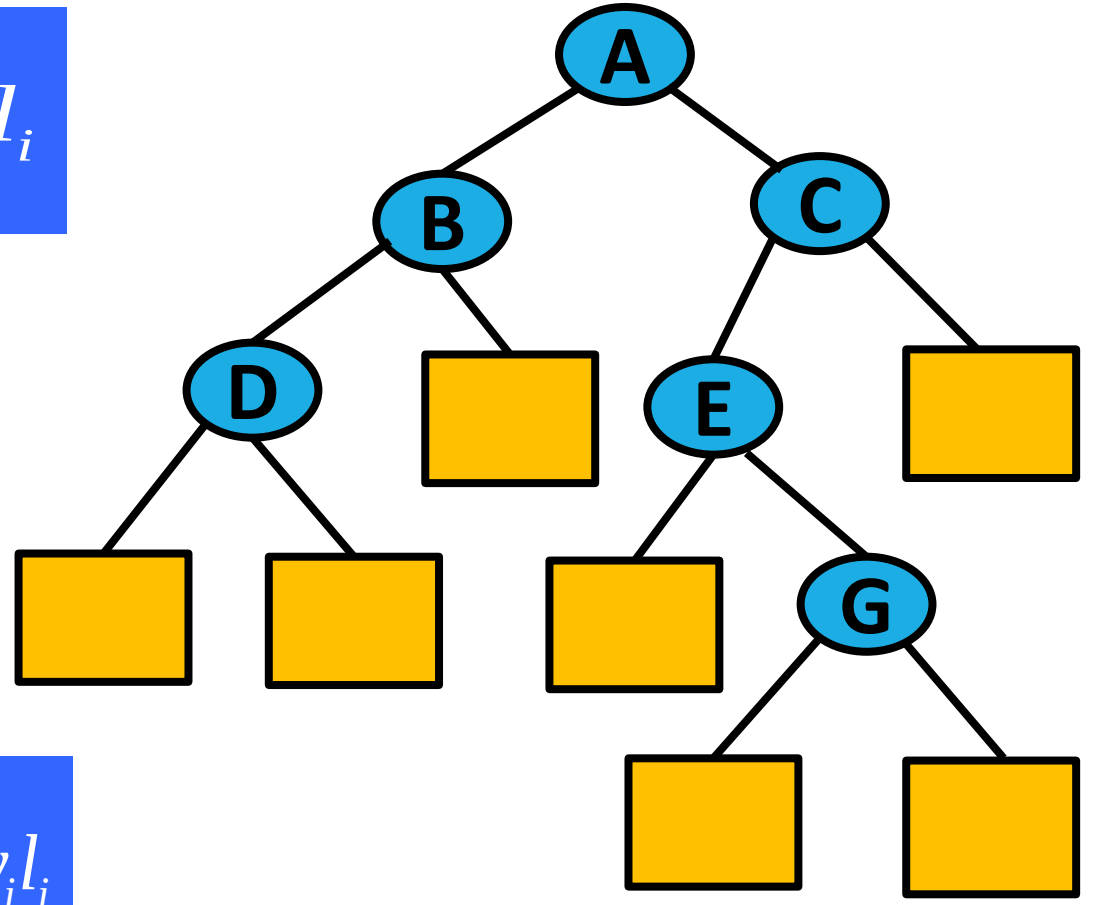
4.13 二叉树的应用-- 哈夫曼树（最优二叉树）

扩充二叉树的外部路径长度：

$$E = \sum_{i=1}^{m} l_i$$

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



扩充二叉树的带权的外部路径长度

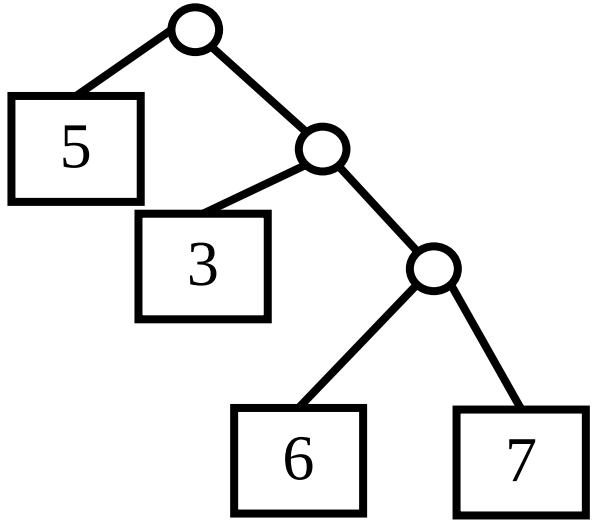
$$WPL = \sum_{i=1}^{m} w_i l_i$$

w_i是第i个外部结点的权值

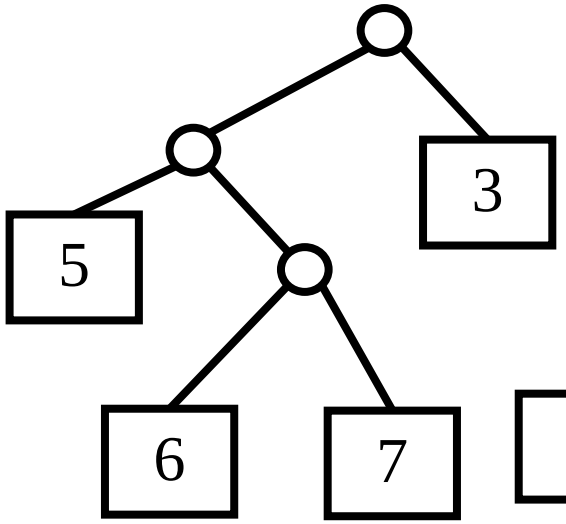
哈夫曼树：最优二叉树

假设有一组（无序）实数 $\{w_1, w_2, w_3, \ldots, w_m\}$，现要构造一棵以$w_i$（i = 1,2,…，m）为权的m个外部结点的扩充的二叉树，使得带权的外部路径长度**WPL最小**。满足这一要求的扩充二叉树就称为**哈夫曼树或最优二叉树**

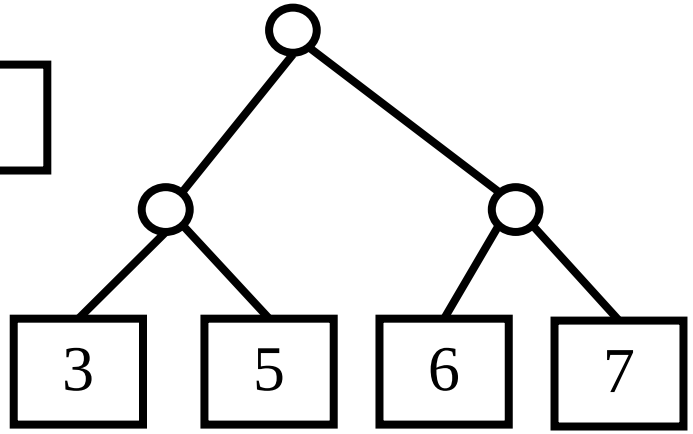
例如：给出权是{ 3,5,6,7}，则可以构造不同的二叉树



WPL=50



WPL=52



WPL=42

哈夫曼算法

(1) 由给定的m个权值$\{w_1,w_2,\ldots,w_m\}$构造成m棵二叉树集$F=\{T_1,T_2,\ldots,T_m\}$，其中每一棵二叉树T_i中只有一个带权为w_i的根结点，且根结点的权值为w_i

(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

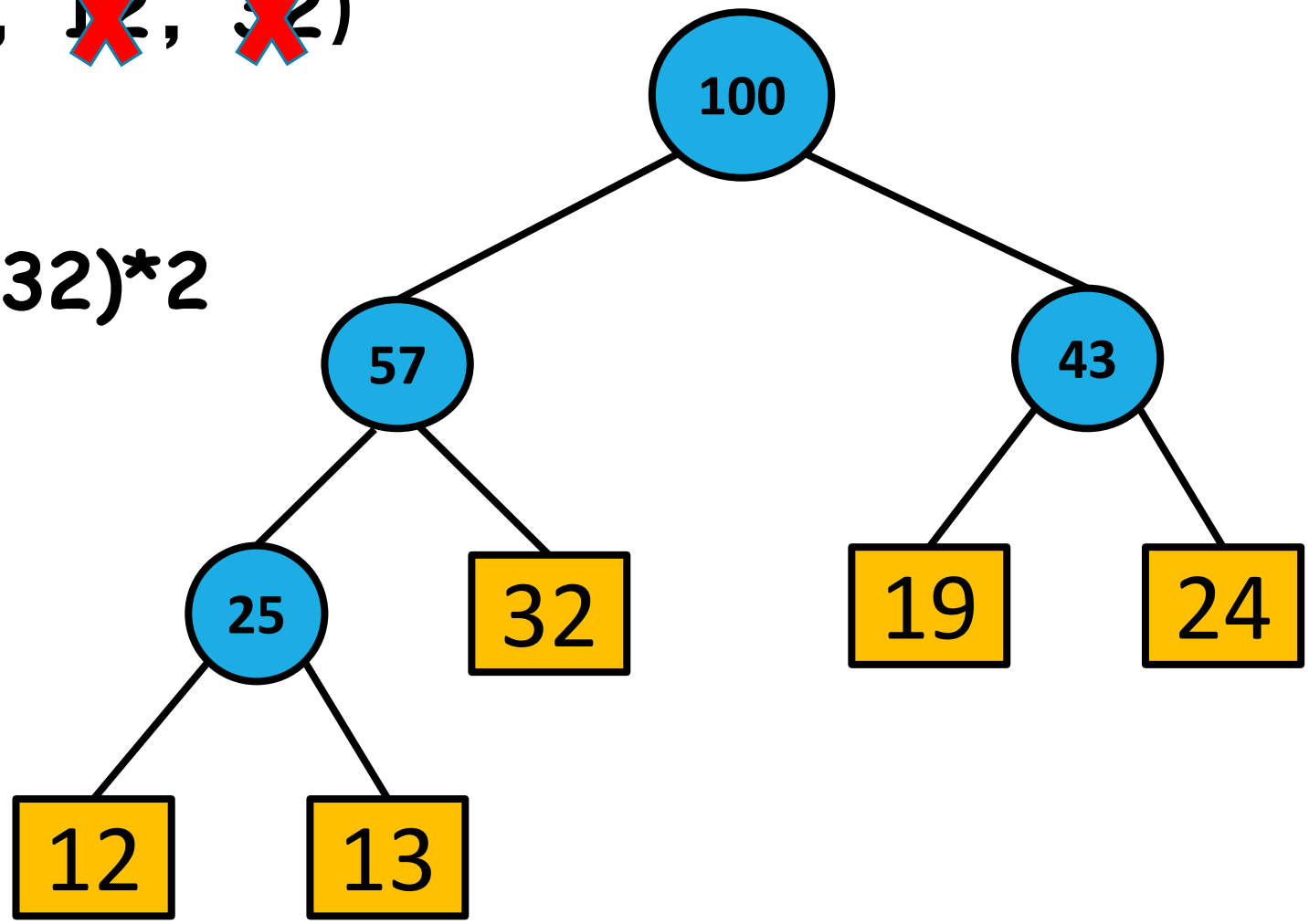
(3) 在F中删除这两棵树，同时将新得到的二叉树加入F中

(4) 重复（2）和（3），直到F中只含一棵树为止

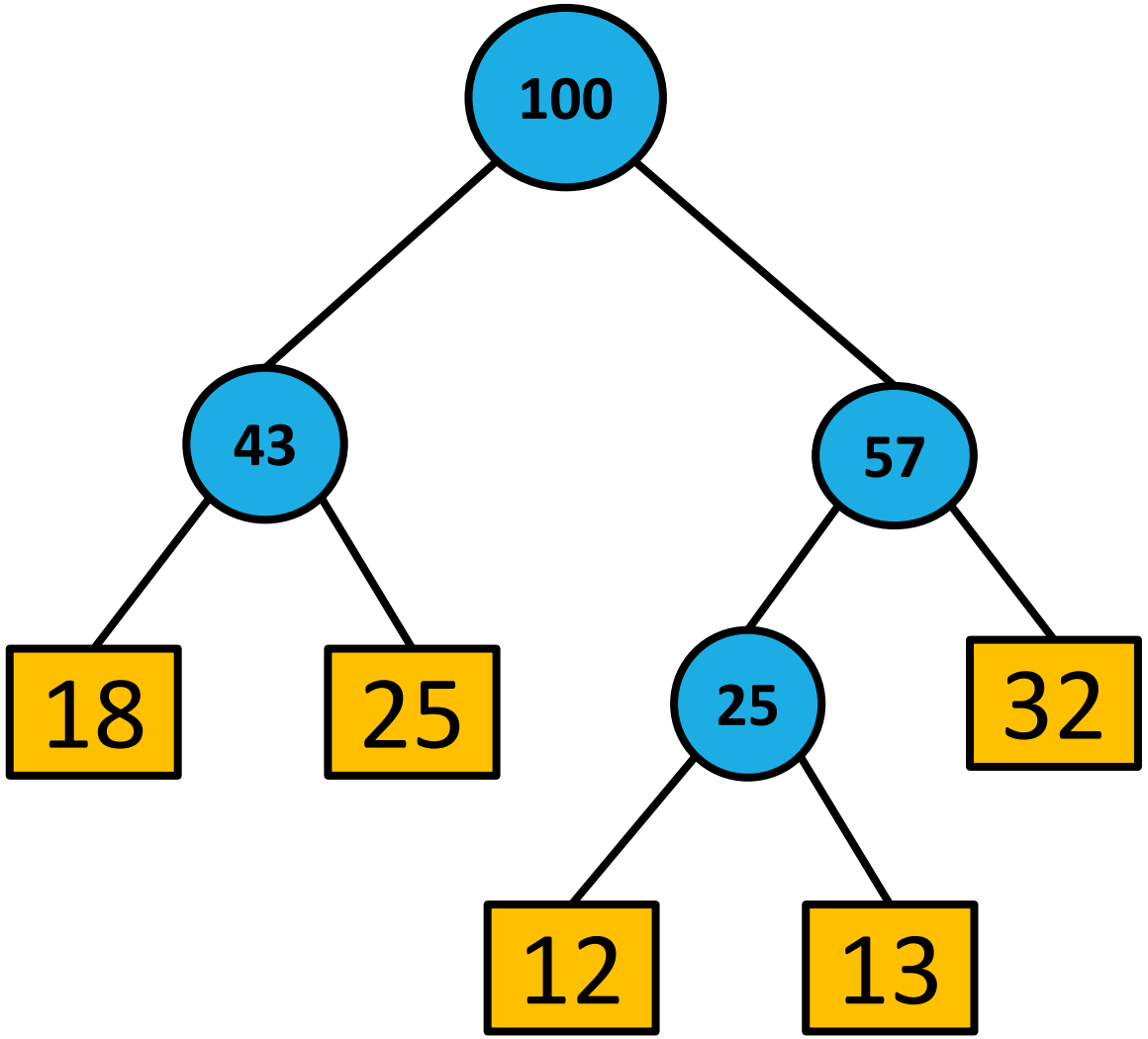
哈夫曼算法

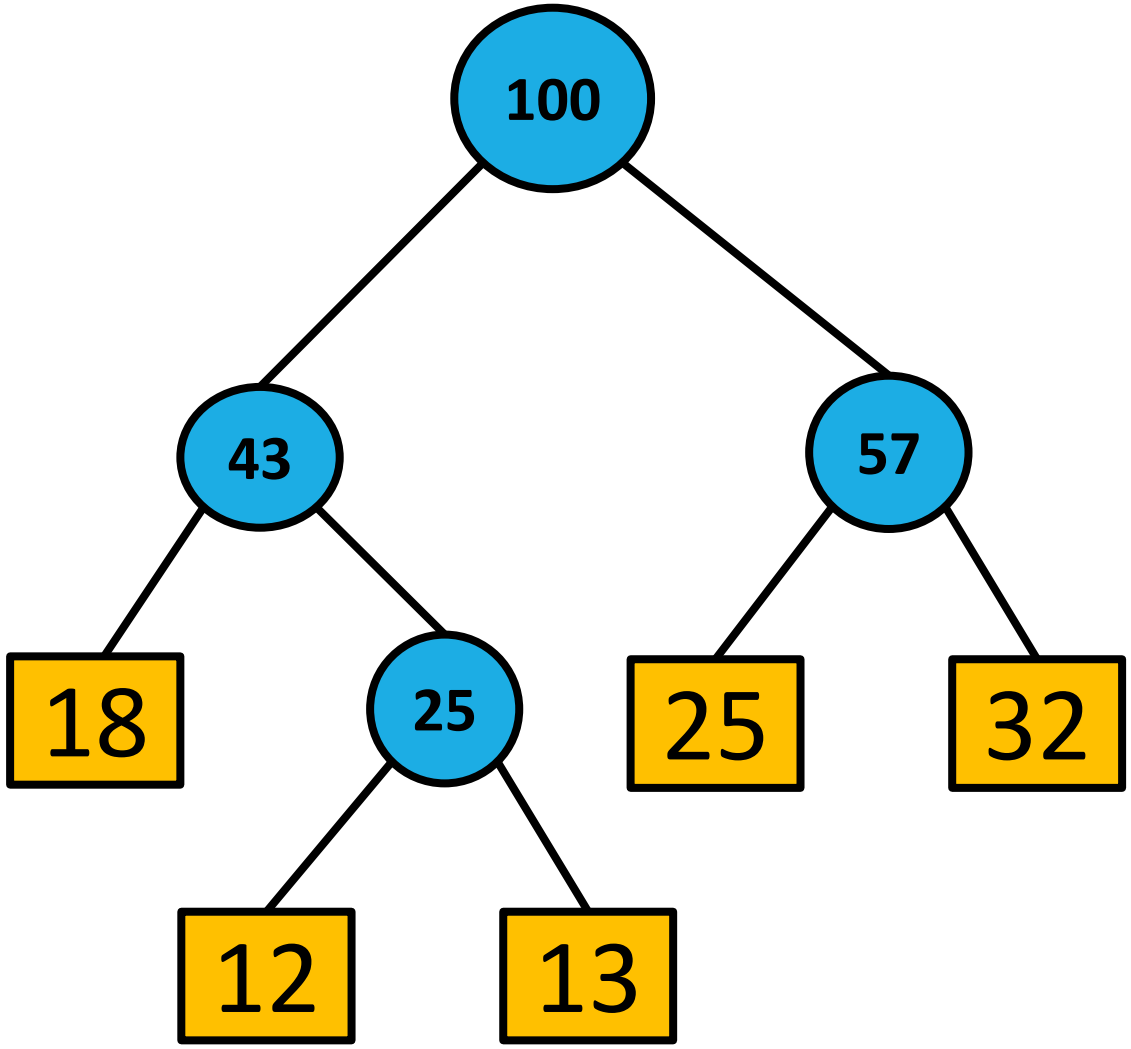
给定权值W=（19，24，13，12，32）
画出哈夫曼树并计算其WPL

WPL=(12+13)*3+(19+24+32)*2
=225

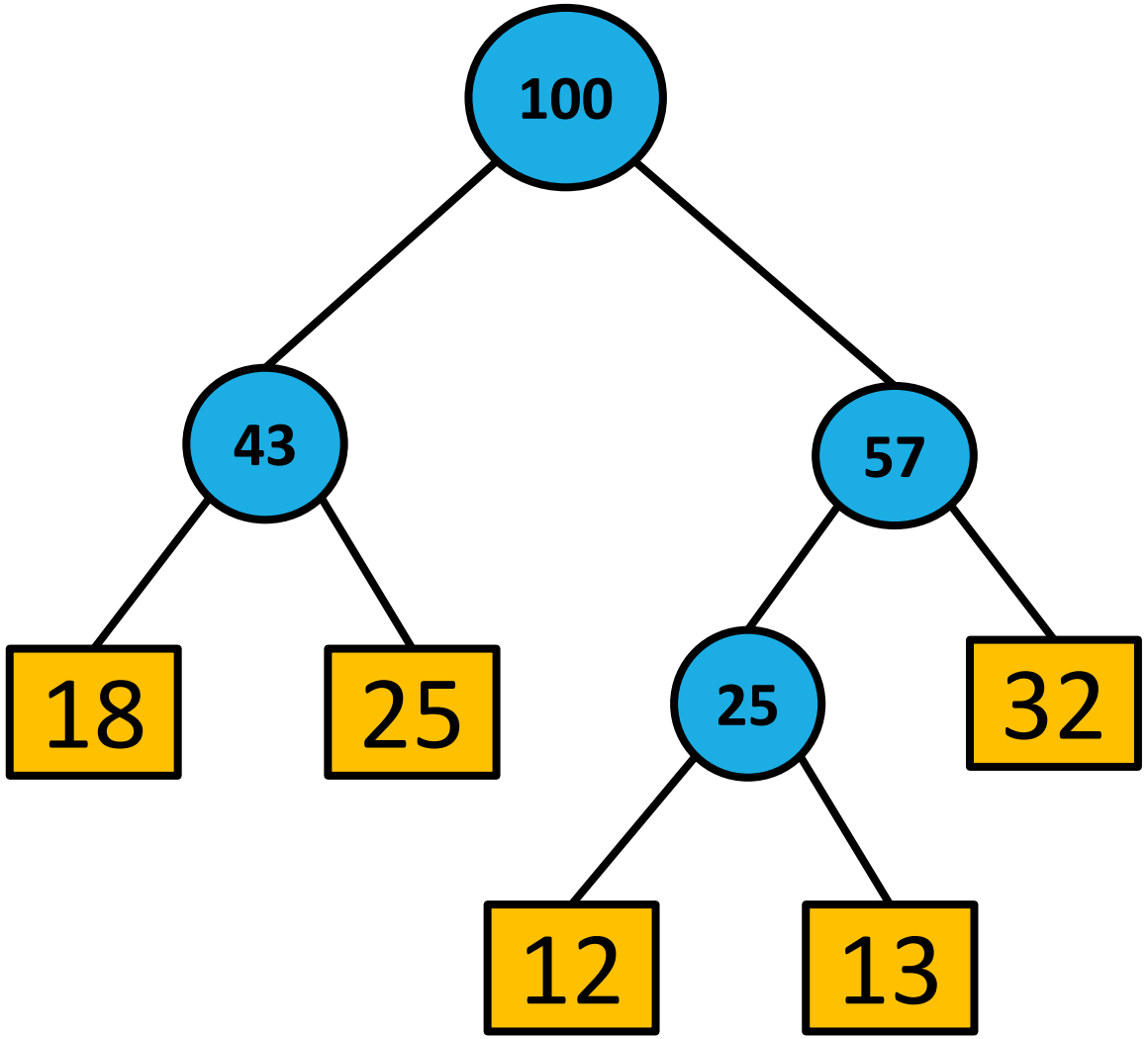


给定权值W=（18，25，13，12，32）

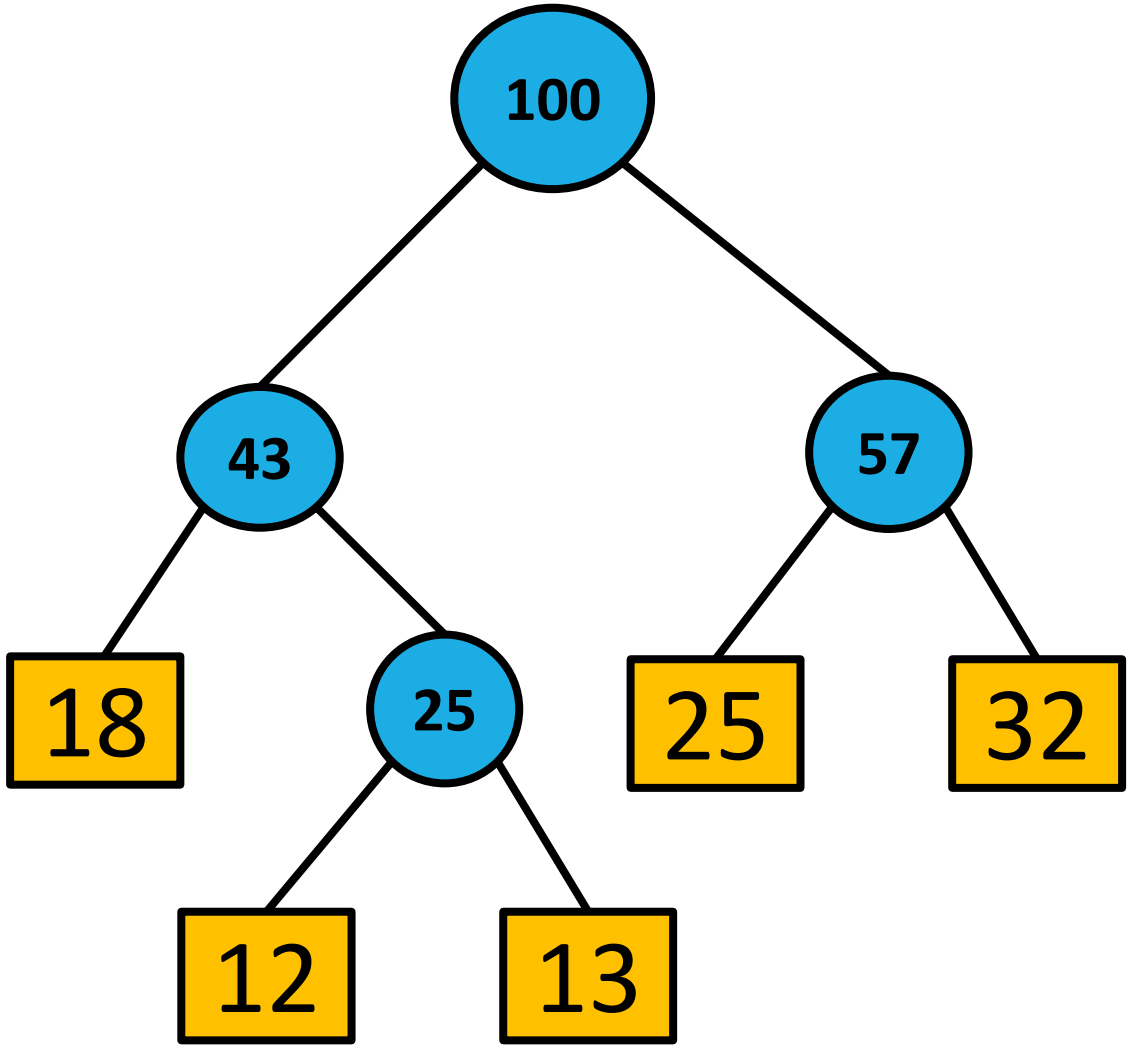




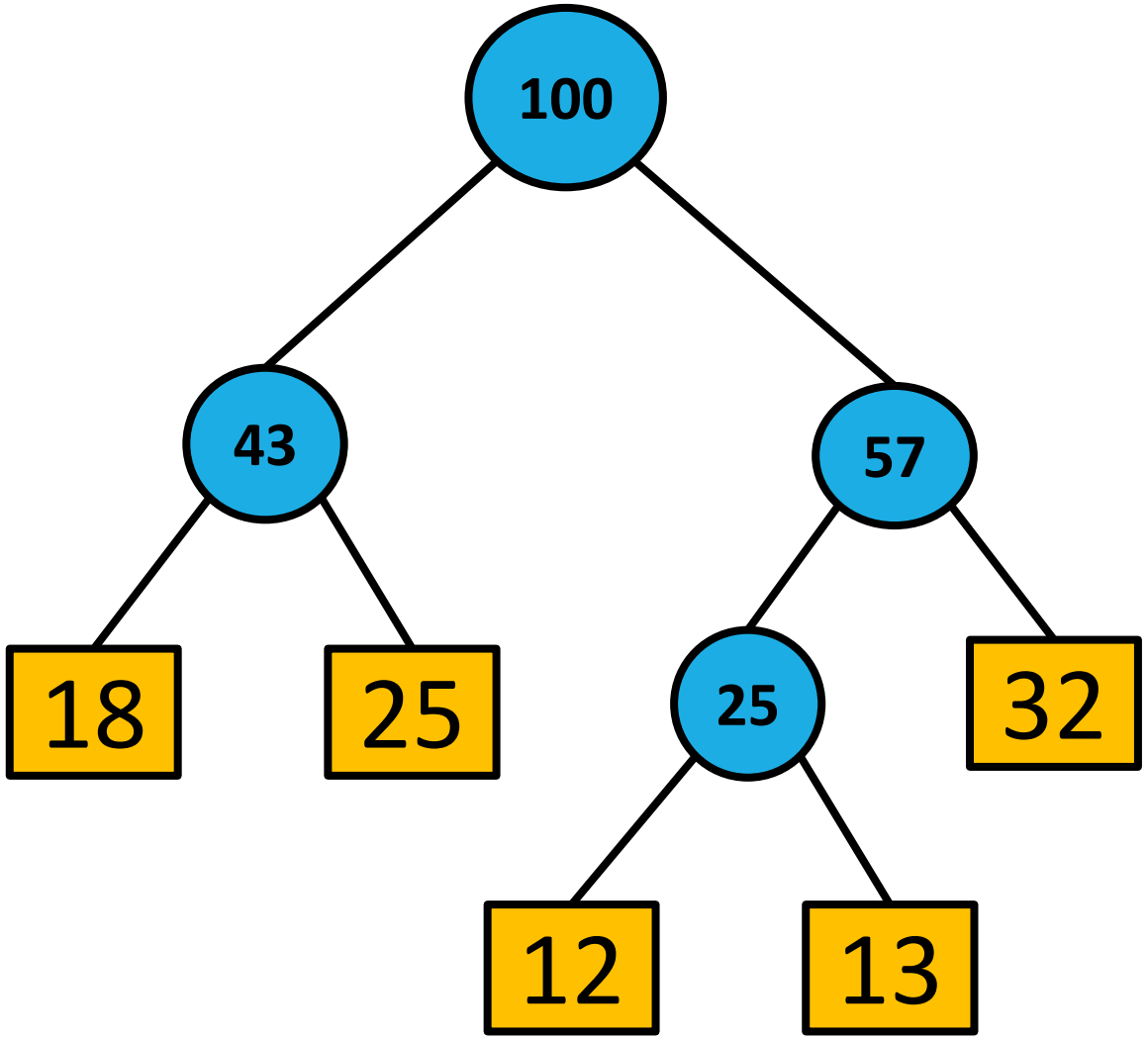
WPL= 225

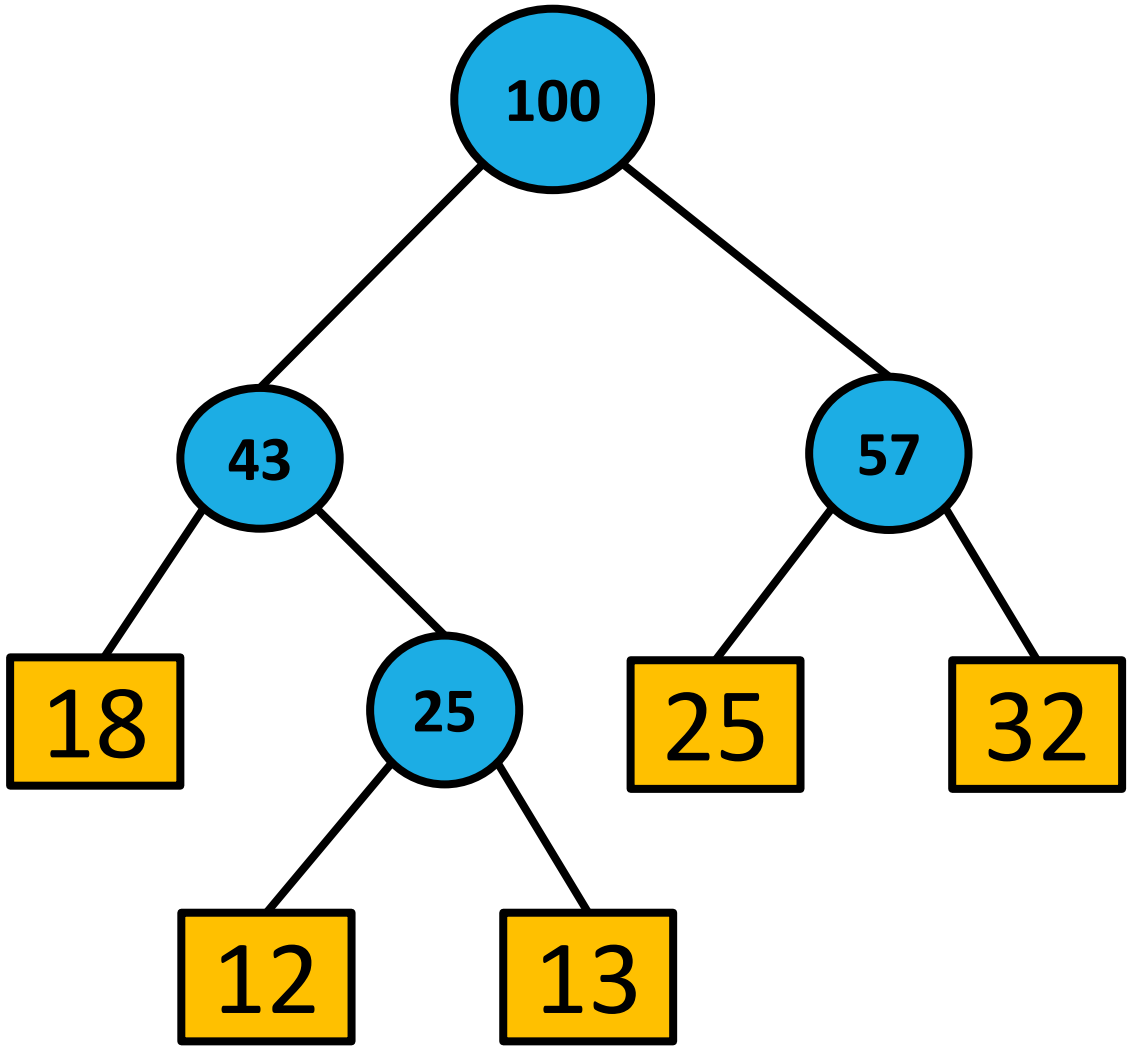


WPL= 225



互不等价的Huffman树

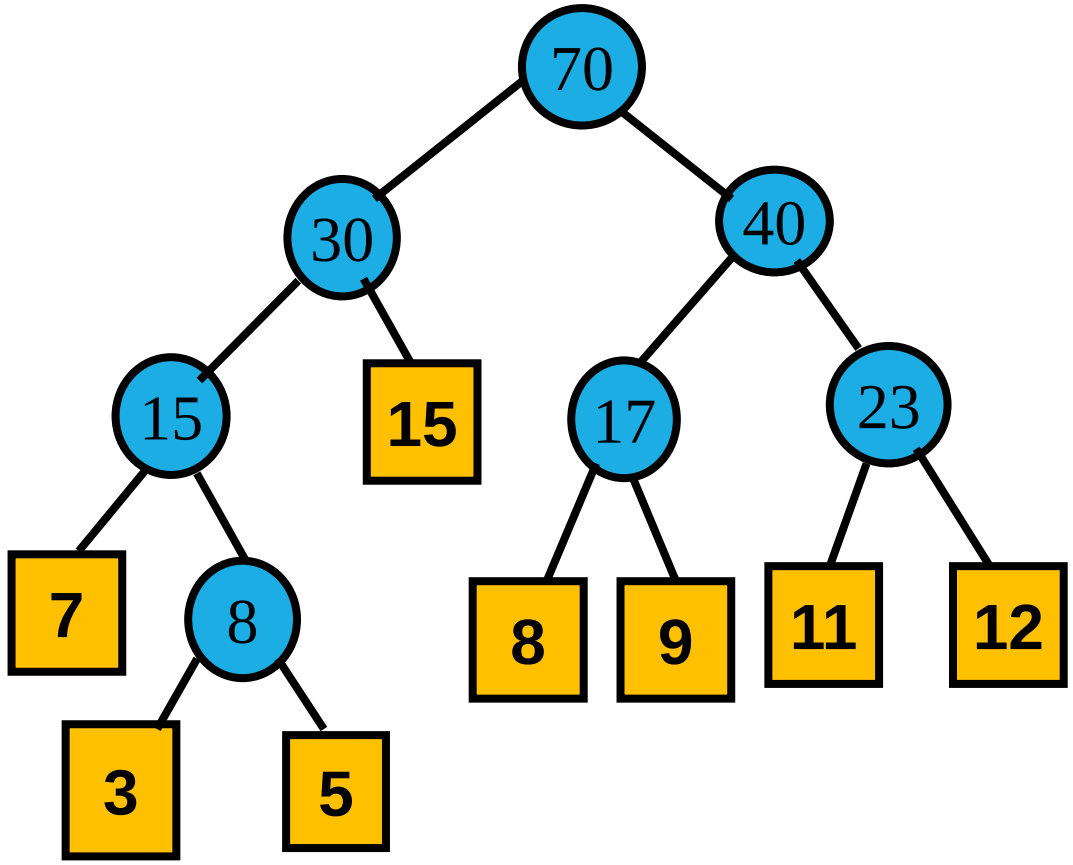


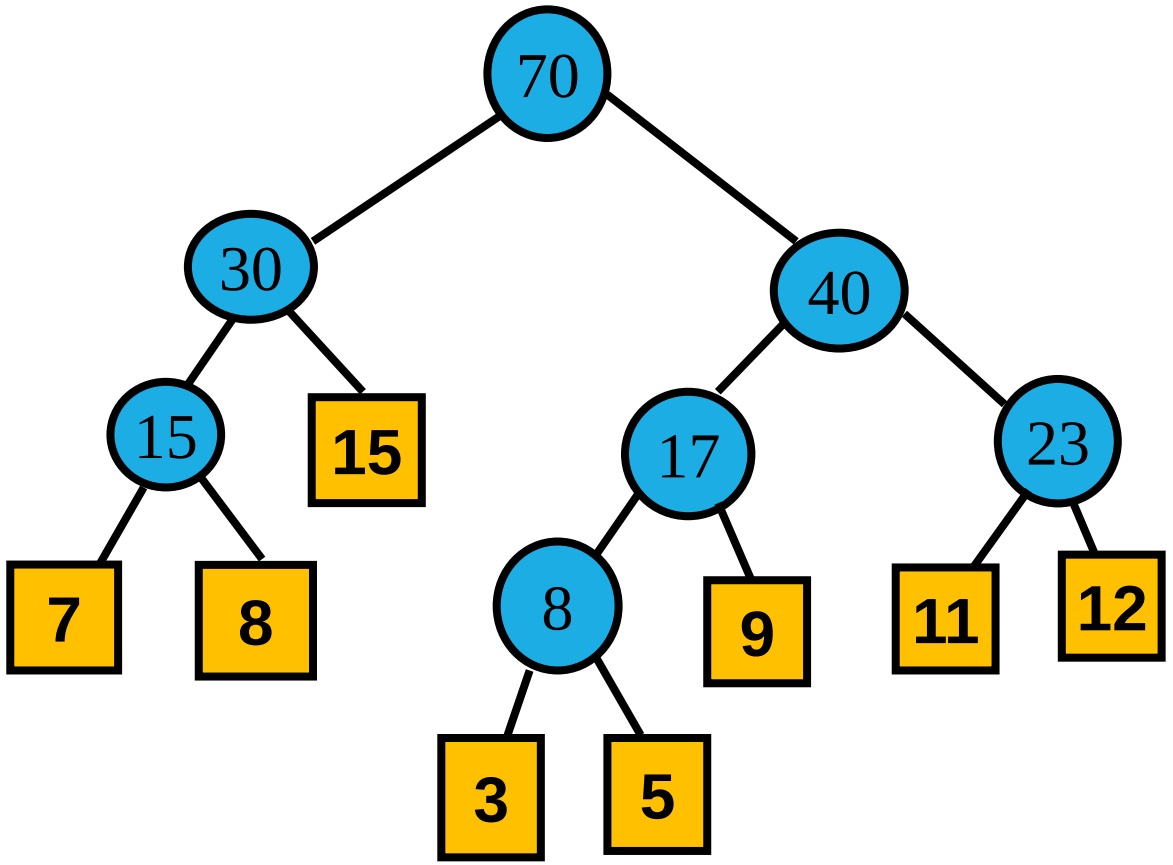


“互不等价”：其中一棵树不能经过交换某些结点的左右子树而得到另一棵树

WPL相等

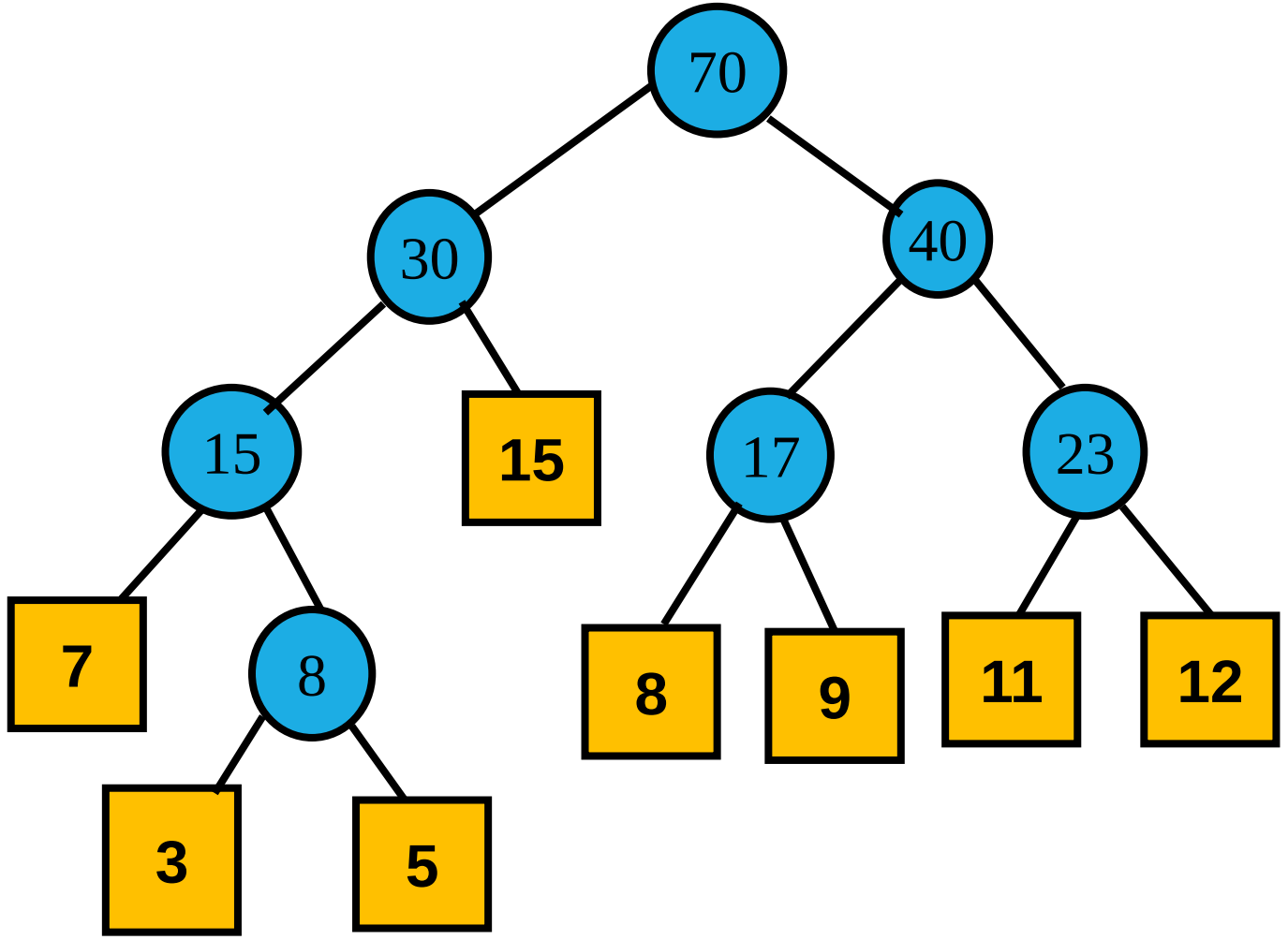
课堂练习：以数据集合{3,5,7,8,9,11,12,15}为权值构造Huffman，并求WPL





$$WPL=(3+5)*4+(7+8+9+11+12)*3+15*2=203$$

	ww	Parent	Lchild	Rchild
0	3	8	-1	-1
1	5	8	-1	-1
2	7	9	-1	-1
3	8	10	-1	-1
4	9	10	-1	-1
5	11	-1	-1	-1
6	12	-1	-1	-1
7	15	-1	-1	-1
8	8	9	0	1
9	15		2	8
10	17		3	4
11				
12				
13				
14				



哈夫曼算法的实现

哈夫曼树数据结构定义

```
1   struct HuffNode                  //定义哈夫曼树结点
2   {
3           int weight;                              //权值
4           int parent, leftchild, rightchild;  //父结点与左右孩子
5   };
6   typedef struct HuffNode *HtNode;
7   typedef struct HuffTreeNode   //定义哈夫曼树
8   {
9           int n;                 //哈夫曼树叶子结点个数
10          int root;               //哈夫曼树树根
11          HtNode  ht;            //指向哈夫曼树的指针
12  }*HuffTree;
```

算法4-21

```
1   HuffTree CreateHuffTree(int n, int *w)              //构造哈夫曼树
2   {
3           HuffTree pht;
4           int i, j, x1, x2, min1, min2;
5           pht = (HuffTree)malloc(sizeof(struct HuffTreeNode));
6           if (pht == NULL){
7                   printf("Out of space!!\n");
8                   return pht;
9           }
10          //为哈夫曼树申请2*N-1个空间
11          pht->ht = (HtNode)malloc(sizeof(struct HuffNode)*(2 * n - 1));
12          if (pht->ht == NULL){
13                  printf("Out of space!!\n");
14                  return pht;
15          }
16          //初始化哈夫曼树
17          for (i = 0; i < 2 * n - 1; i++)
18           {
19                  pht->ht[i].leftchild = -1;          //初始化叶结点左孩子
20                  pht->ht[i].rightchild = -1;         //初始化叶结点右孩子
21                  pht->ht[i].parent = -1;               //初始化叶结点的父亲
22                  if (i < n)
23                          pht->ht[i].weight = w[i];
24                  else
25                          pht->ht[i].weight = -1;
26          }
```

算法4-21

```
27      for (i = 0; i < n - 1; i++)
28          {
29                      min1 = MAX;                     //m1代表最小值
30                      min2 = MAX;                     //m2代表次小值
31/32                   x1 = -1;     x2 = -1;           //最小值下标   和   次小值下标
33                      //找到最小值下标x1并把最小值赋值给m1
34                      for (j = 0; j < n + i; j++)
35                      if (pht->ht[j].weight < min1&&pht->ht[j].parent == -1){
36                              min2 = min1;
37                              x2 = x1;
38                              min1 = pht->ht[j].weight;
39                              x1 = j;
40                       }
41                      //找到次小值下标x2并把次小值赋值给min2
42                      else if (pht->ht[j].weight < min2&&pht->ht[j].parent == -1)
43                       {
44/45                           min2 = pht->ht[j].weight;   x2 = j;
46                      }
47                      //构建x1，x2的父结点
48                      pht->ht[n + i].weight = min1 + min2;//父结点的权值为最小值加次小值
49                      pht->ht[n + i].leftchild = x1;      //父结点的左孩子为x1
50                      pht->ht[n + i].rightchild = x2;     //父结点的右孩子的x2
51                        pht->ht[x1].parent = n + i;       //x1父结点下标
52                      pht->ht[x2].parent = n + i;           //x2父结点下标
53              }
54              pht->root = 2 * n - 2;//哈夫曼树根结点位置
55/56           pht->n = n;   return pht;
57      }
```

哈夫曼树的应用：哈夫曼编码

等长编码

例如：D={A，B，C，D}
假设：A：00　　B：01　　C：10　　D：11
那么：信息　“ ABAC ”
　　编码　“ 00 01 00 10 ”　译码　“ABAC”

不等长编码

例如：D={A，B，C，D}

假设：A：0　　B：1　　C：10　　D：11

那么：信息　“ ABAC ”

　　编码　“ 0 1 0 10 ”

　　译码　“ABAC ”　　“ACC ”　　“ABABA”　等

哈夫曼树的应用：哈夫曼编码

最优前缀编码

- 编码长度最短
- 字符集中任一字符的编码都**不是**其它字符的编码的**前缀**

例如： D={A，B，C，D}

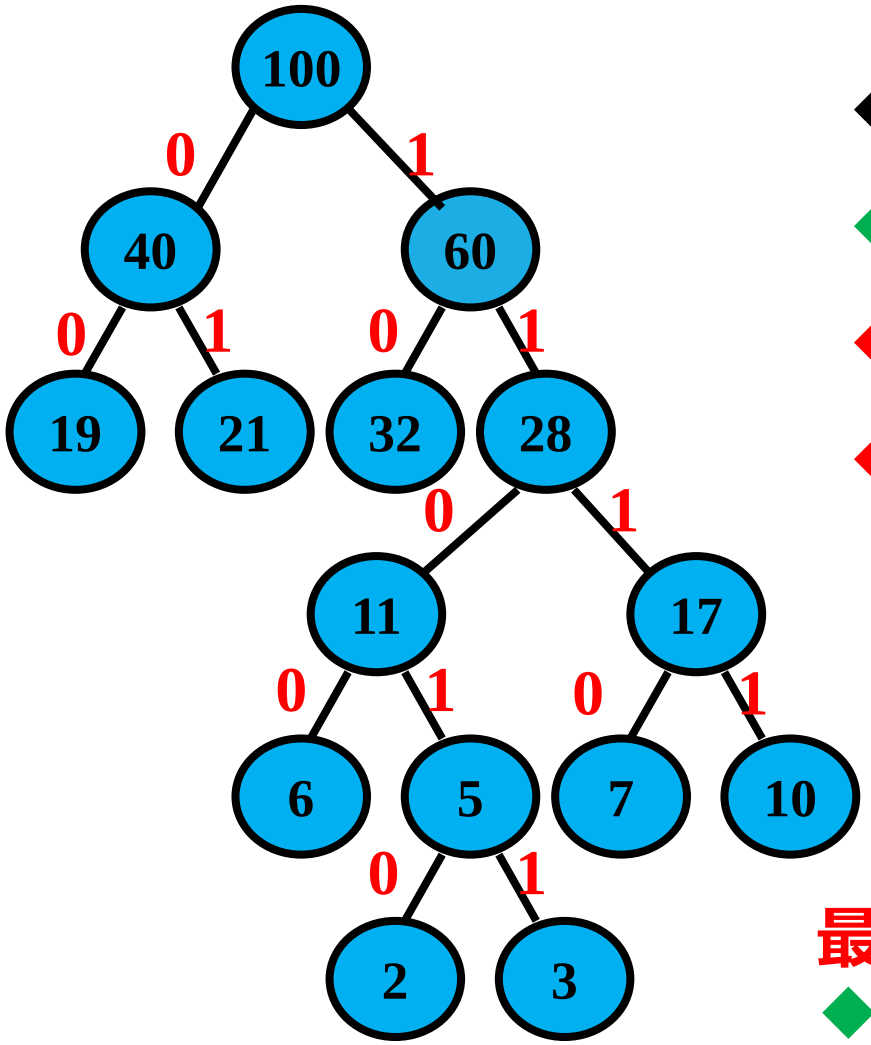
假设： A：0　　B：1　　C：10　　D：11

那么：信息　“ ABAC “

编码　“ 0 1 0 10 ”

译码　“ABAC ”　　“ACC ”　　“ABABA” 等

哈夫曼树的应用：哈夫曼编码



- 构建哈夫曼编码过程：
- 构造哈夫曼树
- 左分支表示字符 ‘0’，右分支表示字 ‘1’
- 从根结点到叶子结点的路径上分支字符组成的字符串作为该叶子结点字符的编码

最优前缀编码

- 编码长度最短
- 字符集中任一字符的编码都不是其它字符的编码的前缀