二叉树

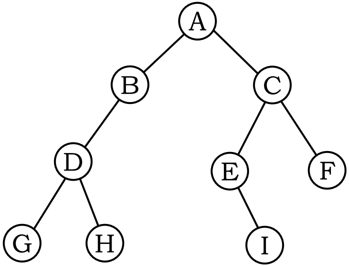
Binary Tree

河南省实验中学信息技术组

2026 年 02 月 23 日

二叉树定义

- 二叉树：每个结点最多有两个孩子 (二叉树的度最大为 2) 的树，这两个孩子分别称为左孩子和右孩子。
- 左孩子和右孩子是有顺序的，即只有一个孩子，也要区分其为左孩子还是右孩子。
- 以左孩子为根的子树被称为左子树，以右孩子为根的子树被称为右子树。
- 二叉树的定义是递归的。



特殊二叉树

- 斜树：所有结点都只有左子树的二叉树叫左斜树，所有结点都只有右子树的二叉树的叫右斜树。

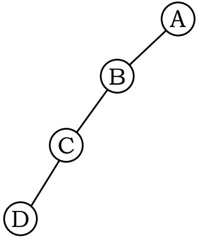


图: 左斜树

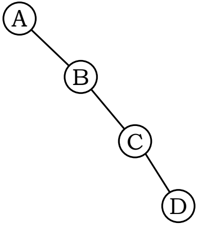
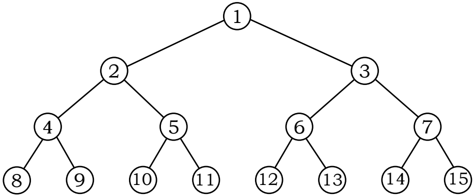


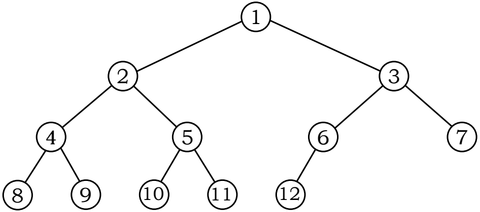
图: 右斜树

特殊二叉树

- 满二叉树：在一棵二叉树中，如果所有内部结点都有左右子树，并且所有的叶子都在同一层上，这样的二叉树称为满二叉树。
- 完全二叉树：对一棵具有 n 个结点的的二叉树按照层序编号，如果编号为 $i(1 \leq i \leq n)$ 的结点与同样深度的满二叉树中编号为 i 的结点在二叉树中的位置完全相同，则这棵二叉树称为完全二叉树。



图：满二叉树



图：完全二叉树

二叉树性质

- 性质 1：二叉树第 i 层上至多有 2^{i-1} 个结点。
- 性质 2：深度为 k 的二叉树最多有 2^k-1 个结点。
  证明：深度为 k 的二叉树结点数如果最多，则第 $i(1\le i\le k)$ 层上有 2^{i-1} 个结点，则总的结点数为

$$\sum_{i=1}^{k}2^{i-1}=\frac{1\times(1-2^k)}{1-2}=2^k-1$$

- 性质 3：对于任意一棵二叉树，如果其叶子结点有 n_0 个，度为 2 的结点有 n_2 个，则 $n_0=n_2+1$。
  证明：二叉树中只有度为 0,1,2 的结点，设度为 1 的结点个数为 n_1，则树的结点总数为 $n=n_0+n_1+n_2$。再看二叉树中边的数目 e，因为二叉树除了根结点之外每个结点都有一条入边，所以 $e=n-1=n_0+n_1+n_2-1$。换个角度思考，度为 2 的结点出发的边有 2 条，度为 1 的结点出发的边有 1 条，度为 0 的结点出发的边有 0 条，故 $e=n_1+2\times n_2$。所以，

$$n_0+n_1+n_2-1=n_1+2\times n_2\Longrightarrow n_0=n_2+1$$

二叉树性质

- 性质 4：具有 n 个结点的完全二叉树的深度为 $\lceil \log_2(n+1) \rceil$。
  证明：深度为 k 的二叉树最多结点个数 $n \le 2^k - 1$，最少结点个数 $n > 2^{k-1} - 1$，因此

$$
\begin{aligned}
2^{k-1} - 1 < n \le 2^k - 1 \\
2^{k-1} < n + 1 \le 2^k \\
k - 1 < \log_2(n+1) \le k
\end{aligned}
$$

  因深度只能是整数，因此 $k = \lceil \log_2(n+1) \rceil$。

二叉树性质

- 性质 5：如果将一个有 n 个结点的完全二叉树的结点按照层次从上向下，每层从左到右编号，则对于任意一个结点 $i(1 \leq i \leq n)$，有：
  1. 如果 $i = 1$，则结点 i 是完全二叉树的根，无双亲；如果 $i > 1$，则其双亲结点编号为 $\lfloor \frac{i}{2} \rfloor$。
  2. 如果 $2i > n$，则结点 i 无左孩子；否则，其左孩子结点编号为 $2i$。
  3. 如果 $2i + 1 > n$, 则结点 i 无右孩子；否则，其右孩子结点编号为 $2i + 1$。
  4. 若结点编号 i 为偶数，则它是左孩子，它的右兄弟结点为 $i + 1$(如果存在)。
  5. 若结点编号 i 为奇数数且 $i \neq 1$，则它是右孩子，它的左兄弟结点为 $i - 1$。
  6. 结点 i 所在的层次为 $\lceil \log_2(i + 1) \rceil$。

二叉树存储

- 在考虑二叉树的存储时，需要考虑如何存储结点中的数据和结点之间的关系。
- 树上的结点信息需要记录：结点数据、双亲、孩子 (最多 2 个)。

$data$	p	lc	rc

```
const int N = 100;
string data[N + 10];  // 数据
int p[N + 10], lc[N + 10], rc[N + 10]; // 双亲 左孩子 右孩子
int n = 0, rt = 0; // 树结点数目 根结点编号
```

二叉树存储

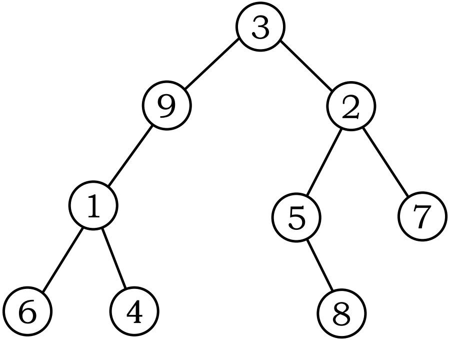


图: 二叉树

下标	p	lc	rc
1	9	6	4
2	3	5	7
3	0	9	2
4	1	0	0
5	2	0	8
6	1	0	0
7	2	0	0
8	5	0	0
9	3	1	0

表: 存储示意

二叉树遍历

- 二叉树的遍历是指从根结点出发，按某种次序依次访问二叉树中的所有结点，使得每个结点被访问一次且仅被访问一次。
- 按照访问次序的不同共有四种遍历方法:
  - 前序遍历
  - 中序遍历
  - 后序遍历
  - 层次遍历

遍历方式

- 前序遍历：先访问根结点，然后前序遍历左子树，再前序遍历右子树。
- 中序遍历：先中序遍历左子树，然后访问根结点，再中序遍历右子树。
- 后序遍历：先后序遍历左子树，然后后序遍历右子树，再访问根结点。
- 层次遍历：从树的第一层开始从上向下逐层遍历，在同一层中，先访问左孩子 (如果有) 再访问右孩子 (如果有)。

前序遍历	3	9	1	6	4	2	5	8	7
中序遍历	6	1	4	9	3	5	8	2	7
后序遍历	6	4	1	9	8	5	7	2	3
层次遍历	3	9	2	1	5	7	6	4	8

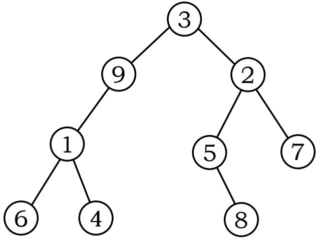


图: 二叉树

【例】二叉树的遍历

【题目描述】

给定一棵有 n 结点的二叉树，它的结点编号从 $1 \sim n$，求出它的前序、中序、后序、层次遍历的序列。

【输入格式】

第一行一个整数 $n(1 \leq n \leq 100)$，表示二叉树的结点个数；

接下来 n 行，其中第 i 行为二叉树第 i 个结点的左孩子和右孩子结点编号。

【输出格式】

输出共 4 行，分别为二叉树的前序、中序、后序、层次遍历的序列。

【样例输入】

```
9
6 4
5 7
9 2
0 0
0 8
0 0
0 0
0 0
1 0
```

【样例输出】

```
3 9 1 6 4 2 5 8 7
6 1 4 9 3 5 8 2 7
6 4 1 9 8 5 7 2 3
3 9 2 1 5 7 6 4 8
```

遍历实现

- 前序遍历、中序遍历、后续遍历算法定义是递归的，所以实现时可以用递归的方法。
- 层次遍历需要使用队列来实现。
  - 首先，将根结点放入队列。
  - 只要队列不为空，取出队头 (访问)，将其孩子结点放入队列。

遍历实现

- 前序遍历：先访问根结点，再前序遍历左子树，最后前序遍历右子树。

```
void preorder(int x) // 前序遍历以 x 为根的子树
{
    if(x == 0) return;      // 先判断访问的树 (后期是子树) 是否为空
    cout << x << " ";       // 访问根结点 打印结点数据
    preorder(lc[x]);        // 前序遍历根结点的左子树
    preorder(rc[x]);        // 前序遍历根结点的右子树
}
```

- 中序遍历：先中序遍历左子树，再访问根结点，最后中序遍历右子树。

```
void inorder(int x) // 中序遍历以 x 为根的子树
{
    if(x == 0) return;       // 先判断访问的树 (后期是子树) 是否为空
    inorder(lc[x]);          // 中序遍历根结点的左子树
    cout << x << " ";        // 访问根结点 打印结点数据
    inorder(rc[x]);          // 中序遍历根结点的右子树
}
```

遍历实现

- 后序遍历：先后序遍历左子树，再访问右子树，最后后序遍历根结点。

```
void postorder(int x) // 后序遍历以 x 为根的子树
{
    if(x == 0) return;      // 先判断访问的树 (后期是子树) 是否为空
    postorder(lc[x]);       // 后序遍历根结点的左子树
    postorder(rc[x]);       // 后序遍历根结点的右子树
    cout << x << " ";       // 访问根结点 打印结点数据
}
```

- 层次遍历：从根结点开始逐层遍历，每一层先遍历左孩子然后右孩子。

```
void levelorder(int rt)   // 二叉树层次遍历
{
    if(rt == 0) return;         // 空树 直接结束
    queue<int> q; q.push(rt);   // 将根入队
    while(!q.empty())
    {
        int x = q.front(); q.pop(); // 队头元素出队
        cout << x << " ";           // 访问当前队头元素 打印值
        if(lc[x]) q.push(lc[x]);    // 左孩子非空 入队
        if(rc[x]) q.push(rc[x]);    // 右孩子非空 入队
    }
}
```

遍历应用

- 二叉树的遍历方法将二叉树的复杂结构转化为线性序列。
- 二叉树的遍历方法提供了对二叉树中结点的不同访问顺序，可以在遍历过程中对结点进行不同的处理。
- 二叉树的后序遍历是先遍历子树，后遍历根，与实际生活中解决问题的思路类似，所以应用较多，也是树型动态规划实现的基本方式。

【例】二叉树大小

【题目描述】
给定一棵有 n 结点的二叉树，它的结点编号从 $1 \sim n$，现在需要询问它的某些结点为根的子树大小。
【输入格式】
第一行一个整数 n，表示树的大小。
接下来 n 行，第 i 行两个整数，分别表示结点 i 的左右孩子，如果没有对应孩子，则数字为 0。
接下来一行一个整数 q，表示询问的次数。
接下来 q 行，每行一个整数 x，表示询问以 x 为根的子树大小。
【输出格式】
共 q 行，每行一个整数，表示对应子树大小。
【数据范围】
对于 60% 的数据,$1 \leq n \leq 2000$。
对于 100% 的数据，$1 \leq n \leq 5 \times 10^4, 1 \leq q \leq n$。

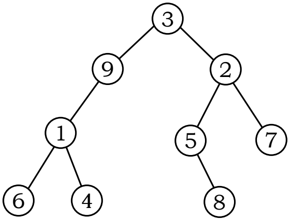
【例】二叉树大小

【样例输入】

```
9
6 4
5 7
9 2
0 0
0 8
0 0
0 0
0 0
1 0
3
9
5
4
```

【样例输出】

```
4
2
1
```



【例】二叉树大小

- 遍历根结点的左子树和右子树，计算左子树和右子树的结点个数，然后加上 1(根结点)。

```
int size(int x)
{
    if(x == 0) return 0;  // 空树 大小为 0
    // 树大小 = 左子树大小 + 右子树大小 + 1
    return size(lc[x]) + size(rc[x]) + 1;
}
```

- 时间复杂度：$O(N^2)$。

【例】二叉树大小

- 以任意一个结点为根的子树的大小不会改变，因此可以预先存储每个子树的大小 (记忆化)。
- 对于每次询问，直接输出存储的子树大小值即可。

```
int d[N + 10];  // d[x] 表示以 x 为根的子树大小

int size(int x)
{
    if(x == 0) return 0;  // 空树 大小为 0
    if(d[x]) return d[x]; // 如果子树已经计算过就直接返回
    // 树大小 = 左子树大小 + 右子树大小 + 1
    return d[x] = size(lc[x]) + size(rc[x]) + 1;
}
```

- 时间复杂度：$O(N)$。

【例】二叉树深度

【题目描述】

给定一棵有 n 结点的二叉树，它的结点编号从 $1 \sim n$，现在需要询问它的某些结点为根的子树深度 (高度)。

【输入格式】

第一行一个整数 n，表示树的大小。

接下来 n 行，第 i 行两个整数，分别表示结点 i 的左右孩子，如果没有对应孩子，则数字为 0。

接下来一行一个整数 q，表示询问的次数。

接下来 q 行，每行一个整数 x，表示询问以 x 为根的子树深度 (高度)。

【输出格式】

共 q 行，每行一个整数，表示对应子树大小。

【数据范围】

对于 60% 的数据,$1 \leq n \leq 2000$。

对于 100% 的数据，$1 \leq n \leq 5 \times 10^4, 1 \leq q \leq n$。

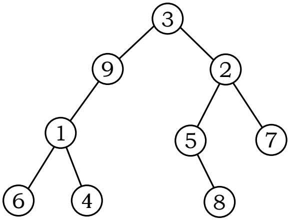
【例】二叉树深度

【样例输入】

```
9
6 4
5 7
9 2
0 0
0 8
0 0
0 0
0 0
1 0
3
9
5
4
```

【样例输出】

```
3
2
1
```



【例】二叉树深度

- 遍历根结点的左子树和右子树，计算左子树和右子树高度并取最大值，然后加 1。

```
int depth(int x)
{
    if(x == 0) return 0;  // 空树 深度为 0
    // 树深度 = 左子树和右子树深度最大值 + 1
    return max(depth(lc[x]), depth(rc[x])) + 1;
}
```

- 时间复杂度：$\mathrm{O}(N^2)$。

【例】二叉树深度

- 以任意一个结点为根的子树的深度 (高度) 不会改变，因此可以预先存储每个子树的深度 (记忆化)。
- 对于每次询问，直接输出存储的子树深度 (高度) 值即可。

```
int d[N + 10];  // d[x] 表示以 x 为根的子树深度 (高度)
int depth(int x)
{
    if(x == 0) return 0;  // 空树 深度为 0
    if(d[x]) return d[x]; // 如果子树已经计算过就直接返回
    // 树深度 = 左子树和右子树深度最大值 + 1
    return d[x] = max(depth(lc[x]), depth(rc[x])) + 1;
}
```

- 时间复杂度：$O(N)$。

【例】判断两棵二叉树是否相同

二叉树

河南省实验中学信息技术组

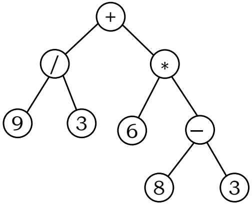

```
int lc1[N + 10], rc1[N + 1];   // 第 1 棵树
int lc2[N + 10], rc2[N + 1];   // 第 2 棵树

bool equal(int x, int y)
{
    if(x == 0 && y == 0) return true;  // 两棵树都是空树 相等
    // 根结点都非空 而且数据相同
    if(x != 0 && y != 0 && x == y)
    {
        // 左子树是否相等
        bool lequal = equal(lc1[x], lc2[y]);
        // 右子树是否相等
        bool requal = equal(rc1[x], rc2[y]);
        if(lequal && requal) return true;
    }
    return false;
}
```

【例】表达式树

【题目描述】

中缀表达式可以转化为一棵由运算符和运算数构成的二叉树。
例如，中缀表达式 9/3+6*(8-3) 可以转化为如下二叉树：



通过观察可以发现，操作数都在叶子结点中，操作符在内部结点中。
表达式树的后序遍历就是后缀表达式，在对子树进行后序遍历后，直接用根结点的运算符对左右子树的结果进行运算即可得出表达式的值。
现在给定一棵二叉树，请你求出对应表达式的值。

【例】表达式树

【输入格式】

第一行一个整数 $n(2 \leq n \leq 50)$，表示操作数的个数。
第一行 n 个整数，表示 n 个操作数 (叶子结点)，它们在二叉树的结点编号分别为 1 到 n。
接下来 $n-1$ 行，表示 $n-1$ 个操作符，它们在二叉树中的结点编号分别为 $n+1$ 到 $2n-1$。
每行首先输入一个运算符 (+-*/)，然后输入两个整数分别表示左孩子和右孩子的编号。对于除法运算，使用整除。

【输出格式】

一行一个整数，表示表达式的值 (输入保证运算中间和最后结果不超过`int`)。

【样例输入】

```
5
9 3 6 8 3
/ 1 2
+ 6 8
* 3 9
- 4 5
```

【样例输出】

```
33
```

【例】表达式树

- 表达式树中有两种结点，下标 $1 \sim n$ 的为数字，下标 $n+1 \sim 2n-1$ 的为运算符，存储结点数据时可以都存储为整数 (字符本质上也是数字)，通过下标的值来区分。
- 对于数字结点，直接返回结点数据；对于运算符结点，先分别计算出左右子树的的值，然后根据运算符返回运算结果。

```
int cal(int x)  // 计算以 x 为根的表达式树的值
{
    if(x >= 1 && x <= n) return data[x];   // 数字结点
    int lval = cal(lc[x]);  // 计算左表达式树的值
    int rval = cal(rc[x]);  // 计算右表达式树的值
    if(data[x] == '+') return lval + rval;
    else if(data[x] == '-') return lval - rval;
    else if(data[x] == '*') return lval * rval;
    else if(data[x] == '/') return lval / rval;
    else return 0;
}
```

【例】前序建树

【题目描述】
给定二叉树的前序遍历序列，请你建立二叉树。
单独给出前序遍历序列无法建树，但是给定的前序遍历序列比较特殊，所有结点都有左右孩子，如果没有以 0 代替。
输入保证数的结点个数 n 不超过 2000，输入的数据一定能建立一棵二叉树。
【输入格式】
一行若干个整数，表示前序遍历序列，孩子为空以 0 表示。
【输出格式】
若干行，第 i 两个整数，分别表示结点 i 的左右孩子，没有孩子则对应位置用 0 表示。

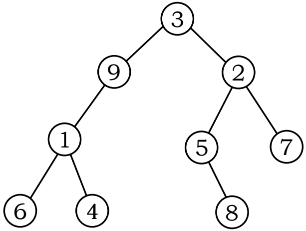
【例】前序建树

【样例输入】

```
3 9 1 6 0 0 4 0 0 0 2 5 0 8 0 0 7 0 0
```

【样例输出】

```
6 4
5 7
9 2
0 0
0 8
0 0
0 0
0 0
1 0
```



【例】前序建树

- 二叉树的前序遍历无法正常建树，但是修改后的前序遍历序列每个结点都有左右孩子，如果没有以 0 代替。
- 第一个输入的数字为根结点，然后是左子树的“前序遍历”，其次是右子树的“前序遍历”，因此可以用递归一边输入一边建树。

```cpp
// 前序遍历建树 返回树根结点下标
int build()
{
    int x; cin >> x;
    n = max(x, n);
    if(x == 0) return 0;  //空树
    lc[x] = build(); // 递归建左树
    rc[x] = build(); // 递归建右树
    return x;
}

rt = build();
```

【例】前序中序建树

【题目描述】

已知一棵二叉树的前序序列和中序序列，建立一棵二叉树。

【输入格式】

第一行为一个整数 $n(n \leq 2000)$，表示树的大小。

第二行为二叉树的前序序列。

第三行为二叉树的中序序列。

【输出格式】

输出共 n 行，第 i 行两个整数，表示结点 i 的左右孩子，没有孩子则对应位置用 0 表示。

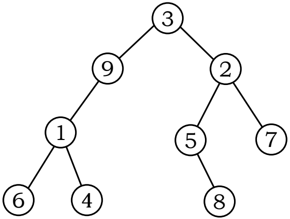
【例】前序中序建树

【样例输入】

```
9
3 9 1 6 4 2 5 8 7
6 1 4 9 3 5 8 2 7
```

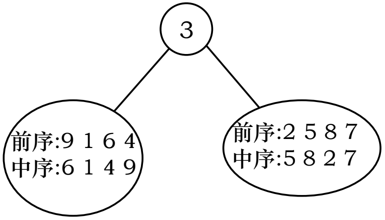
【样例输出】

```
6 4
5 7
9 2
0 0
0 8
0 0
0 0
0 0
1 0
```

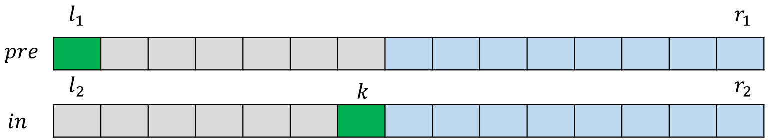


【例】前序中序建树

- 根据前序遍历的的结果3 9 1 6 4 2 5 8 7，可知树的根为 3。
- 再结合中序遍历的结果 6 1 4 9 3 5 8 2 7，可知 6 1 4 9 是左子树结点，5 8 2 7 是右子树结点。
- 此时可知左子树的前序遍历结果为 9 1 6 4，中序遍历结果为 6 1 4 9，递归调用前序中序遍历建树的方法建立左子树，同理可以建立右子树。



【例】前序中序建树




```cpp
int pre[N + 10], in[N + 10];    // 前序中序建树
int build(int l1, int r1, int l2, int r2)  // pre[l1~r1] in[l2~r2]
{
    if(r1 < l1) return 0;          // 首先要判断是否为空树
    int x = pre[l1];                   // 根结点数据为前序遍历序列的第一个元素
    int k = 0;                     // 中序遍历序列的根结点下标
    for(int i = l2; i <= r2; ++i) if(x == in[i]) k = i;
    // 利用左子树的前序和中序遍历序列建树 并设置其为根结点的左孩子
    lc[x] = build(l1 + 1, l1 + k - l2, l2, k - 1);
    // 利用右子树的前序和中序遍历序列建树 并设置其为根结点的右孩子
    rc[x] = build(l1 + k - l2 + 1, r1, k + 1, r2);
    return x;
}

rt = build(1, n, 1, n);
```

【例】中序后序建树

【题目描述】
已知一棵二叉树的中序序列和后序序列，建立一棵二叉树。
【输入格式】
第一行为一个整数 $n(n \leq 2000)$，表示树的大小。
第二行为二叉树的中序序列。
第三行为二叉树的后序序列。
【输出格式】
输出共 n 行，第 i 行两个整数，表示结点 i 的左右孩子，没有孩子则对应位置用 0 表示。

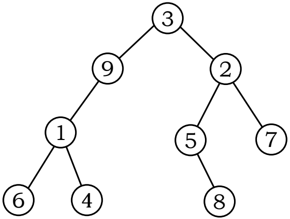
【例】中序后序建树

【样例输入】

```
9
6 1 4 9 3 5 8 2 7
6 4 1 9 8 5 7 2 3
```

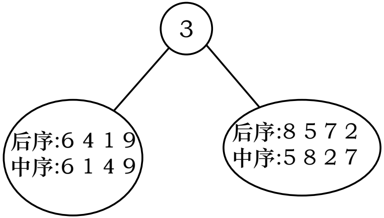
【样例输出】

```
6 4
5 7
9 2
0 0
0 8
0 0
0 0
0 0
1 0
```

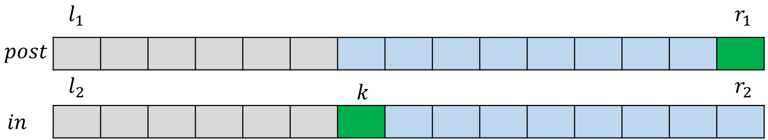


【例】中序后序建树

- 根据后序遍历的的结果 6 4 1 9 8 5 7 2 3，可知树的根为 3。
- 再结合中序遍历的结果 6 1 4 9 3 5 8 2 7，可知 6 1 4 9 是左子树结点，5 8 2 7 是右子树结点。
- 此时可知左子树的后序遍历结果为 6 4 1 9，中序遍历结果为 6 1 4 9，递归调用中序后序遍历建树的方法建立左子树，同理可以建立右子树。



【例】中序后序建树




```cpp
int post[N + 10], in[N + 10];    // 中序后序建树
int build(int l1, int r1, int l2, int r2) // post[l1~r1] in[l2~r2]
{
    if(r1 < l1) return 0;         // 首先要判断是否为空树
    int x = post[r1];           // 根结点数据为后序遍历序列的最后一个元素
    int k = 0;                    // 中序遍历序列的根结点下标
    for(int i = l2; i <= r2; ++i) if(x == in[i]) k = i;
    // 利用左子树的后序和中序遍历序列建树 并设置其为根结点的左孩子
    lc[x] = build(l1, l1 + k - l2 - 1, l2, k - 1);
    // 利用右子树的后序和中序遍历序列建树 并设置其为根结点的右孩子
    rc[x] = build(l1 + k - l2, r1 - 1, k + 1, r2);
    return x;
}

rt = build(1, n, 1, n);
```

【例】中序后序建树

- 上述前序中序建树、中序后序建树的过程实际上与遍历结构类似。
- 那么如果前序中序建树是为了求后序、中序后序建树是为了求前序，可以不进行建树过程，简单修改建树程序即可输出遍历序列。
- 可能也有其他的情况，不需要建树，请读者观察实现。

练习

- 二叉树的遍历 (A0223)
- 二叉树大小 (A0224)
- 二叉树深度 (A0225)
- 表达式树 (COGS 1637)
- 二叉树前序建树 (A0226)
- 前序中序建树 (A0227)
- 中序后序建树 (A0228)
- 树加分 (A0229)
- 树 (COGS 1635)
- 前序中序求后序 (COGS 2038)
- 中序后序求前序 (COGS 2074)
- 求先序遍历 [NOIP 2001](COGS 1424)

练习

- 小球下落 (COGS 1643)
- FBI 树 [NOIP 2004](COGS 1063)
- 单词查找树 [NOI 2000](COGS 293)
- 树的层次遍历 (COGS 1644)
- 下落的树叶 (COGS 1636)
- 对称二叉树 [NOIP 2018](COGS 3052)

二叉树的计数

- 问题：具有 n 个结点的不同的二叉树有多少种?
- 设 b_n 为具有 n 个结点的不同的二叉树的种类。
- 当结点数为 0 时，是空树，$b_0 = 1$。
- 当结点数为 1 时，$b_1 = 1$。
- 当结点数为 2 时，$b_2 = 2$。
- 当结点数为 3 时，$b_3 = 5$。

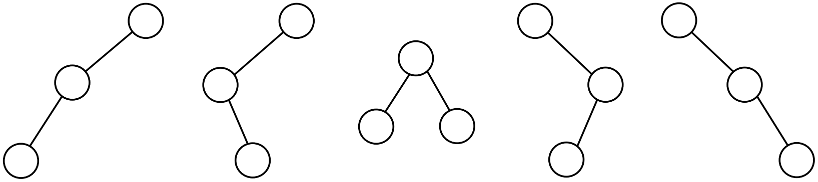
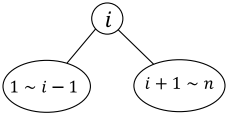


图: $n = 3$ 的情况

二叉树的计数

- 当结点数为 n 时，我们主要考虑哪个结点做根结点，此时我们用比根结点小的结点来组成左子树，用比根结点大的结点来组成右子树。
- 当用第 1 个结点做根时，只有右子树，方案数为 b_{n-1}。
- 当用第 n 个结点做根时，只有左子树，方案数为 b_{n-1}。
- 当用第 i 个结点做根时，用第 $1 \sim i-1$ 号结点做左子树，用 $i+1 \sim n$ 号结点做右子树，方案数为左子树不同形态数乘以右子树不同形态数，即 $b_{i-1} \times b_{n-i}$。



综上，我们可以得到如下公式，这种计数称为卡特兰数 (Catalan 数)。

$$b_n = \begin{cases} 1, & n = 0 \\ \sum_{i=1}^{n} b_{i-1} \times b_{n-i} = \dfrac{1}{n+1} \mathrm{C}_{2n}^{n}, & n \geq 1 \end{cases}$$