DATA STRUCTURES & ALGORITHMS

哈夫曼树与哈夫曼编码

数据结构与算法（Python）- 第06讲

北京大学计算机学院　　刘云淮

HISTORY & STORY

从一个故事开始

David A. Huffman

1925 - 1999

MIT电气工程博士生，1952年发表论文《A Method for the Construction of Minimum-Redundancy Codes》

图灵奖得主评价

"一个不寻常的学生超越教授的明显案例，展示了学生可以做得比教授更好。"

— Donald Knuth

1951年的那个学期

1 期末的选择

教授Robert Fano给了学生两个选择：参加期末考试，或者写一篇学期论文。Huffman选择了后者，论文题目是：**"找到最有效的二进制编码方法"**

2 隐藏的秘密

Fano教授没有告诉学生，这是一个开放问题，他自己也在研究但未能解决。Fano的方法是"Shannon-Fano编码"，但并不总是最优的。

3 灵光乍现

Huffman花了很长时间，几乎要放弃时，突然想到了解决方案。他的方法被证明是最优的，比教授的方法更好！

4 历史影响

这篇论文被引用超过**9,570**次，成为信息理论中被引用最多的论文之一。今天几乎所有无损压缩方法都在使用哈夫曼编码。

PROBLEM BACKGROUND

问题背景：数据压缩的需求

手机存储告急

一张高清照片约**5-10MB**，1分钟视频约**100MB**。128GB手机能存多少照片？

存储空间使用 85%

网络传输成本

下载1GB流量费用约**5-10元**。如果文件能压缩50%，能省多少钱？

1GB	0.5GB
原始大小	压缩后

微信发送限制

微信文件传输限制**100MB**，邮箱附件通常限制**25-50MB**。大文件如何传输？

文件过大，无法发送

核心问题

如何在**保证信息完整**的前提下，用**最少的二进制位**表示数据？

这就是**数据压缩**要解决的核心问题。哈夫曼编码提供了一种最优的解决方案。

ENCODING SCHEMES

定长编码 vs 不定长编码

方案1：定长编码

基本思想

所有字符使用相同长度的二进制编码

示例：26个英文字母

A	→	00000
B	→	00001
C	→	00010
...	→	...
Z	→	11001

每个字符需要 **5位** 二进制

优点

编解码简单，无歧义

缺点

空间浪费，不区分频率

方案2：不定长编码

基本思想

高频字符用短编码，低频字符用长编码

示例：考虑频率

E (12.7%)	01
T (9.1%)	10
A (8.1%)	000
Z (0.1%)	111111

关键难点：前缀码问题

如果 A=0, B=01, C=10，收到"010"如何译码？

歧义！可能是 BA 或 AC

EVERYDAY EXAMPLES

生活中的哈夫曼思想

高频 = 短编码，低频 = 长编码

摩斯电码

1837年发明

电报通信中，用点(·)和划(-)表示字母

编码规则

E	(最常用)	·
T	(次常用)	-
Q	(很少用)	--·-

常用字母用1个符号，少用字母用4个符号

手机输入法

智能排序

根据使用频率，常用词排在前面

输入"da"

大	1次点击
打	2次点击
答	3次点击
达...	更多点击

常用词1次选中，少用词需要多次翻页

图书馆编号

图书分类法

中图法用字母+数字表示图书类别

编号示例

I	文学类
最热门，1位字母	
I247.5	现代长篇小说
细分，6位编码	
TP312.56	Python编程
更细分，7位编码	

大类编号短，小类编号长

DEFINITION

哈夫曼树的定义

什么是哈夫曼树?

定义

给定一组实数 $\{P_1, P_2, \ldots, P_m\}$ 作为权值，构造一棵具有m个叶子结点的二叉树，使得该树的**带权外部路径长度最小**。

带权外部路径长度（WPL）

$$WPL = \Sigma(w_i \times L_i)$$

w_i	L_i	m
第i个叶子的权值	从根到第i个叶子的路径长	外部结点（叶子）个数

核心规律：权值越大的叶子离根越近，WPL就越小

示例对比

字符集 {A, B, C, D}，权值 {2, 3, 4, 11}

树1 — WPL = 34

11×1 + 4×2 + 2×3 + 3×3

树2 — WPL = 40

2×2 + 3×2 + 4×2 + 11×2

树3 — WPL = 53

2×1 + 3×3 + 4×3 + 11×3

最优性

哈夫曼树是**带权外部路径长度最小**的二叉树，这个最小性保证了编码的最优性。

ALGORITHM

哈夫曼树的构造算法

哈夫曼算法步骤

1 初始化

根据给定的n个权值 $\{w_1, w_2, ..., w_n\}$，构成n棵二叉树的集合 $F = \{T_1, T_2, ..., T_n\}$，其中每棵二叉树$T_i$只有一个带权为$w_i$的根结点

2 选取与合并

在F中选取**两棵权值最小**的树作为左右子树构造一棵新的二叉树，新二叉树的根结点权值为左右子树根结点权值之和

3 更新集合

在F中**删除**这两棵树，同时将**新得到的二叉树加入** F

4 重复直到结束

重复步骤2和3，直到F中**只含一棵树**为止。这棵树就是哈夫曼树

构造示例

字符集 {A, B, C, D}，权值 {2, 3, 4, 11}

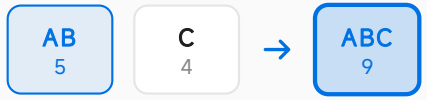
1 初始状态

4棵独立的树（只有根结点）

2 第一次合并

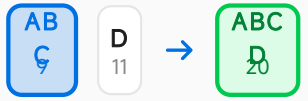
合并A(2)和B(3)，得到新树AB(5)

3 第二次合并



合并C(4)和AB(5)，得到新树ABC(9)

4 第三次合并



合并ABC(9)和D(11)，得到哈夫曼树

HUFFMAN CODING

哈夫曼编码的生成

</>编码规则

0　左分支标记为 0

从父结点指向左子女的边标记为0

1　右分支标记为 1

从父结点指向右子女的边标记为1

编码生成

从根结点到每个叶子结点的路径上的号码连接起来，就是该字符的编码

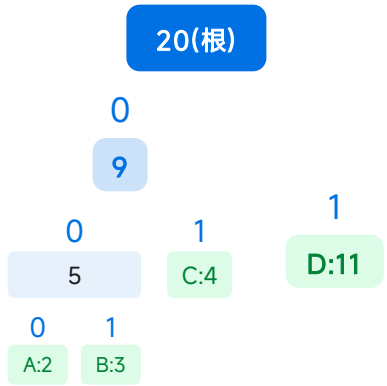
根 → 左 → 左 → 叶

0 0 1 = "001"

编码表示例

字符集 {A, B, C, D}，权值 {2, 3, 4, 11}

哈夫曼树结构



编码表

D (11, 最频繁)	1	C (4)	01
B (3)	001	A (2, 最少)	000

前缀码性质：没有任何编码是其他编码的前缀，保证无二义性

DECODING

哈夫曼编码的译码

译码过程

译码规则

从二叉树的**根结点**开始，对二进制序列的每一位进行匹配：

0 遇到 0

沿**左分支**走

1 遇到 1

沿**右分支**走

译码步骤

1. 从根结点开始，按二进制位遍历树
2. 到达叶子结点时，译出一个字符
3. 回到根结点，继续译码下一位
4. 重复直到所有位都译完

前缀码保证：由于哈夫曼编码是前缀码，译码过程**不会产生歧义**，每个编码都能被唯一确定

译码示例

编码表

A	B	C	D
000	001	01	1

译码过程演示

二进制序列

000100101101
1

000	→	A	根→左→左→左
1	→	D	根→右
001	→	B	根→左→左→右
01	→	C	根→左→右
1	→	D	根→右

译码结果

A D B C D

压缩前
13位

REAL-WORLD APPLICATIONS

哈夫曼编码的实际应用

ZIP文件压缩

ZIP使用 DEFLATE算法 ：LZ77 + 哈夫曼编码

压缩效果 30-70%

MP3音频压缩

MP3心理声学处理后，用哈夫曼编码压缩

压缩率 75-90%

JPEG图像压缩

JPEG最后一步用哈夫曼编码压缩DCT系数

压缩率 ~32%

HTTP压缩

网页传输使用gzip压缩（哈夫曼编码）

加载速度提升 60-80%

压缩效果对比

文本文件压缩

ASCII编码（定长）	8位/字符
哈夫曼编码（变长）	3.1位/字符
压缩率	61.25%

实际应用数据

512×512 Lena图像	262KB → 178KB
1000字符文本	8000位 → 3100位
1分钟MP3音频	10MB → 1MB

注意：哈夫曼编码是 无损压缩 ，解压后数据与原始数据完全一致

PYTHON IMPLEMENTATION

Python实现代码

完整代码实现

```
# 哈夫曼编码Python实现import heapq
from collections import defaultdict
# 定义树结点类classHuffmanNode:
    def__init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = Noneself.right = Nonedef__lt__(self, other):
        returnself.freq < other.freq
# 构建哈夫曼树defbuild_huffman_tree(freq_dict):
    heap = [HuffmanNode(char, freq)
            for char, freq in freq_dict.items()]
    heapq.heapify(heap)

    whilelen(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = HuffmanNode(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)
```

运行示例

```
# 测试代码
text = "this is an example of huffman encoding"
encoded, codes, root = huffman_encoding(text)
print("原始文本:", text)
print("\n哈夫曼编码表:")
for char, code insorted(codes.items()):
    print(f"  '{char}': {code}")
print(f"\n编码后: {encoded}")
print(f"原始长度: {len(text) * 8} 位")
print(f"编码长度: {len(encoded)} 位")
print(f"压缩率: {(1 - len(encoded)/(len(text)*8))*100:.1f}%")
```

SUMMARY & THINKING

思考与总结

核心思想回顾

高频短编码

出现频率高的字符用短编码，频率低的用长编码

贪心算法

每次合并权值最小的两棵树，局部最优达到全局最优

前缀码性质

所有编码都是前缀码，保证译码的唯一性和正确性

最优性

哈夫曼编码是带权外部路径长度最小的编码方案

思考题

编制一个将 **百分制转换成五分制** 的程序，怎样才能使得程序中的比较次数最少？

成绩分布如下：

0-59分	5%	60-69分	15%
70-79分	40%	80-89分	30%
90-100分	10%		

学习收获

- ✓ 理解了数据压缩的基本原理和重要性
- ✓ 掌握了哈夫曼树的构造算法和编码方法
- ✓ 了解了贪心算法在实际问题中的应用
- ✓ 认识到算法思维对解决实际问题的重要性

算法不仅是代码，更是一种**解决问题的思维方式**