数据结构

第 14 周 – 树：基本概念、二叉树、存储表示、遍历

Koczka Ferenc 博士 · 埃斯特哈齐·卡罗伊大学 · 2026

复习

- 到目前为止都是**线性**数据结构：数组、链表、栈、队列 – 每个元素至多有**一个**后继
- **递归**（第 12 周）：把问题归结为较小的实例
- **栈**和**队列**：遍历的工具

今天的问题

- 怎样存储**层次**数据?
- 什么是二叉树？它在内存中怎样表示?
- 怎样访问树的每一个结点?

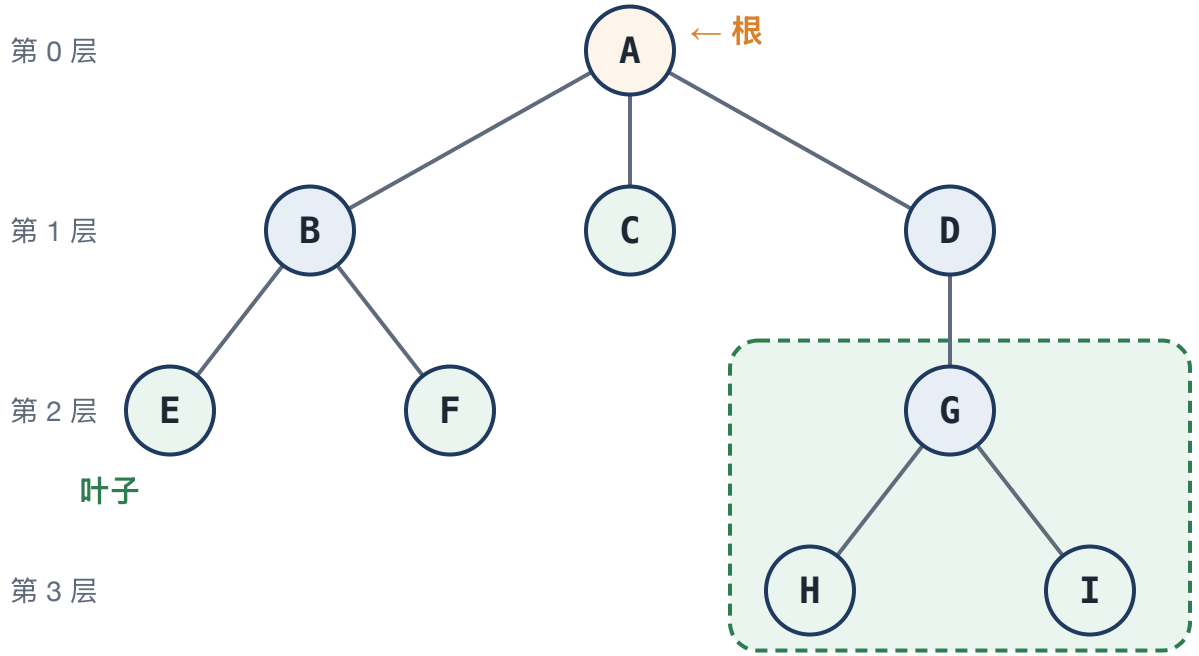
层次数据

很多数据天然是**树形**的：

- **文件系统：**文件夹、子文件夹、文件
- **组织结构图：**大学 → 学院 → 研究所 → 系
- **家谱**
- **网页**（HTML DOM）：`<html>` → `<body>` → `<div>` → ...
- **表达式：**`(3 + 4) * (5 - 2)`
- **决策：**象棋程序的走法树、机器学习中的决策树

> 树是一种**层次**数据结构：每个元素**至多有一个前驱**（父结点），但可以有**多个后继**（孩子）（第 1 周：分类）。

基本概念



B 的父结点：A · A 的孩子：B、C、D

B、C、D 是兄弟

H 的祖先：G、D、A

A 的度：3 · G 的度：2

H 的深度：3

树的高度：3

叶子：C、E、F、H、I

基本概念 – 汇总

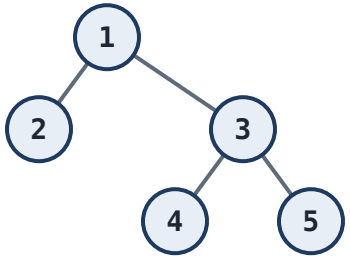
概念	含义
结点、边	元素，以及父子之间的连接
根	唯一没有父结点的结点
叶子	没有孩子的结点
父结点、孩子、兄弟	直接的连接 · 同一个父结点的孩子
祖先、子孙	通向根的路径上的结点，或者向下的结点
子树	一个结点及其所有子孙
深度 / 层	从根出发的路径长度（根为 0）
高度	最长的根–叶子路径的长度
度	孩子的个数 · **森林：**树的集合

递归定义：树或者是**空**的，或者由一个**根**及其**子树**组成。

二叉树

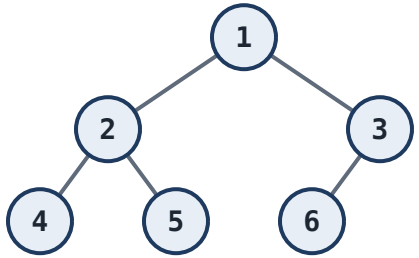
二叉树

二叉树：每个结点**至多有两个**孩子：**左**孩子和**右**孩子。



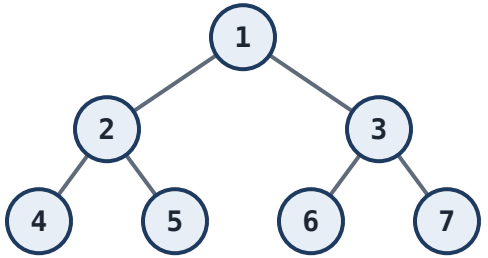
满二叉树（严格）

0 个或 2 个孩子



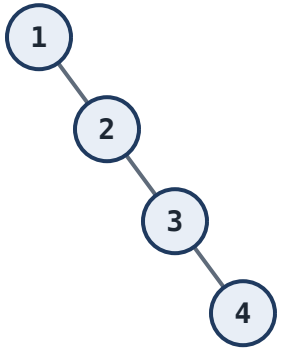
完全二叉树

逐层从左到右填满



完美二叉树

每一层都是满的



退化二叉树

实际上是链表

二叉树的性质

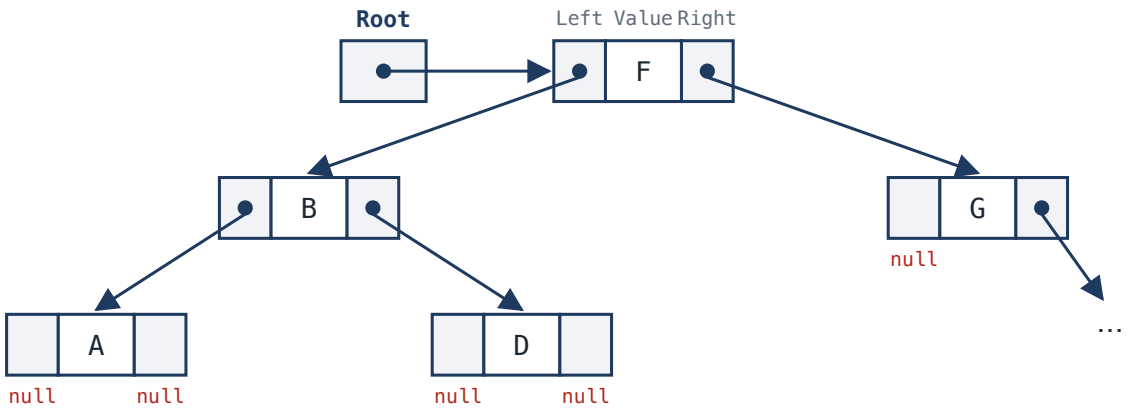
- 第 i 层至多有 2^i 个结点
- 高度为 h 的二叉树至多有 $2^{h+1} - 1$ 个结点（完美二叉树）
- 有 n 个结点的二叉树，高度至少为 $\lfloor \log_2 n \rfloor$，至多为 $n - 1$（退化）

性质	二叉树
元素	同构
关系	**层次**
大小	动态
存储	链式或顺序（数组）
访问	**顺序**（从根开始，沿指针）

> 高度决定了树的操作的**步数** – 所以树是否**平衡**很重要（第 15 周）。

二叉树的存储表示

链式表示



Java

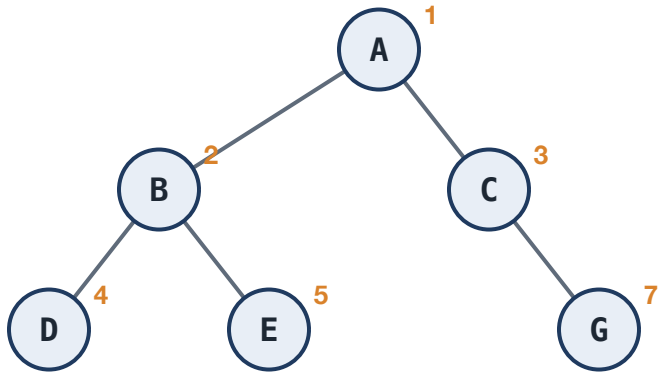
```
class Node {
    char value;
    Node left, right;
}
Node root;   // null:空树
```

Go

```
type Node struct {
    Value       rune
    Left, Right *Node
}
```

伪代码：`Elem[P].Value`、`Elem[P].Left`、`Elem[P].Right`，**Root** 指针指向根。

数组（顺序）表示



A[1..7]

A	B	C	D	E		G
1	2	3	4	5	6	7

i 的左孩子：2·i

i 的右孩子：2·i + 1

i 的父结点：i div 2

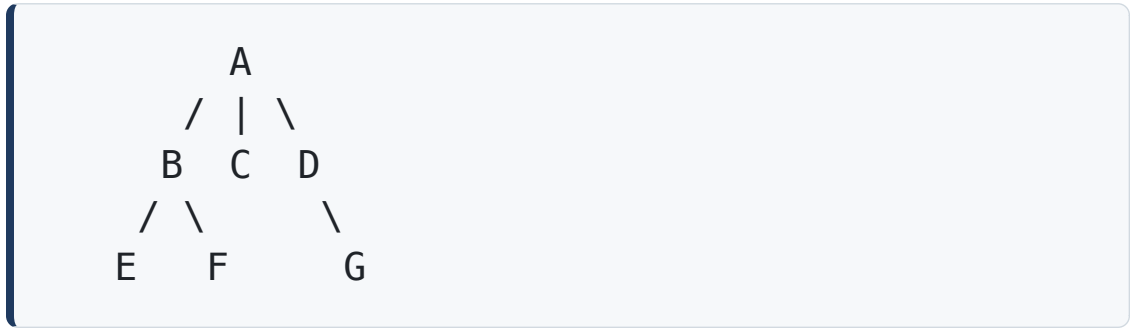
树不完全时出现空位（下标 6）

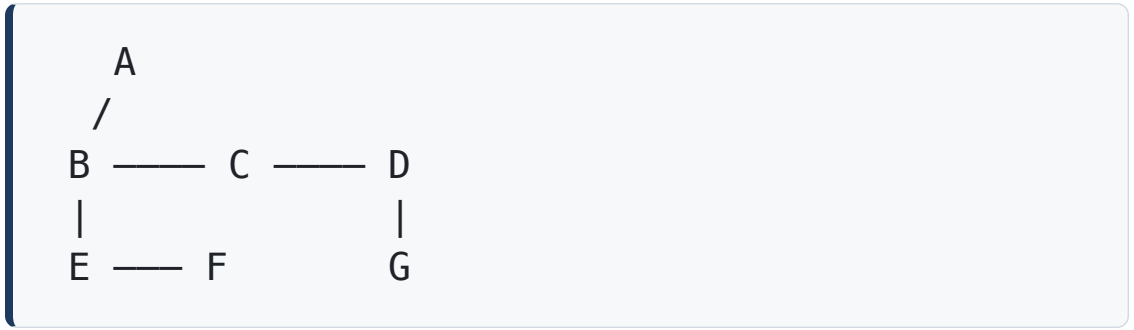
- 不需要指针：孩子和父结点的下标**可以计算**
- 对**完全**二叉树非常合适（没有空位） – **堆**就使用这种表示（第 15 周）
- 对退化的树会有大量空位：高度为 **h** 的链需要 $2^{h+1} - 1$ 个元素的数组！

用二叉树表示一般的树

如果一个结点可以有**任意**个孩子：**左孩子 – 右兄弟**表示法

- **左**指针 → **第一个孩子**
- **右**指针 → **下一个兄弟**



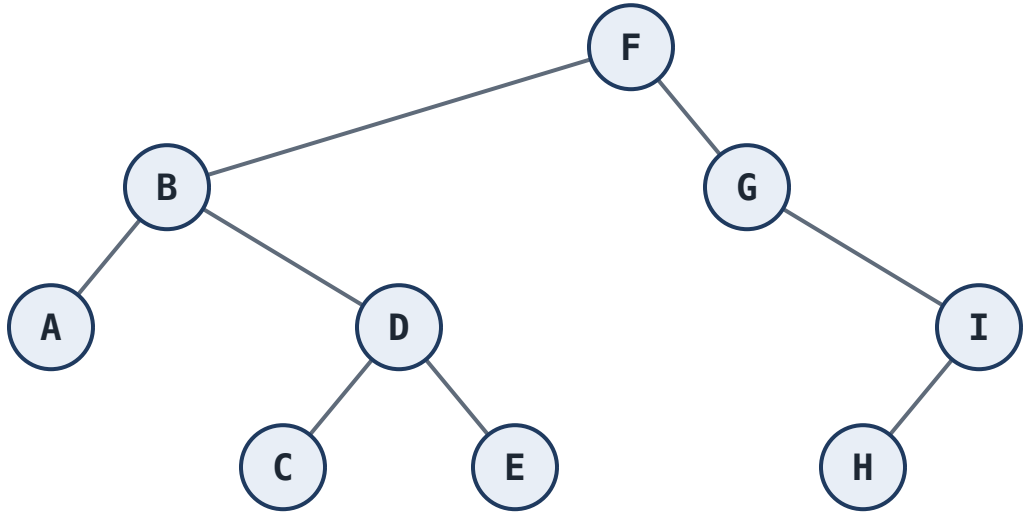


这样每一棵一般的树（以及森林）都可以用**二叉树**存储 – 例如文件系统的文件夹结构。

遍历

树的遍历

遍历：按照一定的顺序，每个结点**恰好处理一次**。



先序（根、左、右）：

F B A D C E G I H

中序（左、根、右）：

A B C D E F G H I

后序（左、右、根）：

A C E D B H I G F

层序（广度优先）：

F B G A D I C E H

深度优先遍历 – 递归实现

伪代码

```
Procedure Preorder(P)
  If P ≠ Nil then
    Process(Elem[P].Value)
    Preorder(Elem[P].Left)
    Preorder(Elem[P].Right)
  End if
End procedure

Procedure Inorder(P)
  If P ≠ Nil then
    Inorder(Elem[P].Left)
    Process(Elem[P].Value)
    Inorder(Elem[P].Right)
  End if
End procedure

Procedure Postorder(P)
  If P ≠ Nil then
    Postorder(Elem[P].Left)
    Postorder(Elem[P].Right)
    Process(Elem[P].Value)
  End if
End procedure
```

Java

```java
static void inorder(Node p) {
    if (p == null) return;
    inorder(p.left);
    System.out.print(p.value + " ");
    inorder(p.right);
}
```

Go

```go
func inorder(p *Node) {
    if p == nil {
        return
    }
    inorder(p.Left)
    fmt.Printf("%c ", p.Value)
    inorder(p.Right)
}
```

三种遍历的**唯一**区别是**何时**处理根。步数：**O(n)**，空间：O(h)。

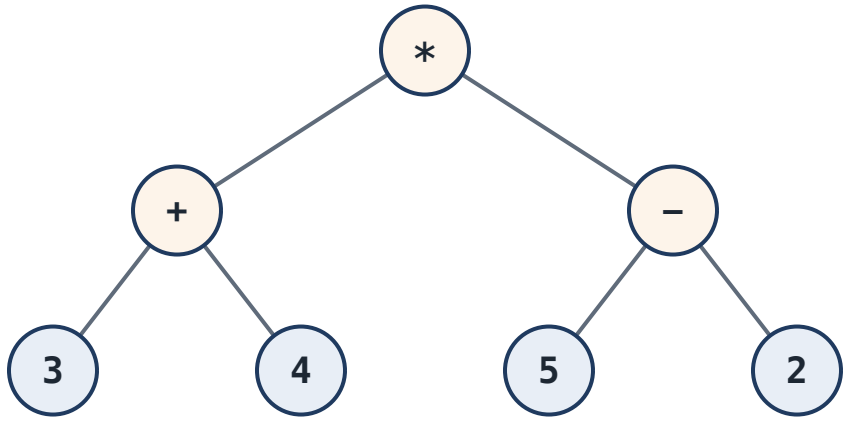
层序（广度优先）遍历

从根开始，**逐层**从左到右 – 用**队列**（第 8 周）：

```
Procedure LevelOrder(Root)
  If Root = Nil then exit
  Enqueue(Root)
  While not IsEmpty
    P ← Dequeue
    Process(Elem[P].Value)
    If Elem[P].Left ≠ Nil then Enqueue(Elem[P].Left)
    If Elem[P].Right ≠ Nil then Enqueue(Elem[P].Right)
  End while
End procedure
```

例：**F B G A D I C E H** · 如果把队列换成**栈**（并且先压入右孩子），得到的就是**先序**遍历！

应用：表达式树



中序（带括号）：

```
(3 + 4) * (5 - 2)
```

先序（前缀）：

```
* + 3 4 - 5 2
```

后序（后缀，第 7 周）：

```
3 4 + 5 2 - *
```

值：7 · 3 = 21

- 内部结点是**运算符**，叶子是**操作数**
- **后序**遍历正好给出**后缀**形式 — 可以用栈由后缀形式构造出这棵树
- **求值：** `Value(P) = Value(left) op Value(right)` — 递归地计算

树上的递归操作

模式：**“处理根，再对两棵子树调用自身”**

```
Function Count(P)
  If P = Nil then Return 0
  Return 1 + Count(Elem[P].Left)
           + Count(Elem[P].Right)
End function

Function Height(P)
  If P = Nil then Return -1
  Return 1 + max(Height(Elem[P].Left),
                 Height(Elem[P].Right))
End function
```

```
Function Leaves(P)
  If P = Nil then Return 0
  If Elem[P].Left = Nil and Elem[P].Right = Nil
    then Return 1
  Return Leaves(Elem[P].Left)
       + Leaves(Elem[P].Right)
End function
```

其他例子：求和、最大值、查找（O(n)）、镜像、复制、比较两棵树

对于示例树：9 个结点，4 片叶子（A、C、E、H），高度为 3。

小结

- **树：**层次数据结构；根、叶子、父结点、孩子、子树、深度、高度
- **二叉树：**至多两个孩子；满、完全、完美、退化
- **链式表示：**Left 和 Right 指针 · **数组表示：**孩子 2i 和 2i + 1，父结点 i div 2
- **遍历：**先序（根-左-右）、中序（左-根-右）、后序（左-右-根） – 递归，O(n)；层序 – 用队列
- 树上的操作**天然是递归的**

下周：二叉搜索树 – 查找、插入、删除；平衡；堆。

复习题

1. 判断对错：二叉树是层次数据结构。
2. 写出遍历序列：根为 1;1 的左孩子是 2，右孩子是 3;2 的孩子是 4 和 5;3 的右孩子是 6。
3. 二叉树的元素存放在数组 A 中（A[1] 是根，第 i 个元素的孩子是 A[2i] 和 A[2i + 1]）。A = [K, F, R, B, H, –, T]。画出这棵树，并写出中序遍历！
4. 高度为 4 的二叉树至多有多少个结点？
5. 哪种遍历能从表达式树得到后缀形式？
6. 只有一个结点的树，`Height(P)` 的值是多少？

实践课

✎ 示例程序

程序	Java	Go	演示内容
二叉树	`BinaryTree.java`	`w14/binarytree`	四种遍历、结点数、叶子数、高度、横向打印
数组表示的树	`ArrayTree.java`	`w14/arraytree`	下标公式、用下标遍历
表达式树	`ExpressionTree.java`	`w14/expressiontree`	由后缀形式构造、求值、中缀/前缀
用栈遍历	`IterativeTraversal.java`	`w14/iterative`	不用递归的先序和中序遍历

✎ 练习

1. **和与最大值：**计算存放整数的二叉树中元素的和与最大值！
2. **只有一个孩子的结点：**有多少个结点恰好有一个孩子？
3. **满二叉树：**判断是否每个结点都有 0 个或 2 个孩子！
4. **第 k 层：**输出第 **k** 层的元素！
5. **路径：**输出所有根–叶子路径！
6. **相同的树：**判断两棵二叉树的结构和值是否都相同！

参考答案：`Exercises14.java`、`w14/exercises`

✎ 课后作业

1. 编写函数，把链式表示的二叉树转换成**数组**表示（以及反过来）！需要多大的数组?
2. 由**先序**和**中序**遍历序列构造二叉树！（例如先序：F B A D C E G I H，中序：A B C D E F G H I）
3. **逐层、每层一行**地输出这棵树（每一层另起一行）！
4. 编写程序，以**树**的形式输出一个文件夹的内容（就像 `tree` 命令）！
   （Java: `File.listFiles()`，Go: `os.ReadDir`）