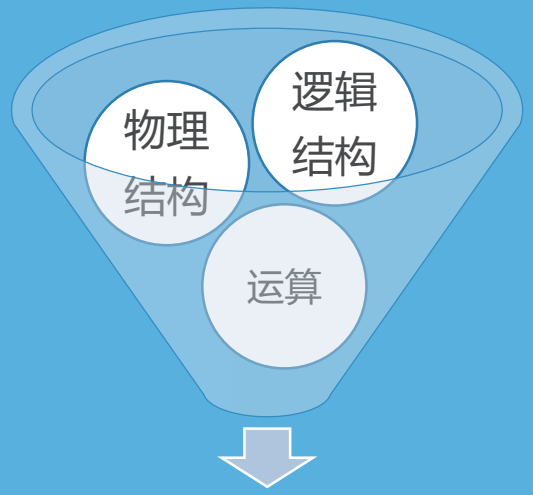


DSA 2024

数据结构与算法（Python）-06/哈夫曼树

刘云淮 Yunhuai.liu@pku.edu.cn

北京大学计算机学院

Huffman coding-问题的提出

In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding or using such a code is Huffman coding, an algorithm developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes"

二进制编码问题描述：

设需要编码的字符集合$d = \{ d_1, d_2, \ldots, d_n \}$，$p = \{ p_1, p_2, \ldots, p_n \}$ 为d中各种字符出现的频率，要对d里的字符进行二进制编码，使得

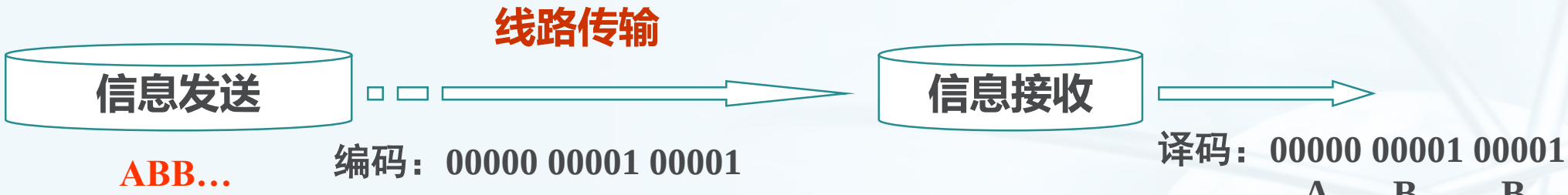
① 通信编码总长最短

② 不能出现解码歧义

方案1：定长编码

- 所有的字符都具有相同的编码长度

  如26个字符：A，B，C，D，…，Z



A：00000
B：00001
C：00010
D：00011
……
Z：11001

方案1：定长编码

定长编码的优点

编码简单，译码更简单，用于字符等概率出现时。

定长编码的缺点

对于各个字符出现频率不等概率情况(实际情况）下，定长编码不是最好的，

如果能够使得经常出现的字符编码长度短，不经常出现的可以长一些，那么整个信息串的编码长度会减少。

比如：aaaaaaaaaaaaaaaaaaab

方案2：不定长编码

- **不定长编码**：假定字符出现频率分布如下{p_1，p_2，….，p_m}，根据减少信息串编码长度的要求，频率最大的字符其编码位数应该最小。

  假定各个字符得到的编码位数为{$l_1,l_2,...,l_m$}，则总的信息串的长度为：

$$L = \sum_{i=1}^{m} P_i L_i$$

方案2:不定长编码

不定长编码的难点

为了译码时不出现歧义问题，必须保证任何字符的编码都不是其它字符编码的前缀。

如A：0， B：10， C：010，这样如果接收到 01010 序列，如何译码？是ABB还是CB？

无法确定。

不定长编码问题总结：

最经常出现的字符编码最短

避免出现二义性问题

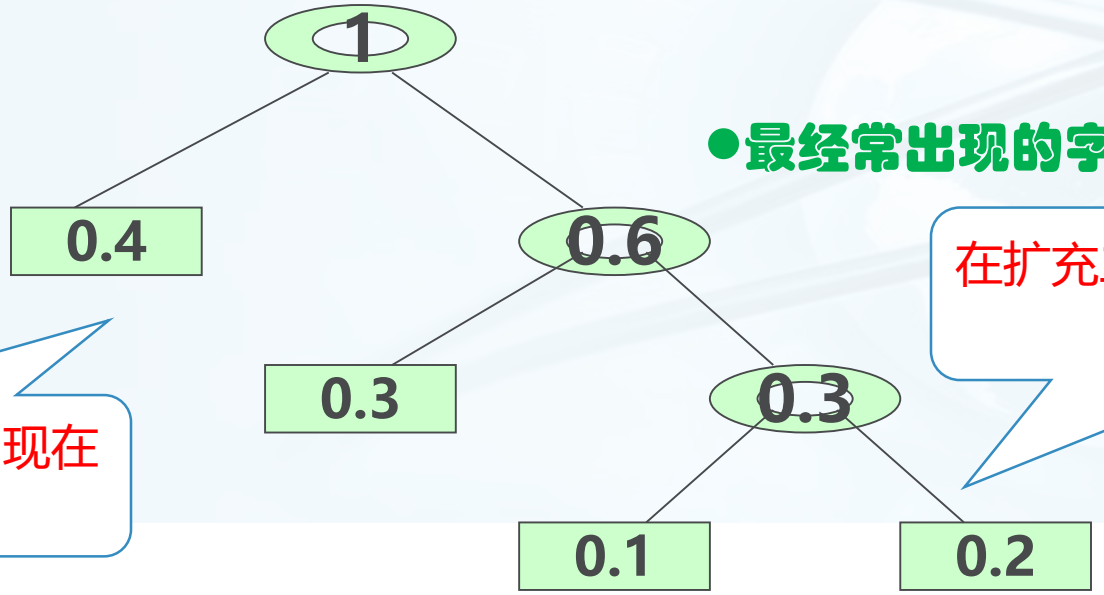
哈夫曼树（Huffman Code）

- **构造哈夫曼树是不定长编码的一种解决方案**

  所构造哈夫曼树是一棵二叉树

  利用根节点到叶结点的路径进行编码

- **例对字符集 {A(0.1), B(0.2), C(0.3), D(0.4)}**



●最经常出现的字符编码最短

在扩充二叉树中，**带权外部路径长度**最小

●避免出现二义性问题

所有需要被编码的字符出现在二叉树的叶结点上

哈夫曼树

哈夫曼（Huffman）树定义：

设有一组实数{P_1，P_2，P_3，…，P_m}，现构造一棵以P_i（i = 1, 2, …, m）为权的m个叶子节点的二叉树，使得该树

带权外部路径长度

最小。

带权外部路径长度

- 设二叉树具有m个带权值的叶子结点，那么从根结点到各个叶子结点的路径长度与相应结点权值的乘积的和，叫做二叉树的**带权的外部路径长度**。

w_i是第i个外部结点的权值

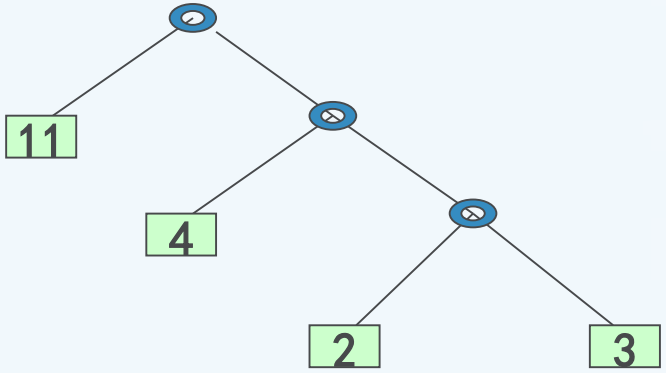
l_i为从根到第个外部结点的路径长度

m为外部结点的个数

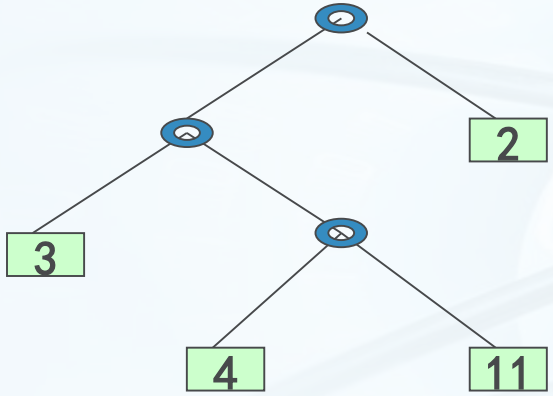
$$L = \sum_{i=}^{m} w_i l_i$$

带权外部路径长度

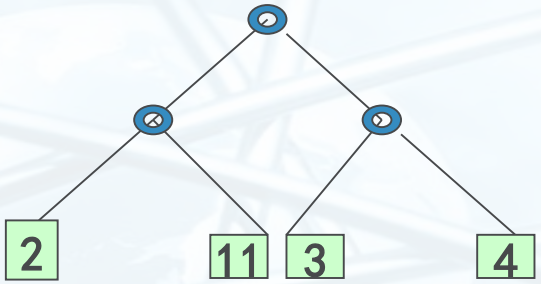
- 设字符集 {A, B, C, D}出现的频率（对应的权值）分别为{ 2，3，4，11}，我们可以形成下列三种二叉树。它们的带权外部路径长度分别为：



WPL=1*11+2*4+3* (2+3)
=34



WPL=2*3+3* (4+11)+1*2
=53



WPL=2* (2+11+3+4)
=40

规律: 权越大的叶子离根越近，则二叉树的带权外部路径长度就越小

构造哈夫曼树—哈夫曼算法

- **如何构造一棵哈夫曼树呢?**

  最早给出了带有一般规律的算法，称**哈夫曼算法**

① 根据给定的n个权值$\{w_1,w_2,\ldots,w_n\}$，构成n棵二叉树的集合$F=\{T_1,T_2,\ldots,T_n\}$，其中每一棵二叉树T_i中只有一个带权为w_i的根结点，其左右子树为空。

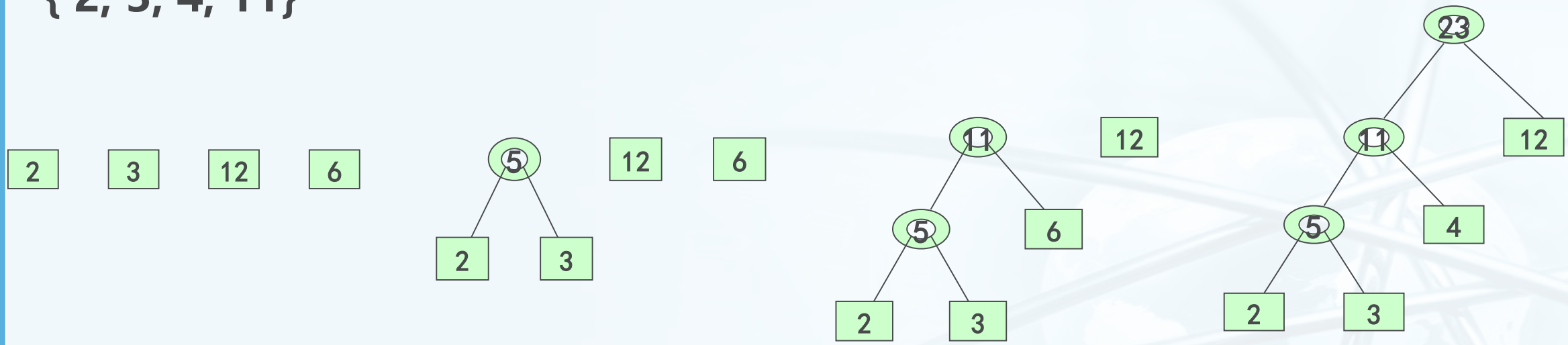
② 在F中选取两棵权值最小的树作为左右子树以构造一棵新的二叉树，且新二叉树的根结点的权值为其左右子树根结点权值之和。

③ 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(2)和(3)，直到F中只含一棵树为止。

哈夫曼树的构造过程

› 设字符集 {A, B, C, D}出现的频率（对应的权值）分别为 { 2, 3, 4, 11}

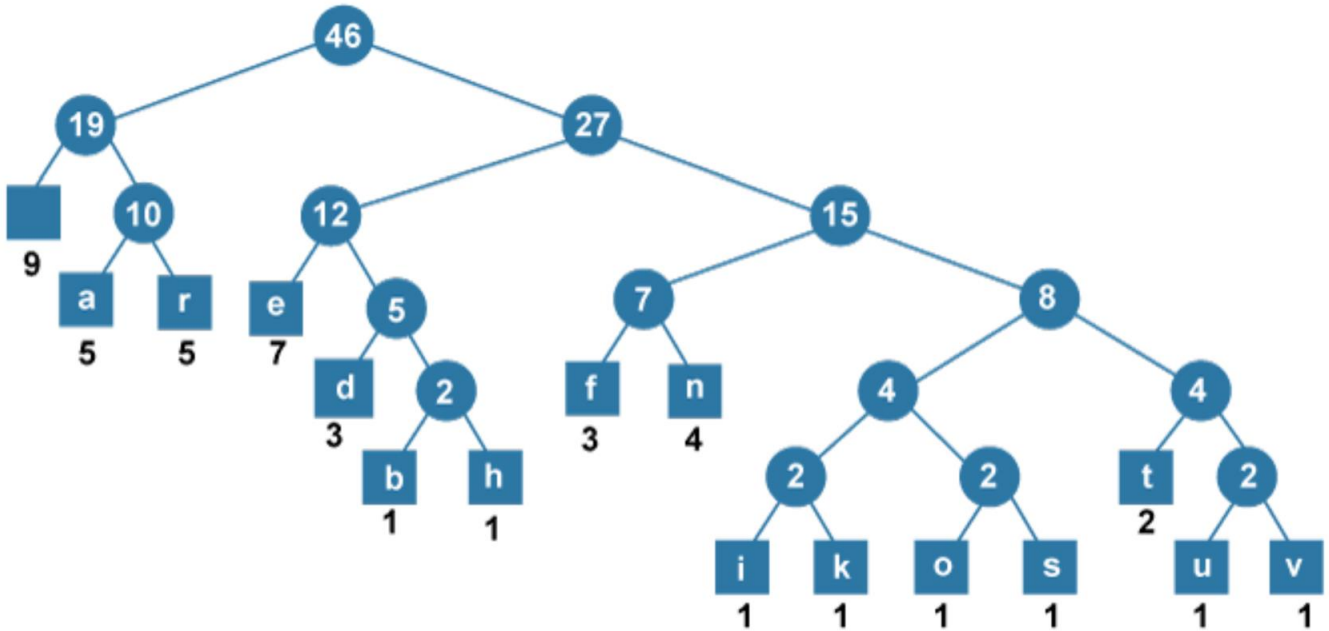


4棵只有根的二叉树　　2、3合并得到3棵二叉树　　5、6合并得到2棵二叉树　　11、12合并得到1棵二叉树

左右选择不同，得到形态不同的HuffMan树，
但它们的WPL相同。

另外一个示例

Character	a	b	d	e	f	h	i	k	n	o	r	s	t	u	v	
Frequency	9	5	1	3	7	3	1	1	1	4	1	5	1	2	1	1



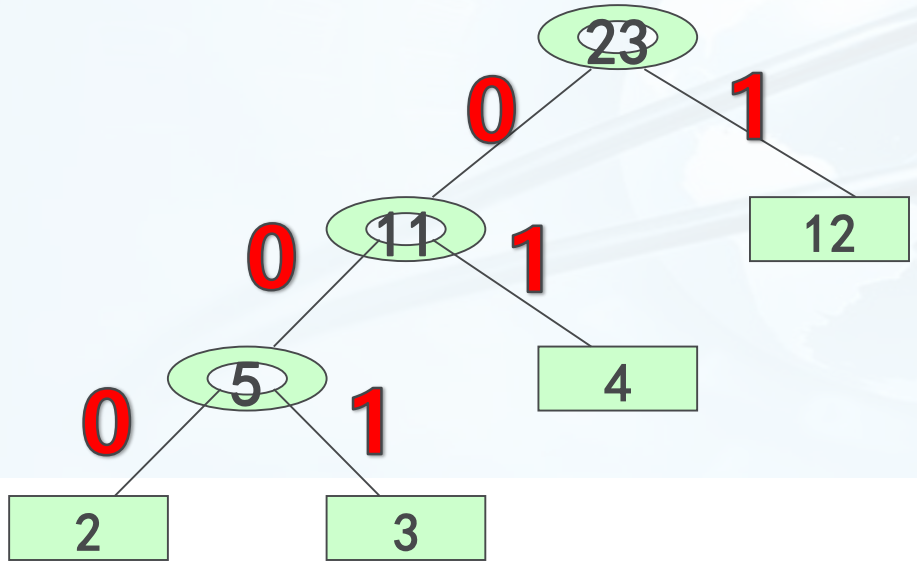
哈夫曼编码

- 根据字符出现的概率，建立HuffMan树以后，

  把从每个结点引向其左子女的边标上号码 0

  把从每个结点引向右子女的边标上号码 1

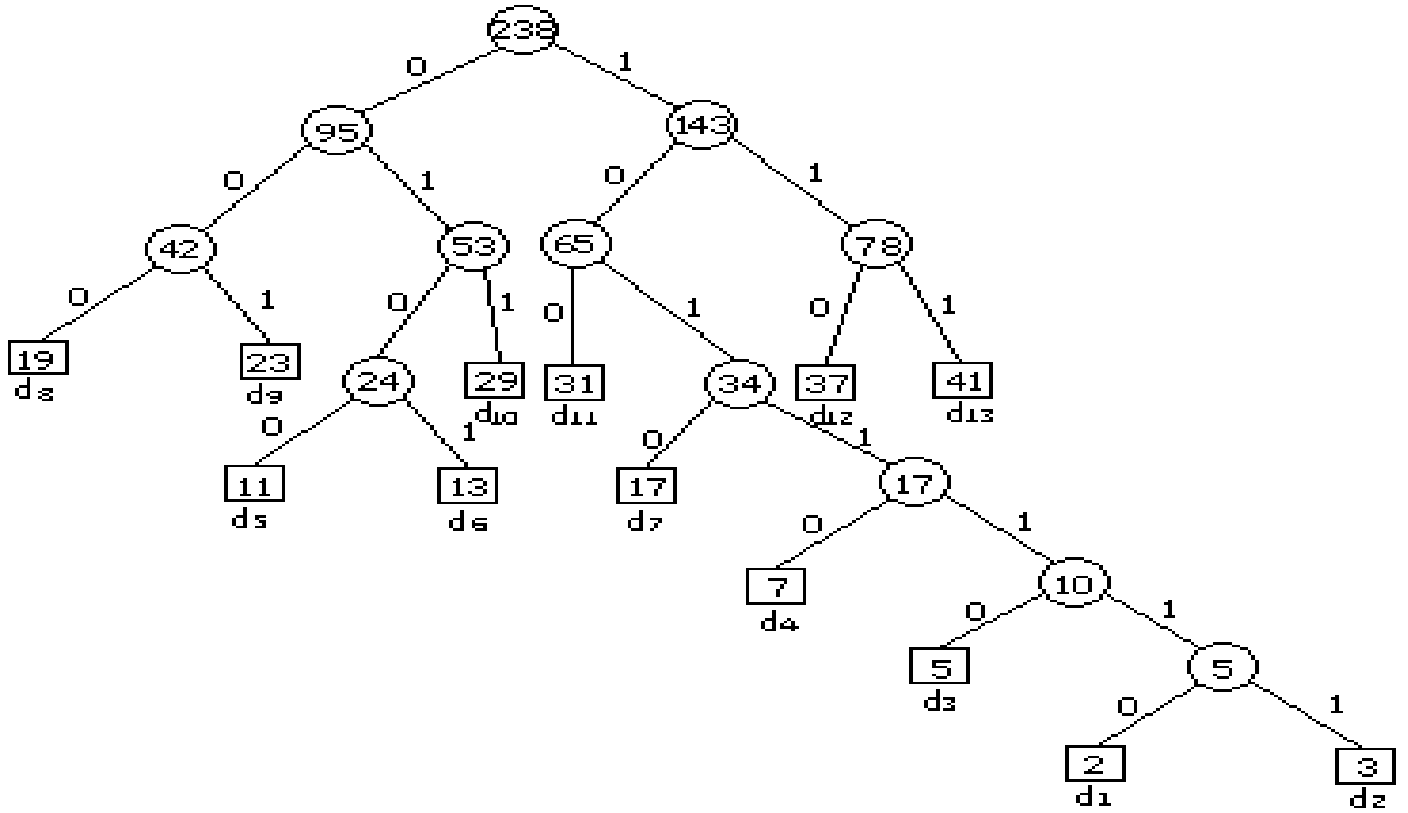
从根到每个叶子的路径上的号码连接起来就是这个叶子代表的字符的编码。



哈夫曼编码

设字符集d = { d_1，d_2，…，d_m }

对应权值w={ 2，3，5，7，11，13，17，19，23，29，31，37，41 }



部分编码结果

d_1：1011110，　d_2：1011111，　d_3：101110，　d_4：10110

哈夫曼编码的译码

由HuffMan编码得到的二进制序列如何译码?

从二叉树的根结点开始，对二进制序列的第一个字符开始匹配，如果为0，沿左分支走，如果为1，沿右分支走，直到找到一个叶结点为止，则**确定一条到达树叶的路径，译出一个字符**。

回到树根，从二进制位串中的下一位开始继续译码

如：5个字符（A，B，C，D，E)的出现频率为｛20，5，30，30，15)，

- 可能得到的编码为：　00, 010, 10, 11, 011
- 二进制序列：　00010010110110111010 译码

A B B D E E D C C

Huffman Code


```
# Huffman Coding in python

string = 'BCAADDDCCACACAC'


# Creating tree nodes
class NodeTree():

    def __init__(self, left=None, right=None):
        self.left = left
        self.right = right

    def children(self):
        return (self.left, self.right)

    def nodes(self):
        return (self.left, self.right)

    def __str__(self):
        return '%s_%s' % (self.left, self.right)


# Main function implementing huffman coding
def huffman_code_tree(node, left=True, binString=''):
    if type(node) is str:
        return {node: binString}
    (l, r) = node.children()
    d = dict()
    d.update(huffman_code_tree(l, True, binString + '0'))
    d.update(huffman_code_tree(r, False, binString + '1'))
    return d
```

```
# Calculating frequency
freq = {}
for c in string:
    if c in freq:
        freq[c] += 1
    else:
        freq[c] = 1

freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

nodes = freq

while len(nodes) > 1:
    (key1, c1) = nodes[-1]
    (key2, c2) = nodes[-2]
    nodes = nodes[:-2]
    node = NodeTree(key1, key2)
    nodes.append((node, c1 + c2))
    nodes = sorted(nodes, key=lambda x: x[1], reverse=True)

huffmanCode = huffman_code_tree(nodes[0][0])

print(' Char | Huffman code ')
print('----------------------')
for (char, frequency) in freq:
    print(' %-4r |%12s' % (char, huffmanCode[char]))
```

哈夫曼编码应用

- Huffman编码适合于
- 字符 频率不等，差别较大的情况
- 数据通信的二进制编码
- 不同的频率分布，会有不同的压缩比率
- 大多数的商业压缩程序都是采用几种编码方式以应付各种类型的文件
- Zip 压缩就是 LZ77 与 Huffman 结合

思考

- 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：

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