数据结构与算法B

07-Huffman树

问题的提出

- 数据通信的二进制编码问题描述：
  - 设需要编码的字符集合$d=\{d_1, d_2, \ldots, d_n\}$，d中各种字符出现的频率$w=\{w_1, w_2, \ldots, w_n\}$，要对d里的字符进行二进制编码，使得

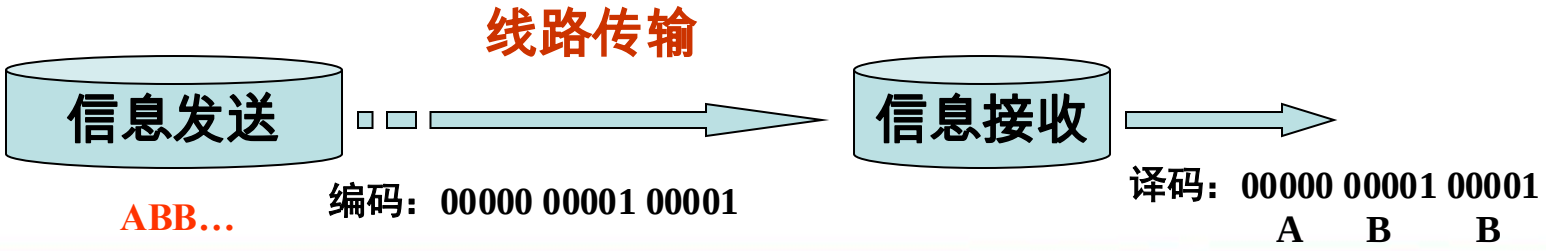
① 通信编码总长最短

② 若$d_i \neq d_j$，则d_i的编码不可能是d_j的编码的前缀。（这样就使得译码可以一个字符一个字符地进行，不需要在字符与字符之间添加分隔符）

解决方案-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



解决方案-1

- 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。

解决方案-2

- 不定长编码：假定字符出现频率分布如下$\{p_1, p_2, \ldots, p_m\}$，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$\{l_1, l_2, \ldots, l_m\}$，则总的信息串的长度为：

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

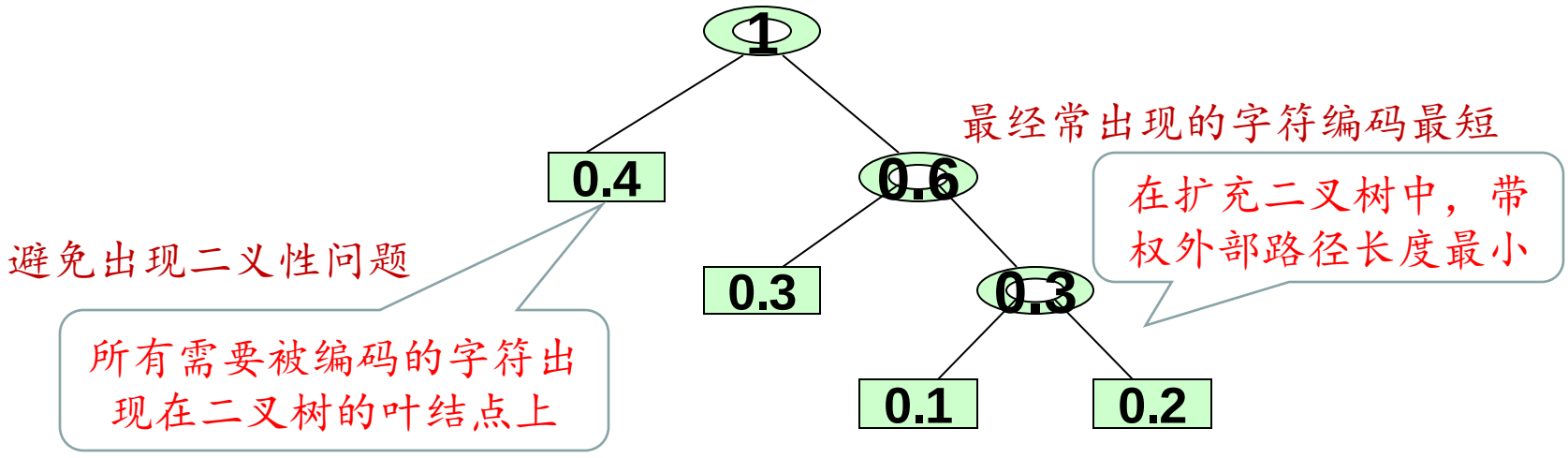
解决方案-2

- 不定长编码的难点
  - 为了译码时不出现二义问题，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

概述

- 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)}



哈夫曼树

- 哈夫曼（ Huffman ）树定义：
- 设有一组实数｛$w_1, w_2, w_3, \ldots, w_m$｝，现构造一棵以$w_i$（i = 1, 2, …， m）为权的m个外部结点的扩充的二叉树，使得该树带权外部路径长度最小。

带权外部路径长度

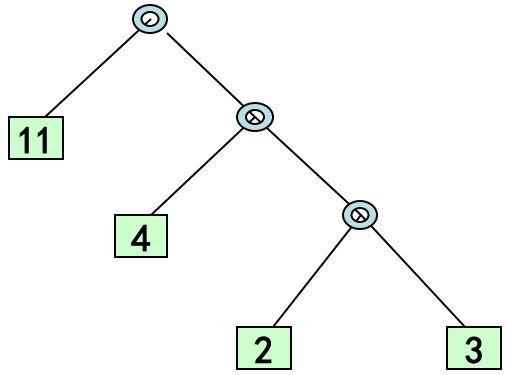
- 设扩充二叉树具有m个带权值的外部结点，那么从根结点到各个外部结点的路径长度与相应结点权值的乘积的和，叫做扩充二叉树的带权的外部路径长度。

$$WPL = \sum_{i=1}^{m} w_i l_i$$

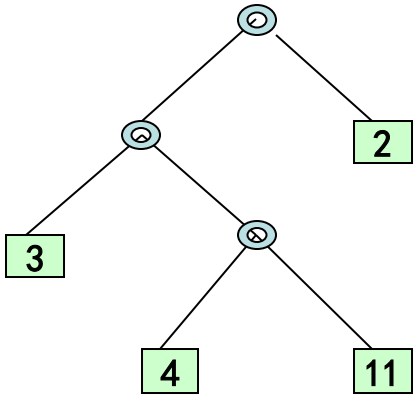
  - w_i是第i个外部结点的权值
  - L_i 为从根到第i个外部结点的路径长度
  - m为外部结点的个数

带权外部路径长度

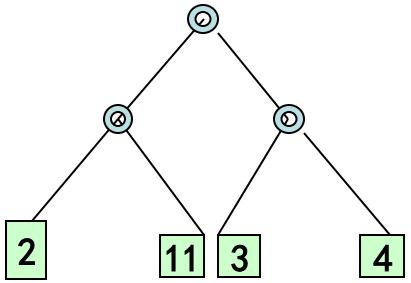
- 设字符集 {A, B, C, D}出现的频率（对应的权值）分别为{ 2, 3, 4, 11}，我们可以形成下列三种二叉树。它们的带权外部路径长度分别为：



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。

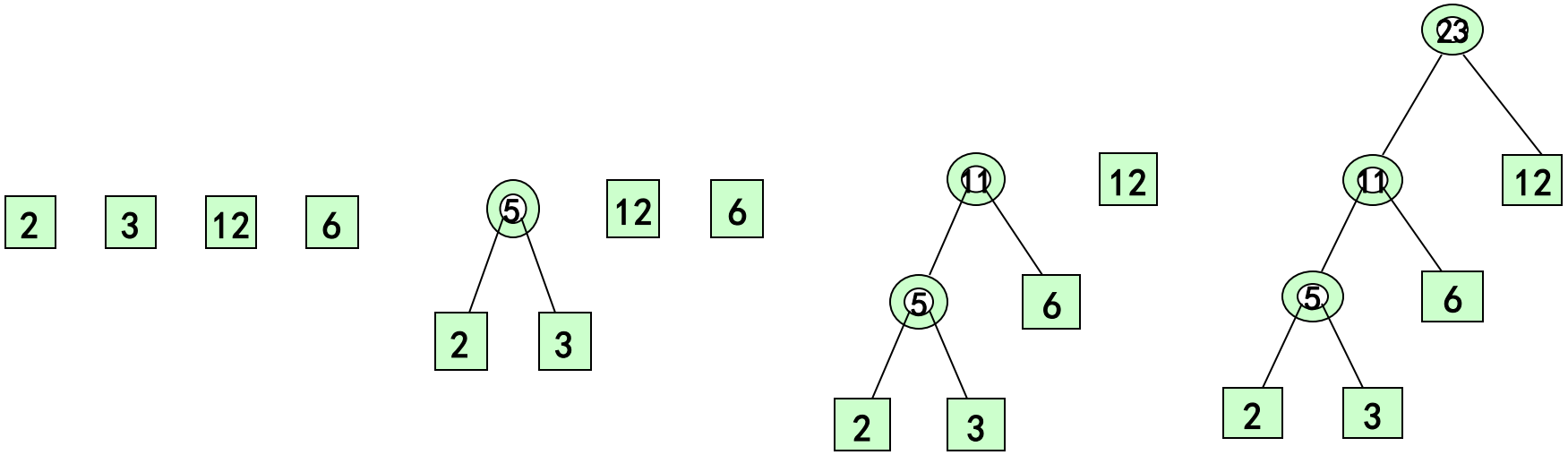
② 在F中选取两棵权值最小的树作为左右子树以构造一棵新的二叉树，且新二叉树的根结点的权值为其左右子树根结点权值之和。

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, 6, 12}



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。

哈夫曼树的顺序表示

- 假定外部结点个数为m，则内部结点个数必为m-1，因此最后得到的Huffman树必定有2m-1个结点。因此，用2m-1个元素的一维数组就可以存储该Huffman树。
- 结点结构：

ww	parent	llink	rlink

–ww: 以该结点为根的子树中所有外部结点的加权和。
–parent: 父结点在数组中的存储位置（下标），根无父，则可以设为-1。
–llink: 左子节点位置；对于外部结点，设为-1。
–rlink: 右子节点位置；对于外部结点，设为-1。

哈夫曼算法-part1

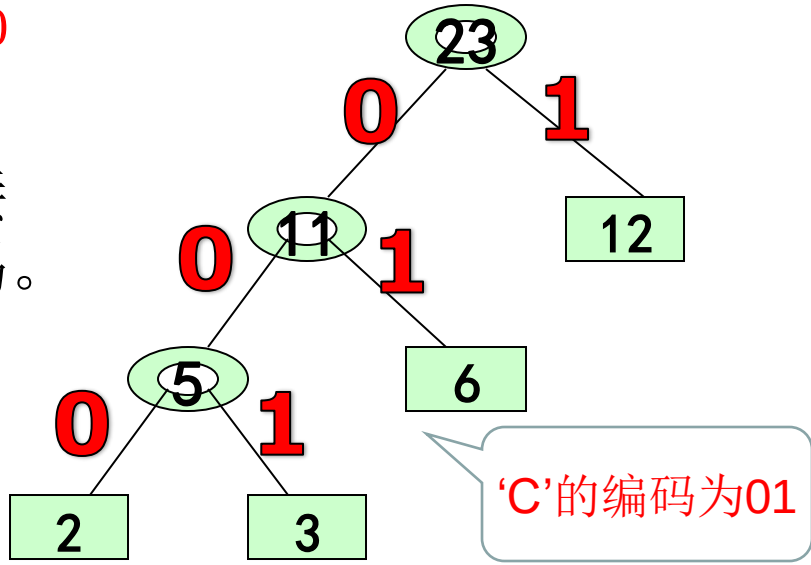
```
class HuffmanNode:
    def __init__(self, weight, char=None, parent=None):
        self.weight = weight  # 权值
        self.char = char      # 字符
        self.parent = parent  # 父节点
        self.left = None      # 左子节点
        self.right = None     # 右子节点
```

哈夫曼算法-part2

```
def build_huffman_tree(weights, chars):
    nodes = [HuffmanNode(weights[i], chars[i]) for i in
range(len(weights))]
    while len(nodes) > 1:
        nodes.sort(key=lambda x: x.weight)  # 按权值排序
        left = nodes.pop(0)  # 取出权值最小的节点
        right = nodes.pop(0)  # 取出权值次小的节点
        new_node = HuffmanNode(left.weight + right.weight)  # 创建新节
点
        new_node.left = left
        new_node.right = right
        left.parent = new_node  # 设置左子节点的父节点
        right.parent = new_node  # 设置右子节点的父节点
        nodes.append(new_node)  # 将新节点加入列表
    return nodes[0]  # 返回根节点
```

哈夫曼编码

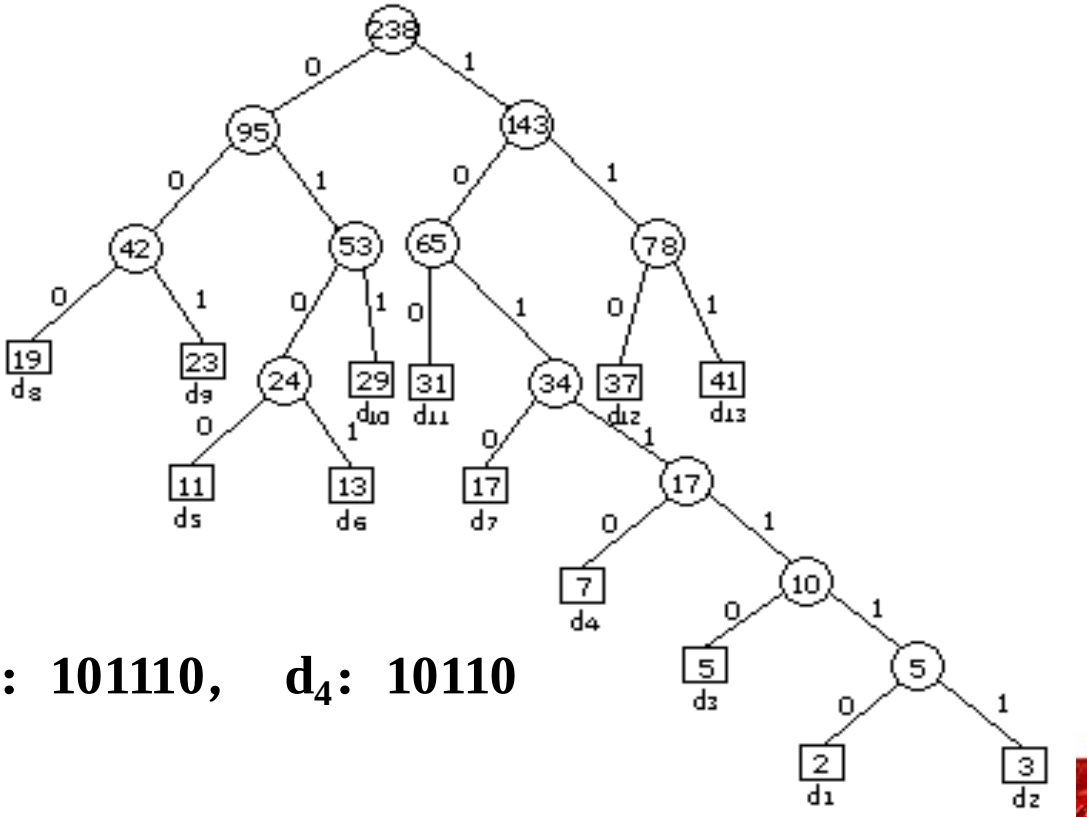
- 根据字符出现的概率，建立Huffman树以后，
  - –把从每个结点引向其左子女的边标上号码0
  - –把从每个结点引向右子女的边标上号码1
- 从根到每个叶子的路径上的号码连接起来就是这个叶子代表的字符的编码。



哈夫曼编码—书上的例子

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：　0001001011011011110 10
    - 译码：A　B　B　D　E　E　D　C　C

Huffman树的应用

- 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