

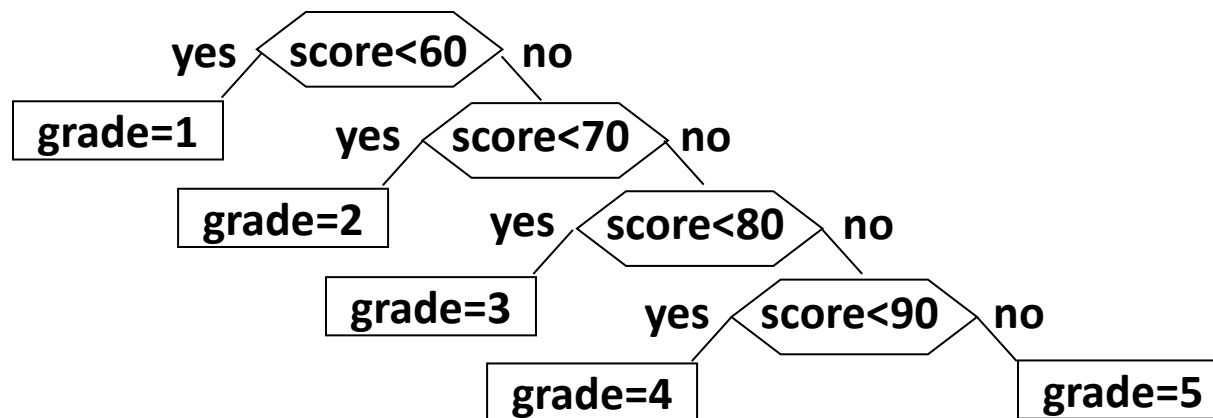
## 5.2 哈夫曼树与哈夫曼编码

# 什么是哈夫曼树 (Huffman Tree)

[例] 将百分制的考试成绩转换成五分制的成绩

```
if( score < 60 ) grade =1;  
else if( score < 70 ) grade =2;  
else if( score < 80 ) grade =3;  
else if( score < 90 ) grade =4;  
else grade =5;
```

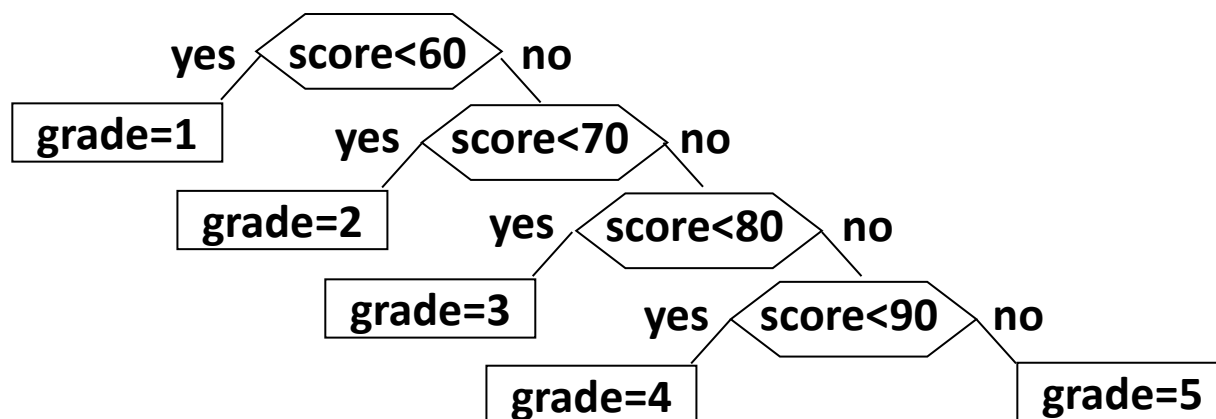
□ 判定树:



□ 如果考虑学生成绩的分布的概率：

分数段	0-59	60-69	70-79	80-89	90-100
比例	0.05	0.15	0.40	0.30	0.10

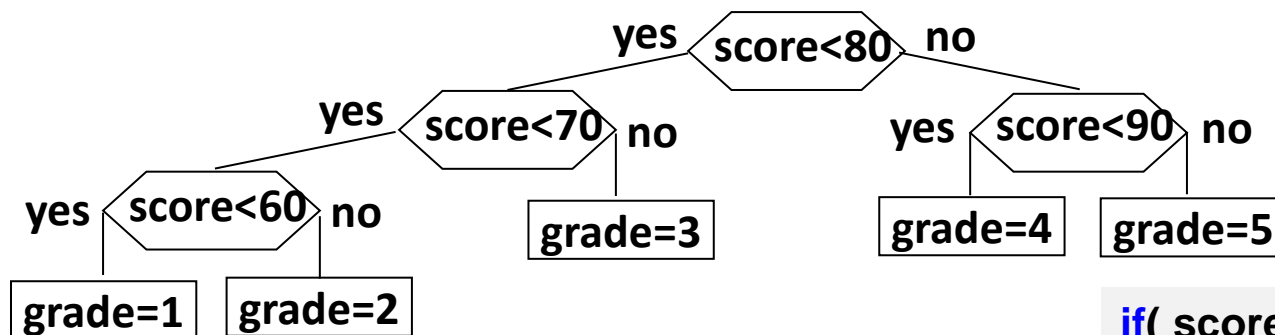
➤ 查找效率：  $0.05 \times 1 + 0.15 \times 2 + 0.4 \times 3 + 0.3 \times 4 + 0.1 \times 5 = 3.15$



□ 如果考虑学生成绩的分布的概率：

分数段	0-59	60-69	70-79	80-89	90-100
比例	0.05	0.15	0.40	0.30	0.10

□ 修改判定树：



效率:  $0.05 \times 3 + 0.15 \times 3 + 0.4 \times 2 + 0.3 \times 2 + 0.1 \times 2$   
**= 2.2**

```
if( score < 80 )
{
    if( score < 70 )
        if( score < 60 ) grade =1;
        else grade = 2;
    else grad=3;
}else if( score < 90 ) grade =4;
else grade =5;
```

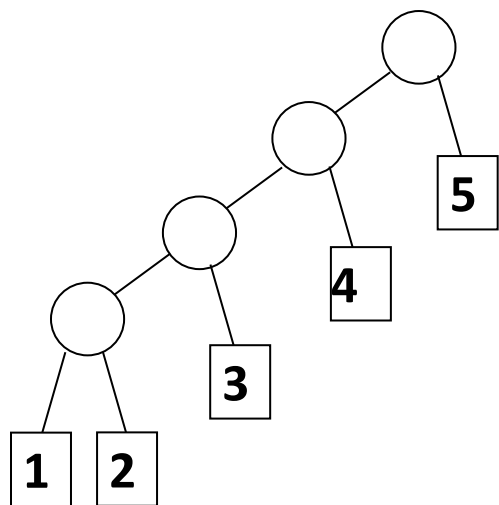
如何根据结点不同的查找频率构造更有效的搜索树？

## ❖ 哈夫曼树的定义

**带权路径长度(WPL):** 设二叉树有  $n$  个叶子结点, 每个叶子结点带有权值  $w_k$ , 从根结点到每个叶子结点的长度为  $l_k$ , 则每个叶子结点的带权路径长度之和就是: 
$$WPL = \sum_{k=1}^n w_k l_k$$

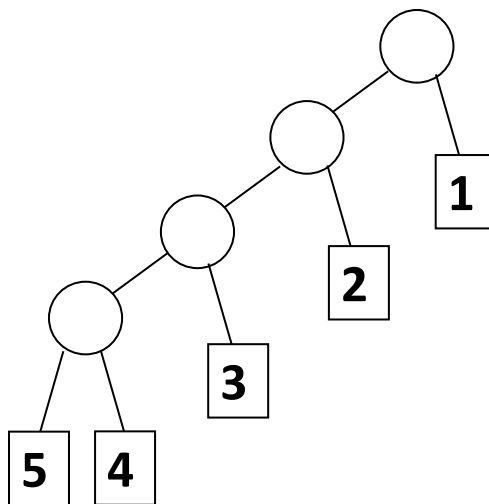
**最优二叉树或哈夫曼树:** WPL最小的二叉树

【例】有五个叶子结点，它们的权值为{1,2,3,4,5}，用此权值序列可以构造出形状不同的多个二叉树。



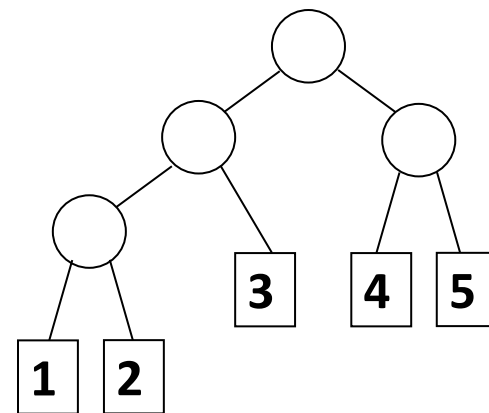
$$WPL = 34$$

$$WPL = 5 \times 1 + 4 \times 2 + 3 \times 3 + 2 \times 4 + 1 \times 4 = 34$$



$$WPL = 50$$

$$WPL = 1 \times 1 + 2 \times 2 + 3 \times 3 + 4 \times 4 + 5 \times 4 = 50$$



$$WPL = 33$$

$$WPL = 1 \times 3 + 2 \times 3 + 3 \times 2 + 4 \times 2 + 5 \times 2 = 33$$

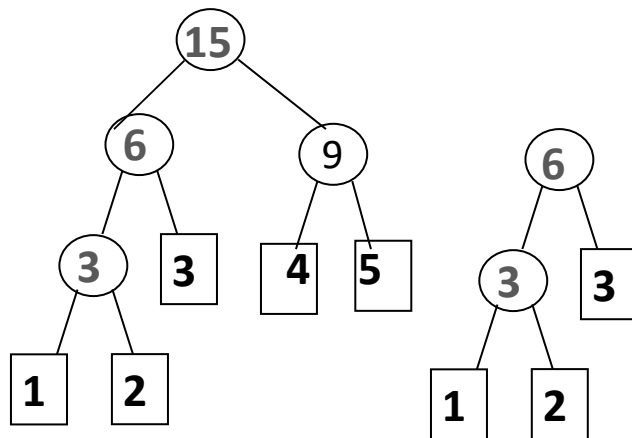
# 哈夫曼树的构造

把权值按照从小到大排序



每次把**权值最小的两棵**二叉树合并

1 2 3 4 5



```
typedef struct TreeNode *HuffmanTree;
```

```
struct TreeNode{
```

```
    int Weight;
```

```
    HuffmanTree Left, Right;
```

```
}
```

```
HuffmanTree Huffman( MinHeap H )
```

```
{    /* 假设H->Size个权值已经存在H->Elements[]->Weight里 */
```

```
    int i;    HuffmanTree T;
```

```
    BuildMinHeap(H); /*将H->Elements[]按权值调整为最小堆*/
```

```
    for (i = 1; i < H->Size; i++) { /*做H->Size-1次合并*/
```

```
        T = malloc( sizeof( struct TreeNode) ); /*建立新结点*/
```

```
        T->Left = DeleteMin(H);
```

```
                /*从最小堆中删除一个结点，作为新T的左子结点*/
```

```
        T->Right = DeleteMin(H);
```

```
                /*从最小堆中删除一个结点，作为新T的右子结点*/
```

```
        T->Weight = T->Left->Weight+T->Right->Weight;
```

```
                /*计算新权值*/
```

```
        Insert( H, T ); /*将新T插入最小堆*/
```

```
    }
```

```
    T = DeleteMin(H);
```

```
    return T;
```

```
}
```

如何选取两个最小的?

---用堆实现

排序方法效率不如堆!

整体复杂度为 $O(N \log N)$

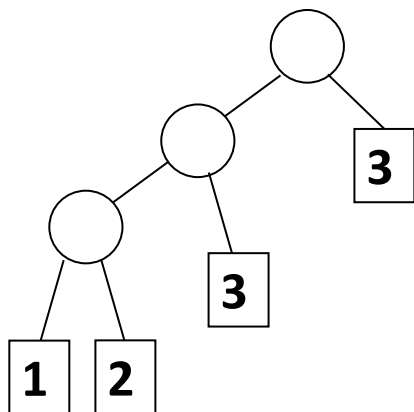


## ❖ 哈夫曼树的特点:

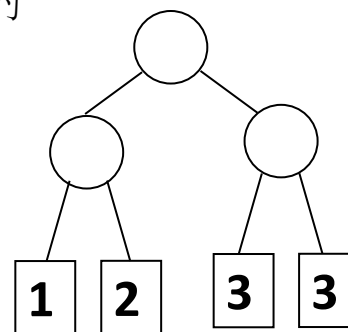
- 👉 没有度为1的结点;
- 👉  $n$ 个叶子结点的哈夫曼树共有 $2n-1$ 个结点;
- 👉 哈夫曼树的任意非叶节点的左右子树交换后仍是哈夫曼树;
- 👉 对同一组权值 $\{w_1, w_2, \dots, w_n\}$ , 是否存在不同构的两棵哈夫曼树呢?

对一组权值 $\{1, 2, 3, 3\}$ , 不同构的两棵哈夫曼树:

但是WPL值是一样的



WPL = 18



WPL = 18

$n_0$ : 叶结点总数  
 $n_1$ : 只有一个儿子的结点总数  
 $n_2$ : 有2个儿子的结点总数  
 $n_2 = n_0 - 1$

# 哈夫曼编码

□ 给定一段字符串，如何**对字符进行编码**，可以使得该字符串的编码**存储空间最少**？

[例] 假设有一段文本，包含**58**个字符，并由以下**7**个字符构：**a, e, i, s, t, 空格 (sp), 换行 (nl)**；这**7**个字符出现的次数不同。如何对这**7**个字符进行编码，使得总编码空间最少？

【分析】

- (1) 用等长**ASCII**编码： **$58 \times 8 = 464$** 位；
- (2) 用等长**3位**编码： **$58 \times 3 = 174$** 位；因为 $2^3=8$ , 用3位就可以表示8种不同的对象
- (3) 不等长编码：出现频率高的字符用的**编码短些**，出现频率低的字符则可以**编码长些**？

怎么进行不等长编码？

如何避免二义性？

👁 **前缀码** *prefix code*: 任何字符的编码都不是另一字符编码的前缀

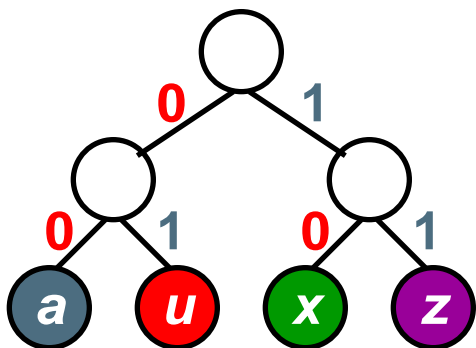
◆ 可以无二义地解码

## ❖ 二叉树用于编码

用二叉树进行编码:

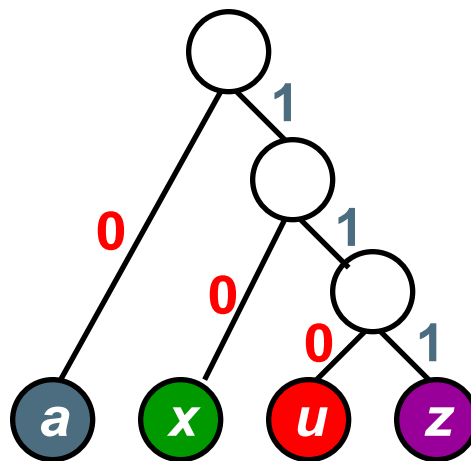
- (1) 左右分支: 0、1
- (2) 字符只在叶结点上

四个字符的频率: a:4, u:1, x:2, z:1

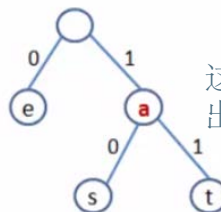


$$\text{Cost} (aaaxuaxz \rightarrow 0000001001001011) \\ = 2 \times 4 + 2 \times 1 + 2 \times 2 + 2 \times 1 = 16$$

当你的所有的节点都在叶节点上的时候,就不可能出现一个字符的编码是另一个字符编码的前缀



$$\text{Cost} (aaaxuaxz \rightarrow 00010110010111) \\ = 1 \times 4 + 3 \times 1 + 2 \times 2 + 3 \times 1 = 14$$

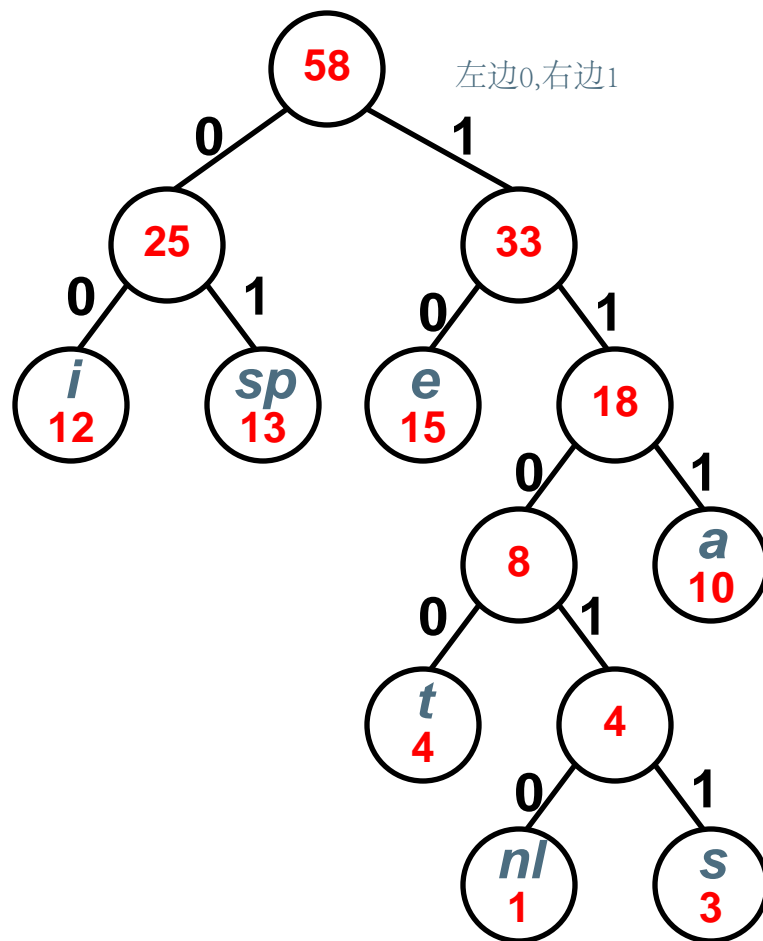


这种情况:a在s,t叶节点的上面,就会出现前缀的情况

怎么构造一颗编码代价最小的二叉树?

# 〔例〕哈夫曼编码

$C_i$	<i>a</i>	<i>e</i>	<i>i</i>	<i>s</i>	<i>t</i>	<i>sp</i>	<i>nl</i>
$f_i$	10	15	12	3	4	13	1



*a* : 111

*e* : 10

*i* : 00

*s* : 11011

*t* : 1100

*sp* : 01

*nl* : 11010

代价最小,并且没有二义性

$$\begin{aligned}
 \text{Cost} &= 3 \times 10 + 2 \times 15 \\
 &\quad + 2 \times 12 + 5 \times 3 \\
 &\quad + 4 \times 4 + 2 \times 13 \\
 &\quad + 5 \times 1 \\
 &= 146
 \end{aligned}$$