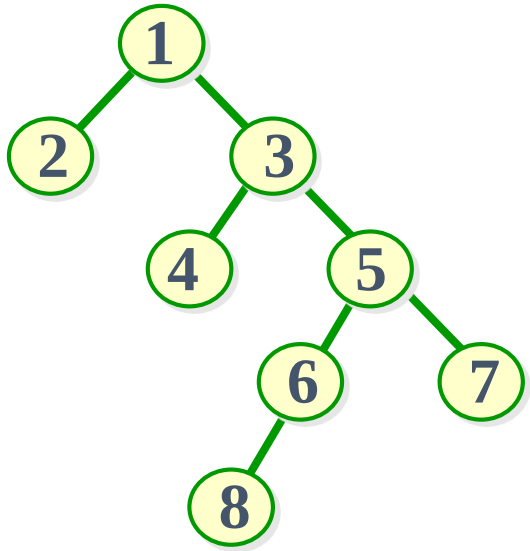
Huffman树

1. 路径长度 (Path Length)

- 两个结点之间的路径长度 PL 是连接两结点的路径上的分支数。
- 右图中，结点4与结点6间的路径长度为 3。
- 树的路径长度是各结点到根结点的路径长度之和PL。
- 右图中，PL = 0+1+1+2+2+3+3+4 = 16。

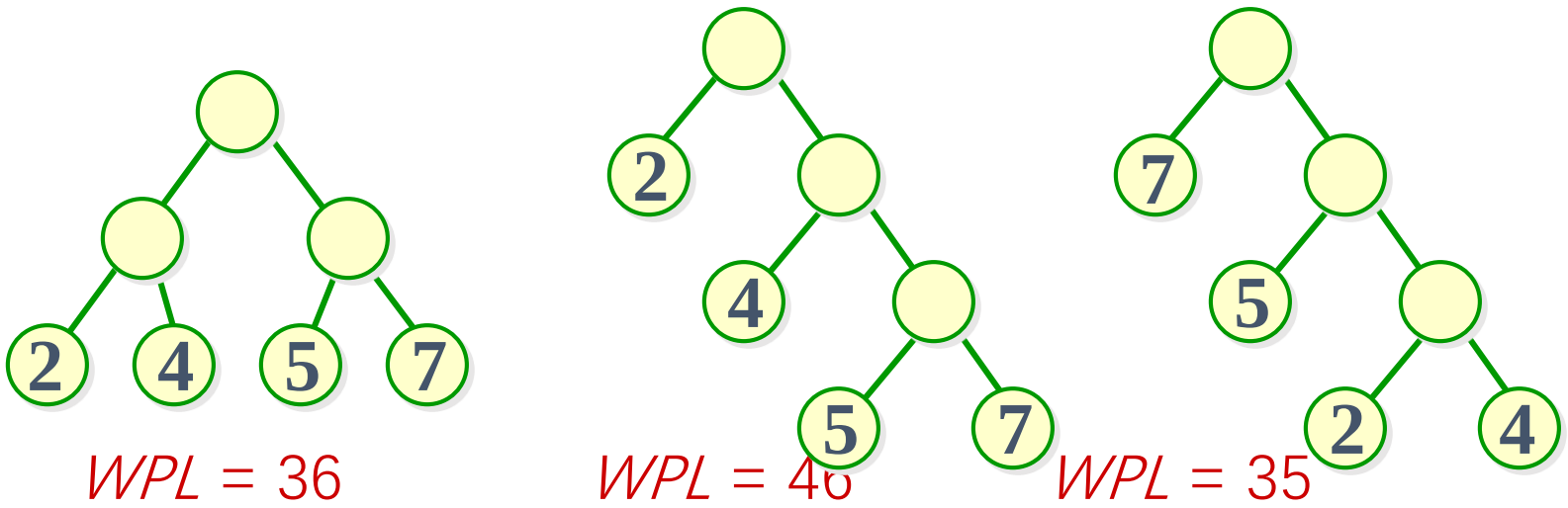


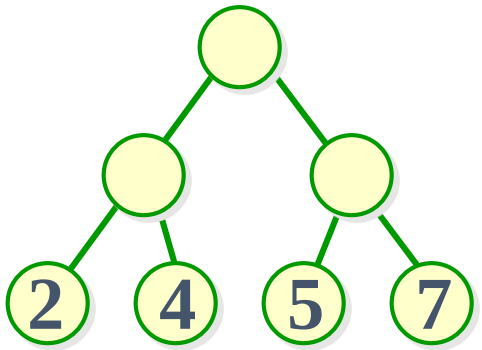
课件来自北京师范大学人工智能学院-郑新老师

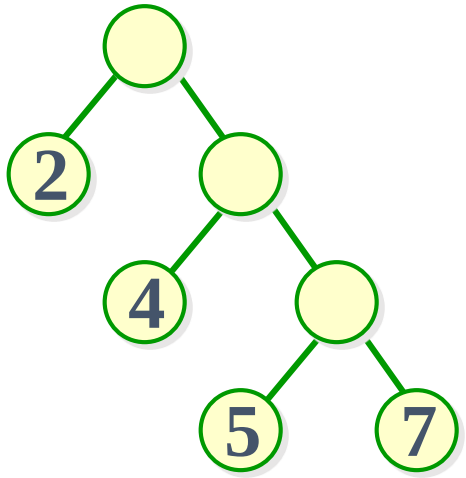
2. 带权路径长度 (Weighted Path Length, WPL)

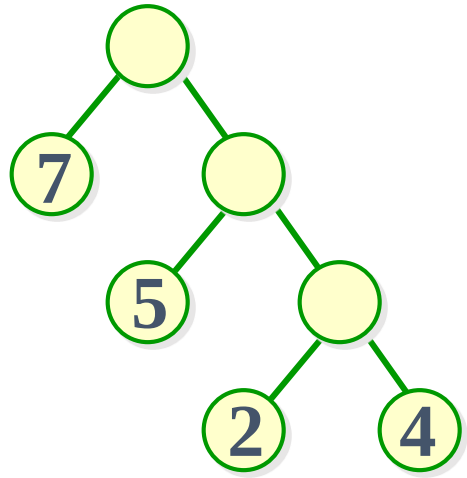
- 二叉树的带权路径长度是各叶结点所带权值 w_i 与该结点到根的路径长度 l_i 的乘积的和。

$$WPL = \sum_{i=0}^{n-1} w_i * l_i$$









WPL = 2*2+
4*2+5*2+
7*2 = 36

WPL = 2*1+
4*2+5*3+
7*3 = 46

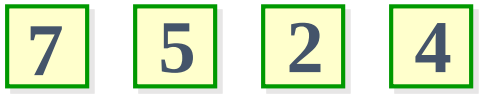
WPL = 7*1+
5*2+2*3+
4*3 = 35

- 带权路径长度达到最小的二叉树即为Huffman树。
- 在Huffman树中，权值大的结点离根最近。

Huffman树的构造算法

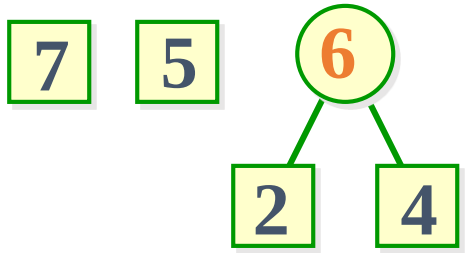
- 由给定 n 个权值 $\{w_0, w_1, w_2, \ldots, w_{n-1}\}$，构造 具有 n 棵二叉树的森林 $F = \{ T_0, T_1, T_2, \ldots, T_{n-1} \}$，其中每棵二叉树 T_i 只有一个带权值 w_i 的根结点，其左、右子树均为空。
- 重复以下步骤，直到 F 中仅剩一棵树为止：
  (1) 在 F 中选取两棵根结点权值最小的二叉树，做为左、右子树构造一棵新的二叉树。置新的二叉树的根结点的权值为其左、右子树上根结点的权值之和。
  (2) 在 F 中删去这两棵二叉树。
  (3) 把新的二叉树加入 F。

F : {7} {5} {2} {4}



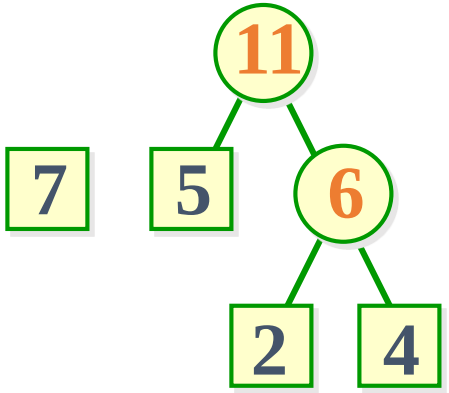
初始

F : {7} {5} {6}



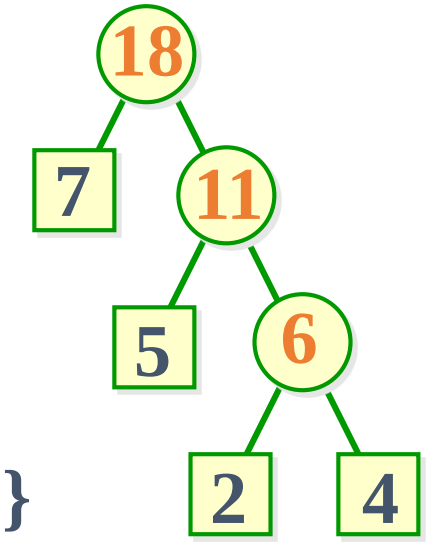
合并{2} {4}

F : {7} {11}



合并{5} {6}

F : {18}



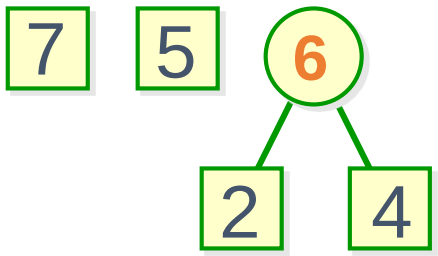
合并{5} {6}

采用静态链表的Huffman树

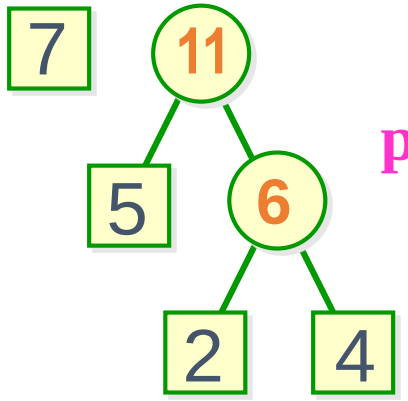
```
#define leafNumber 20                    //默认权重集合大小
#define totalNumber 39                   //树结点最大个数
typedef struct {
  char data;                        //结点的值
  int weight;                       //结点的权
  int parent, lchild, rchild;       //双亲、左、右子女
} HTNode;
typedef struct {
  HTNode elem[totalNumber];   //树存储数组
  int num, root;                          //外结点数与根
} HFTree;
```

7 5 2 4

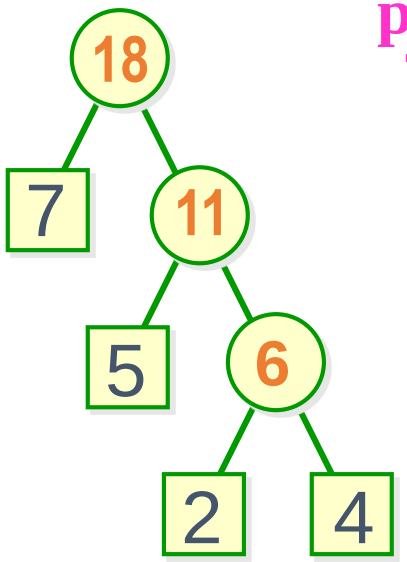
	weight	parent	lchild	rchild
0	7	−1	−1	−1
1	5	−1	−1	−1
2	2	−1	−1	−1
3	4	−1	−1	−1
4		−1	−1	−1
5		−1	−1	−1
6		−1	−1	−1



	weight	parent	lchild	rchild
0	7	−1	−1	−1
1	5	−1	−1	−1
p1 → 2	2	4 ~~−1~~	−1	−1
p2 → 3	4	4 ~~−1~~	−1	−1
i → 4	6	−1	2 ~~−1~~	3 ~~−1~~
5		−1	−1	−1
6		−1	−1	−1



	weight	parent	lchild	rchild
0	7	−1	−1	−1
p1 → 1	5	~~−1~~	−1	−1
2	2	4	−1	−1
3	4	4	−1	−1
p2 → 4	6	~~−1~~	2	3
i → 5	11	−1	~~−1~~	~~−1~~
6		−1	−1	−1



	weight	parent	lchild	rchild
p1 → 0	7	6 ~~−1~~	−1	−1
1	5	5	−1	−1
2	2	4	−1	−1
3	4	4	−1	−1
4	6	5	2	3
p2 → 5	11	6 ~~−1~~	1	4
i → 6	18	−1	0 ~~−1~~	5 ~~−1~~

建立Huffman树的算法

```
#include "Huffman.h"
#define maxWeight 32767
void createHFTree ( HFTree& HT, char value[ ],
          int fr[ ], int n ) {
//输入数据value[n]和相应权值fr[n]，构造用三叉链
//表表示的Huffman树HT
  int i, k, s1, s2;  int min1, min2;
  for ( i = 0; i < n; i++ ) {                    //外结点赋值
          HT.elem[i].data = value[i];
          HT.elem[i].weight = fr[i];
  }
```

```
  for ( i = 0; i < leafNumber; i++ ) {  //指针置空
    HT.elem[i].parent = -1;
    HT.elem[i].lchild =     HT.elem[i].rchild = -1;
}
for ( i = n; i < 2*n-1; i++ ) {              //逐步构造树
    min1 = min2 = maxWeight;
       //min1是最小权值, min2是次小权值
    s1 = s2 = 0;
       //s1是最小权值点, s2是次小权值点
    for ( k = 0; k < i; k++ )
       if ( HT.elem[k].parent == -1 )
          if ( HT.elem[k].weight < min1 ) {      //最小
```

```
            min2 = min1;  s2 = s1;              //原最小变次小
            min1 = HT.elem[k].weight;  s1 = k;     //新最小
        }
        else if ( HT.elem[k].weight < min2 )       //新次小
            { min2 = HT.elem[k].weight;  s2 = k; }
        HT.elem[s1].parent = HT.elem[s2].parent = i;
        HT.elem[i].lchild = s1;  HT.elem[i].rchild = s2;
        HT.elem[i].weight =
            HT.elem[s1].weight+HT.elem[s2].weight;
    }
    HT.num = n;  HT.root = 2*n−2;
}
```

有关Huffman树的几个要点

- Huffman树的叶结点又称为外结点，分支结点又称为内结点。外结点是结果，内结点是过程。
- Huffman树是严格二叉树。有 n 个外结点，就有 n−1 个内结点，表示需要构造 n−1 次二叉树。树中总结点数为2n−1。
- Huffman算法每次选根结点权值最小的子树来构造新二叉树时，未明确规定谁做左子树，谁做右子树，所以构造出来的Huffman树可能不惟一。
- Huffman树的最小带权路径长度是惟一的。

应用： Huffman编码

- 主要用途是实现数据压缩。设给出一段报文:

  CAST CAST SAT AT A TASA

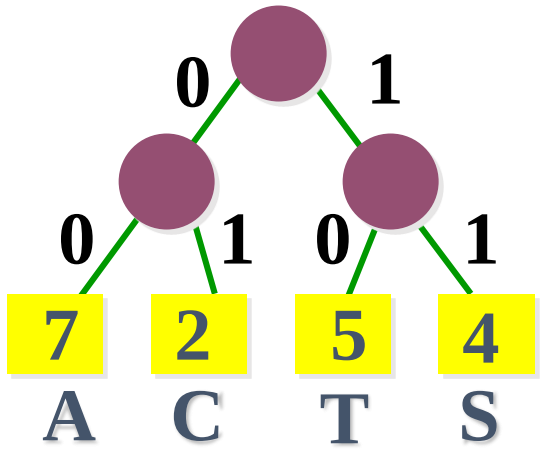
- 字符集合是 { C, A, S, T }，各个字符出现的频度(次数) 是 $W = \{ 2, 7, 4, 5 \}$。
- 若给每个字符以等长编码

  A: 00　T: 10　C: 01　S: 11

  则总编码长度为

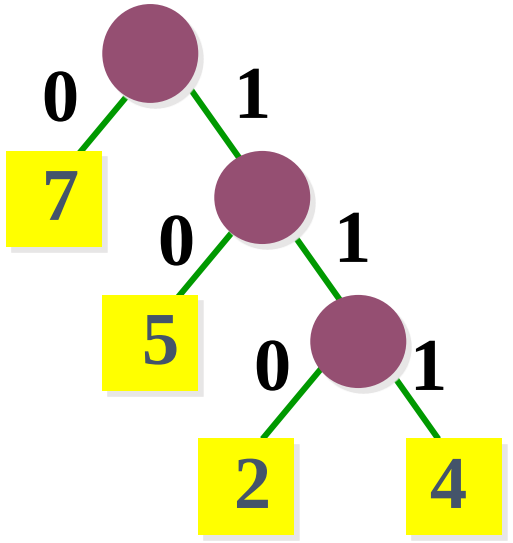
  (2+7+4+5)*2 = 36.

- 能否减少发出的报文编码数?



- 若按各个字符出现的概率不同而给予不等长的编码，可望减少总编码长度。
- 各字符出现概率为{ 2/18, 7/18, 4/18, 5/18 }，化整为 { 2, 7, 4, 5 }。以它们为各叶结点上的权值，建立Huffman树。左分支赋 0，右分支赋 1，可得Huffman编码 (变长编码)。

  A: 0　T: 10　C: 110　S: 111



- 它的总编码长度:

  7*1+5*2+(2+4)*3 = 35。

- 比等长编码的情形要短。

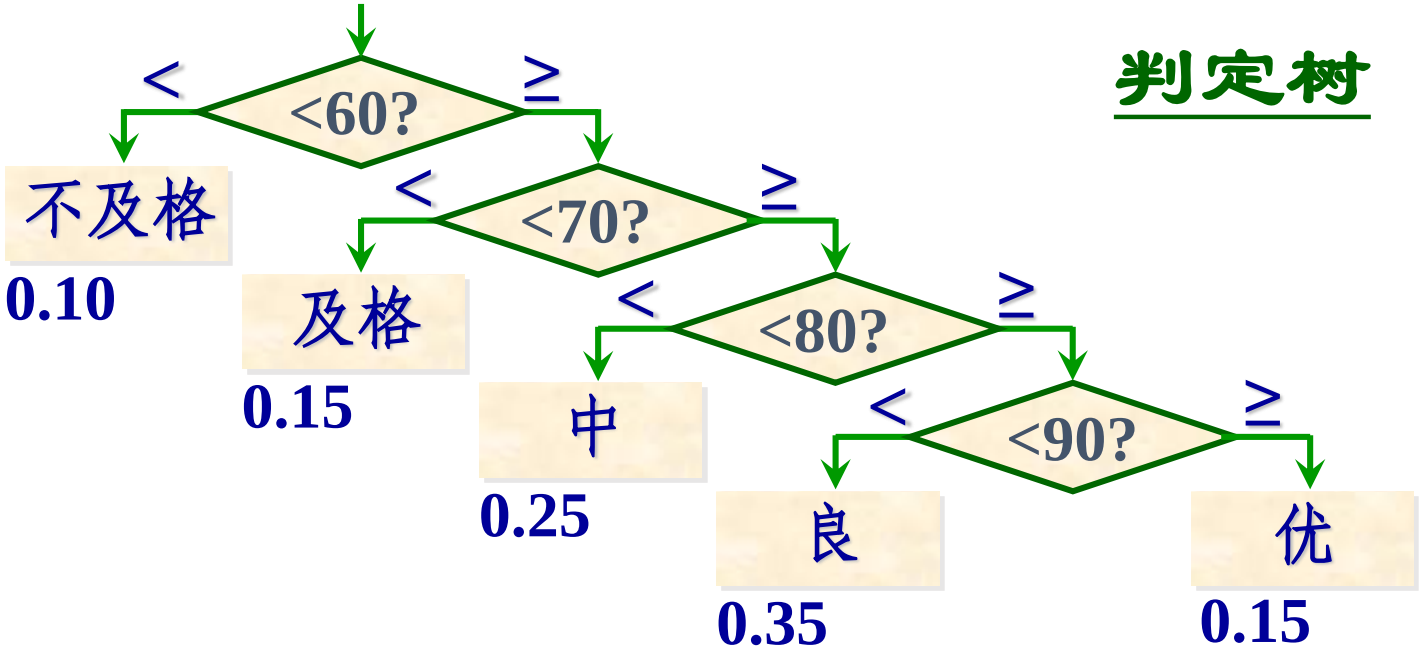
- 用Huffman编码得到的报文总编码长度正好等于Huffman树的带权路径长度WPL。
- Huffman编码是一种前缀编码，任何一个字符的编码不是其他字符编码的前缀，因此在解码时不会混淆。

应用：最佳判定树

- 利用Huffman树，可以在构造判定树（决策树）时让平均判定（比较）次数达到最小。
- 判定树是一棵扩展二叉树，外结点是比较结果，内结点是比较过程，外结点所带权值是概率。

例：考试成绩分布表

[0, 60)	[60, 70)	[70, 80)	[80, 90)	[90, 100]
不及格	及格	中	良	优
0.10	0.15	0.25	0.35	0.15



- 该判定树的带权路径长度

  WPL = 0.10*1+0.15*2+0.25*3+0.35*4+0.15*4

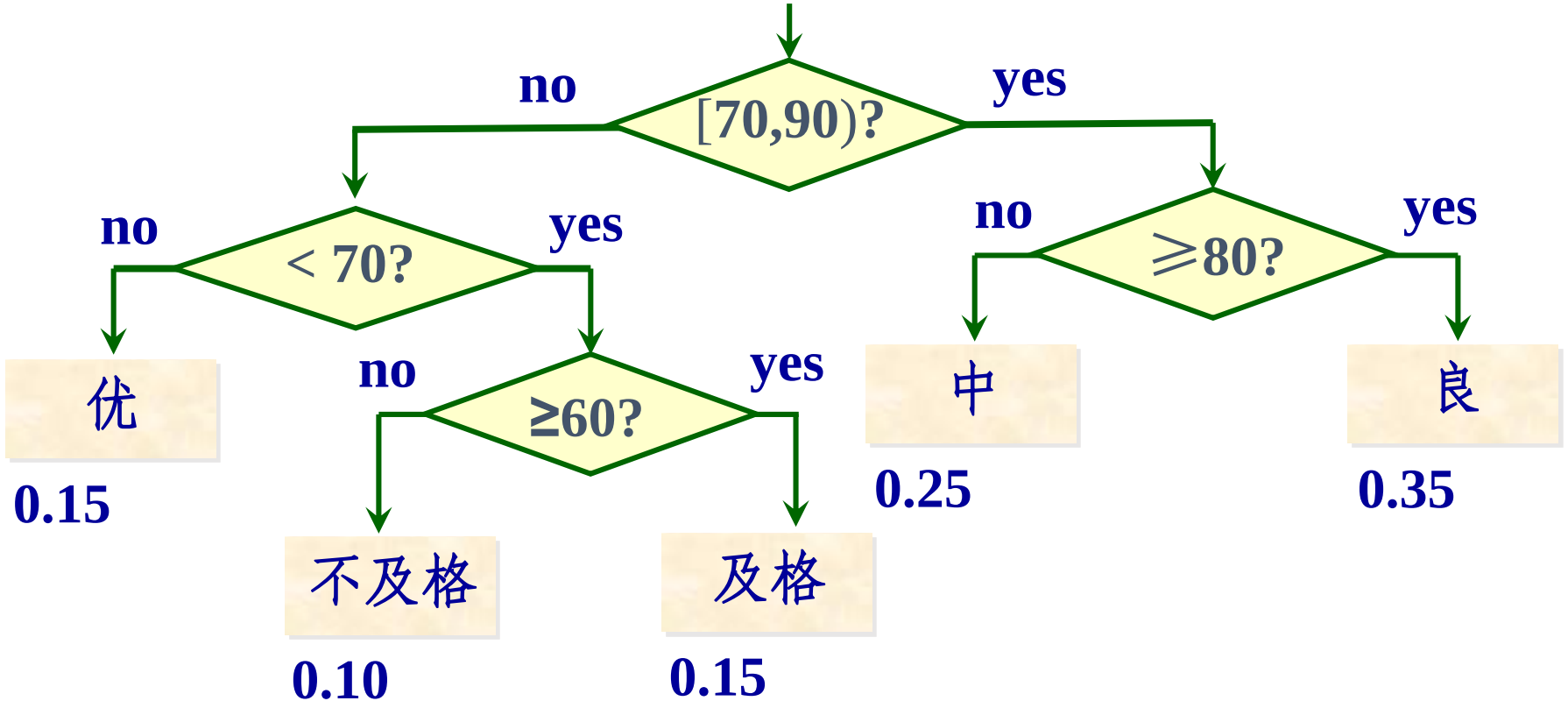
  = 3.15

- 此带权路径长度描述了在树中查找到任一外结点时的平均比较次数。此平均比较次数越少越好。
- 如果按照Huffman算法的思想构造Huffman树，可望得到平均比较次数更少的判定树。
- 下图就是按Huffman算法构造出的判定树。其带权路径长度为:

  WPL = 0.10*3+0.15*3+0.25*2+0.35*2+0.15*2

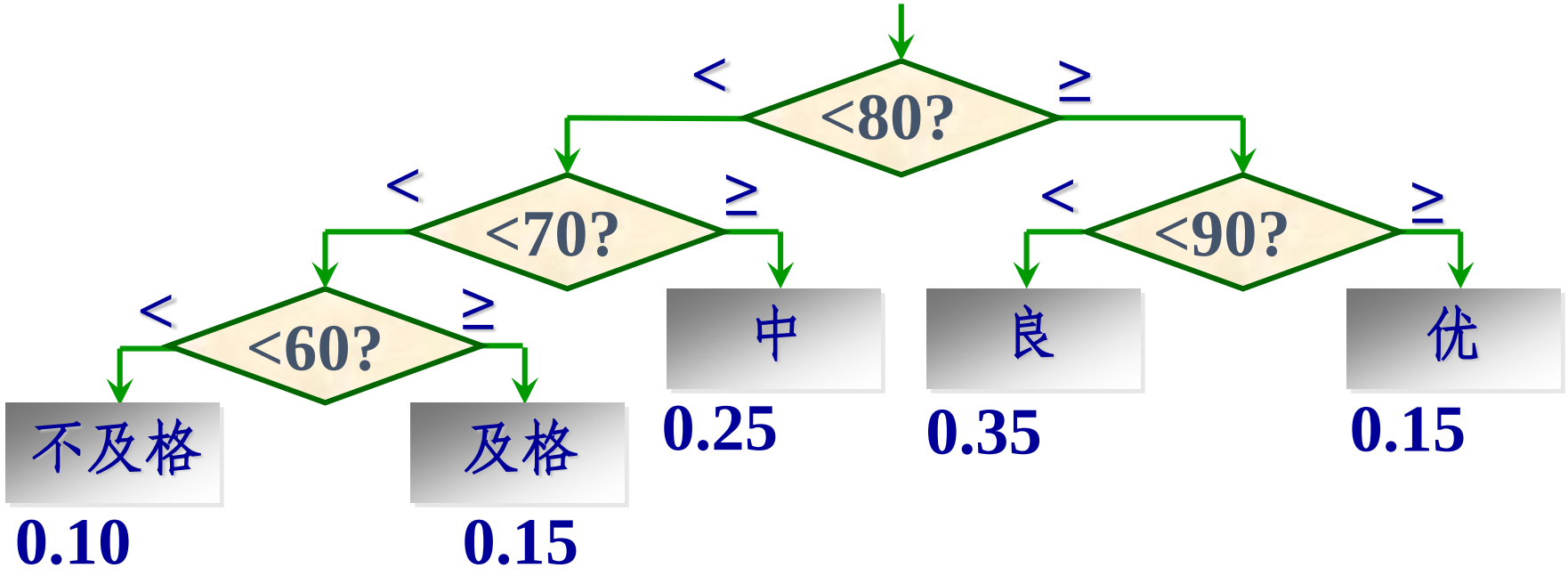
  = 0.3+0.45+0.5+0.7+0.3 = 2.25

按Huffman算法改造判定树



- 此判定树的问题是：根结点的判定需要2次比较。为此，对它加以调整，可得到最合理的判定树。

最佳判定树

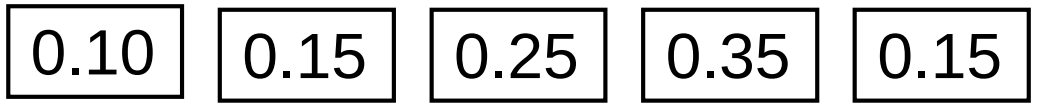


- WPL = 0.10*3+0.15*3+0.25*2+0.35*2+0.15*2

  = 0.3+0.45+0.5+0.7+0.3 = 2.25

- 其构造过程如下：

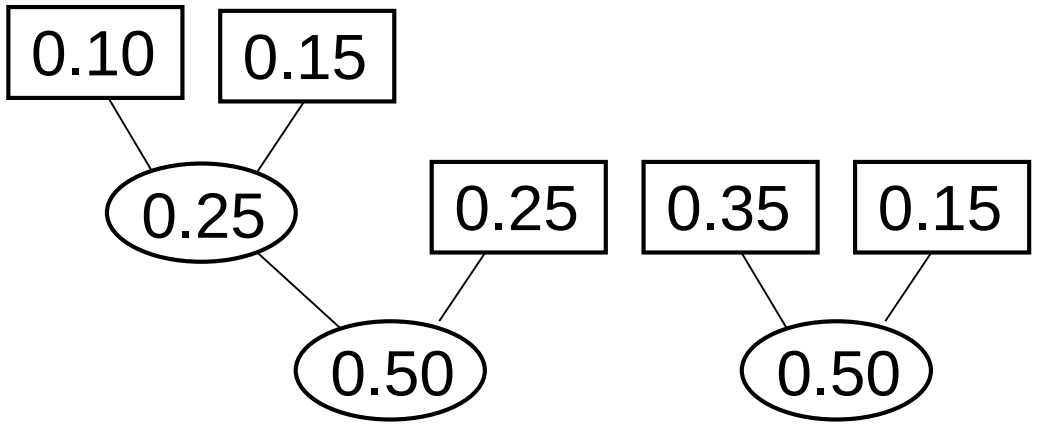
① 初始排列



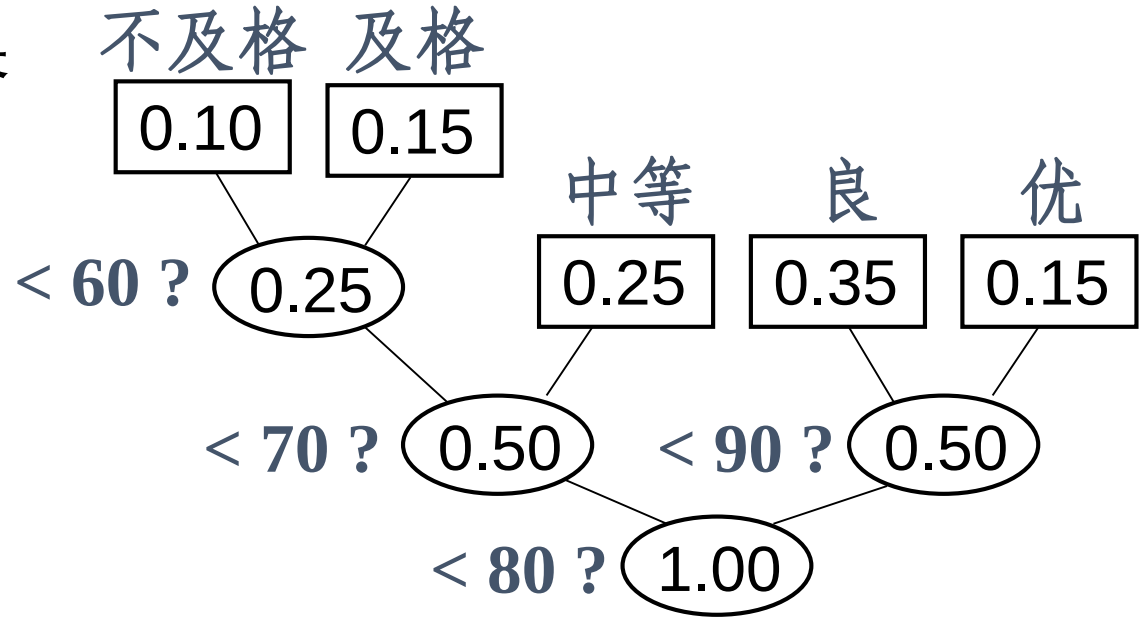
② 相邻结点权值相加，取其和最小者构造二叉树。注意不要打乱原来的排列。

③ 重复上一步，直到构成一棵树为止。

④ 继续


0.10
0.15
0.25
0.25
0.35
0.15
0.50
0.50

⑤ 结果


不及格
及格
0.10
0.15
中等
良
优
< 60 ?
0.25
0.25
0.35
0.15
< 70 ?
0.50
< 90 ?
0.50
< 80 ?
1.00