二叉搜索树

TEACHING COURSEWARE POWPOINT

授课时间：2025.08.19

目录

链式存储

引入：为什么需要链式存储？

顺序存储（数组）适合完全二叉树，但对于一般二叉树会存在空间浪费问题：

需要为缺失的结点保留位置
插入/删除操作效率低

链式存储通过指针动态连接结点，更适合表示任意形状的二叉树

每个结点最多有两个指针（左孩子和右孩子）
叶结点的左右指针均为空（NULL）
根结点是树的唯一入口点

01

概念引入

TEACHING
COURSEWARE

TEACH

概念引入

问题： 假设有一堆数字（比如：5，3，8，1，4，7，9），如何快速找到某个数字（比如4）是否存在？
简单方法： 挨个找（顺序查找
有没有一种组织数据的方式，能快速缩小查找范围。

简单回顾： 树、根节点、左子树、右子树、叶子节点的概念。
二叉树是每个节点最多有两个子节点（左孩子、右孩子）的树。

02

概念

TEACHING
COURSEWARE

TEACH

概念

二叉搜索树（BST）定义与核心规则

定义： 一棵二叉树，并且满足以下关键性质：

对于树中任意一个节点：

其左子树上所有节点的值都小于该节点的值。

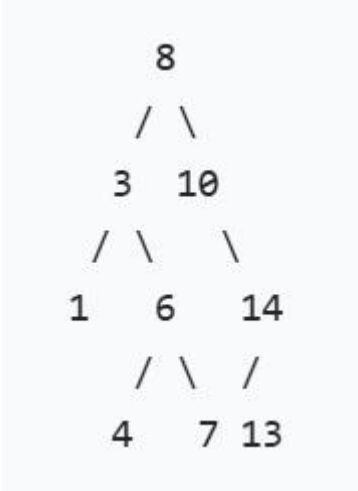
其右子树上所有节点的值都大于该节点的值。

（补充说明： 通常约定树中没有值相等的节点，如果有重复值，需要额外定义放左边还是右边。

把节点想象成一个分叉点。比它小的，统统去左边“排队”；比它大的，统统去右边“排队”。左边和右边内部，也遵循同样的规则。

核心规则：左子树 < 根 < 右子树

概念



二叉搜索树的重要性质

性质：对BST进行中序遍历（左->根->右），得到的序列是一个升序的有序序列！

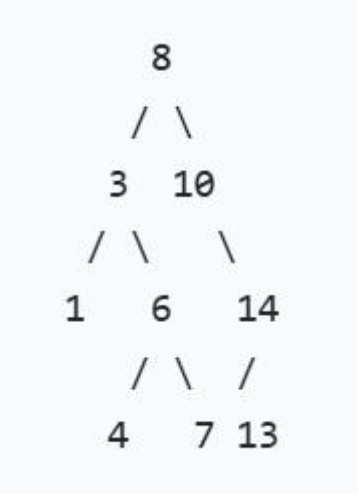
因为规则就是“左 < 根 < 右”，中序遍历正好按这个顺序访问节点。

意义： 这是二叉搜索树的核心特征，也是判断一棵树是否是二叉搜索树的关键依据。

如果一棵二叉树中序遍历不是升序，那它一定不是二叉搜索树！

概念

一、查询7的过程 假设二叉搜索树的结构如下
根节点是8
从根节点8开始
比较目标值7与8
7 < 8
因此进入左子树
在节点3处
比较7与3 7 > 3
因此进入右子树
在节点6处，比较7与6
7 > 6
在节点7处，比较7与
两者相等，查询成功，找到目标值7。



概念

插入9的
插入遵循“左小右大”规则
需找到合适的空位置
从根节点8开始
比较9与8
9 > 8
因此进入右子树
在节点10处
比较9与10
9 < 10，因此进入左子树
由于10的左子树为空，直接将9作为10的左孩子插入

03

哈夫曼树

TEACHING
COURSEWARE

TEACH

哈夫曼树

理解哈夫曼树要解决什么问题（压缩编码）。
掌握哈夫曼树的构建过程（贪心算法：反复合并最小权值的两棵树）。
理解并会计算带权路径长度（WPL）及其意义。
掌握哈夫曼编码规则（左0右1，路径即编码）及其特点（前缀码）。
理解哈夫曼编码为什么是最优的（WPL最小）。
能根据给定的字符频率/权值构建哈夫曼树或写出哈夫曼编码。

哈夫曼树

信息编码

问题： 计算机存储或传输信息（文本、图像等）都需要编码。比如，一篇文章由字母组成，每个字母可以用二进制码表示。

等长编码： 如ASCII码，每个字符用固定长度的二进制数（如8位）表示。简单但效率不高，尤其当字符出现频率差异很大时。

不等长编码： 给出现频率高的字符分配短的码字，出现频率低的字符分配长的码字。目标是让整个编码后的信息总长度尽可能短。

哈夫曼树

不等长编码
不等长编码想靠 “高频短码、低频长码” 压缩长度，但直接随意分配会出问题：
假设字符 A(高频) 编码 0，B(中频) 编码 10，C(低频) 编码 101
若收到编码串 101，到底是 B + A 还是 C

哈夫曼树的作用
哈夫曼树（最优二叉树）的核心价值，就是用树形结构构造「前缀编码」：
树的叶子节点对应字符，路径左分支记 0、右分支记 1
由于编码是从根到叶子的路径，任何字符的编码都不会是其他字符编码的前缀

04

构建过程

TEACHING
COURSEWARE

TEACH

构建过程

```
[ A(5) ]  [ B(8) ]  [ C(12) ]  [ D(20) ]  [ E(25) ]
```

步骤 1：合并最小的两个节点（A 和 B）

1. 选频率最小的节点：A（5）、B（8）
2. 合并为新节点N1(13)（5+8=13），规则：频率小的放左，大的放右（便于后续生成编码）
3. 此时所有节点：N1(13)、C(12)、D(20)、E(25)

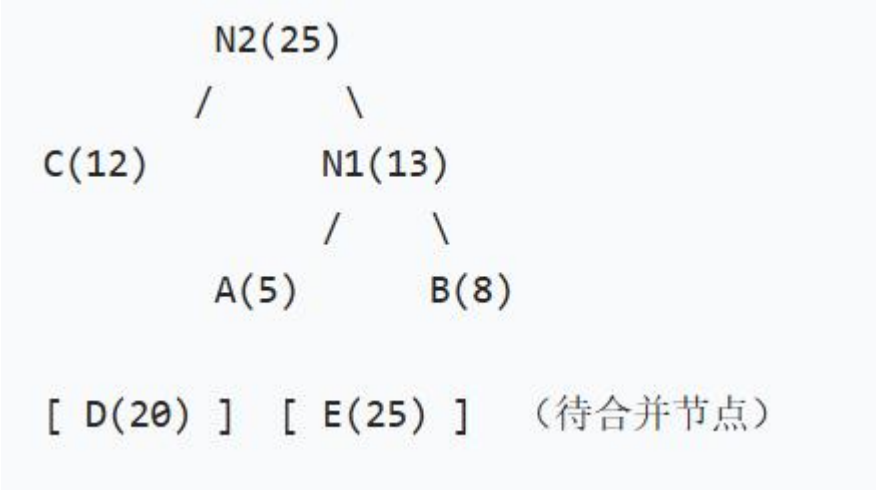
```
      N1(13)
     /      \
A(5)         B(8)

[ C(12) ]  [ D(20) ]  [ E(25) ]   （待合并节点）
```

构建过程

步骤 2：合并新的最小节点（C 和 N1）

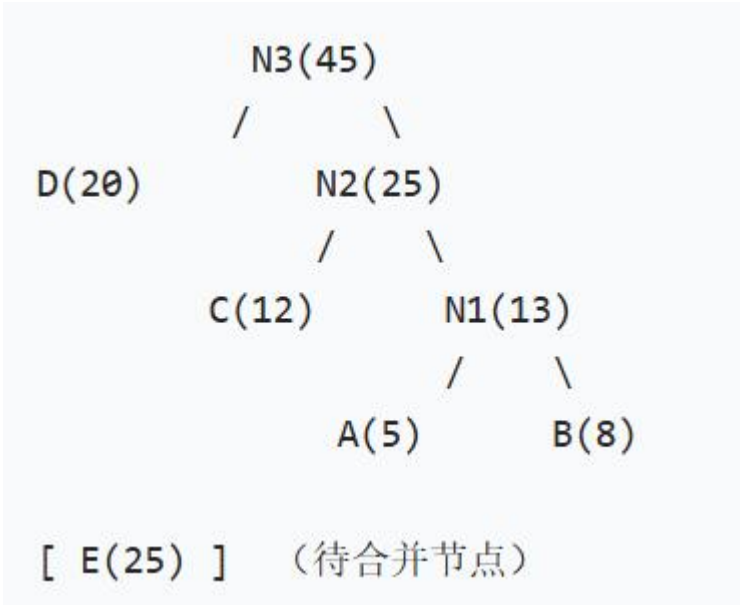
1. 选当前最小节点：C（12）、N1（13）
2. 合并为新节点N2(25)（12+13=25）
3. 此时所有节点：N2（25）、D（20）、E（25）



构建过程

步骤 3：合并新的最小节点（D 和 N2）

1. 选当前最小节点：D（20）、N2（25）
2. 合并为新节点N3(45)（20+25=45）
3. 此时所有节点：N3（45）、E（25）



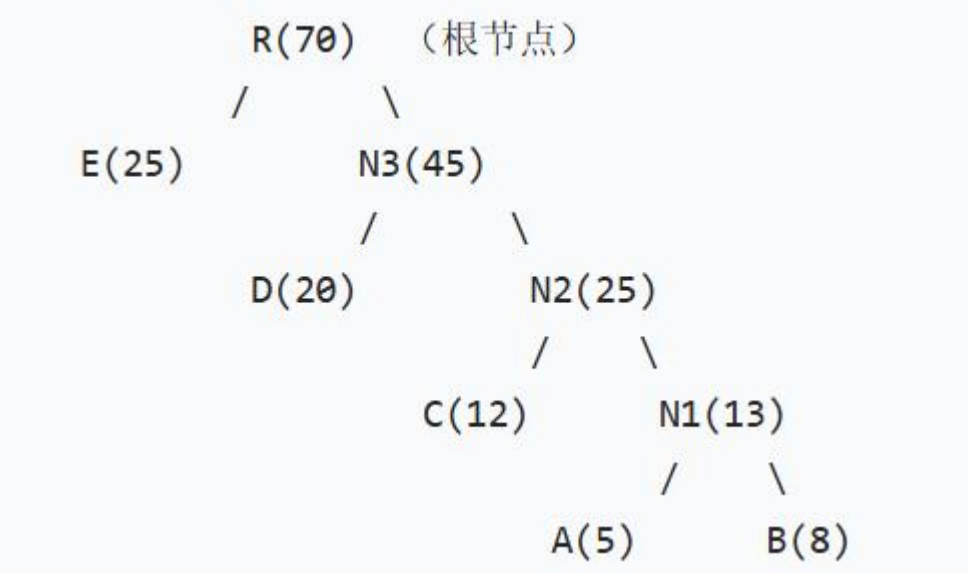
构建过程

步骤 4：合并最后两个节点（E 和 N3）

1. 选剩余两个节点：E（25）、N3（45）

2. 合并为根节点 R（70）（25+45=70，所有字符频率总和）

3. 哈夫曼树构建完成



构建过程

（左 0 右 1）

从根节点出发，向左走记 “0”、向右走记 “1”，直到叶子节点，路径即为该字符的编码（频率越高，编码越短，符合压缩逻辑）：

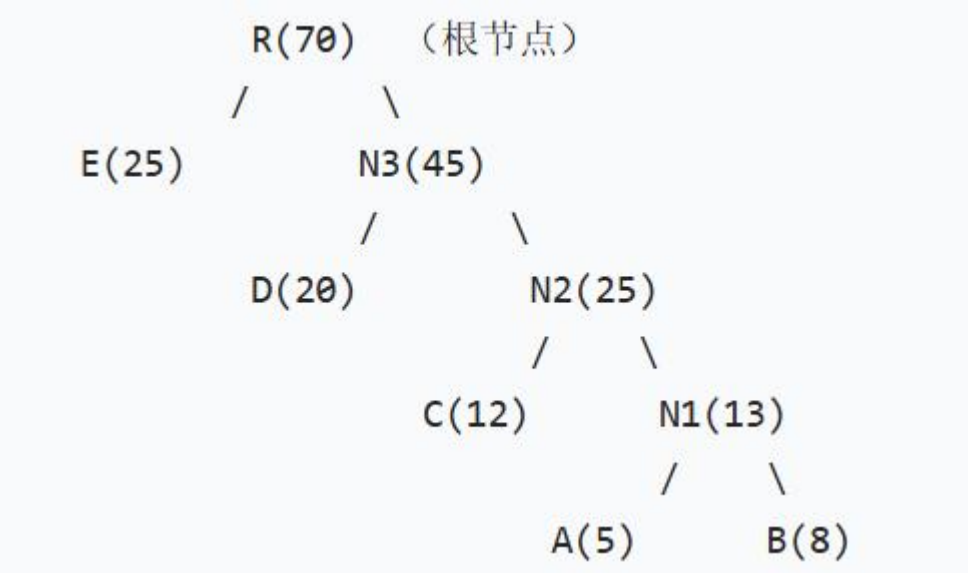
E（25）：根→左 → 0

D（20）：根→右→左 → 10

C（12）：根→右→右→左 → 110

A（5）：根→右→右→右→左 → 1110

B（8）：根→右→右→右→右 → 1111



哈夫曼树

WPL（Weighted Path Length，加权路径长度）是衡量哈夫曼树性能的核心指标

代表所有叶子节点的 “频率（权重）× 到根节点的路径长度” 之和。

路径长度：叶子节点到根节点经过的边数（如根节点到 E 的路径长度是 1，到 A 的路径长度是 4）；

权重：叶子节点的频率（本题中即 A~E 的出现次数：5、8、12、20、25）。

哈夫曼树的核心优势是：通过 “频率高的节点离根更近” 的结构，让最终 WPL 达到所有可能二叉树中的最小值，从而实现最优数据压缩。

叶子节点	频率（权重）	到根节点的路径长度
E	25	1
D	20	2
C	12	3
A	5	4
B	8	4

题目

1. A 的贡献：5 × 4 = 20
2. B 的贡献：8 × 4 = 32
3. C 的贡献：12 × 3 = 36
4. D 的贡献：20 × 2 = 40
5. E 的贡献：25 × 1 = 25

总 WPL = 20 + 32 + 36 + 40 + 25 = 153

哈夫曼树的 WPL 是 “最优解”：任何其他结构的二叉树（如按 “频率从大到小直接作为根→左→右”），WPL 都会大于 153；

与编码的关联：WPL 本质是 “所有字符的编码长度 × 频率之和”，对应压缩后的数据总长度 —— 本题中，用哈夫曼编码存储 A~E 的所有出现次数，总数据长度为 153。

题目

第 11 题

在数据压缩编码中的哈夫曼编码方法，在本质上是一种（ ）的策略。

- A. 枚举
- B. 贪心
- C. 递归
- D. 动态规划

题目

假设字母表 $\{a, b, c, d, e\}$ 在字符串出现的频率分别为 10%，15%，30%，16%，29%。若使用哈夫曼编码方式对字母进行不定长的二进制编码，字母 d 的编码长度（ ）位。

- A. 1
- B. 2
- C. 2 或 3
- D. 3

题目

假设有一组字符 `{a,b,c,d,e,f}`，对应的频率分别为 $5\%, 9\%, 12\%, 13\%, 16\%, 45\%$。请问以下哪个选项是字符abcdef分别对应的一组哈夫曼编码?

- A. `1111,1110,101,100,110,0`
- B. `1010,1001,1000,011,010,00`
- C. `000,001,010,011,10,11`
- D. `1010,1011,110,111,00,01`