Data Structures

Ch5

树和二叉树

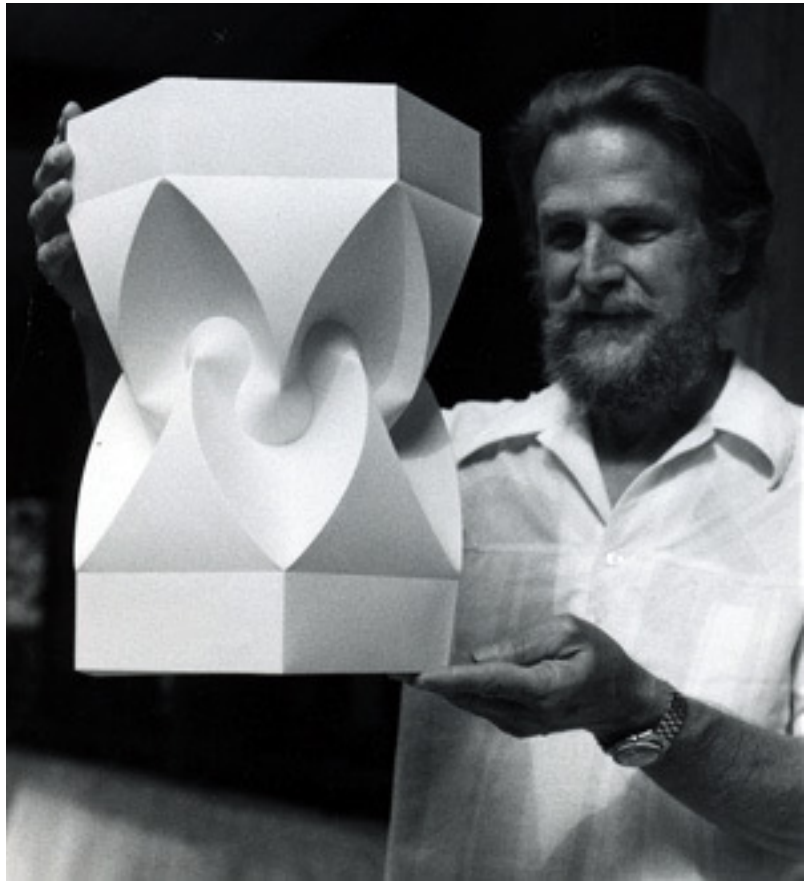
Trees & Binary Trees

2024 年 10 月 29 日

学而不厌 诲人不倦

Chapter 5　树和二叉树

Huffman Codes



1098 PROCEEDINGS OF THE I.R.E. September

A Method for the Construction of Minimum-Redundancy Codes*

DAVID A. HUFFMAN†, ASSOCIATE, IRE

Summary—An optimum method of coding an ensemble of messages consisting of a finite number of members is developed. A minimum-redundancy code is one constructed in such a way that the average number of coding digits per message is minimized.

Introduction

ONE IMPORTANT METHOD of transmitting messages is to transmit in their place sequences of symbols. If there are more messages which might be sent than there are kinds of symbols available, then some of the messages must use more than one symbol. If it is assumed that each symbol requires the same time for transmission, then the time for transmission (length) of a message is directly proportional to the number of symbols associated with it. In this paper, the symbol or sequence of symbols associated with a given message will be called the "message code." The entire number of messages which might be transmitted will be called the "message ensemble." The mutual agreement between the transmitter and the receiver about the meaning of the code for each message of the ensemble will be called the "ensemble code."

Probably the most familiar ensemble code was stated in the phrase "one if by land and two if by sea." In this case, the message ensemble consisted of the two individual messages "by land" and "by sea", and the message codes were "one" and "two."

In order to formalize the requirements of an ensemble code, the coding symbols will be represented by numbers. Thus, if there are D different types of symbols to be used in coding, they will be represented by the digits $0, 1, 2, \cdots, (D-1)$. For example, a ternary code will be constructed using the three digits 0, 1, and 2 as coding symbols.

The number of messages in the ensemble will be called N. Let $P(i)$ be the probability of the ith message. Then

$$\sum_{i=1}^{N} P(i) = 1. \tag{1}$$

The length of a message, $L(i)$, is the number of coding digits assigned to it. Therefore, the average message length is

$$L_{av} = \sum_{i=1}^{N} P(i)L(i). \tag{2}$$

The term "redundancy" has been defined by Shannon[1] as a property of codes. A "minimum-redundancy code" will be defined here as an ensemble code which, for a message ensemble consisting of a finite number of members, N, and for a given number of coding digits, D, yields the lowest possible average message length. In order to avoid the use of the lengthy term "minimum-redundancy," this term will be replaced here by "optimum." It will be understood then that, in this paper, "optimum code" means "minimum-redundancy code."

The following basic restrictions will be imposed on an ensemble code:

(a) No two messages will consist of identical arrangements of coding digits.

(b) The message codes will be constructed in such a way that no additional indication is necessary to specify where a message code begins and ends once the starting point of a sequence of messages is known.

Restriction (b) necessitates that no message be coded in such a way that its code appears, digit for digit, as the first part of any message code of greater length. Thus, 01, 102, 111, and 202 are valid message codes for an ensemble of four members. For instance, a sequence of these messages 11110220201011111102 can be broken up into the individual messages 111-102-202-01-01-111-102. All the receiver need know is the ensemble code. However, if the ensemble has individual message codes including 11, 111, 102, and 02, then when a message sequence starts with the digits 11, it is not immediately certain whether the message 11 has been received or whether it is only the first two digits of the message 111. Moreover, even if the sequence turns out to be 11102, it is still not certain whether 111-02 or 11-102 was transmitted. In this example, change of one of the two message codes 111 or 11 is indicated.

C. E. Shannon[1] and R. M. Fano[2] have developed ensemble coding procedures for the purpose of proving that the average number of binary digits required per message approaches from above the average amount of information per message. Their coding procedures are not optimum, but approach the optimum behavior when N approaches infinity. Some work has been done by Kraft[3] toward deriving a coding method which gives an average code length as close as possible to the ideal when the ensemble contains a finite number of members. However, up to the present time, no definite procedure has been suggested for the construction of such a code

* Decimal classification: R531.1. Original manuscript received by the Institute, December 6, 1951.

† Massachusetts Institute of Technology, Cambridge, Mass.

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Sys. Tech. Jour.*, vol. 27, pp. 398–403; July, 1948.

[2] R. M. Fano, "The Transmission of Information," Technical Report No. 65, Research Laboratory of Electronics, M.I.T., Cambridge, Mass.; 1949.

[3] L. G. Kraft, "A Device for Quantizing, Grouping, and Coding Amplitude-modulated Pulses," Electrical Engineering Thesis, M.I.T., Cambridge, Mass.; 1949.

Figure 1. David Huffman in 1978 and in 1999 (both photos courtesy of University of California, Santa Cruz)

https://www.maa.org/press/periodicals/convergence/discovery-of-huffman-codes

5.7 最优二叉树

5-7-1　Huffman算法

1. 最优二叉树

- 叶子结点的权值：对叶子结点赋予的一个有意义的数值量

⇩

取决于具体问题

- 二叉树的**带权路径长度（WPL）**：从根结点到各个叶子结点的路径长度与相应叶子结点权值的乘积之和

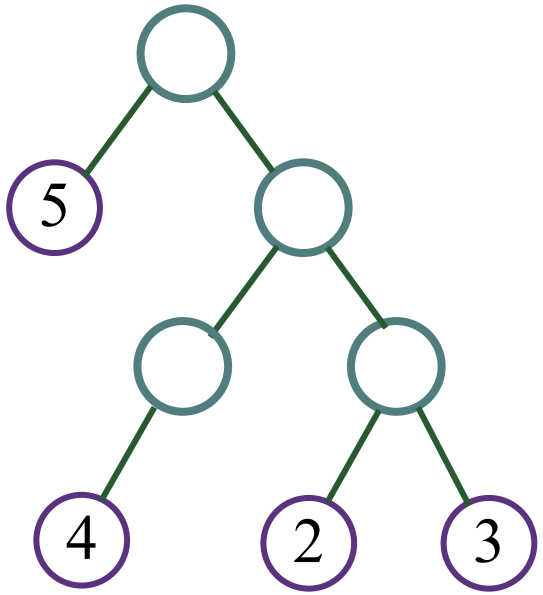
$$\sum_{k=1}^{n} w_k l_k$$

w_k：第k个叶子的权值

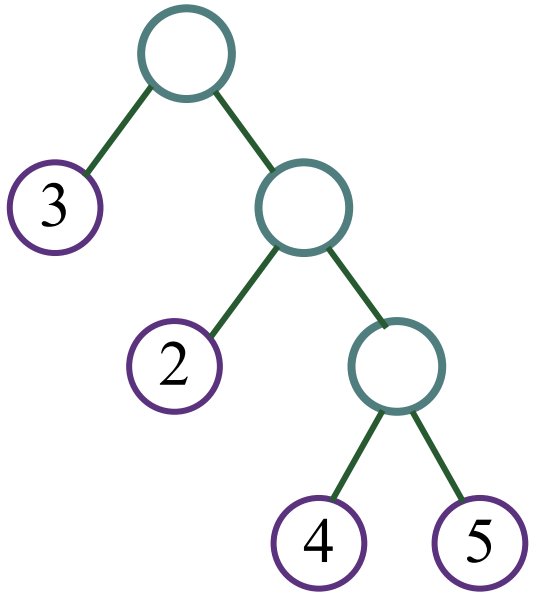
l_k：从根结点到第k个叶子的路径长度

1. 最优二叉树

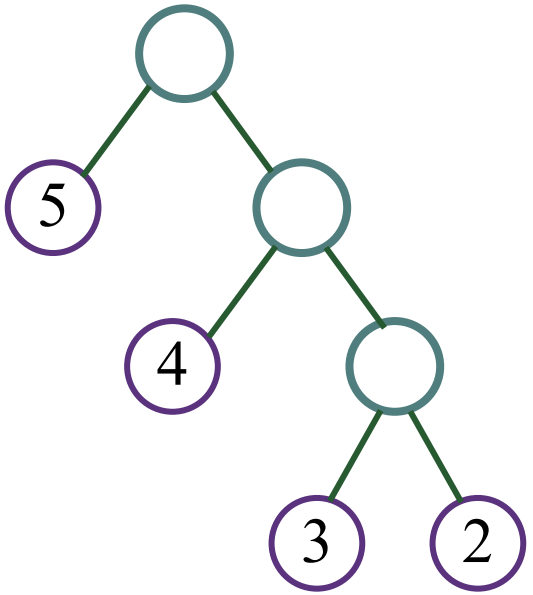
- 最优二叉树（Huffman树）：给定一组具有确定权值的叶子结点，带权路径长度最小的二叉树



32



34

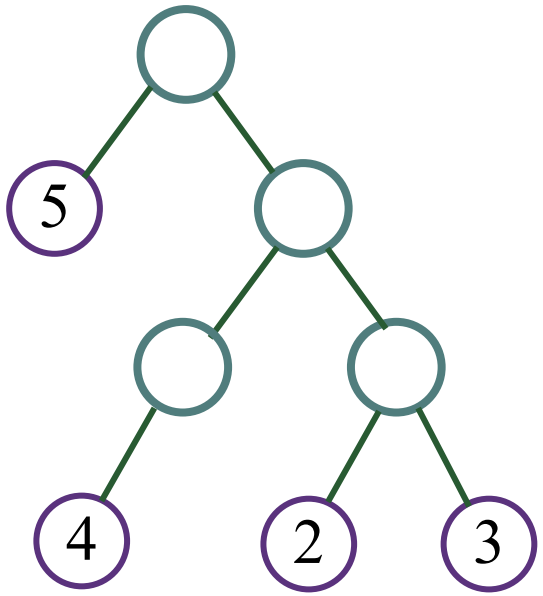


28

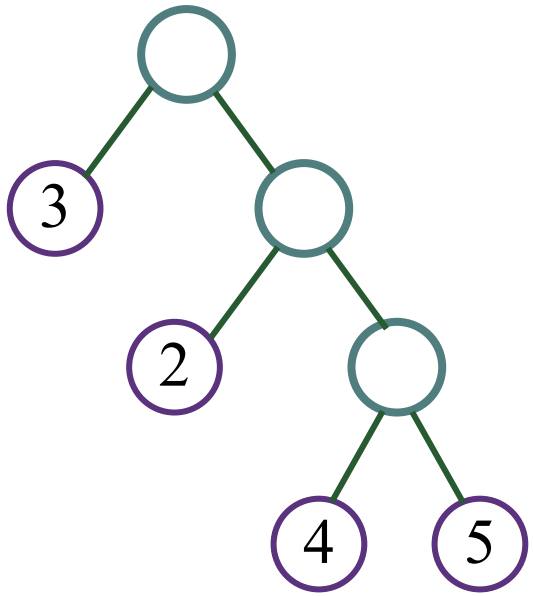
2. 最优二叉树的特点

(1) 权值越大的叶子结点越靠近根结点

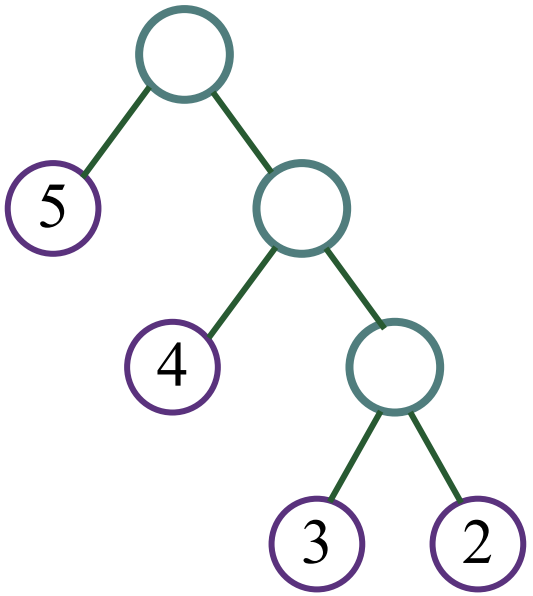
(2) 只有度为 0 和度为 2 的结点，不存在度为 1 的结点



32



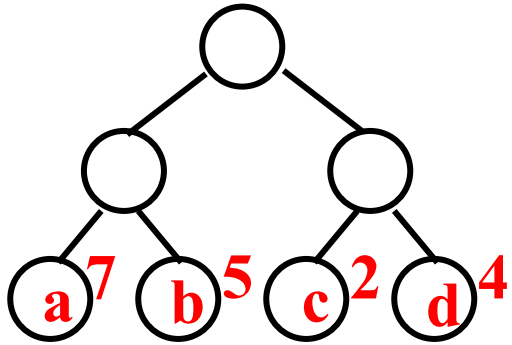
34



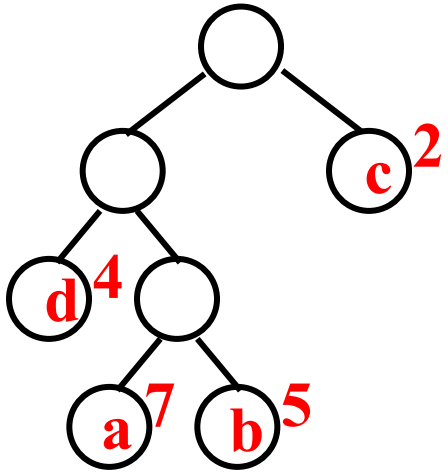
28

2. 最优二叉树的特点

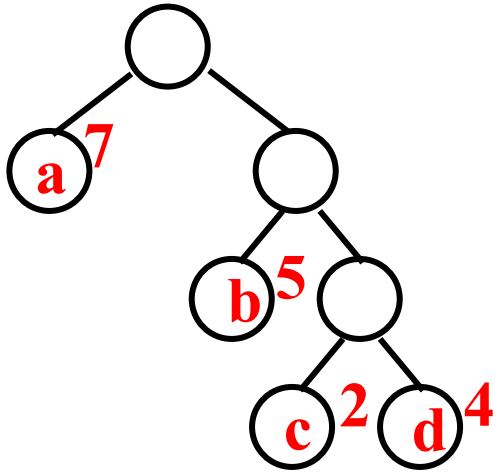
例，3 棵二叉树，都有 4 个叶子结点 a、b、c、d，分别带权7、5、2、4，求它们各自的带权路径长度。



(1)



(2)



(3)

(1) WPL = 7×2 + 5×2 + 2×2 + 4×2 = 36

(2) WPL = 7×3 + 5×3 + 2×1 + 4×2 = 46

(3) WPL = 7×1 + 5×2 + 2×3 + 4×3 = 35

3. Huffman算法

【问题】给定一组权值，构造最优二叉树

【想法——Huffman算法的基本思想】

1. 初始化：由 n 个权值构造 n 棵只有一个根结点的二叉树，得到一个二叉树集合 $F=\{T_1, T_2, \ldots, T_n\}$；
2. 重复下述操作，直到集合 F 中只剩下一棵二叉树
   - 2.1 选取与合并：在 F 中选取根结点的权值最小的两棵二叉树分别作为左右子树构造一棵新的二叉树，这棵新二叉树的根结点的权值为其左右子树根结点的权值之和；
   - 2.2 删除与加入：在 F 中删除作为左右子树的两棵二叉树，并将新建立的二叉树加入到F中；

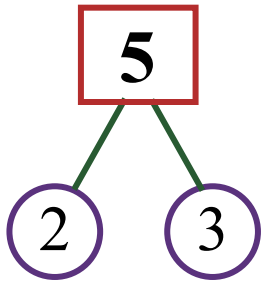
3. Huffman算法

Huffman算法的运行实例

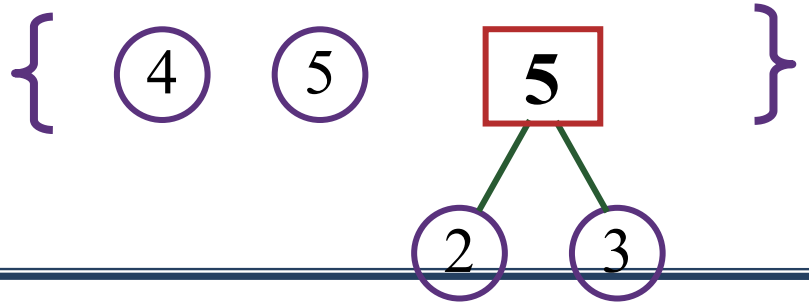
例 给定权值集合{2，4，5，3}

初始化

选取与合并



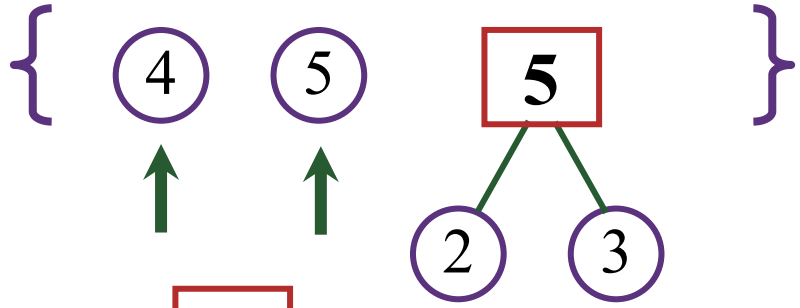
删除与加入



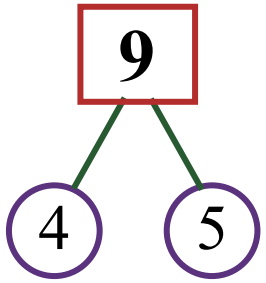
3. Huffman算法

Huffman算法的运行实例

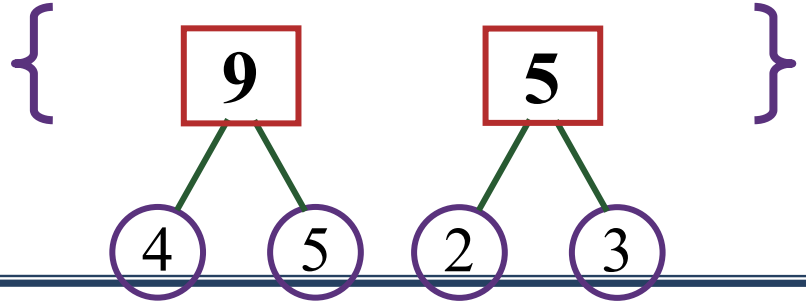
例 给定权值集合{2，4，5，3}

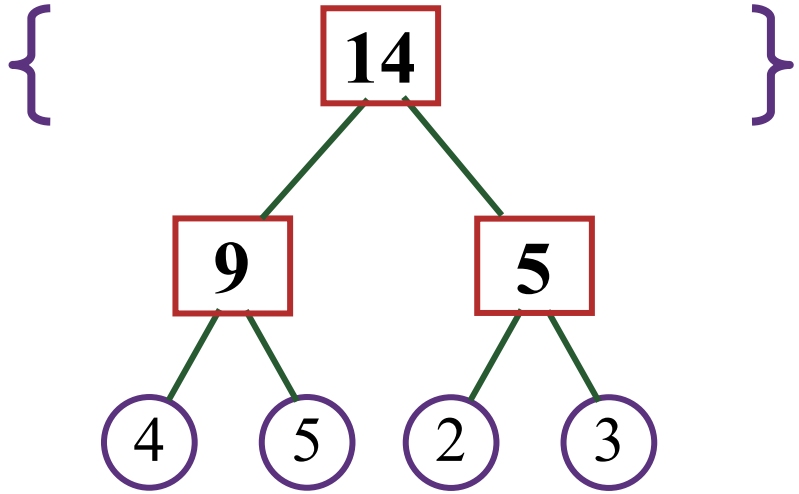


选取与合并



删除与加入





3. Huffman算法

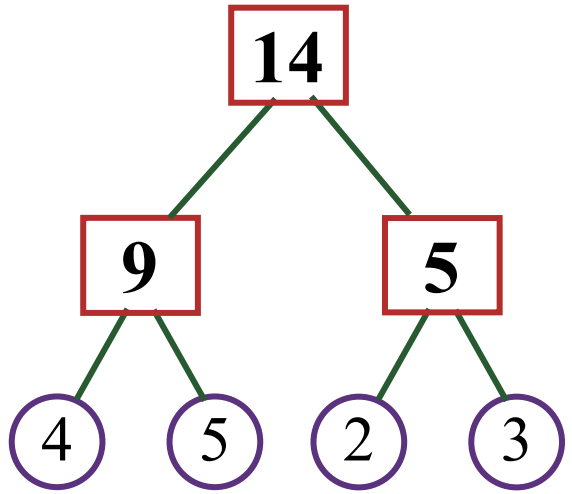
【算法——数据表示】如何存储Huffman树呢?

采用数组还是链表呢?

weight	lchild	rchild	parent

n 个叶子结点 ⇨ 合并 $n-1$ 次 ⇨ $n-1$个分支结点

哈夫曼树共有 $2n-1$个结点 ⇨ 设数组huffTree[2n−1]



须存储哪些关系呢?

选取根结点权值最小的二叉树 ⇨ 存储parent信息

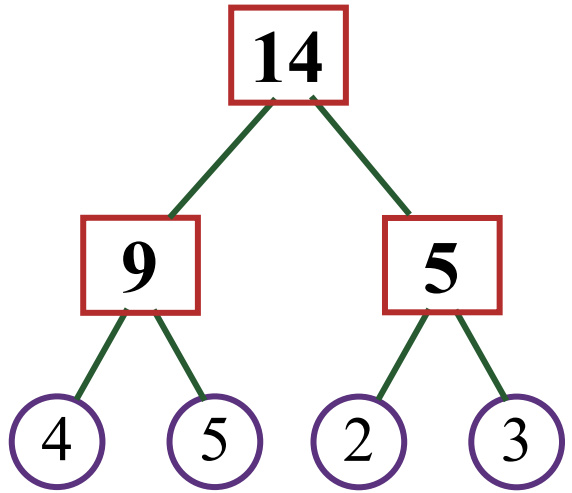
作为左右子树进行合并 ⇨ 存储lchild和rchild信息

3. Huffman算法

【算法——数据表示】如何存储Huffman树呢?

weight	lchild	rchild	parent

```
struct ElemType
{
    int weight;      //假定权值为整数
    int parent, lchild, rchild;    //游标
};
```



函数原型：void HuffmanTree(ElemType huffTree[ ], int w[ ], int n)

3. Huffman算法

【算法——数据处理过程】

例 给定权值集合{2，4，5，3}

{ ② ④ ⑤ ③ }

算法描述：

```
for (i = 0; i < 2*n−1; i++)
{
    huffTree[i].parent = −1;
    huffTree[i].lchild = −1;
    huffTree[i].rchild = −1;
}
```

	weight	parent	lchild	rchild
0		−1	−1	−1
1		−1	−1	−1
2		−1	−1	−1
3		−1	−1	−1
4		−1	−1	−1
5		−1	−1	−1
6		−1	−1	−1

3. Huffman算法

【算法——数据处理过程】

例 给定权值集合{2，4，5，3}

{ ② ④ ⑤ ③ }

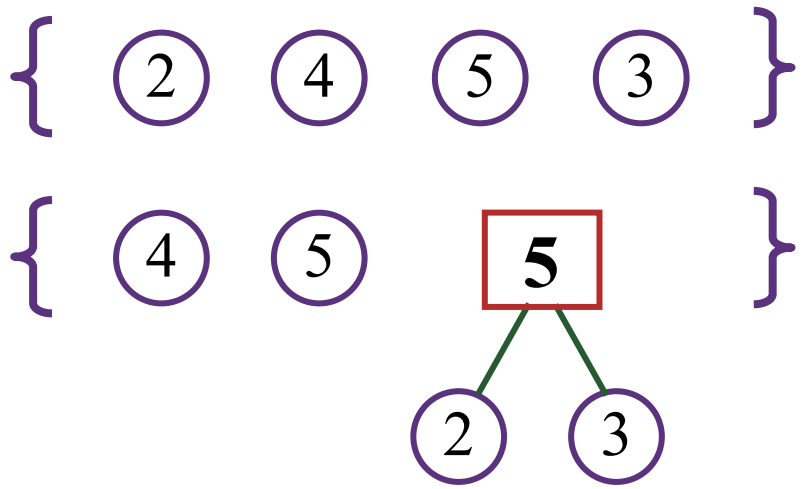
算法描述：

```
for (i = 0; i < n; i++)
    huffTree[i].weight = w[i];
```

	weight	parent	lchild	rchild
0	2	−1	−1	−1
1	4	−1	−1	−1
2	5	−1	−1	−1
3	3	−1	−1	−1
4		−1	−1	−1
5		−1	−1	−1
6		−1	−1	−1

3. Huffman算法

例 给定权值集合{2，4，5，3}



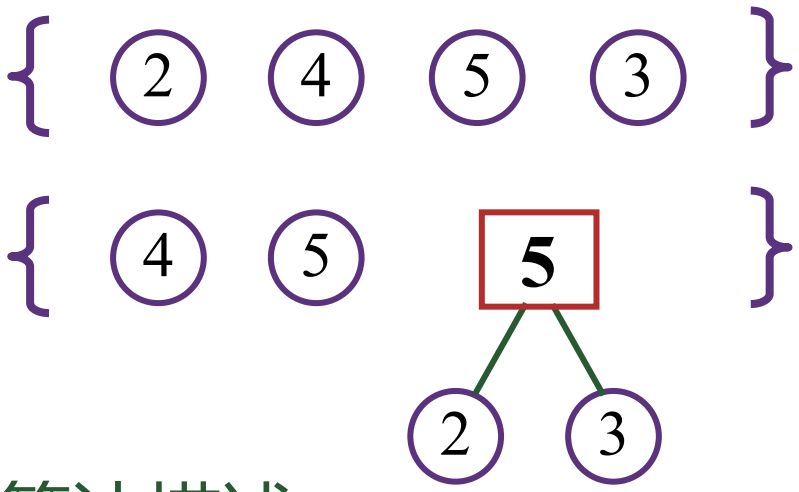
	weight	parent	lchild	rchild
i1 → 0	2	−1	−1	−1
1	4	−1	−1	−1
2	5	−1	−1	−1
i2 → 3	3	−1	−1	−1
k → 4	**5**	−1	−1	−1
5		−1	−1	−1
6		−1	−1	−1

算法描述：

```
huffTree[k].weight = huffTree[i1].weight + huffTree[i2].weight;
```

3. Huffman算法

例 给定权值集合{2，4，5，3}



	weight	parent	lchild	rchild
0	2	4 ~~-1~~	-1	-1
1	4	-1	-1	-1
2	5	-1	-1	-1
3	3	4 ~~-1~~	-1	-1
4	5	-1	0 ~~-1~~	3 ~~-1~~
5		-1	-1	-1
6		-1	-1	-1

i1 → 0

i2 → 3

k → 4

算法描述：

```
huffTree[i1].parent = k;  huffTree[i2].parent = k;
huffTree[k].lchild = i1;  huffTree[k].rchild = i2;
```

3. Huffman算法

【程序】

```
void HuffmanTree(ElemType huffTree[ ], int w[ ], int n )
{
    int i, k, i1, i2;
    for (i = 0; i < 2*n-1; i++)          /*初始化，所有结点均没有双亲和孩子*/
    {
        huffTree[i].parent = -1;   huffTree[i].lchild = -1;   huffTree[i].rchild = -1;
    }
    for (i = 0; i < n; i++)              /*构造n棵只含有根结点的二叉树*/
        huffTree[i].weight = w[i];
    for (k = n; k < 2*n-1; k++)          /*n-1次合并*/
    {
        Select(huffTree, i1, i2);        /*权值最小的两个根结点下标为i1和i2*/
        huffTree[k].weight = huffTree[i1].weight + huffTree[i2].weight;
        huffTree[i1].parent = k;  huffTree[i2].parent = k;
        huffTree[k].lchild = i1;  huffTree[k].rchild = i2;
    }
}
```

3. Huffman算法

	weight	parent	lchild	rchild
0	2	4 ~~-1~~	-1	-1
1	4	-1	-1	-1
2	5	-1	-1	-1
3	3	4 ~~-1~~	-1	-1
4	5	-1	0 ~~-1~~	3 ~~-1~~
5		-1	-1	-1
6		-1	-1	-1

i1 → 0

i2 → 3

k → 4

```
void HuffmanTree :: Select(int n, int &i1, int &i2)
{
    int i = 0, temp;
    for ( ; i < n; i++)
        if (huffTree[i].parent == -1) {i1 = i; break;}
    for (i = i + 1; i < n; i++)
        if (huffTree[i].parent == -1) {i2 = i;break;}
    if (huffTree[i1].weight > huffTree[i2].weight)
    {
        temp = i1; i1 = i2;i2 = temp;
    }
    for (i = i + 1; i < n; i++)
    {
        if (huffTree[i].parent == -1)
        {
            if (huffTree[i].weight < huffTree[i1].weight)
            {
                i2 = i1; i1 = i;
            }
            else if (huffTree[i].weight < huffTree[i2].weight)
            {
                i2 = i;
            }
        }
    }
}
```

```
Select(k, i1, i2);          //权值最小的根结点下标为i1和i2
```

3. Huffman算法

```
struct ElemType
{
  int weight;     //假定权值为整数
  int parent, lchild, rchild;    //游标
};

class HuffmanTree
{
public:
    HuffmanTree(int w[ ], int n);
    HuffmanTree( );
    void Print( );
private:
    ElemType *huffTree;
    int num;
    void Select(int n, int &i1, int &i2);
};
```

```
HuffmanTree :: HuffmanTree(int w[ ], int n)
{
    int i, k, i1, i2;
    huffTree = new ElemType [2*n-1];
    num = n;
    for (i = 0; i < 2*num-1; i++)          //初始化，所有结点均没有双亲和孩子
    {
        huffTree[i].parent = -1;
        huffTree[i].lchild = huffTree[i].rchild = -1;
    }
    for (i = 0; i < num; i++)              //存储叶子结点的权值
        huffTree[i].weight = w[i];
    for (k = num; k < 2*num-1; k++)         //n-1次合并
    {
        Select(k, i1, i2);         //权值最小的根结点下标为i1和i2
        huffTree[k].weight = huffTree[i1].weight + huffTree[i2].weight;
        huffTree[i1].parent = k; huffTree[i2].parent = k;
        huffTree[k].lchild = i1; huffTree[k].rchild = i2;
    }
}
```

3. Huffman算法

```
void HuffmanTree :: Print( )
{
    int i, k;
    cout << "每个叶子到根结点的路径是:" << endl;
    for (i = 0; i < num; i++)
    {
        cout << huffTree[i].weight;
        k = huffTree[i].parent;
        while (k != -1)
        {
            cout << "-->" << huffTree[k].weight;
            k = huffTree[k].parent;
        }
        cout << endl;
    }
}
```

```
int main( )
{
    int w[] = {2,4, 5, 3};
    HuffmanTree T(w, 4);
    T.Print();
    return 0;
}
```

```
每个叶子到根结点的路径是:
2-->5-->14
4-->9-->14
5-->9-->14
3-->5-->14
```

5.7 最优二叉树

5-7-2　Huffman编码

1. 编码 Coding

- 编码：给每一个对象标记一个二进制位串来表示一组对象
- 等长编码：用长度相等的二进制位串表示一组对象

酸	甜
0	1

酸	甜	苦	辣
00	01	10	11

酸	甜	苦	辣	咸	麻
000	001	010	011	100	101

	省	市	区	出生日期	顺序	校验
身份证号码	22	01	04	19870126	444	7

- 编码的目的是什么？⇨ 数字化 ⇨ 编码效率取决于编码长度
- 如果每个对象的使用频率不等，如何获得较高的编码效率？

2. 解码 Decoding

不等长编码：表示一组对象的二进制位串的长度不相等

设计不等长编码时，必须考虑解码的唯一性

一组对象	A	B	C	D	E
使用频率	35	25	15	15	10
不等长编码	0	1	01	10	11

AABACD 00100110

0 0 100110

0 01 00110

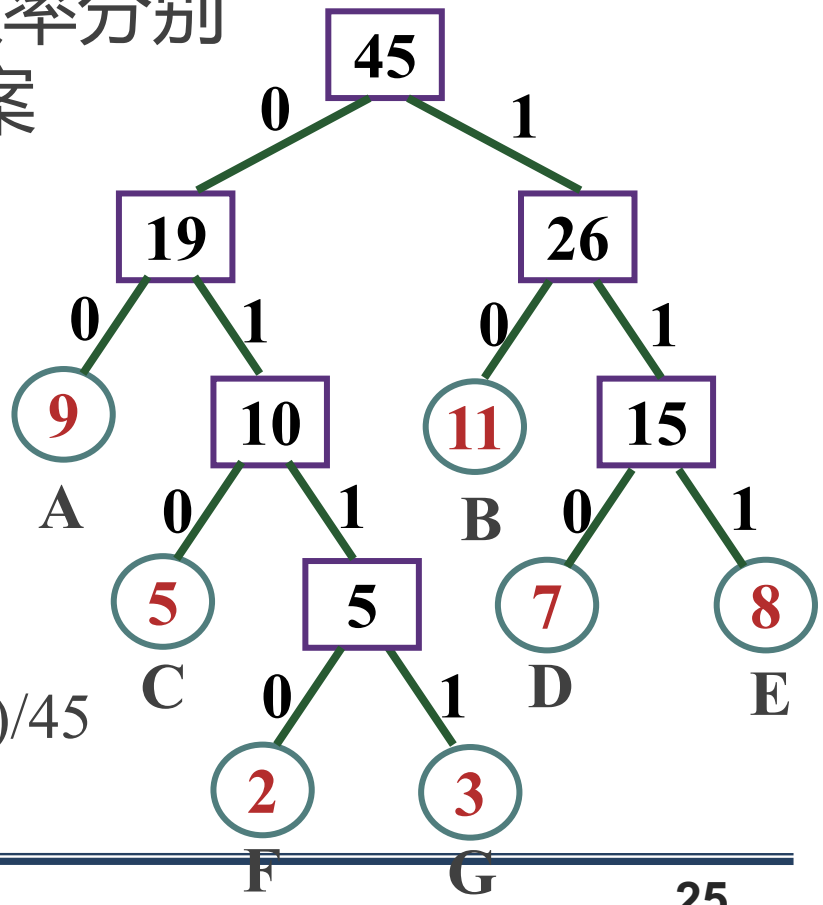
前缀编码：在一组编码中，任一编码都不是其它任何编码的前缀

前缀（无歧义）编码保证了在解码时不会有多种可能

3. Huffman编码

如何设计一个编码效率最高的前缀编码呢？⇨ Huffman编码

例：一组字符{A, B, C, D, E, F, G}出现的频率分别是{9, 11, 5, 7, 8, 2, 3}，设计最经济的编码方案



哈夫曼编码方案：

A：00
B：10
C：010
D：110
E：111
F：0110
G：0111

等长编码的长度：3

哈夫曼编码的平均长度：

$(2\times9+3\times5+4\times2+4\times3+2\times11+3\times7+3\times8)/45$

$=2.67$

3. Huffman编码

对“BACBAD”进行编码： 100001010001l0

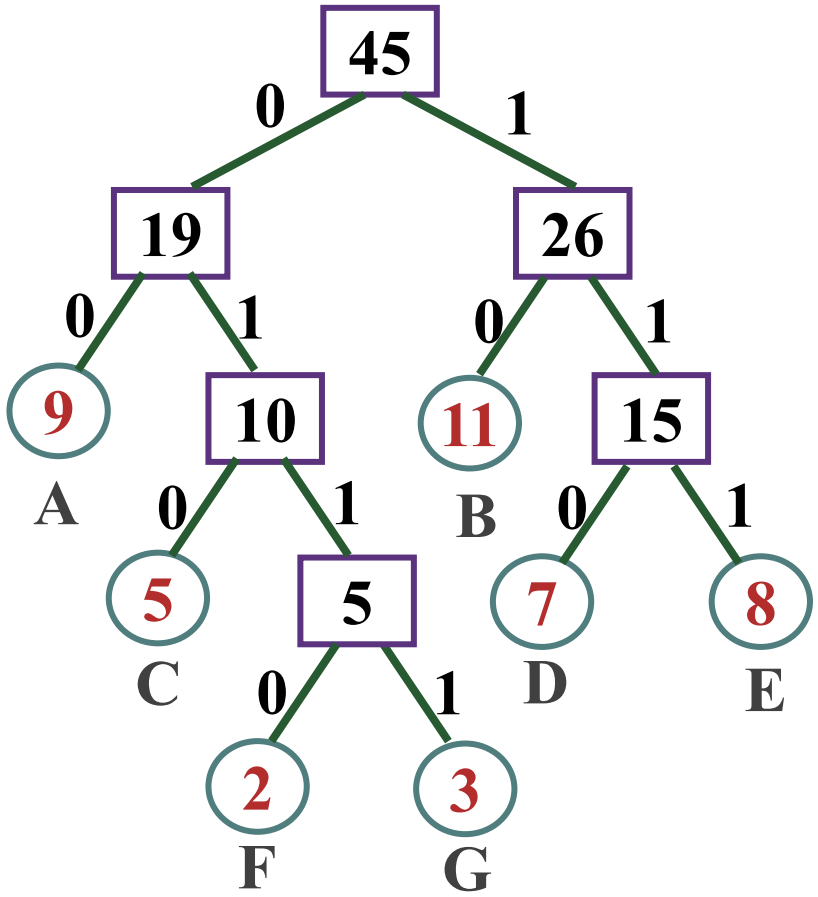
对“100001010001l0”进行解码：

10B-00A-010C-10B-00A-110D

哈夫曼编码方案：

A：00
B：10
C：010
D：110
E：111
F：0110
G：0111

如何进行*m*进制不等长编码?

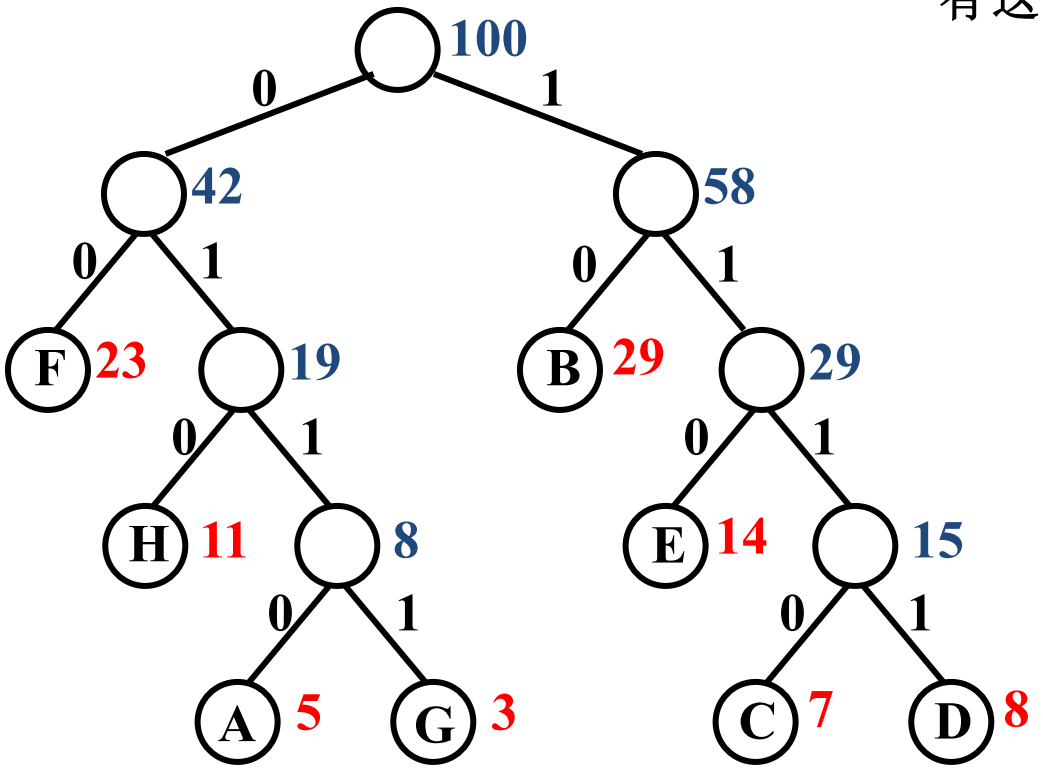


3. Huffman编码

例，某通信可能出现 A B C D E F G H 8 个字符，其概率分别为 0.05 , 0.29 , 0.07 , 0.08 , 0.14 , 0.23 , 0.03 , 0.11 ， 试设计霍夫曼编码

不妨设 w = { 5 , 29 , 7 , 8 , 14 , 23 , 3 , 11 }

排序后 w = { 100 }



计算按哈夫曼编码压缩存储有这段文字，共需多少字节？

A (0110)
B (10)
C (1110)
D (1111)
E (110)
F (00)
G (0111)
H (010)

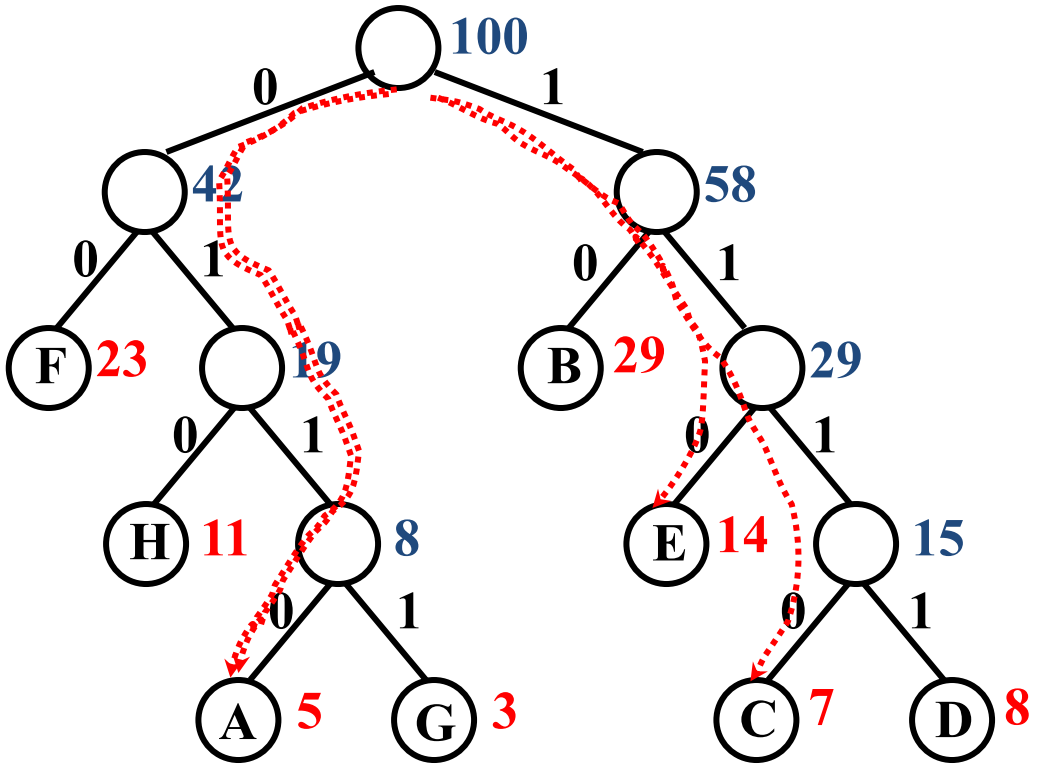
3. Huffman编码

如何译码?

1. 从根结点出发，从左至右扫描编码，

2. 若为 ‘0’ 则走左分支，若为 ‘1’ 则走右分支，直至叶结点为止，

3. 取叶结点字符为译码结果，返回重复执行 1,2,3 直至全部译完为止



A (0110)
B (10)
C (1110)
D (1111)
E (110)
F (00)
G (0111)
H (010)

ACEA 编码为 0110 1110 110 0110

A C E A

小结

1. 掌握建立最优二叉树和哈夫曼编码的方法

作业

假设用于通讯的电文由 8 个字母组成，A B C D E F G H，字母在电文中的出现频率分别为

0.07，0.19，0.02，0.06，0.32，0.03，0.21，0.10，

试设计Huffman编码。

选做作业

提高1：找一篇英文文章，统计26个英文字母出现的概率，并计算每个字母的Huffman编码。

提高2：统计Lena图像中像素值0~255出现的概率，计算每种像素的Huffman编码，实现Lena图像的压缩。

本章小结

1. 掌握树的定义（递归）、ADT定义及遍历方法
2. 掌握树的存储结构（双亲表示/孩子表示/孩子兄弟表示）
3. 熟练掌握二叉树的递归定义和基本性质
4. 理解二叉树的抽象数据类型定义
5. 熟练掌握二叉树的遍历方法（前序、后序、中序、层次）
6. 理解二叉树的顺序存储结构及其特点
7. 熟练掌握二叉链表的定义及节点结构
8. 熟练掌握二叉树遍历的递归方法
9. 掌握二叉树的层序遍历方法
10. 掌握二叉树的建立和销毁方法
11. 掌握树、森林和二叉树之间的转换方法
12. 掌握建立最优二叉树和哈夫曼编码的方法

Thank You !

Q & A