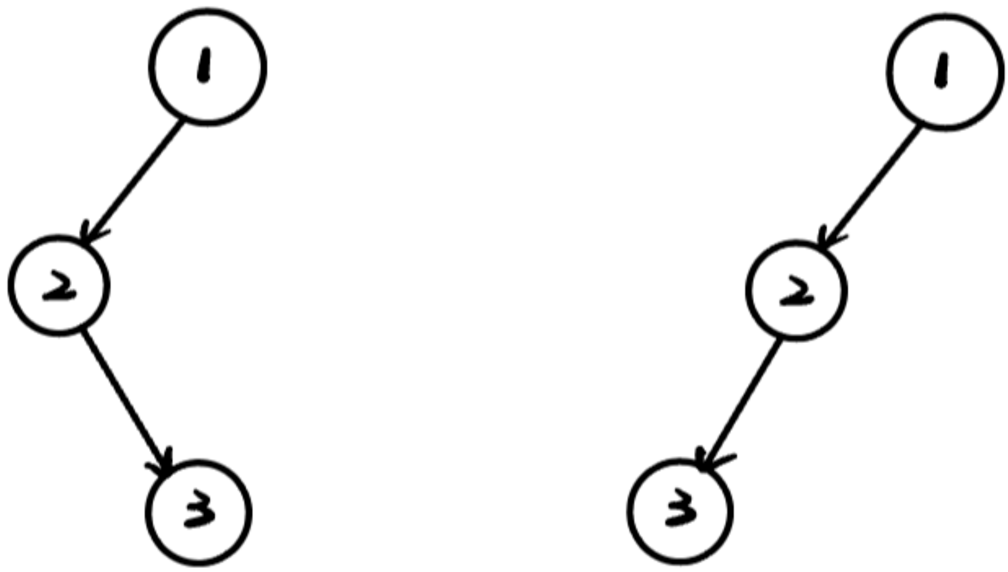
第五章 二叉树

1. 证明:判断以下叙述是否成立，并给出证明，若不成立，给出反例: 已知先序遍历序列和后序遍历序列可以确定唯一一棵二叉树。

解：该命题不成立，反例如下：
考虑如下先序遍历序列和后序遍历序列
preOrder:1,2,3
postOrder:3,2,1
根据以上两个序列，我们可以构造出如下两个二叉树，它们的前序与后序遍历序列均为以上两个序列，故已知先序遍历序列和后序遍历序列不可以确定唯一一棵二叉树。

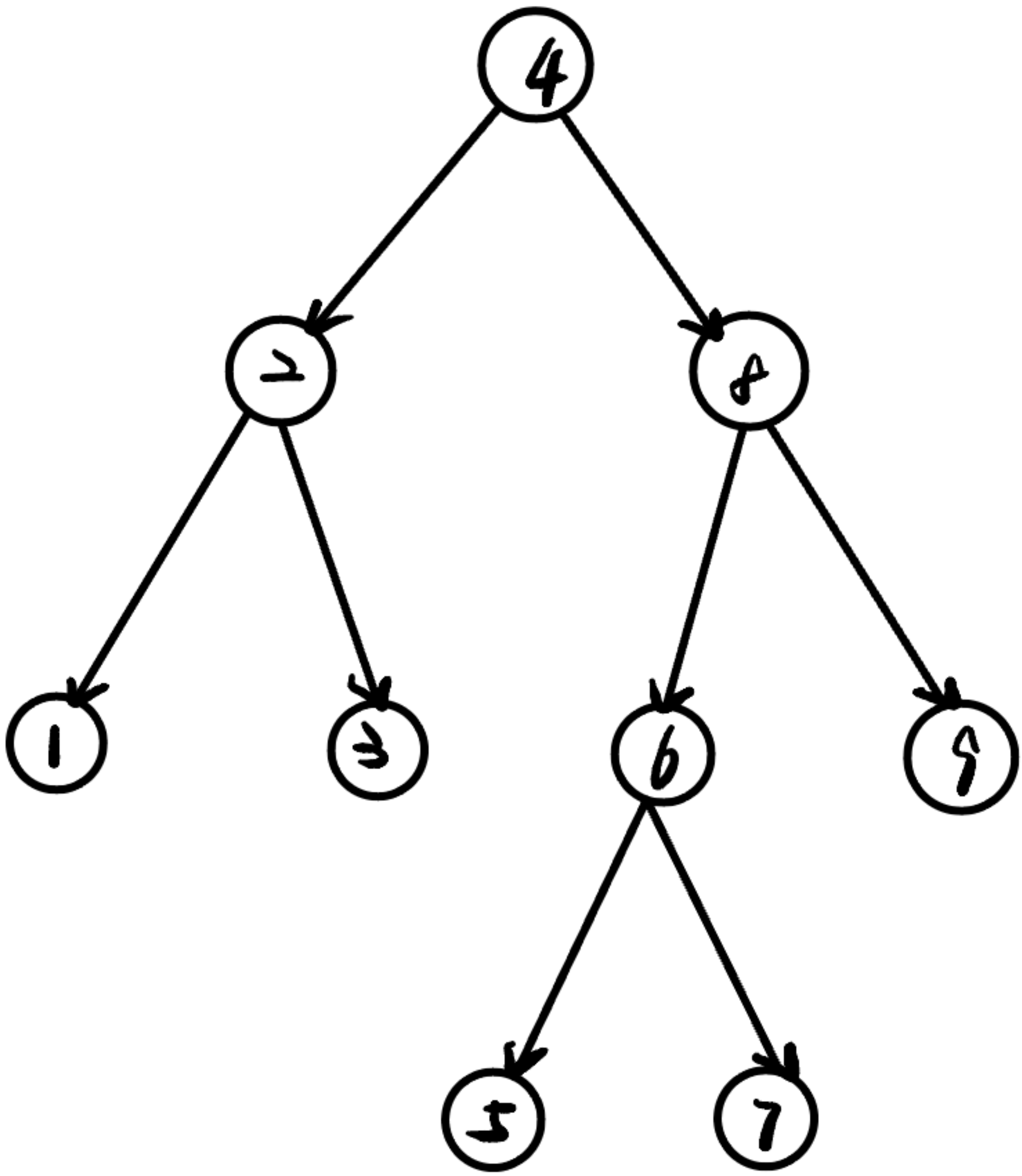


2. 在一棵表示有序集 S 的无重复元素二叉搜索树中，任意一条从根到叶子结点的路径将 S 分为 3 个部分:在该路径左边结点中的元素组成的集合 S1;在该路径上的结点中的元素组成的集合 S2;在该路径右边结点中的元素组成的集合 S3。S = S1∪S2∪S3。若对于任意的 a∈S1，b∈S2，c∈S3，判断以下表达式是否总是成立，若成立，简要叙述理由，若不成立， 给出反例:

1)a<b
2)b<c
3)a<c

解：
1)2)不成立，反例如下：

考虑如下一颗二叉树



根据二叉搜索树的中序有序性，这是一个二叉搜索树。
取一条路径为4->8->6->7，则S1={1,2,3,5},S2={4,8,6,7},S3={9}，显然取a=5,b=4有a>b。故1)不成立；
取一条路径为4->8->6->5，则S1={1,2,3},S2={4,8,6,5},S3={7,9}，显然取b=8,c=7有b>c。故2)不成立；
3)成立，理由如下： 寻找S1和S3中所有结点的最近共同祖先，根据二叉搜索树的定义知这

样的最近共同祖先结点必然存在。根据路径划分的规则，最近共同祖先必然是路径上某一结点且S1必然在共同祖先结点的左子树上，S3必然在共同祖先的右子树上，由二叉搜索树的定义知：任意a∊S1，c∊S3有a<c，证明完毕。

3. 设计一种算法，判断一颗二叉树的对称性(二叉树对称性即树中对称的节点 val 值相同， 如下图所示)。

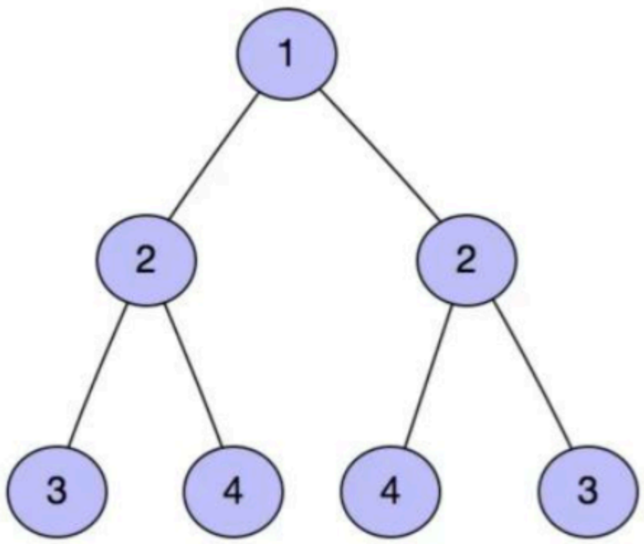


图 a 对称二叉树

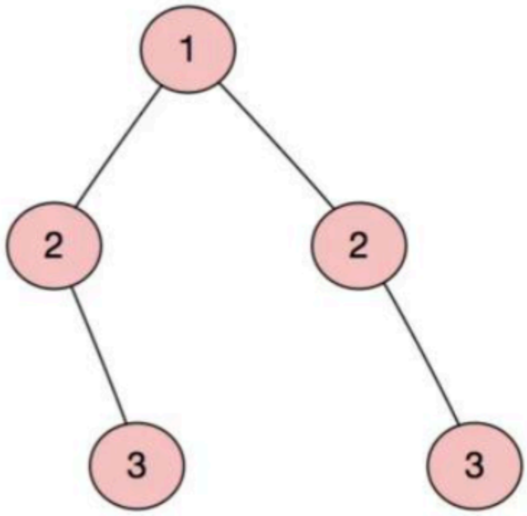


图 b 非对称二叉树

解：
算法描述：可以利用递归实现。首先判断二叉树根结点的左子结点与右子结点值是否相等。若值不相等则说明该二叉树不可能对称；若值相等则说明根结点及其左右子结点是对称的，之后判断左子树是否与右子树对称，使问题规模缩小，从而可以利用递归实现算法。 判断左子树是否与右子树对称即为判断左结点的左（右）子树是否与右结点的右（左）子树对称。

```
//算法代码如下
template<class T>
class treeNode{//定义树结点
public:
  T val;
  treeNode*left,*right;
  treeNode(int v=0,treeNode*l=nullptr,treeNode*r=nullptr):
    val(v),left(l),right(r){}
};
bool Symmetric(treeNode*p,treeNode*q){//判断左子树是否与右子树对称
  if(!p)//左子树为空
    return q==nullptr;//若右子树也为空则返回true，否则返回false
  if(!q)//右子树为空
    return p==nullptr;//若左子树也为空则返回true，否则返回false
  if(p->val!=q->val)//左、右结点值不等
```

```
    return false;
  //左、右结点不为空且值相等
  return Symmetric(p->left&&q->right)&&//判断左结点的左子树是否与右结点的右子树对称
         Symmetric(p->right&&q->left);//判断左结点的右子树是否与右结点的左子树对称
}
bool isSymmetric(treeNode*root){
  if(!root)//空树默认是对称的
    return true;
  return Symmetric(root->left,root->right);
}
```

复杂度分析： 假设二叉树有n个结点。
时间上算法遍历了整个二叉树，故时间复杂度为O(n)；
空间上递归调用栈的高度与二叉树的高度有关，最坏情况下为n，故空间复杂度为O(n)。

4. 设计一种算法，检查一个长度为 m(m>0)的 int 数组是否为一个大顶堆。

解： **算法描述**：根据堆的局部有序性，我们只需要遍历int数组对应的完全二叉树的所有内部结点，对于每一个内部结点检查其子结点的值是否严格小于它的值。若每一个结点都满足局部有序性，则说明该数组是一个大顶堆；反之则不是大顶堆。

```
//算法代码如下
bool isMaxHeap(int nums[],int m){//nums是给定数组，m是该数组的长度
  int i=0;
  while(i<m/2){//遍历所有内部结点
    if(i*2+1<m){//有左子结点（最后一个内部结点可能只有一个子结点）
      if(nums[i*2+1]>=nums[i])//不满足局部有序性，不是大顶堆
        return false;
    }
    if(i*2+2<m){//有右子结点 （最后一个内部结点可能只有一个子结点）
      if(nums[i*2+2]>=nums[i])//不满足局部有序性，不是大顶堆
        return false;
    }
    i++;//满足，继续向下遍历
  }
  return true;
}
```

复杂度分析：
时间上，该算法遍历了堆对应的完全二叉树的所有内部结点，共m/2个，故时间复杂度为O(m)；
空间上，该算法通过比较大小完成，需要的空间是常数的，并不需要额外开辟空间，故空间

复杂度为O(1)。

5. 对于一组权 W0, W1,..., Wn−1，说明怎么构造一个具有最小带权外部路径长度的扩充k叉树。试对权集 1,4,9,16,25,36,49,64,81,100 来具体构造一个这样的扩充三叉树。

解：
首先计算构造扩充k叉Huffman树需要的结点个数。
设需要n个结点，其中结点度数为i的结点个数为ni(i=0,k)，边的数目为e。
则有

$$n = n0 + nk, e = n - 1, e = k * nk$$

可以推出

$$n0 = (k-1) * nk + 1$$

即

$$n0 mod(k-1) = 1$$

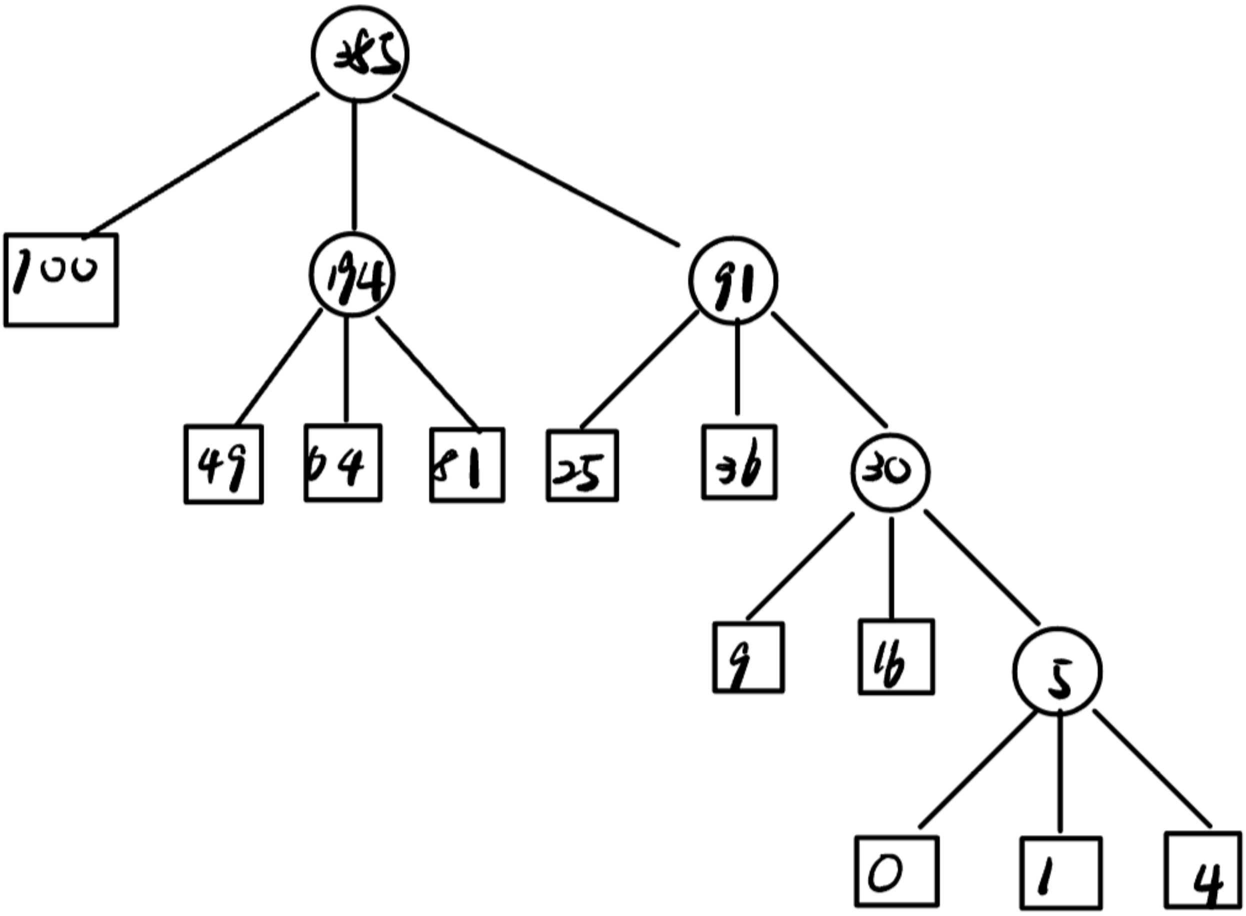
若权的个数n不满足该条件则应补加权值0直到满足该条件（权值为0的结点对带权外部路径长度无影响）。之后利用贪心法构造扩充k叉Huffman树。过程如下：
a.将最小的k个权构造成一棵k叉树，根结点为它们的权值之和；
b.将它们的权值从数组中删去并加入它们的权值之和；
c.再取最小的k个权，构造k叉树，根结点为它们的权值之和；
d.以此类推即可构造出一棵具有最小带权外部路径长度的扩充k叉树。

构造扩充三叉树的过程如下：

$\{1, 4, 9, 16, 25, 36, 49, 64, 81, 100\}$.

$n=10$, $k=3$, $n \% (k-1) = 0 \neq 1$

补上一个0.



6. 给定结点类型为 BinaryTreeNode 的 3 个指针 p、q、rt,假设 rt↑ 为根结点，求距离结点 p↑ 和结点 q↑ 最近的这两个结点的共同祖先结点。

解：

算法描述：

利用后序遍历实现算法，由于后序遍历先遍历左子树，再遍历右子树，最后遍历根结点的特性，可以在遍历过程中判断p和q在当前结点的左子树还是右子树上；若p,q分别在当前结点的两个子树上，则当前结点即为最近共同祖先（后序遍历是“从底层向上”遍历的）。考虑一些特殊情况，若当前结点为p(q)且q(p)在当前结点的子树上，则p(q)即为最近共同祖先。

```
//算法代码如下
BinaryTreeNode*ans=nullptr;
bool dfs(BinaryTreeNode*root, BinaryTreeNode*p, BinaryTreeNode*q){
  //dfs判断p,q是否在root及其子树中
  if(ans)//已经找到了最近共同祖先，直接返回
    return false;
  if(!root)//空树，p，q必然不在其中
    return false;
  //后序遍历框架
  bool inLeft=dfs(root->left,p,q);//判断p,q是否在左子树中
  bool inright=dfs(root->right,p,q); //判断p,q是否在右子树中
  if((inLeft&&inRight)||((root==p||root==q)&&(inLeft||inRight)))
  //两种找到最近共同祖先的条件
    ans=root;
  return inLeft||inRight||(root==p||root==q);//p,q在当前结点或其子树上
}
BinaryTreeNode* findLastCommonAncestor(
  BinaryTreenode*p,BinaryTreeNode*q,BinaryTreeNode*rt){
     dfs(rt,p,q);
     return ans;
}
```

复杂度分析：

算法利用了后序遍历一次遍历了二叉树，故时间复杂度为O(n)；

递归调用栈的高度与二叉树高度有关，最坏情况下二叉树高度即为n，故空间复杂度为O(n)。