第三章：树结构

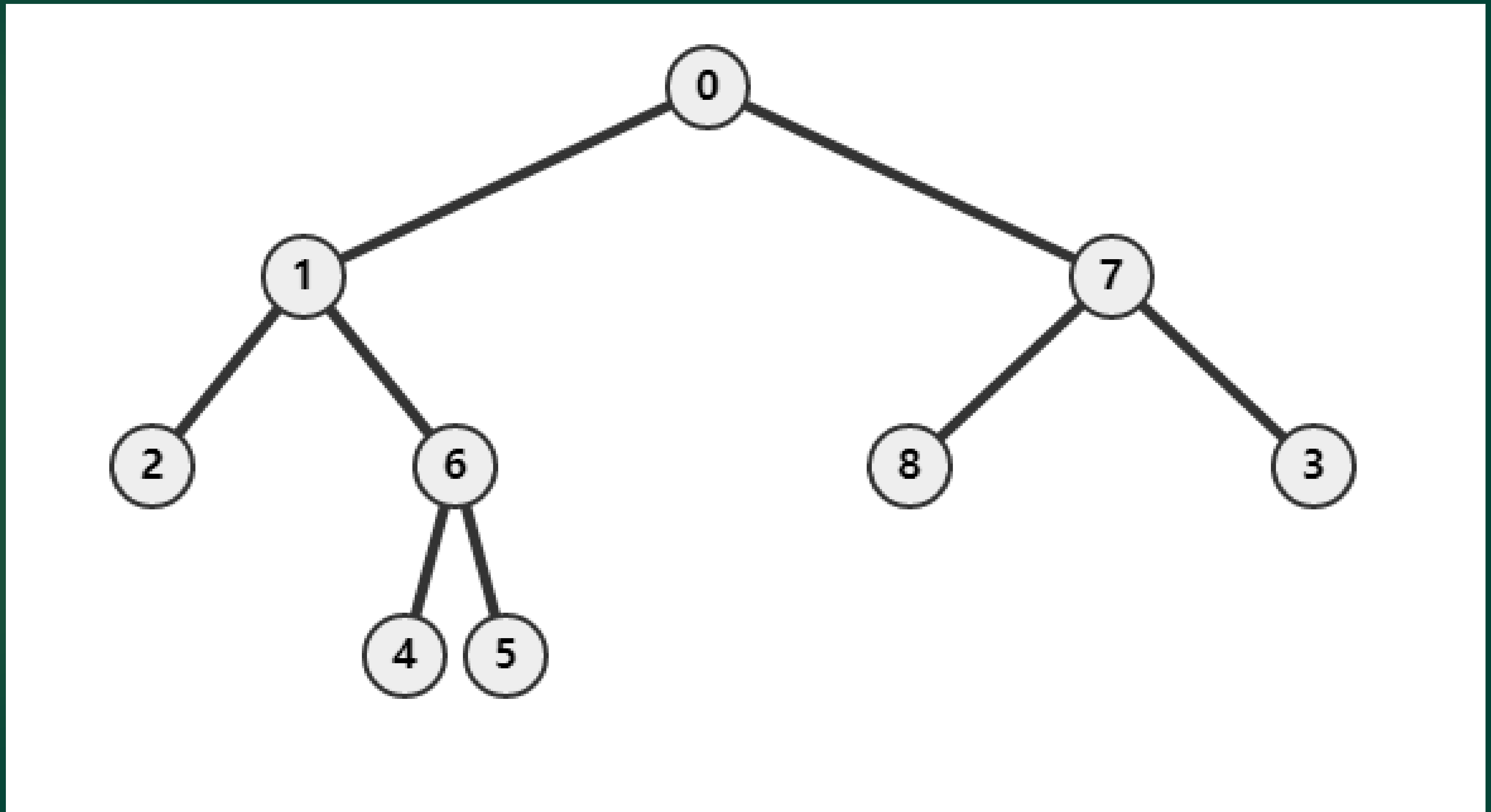
第二节：树模型

特殊的树-二叉树

在计算机科学中，二叉树是每个结点最多有两个子树的树结构。通常子树被称作“左子树”（left subtree）和“右子树”（right subtree）。

这部分知识，我们在之前的学习中已经介绍过，这里只是回顾一下。下图就是一棵二叉树。

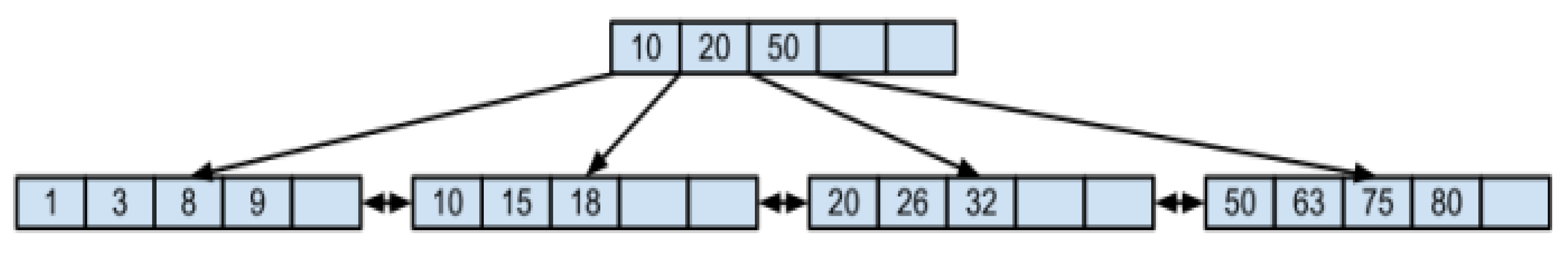
二叉树是特殊的树，也是很常用的数据结构，许多树模型以及数据结构模型，都是建立在二叉树的基础上



B树

1970 年，R.Bayer和E.mccreight提出了一种适用于外查找的，它是一种平衡的多叉树，称为**B**树。

与平衡二叉树类似，只是他是多叉的。



可以想象，假如每一层的分叉达到上千个，在这个树里做查找，近乎于顺序查找。

相比平衡二叉树所对应的二分查找，效率要低很多，那么这样的数据结构，有什么用处呢？

实际上是因为对于很多存储设备来说，顺序读的速度经常是随机读的数十倍，比如我们的机械硬盘，在这种设备上，用二叉的方式来做查询，时间将全部花在寻道上。这时B-Tree的优势就体现出来了。

3039叶子节点的路径

题目描述:

给出一棵n个节点的树，节点编号为1-n（根节点编号为1）。对于每一个叶子节点，输出从根到叶子的路径。（按照路径的字典序）。

输入描述:

第一行：1个数n（2 < n <= 1000），表示树的节点数量。

后面n-1行：每行2个数x y，表示节点x是节点y的父节点（1 <= x, y <= n）。

输出描述:

输出行数等于叶子节点的数量，每行对应从根到叶子节点的路径。路径中的数字为经过节点的编号。按照路径的字典序从小到大输出。

输入样例:
5
1 2
1 3
2 4
4 5
输出样例:
1 2 4 5
1 3

3039叶子节点的路径-解题思路

对每个节点的子节点，按照节点编号从小到大排序，这样在遍历树的过程中，路径的字典序是从小到大的。

遍历过程中，用一个数组保存保存从根到当前节点路径中的所有节点。遍历到叶子节点时直接输出即可。

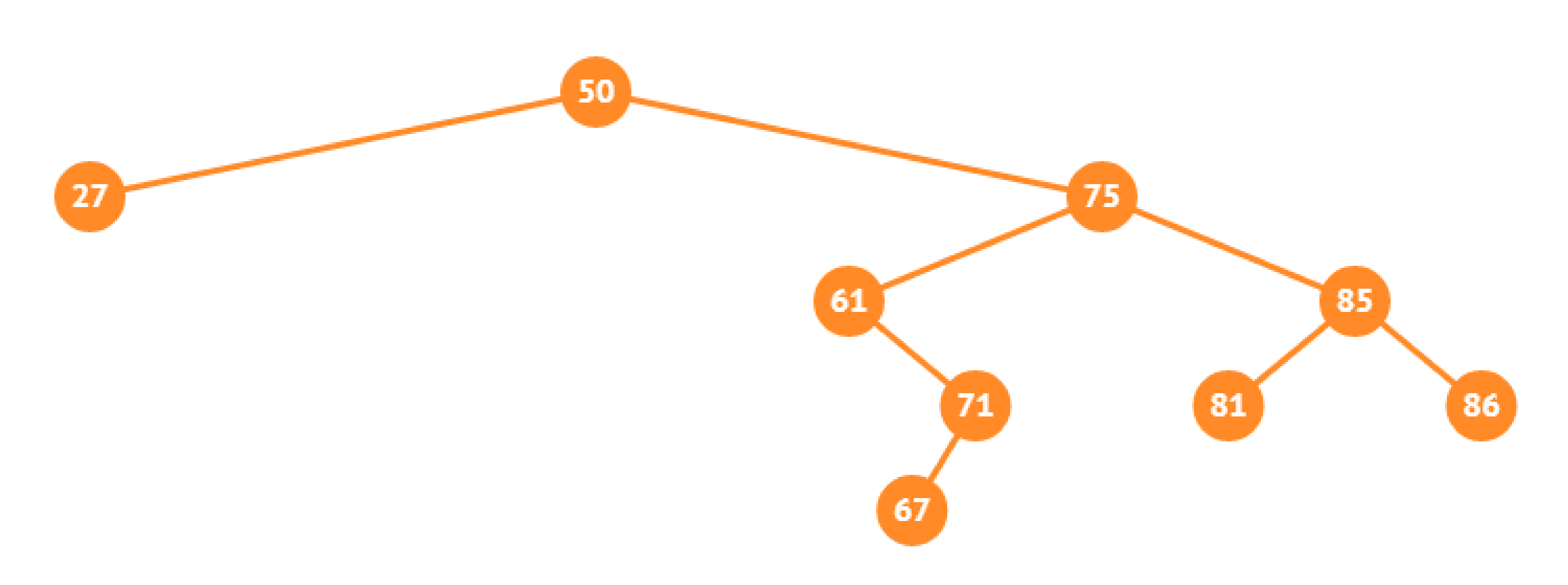
叶子节点的路径
参考答案

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e5+50;
vector<int> G[maxn];
int path[maxn];
void dfs(int rt, int fa, int deep)
{
    int count = 0;
    path[deep] = rt;
    for(int i = 0; i < G[rt].size(); i++)
      {
            int to = G[rt][i];
            if(to == fa) continue;
            count++;
            dfs(to, rt, deep + 1);
      }
      if(count == 0)
```

```
{
            for(int i = 0; i <= deep; i++)
               cout << path[i] << " ";
            cout << endl;
      }
}
int main(){
    int n, u, v;
    cin >> n;

    for(int i = 0; i < n - 1; i++){
        cin >> u >> v;
            G[u].push_back(v);
            G[v].push_back(u);
    }
    dfs(1,0,0);
    return 0;
}
```

最近公共祖先



最近公共祖先是相对于两个节点来说的，一般来说，最近公共祖先为节点u和节点v的最近的公共祖先。若u为v的祖先或者v为u的祖先，则LCA(u,v)就是作为祖先的那个节点。示例图中86和67的LCA是75， 61和85的LCA也是75 。

树的权值

在之前的内容中，叶子的深度代表从根到叶子，经过的边的数量。假如我们认为每条边的长度是不同的，

那么叶子的深度就是从根到叶子的所有边长度的和。

在这里边的长度称为边权（边的权值）。

同样，在计算子树大小时，如果每一个点都有不同的大小，那么子树的大小就是当前节点大小 + 所有子

孙的大小之和。

在这里点的大小称为点权（点的权值）。

之前所学的一些模型是比较特殊的模型，即边权和点权都等于 。作为更一般的情况，则可能边有边权，

点有点权。