数据结构

深圳技术大学
大数据与互联网学院

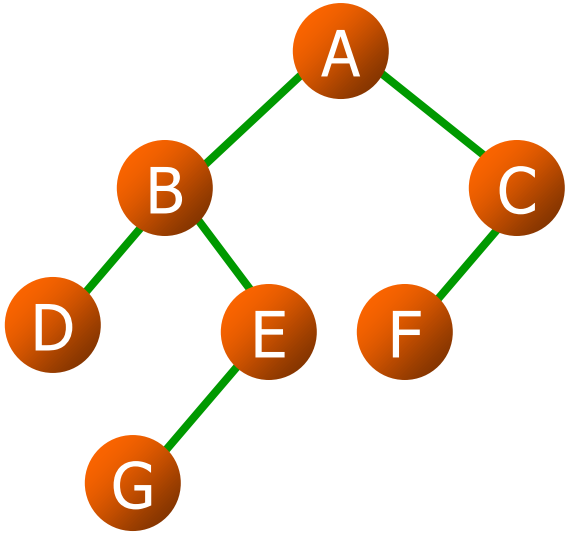
第六章 树和·二叉树

6.1　树的定义和基本术语
6.2　二叉树
6.3　遍历二叉树和线索二叉树
6.4 树和森林
6.5 树与等价问题
6.6 赫夫曼树及其应用
6.7 回溯法与树的遍历
6.8 树的计数

6.6 赫夫曼树

一. 最优二叉树

- 赫夫曼树又称为最优树，是一类带权路径长度最短的树
- 树的路径和路径长度
  - 路径：从树中一个结点到另一个结点之间的分支构成这两个结点之间的路径
  - 路径长度：路径上的分支数目
  - 树的路径长度：从树根到每个结点的路径长度之和



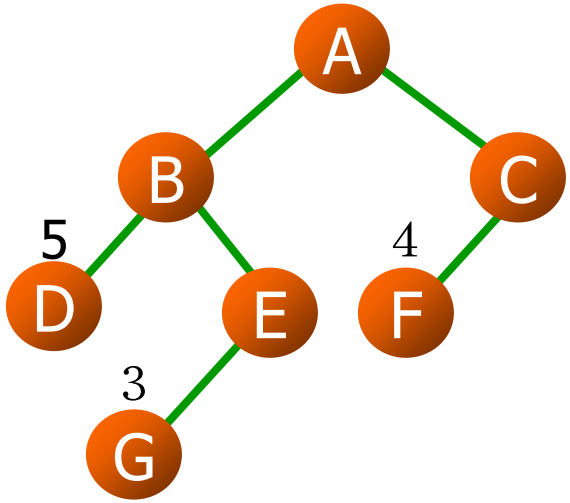
树路径长度为：2*1 + 3*2 + 1*3 = 11

6.6 赫夫曼树

一. 最优二叉树

- 带权路径长度：从结点到树根之间的路径长度与结点上权的乘积
- 树的带权路径长度(WPL)：树中所有叶子结点的带权路径长度之和

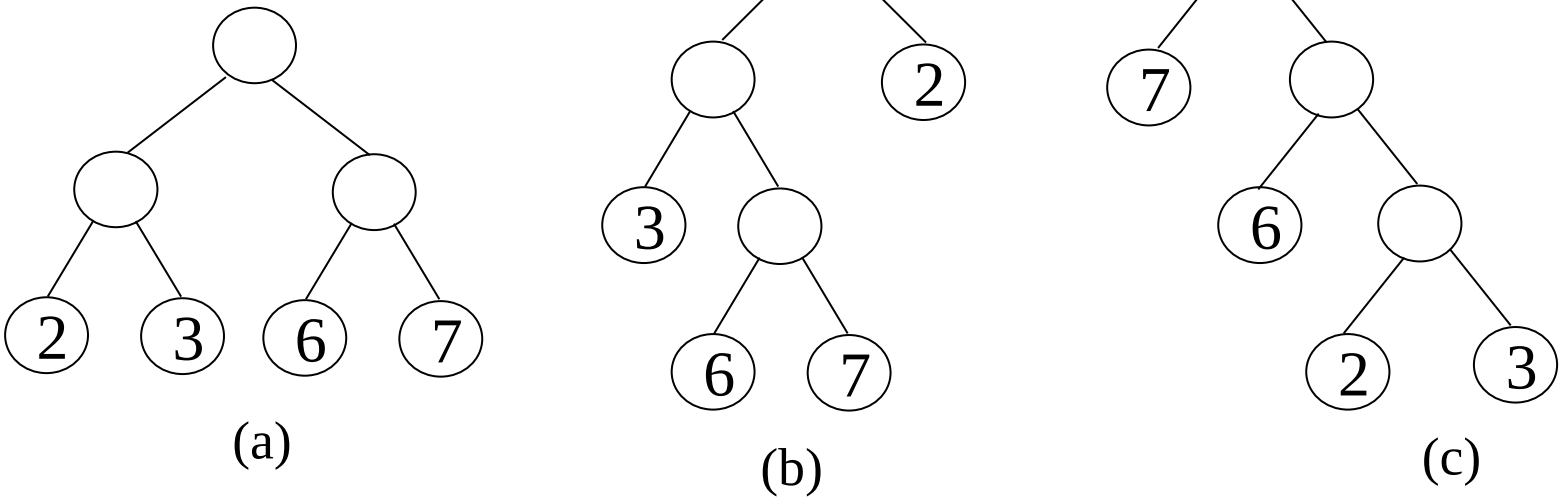
WPL = 2*5+3*3+2*4=27



6.6 赫夫曼树

一. 最优二叉树

- 具有n个叶子结点(每个结点的权值为w_i) 的二叉树不止一棵，但在所有的这些二叉树中，必定存在一棵WPL值最小的树，称这棵树为Huffman树(或称最优树)
- 如图是权值分别为2、3、6、7，具有4个叶子结点的二叉树



具有相同叶子结点，不同带权路径长度的二叉树

6.6 赫夫曼树

一. 最优二叉树

- 它们的带权路径长度分别为：

  (a) WPL=2×2+3×2+6×2+7×2=36；

  (b) WPL=2×1+3×2+6×3+7×3=47；

  (c) WPL=7×1+6×2+2×3+3×3=34。

- 其中(c)的 WPL值最小，称这棵树为最优二叉树或Huffman树

6.6 赫夫曼树

一. 最优二叉树

- 赫夫曼树的简单应用——百分制转五级制
  - 将0～100分转变为不及格、及格、中等、良好、优良五个级别

6.6 赫夫曼树

一. 最优二叉树

- 赫夫曼树的简单应用——百分制转五级制
  - 将0～100分转变为不及格、及格、中等、良好、优良五个级别
  - 分数的分布比例为：

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

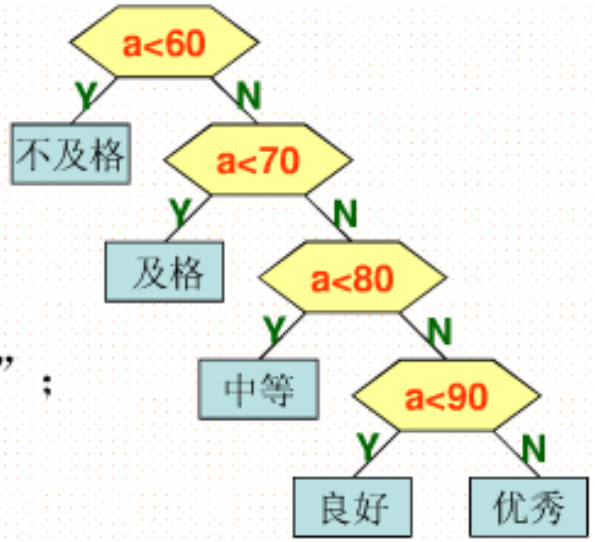
6.6 赫夫曼树

一. 最优二叉树

- 赫夫曼树的简单应用——百分制转五级制
  - 采用普通方法用四次判断语句
  - 若有10000个输入数据，根据分数分布比例需要31500次
  - 80%以上数据需要经过3次以上的比较才能得到结果

a是一个百分制分数

```
If (a<60)  b="不及格";
else   if (a<70) b="及格";
       else  if (a<80)  b="中等";
             else  if (a<90)  b="良好";
                   else b="优秀";
```

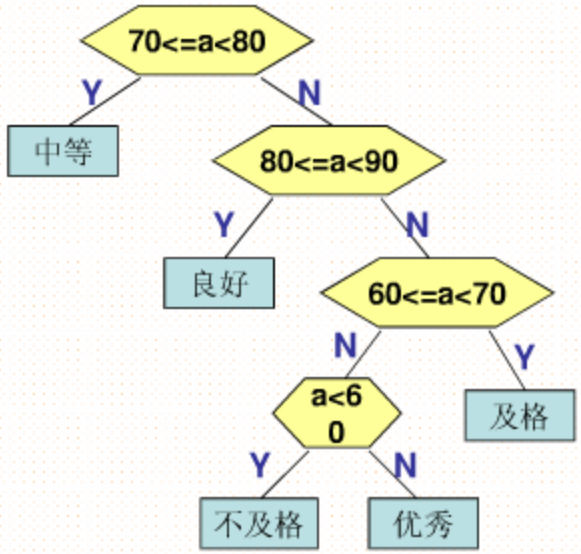


6.6 赫夫曼树

一. 最优二叉树

- 赫夫曼树的简单应用——百分制转五级制
  - 把高比例数据先做比较，程序调整如下，

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

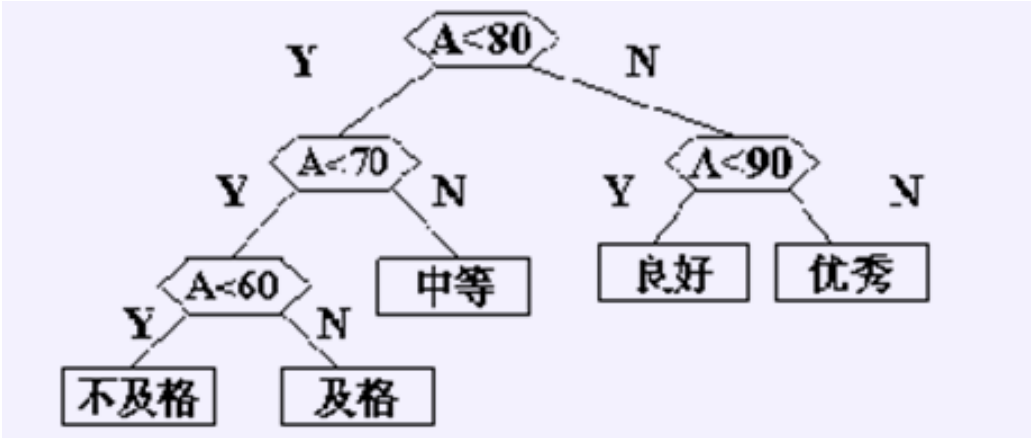


赫夫曼树

6.6 赫夫曼树

一. 最优二叉树

- 赫夫曼树的简单应用——百分制转五级制
  - 改进方法中每个比较有包含两次判断，再做细化，得到如下的判定树
  - 10000个输入数据需要22000次判断

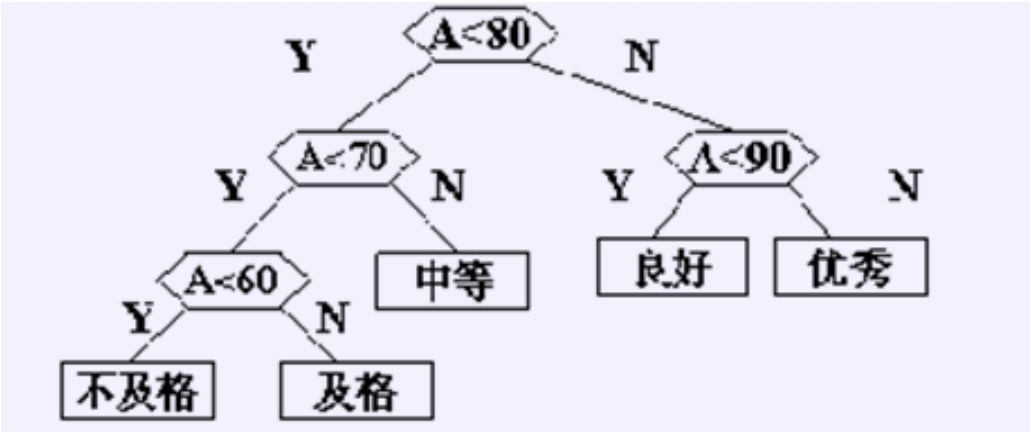


6.6 赫夫曼树

一. 最优二叉树

- 赫夫曼树的简单应用——百分制转五级制
  - 以5、15、40、30和10为权值，构造一颗有5个叶子的赫夫曼树，也就得到同样的这颗二叉树

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



6.6 赫夫曼树

二. 赫夫曼树的构造

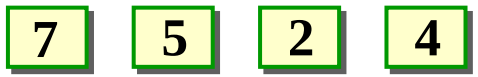
- 在Huffman树中，权值最大的结点离根最近，权值最小的结点离根最远
- 构造算法
  - ① 根据给定的n个权值$(w_1, w_2, \ldots, w_n)$构成n棵二叉树的集合$F=\{T_1, T_2, \ldots, T_n\}$，其中每棵二叉树$T_i$中只有一个带树为$T_i$的根结点
  - ② 在F中选取两棵根结点的权值最小的树作为左右子树构造一棵新的二叉树，且置其根结点的权值为其左右子树权值之和
  - ③ 在F中删除这两棵树，同时将新得到的二叉树加入F中
  - ④ 重复2，3，直到F只含一棵树为止

6.6 赫夫曼树

二. 赫夫曼树的构造

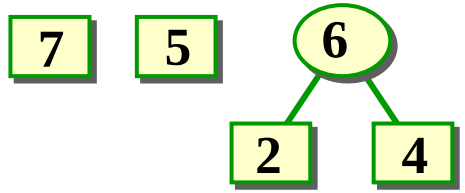
- 举例

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



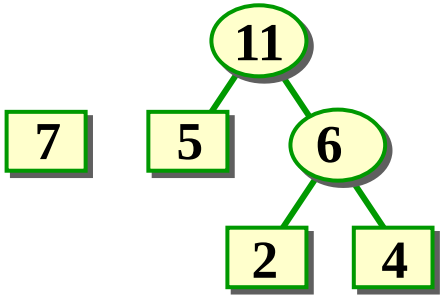
初始

F : {7} {5} {6}



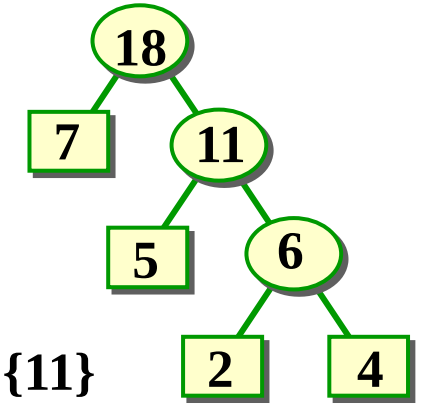
合并{2} {4}

F : {7} {11}



合并{5} {6}

F : {18}



合并{7} {11}

6.6 赫夫曼树

二. 赫夫曼树的构造

- 赫夫曼树的实现 （示例采用静态链表）

```
class HTNode{
    char data;
    unsigned int weight;
    unsigned int parent, lchild, rchild;
};
class HuffmanTree {
    HTNode* tree;  // 结点数组，保存整棵树
    int leaves_number;
    ... // 其他成员函数
}
```

6.6 赫夫曼树

二. 赫夫曼树的构造

- 赫夫曼树的构建

```
status HuffmanTree::HuffmanCreate(char c[], int w[], int n) {
  if (n<=1) return ERROR;
  int m = 2 * n - 1; // 最大需要结点数
  tree = new HTNode[m+1];  // 0号单元未用
  for (int i=1; i<=n; i++) { //初始化叶子结点
    tree[i].data = c[i-1]; tree[i].weight=w[i-1];
    tree[i].parent=0; tree[i].lchild=0; tree[i].rchild=0;
  }
  for (int i=n+1; i<=m; i++) { //初始化非叶子结点
    HT[i].weight=0;     HT[i].parent=0;
    HT[i].lchild=0;       HT[i].rchild=0;
  }
  for (i=n+1; i<=m; i++) {  // 建哈夫曼树
    // 在tree[1..i-1]中选择parent为0且weight最小的两个结点
    int s1, s2; Select(tree, i-1, s1, s2);
    tree[s1].parent = i;  tree[s2].parent = i;
    tree[i].lchild = s1;  tree[i].rchild = s2;
    tree[i].weight = tree[s1].weight + tree[s2].weight;
  }
  leaves_number = n; return OK;
```

6.6 赫夫曼树

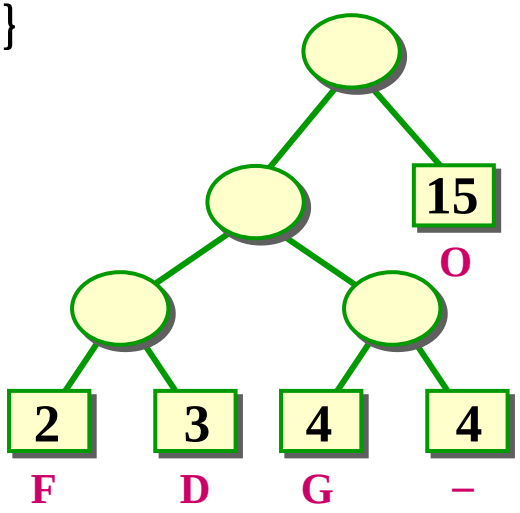
三. 赫夫曼编码

- 通过赫夫曼树把电文缩短，减少传送时间
- 编码前GOOD_G
  - 设给出一段报文： GOOD_GOOD_GOOD_GOOOOOOO_OFF
  - 字符集合是｛O, G, _, D, F｝，各个字符出现的频度(次数)是W＝｛15, 4, 4, 3, 2｝。
  - 若给每个字符以等长编码
  - O: 000 G: 001 _: 010 D: 011 F: 100
  - 则总编码长度为（15+4+4+3+2）＊ 3 = 84.

6.6 赫夫曼树

三. 赫夫曼编码

- 编码后
  - 若按各个字符出现的概率不同而给予不等长编码，可望减少总编码长度。
  - 各字符{ O, G, _, D, F }出现概率为{ 15/28, 4/28, 4/28, 3/28, 2/28 }，化整为 { 15, 4, 4, 3, 2 }
  - 各字符出现概率[取整数]为{15, 4, 4, 3, 2}
  - 如果规定，Huffman树的左子树小于右子树，则可构成右图所示Huffman树



6.6 赫夫曼树

三. 赫夫曼编码

- 编码后
  - 令左孩子分支为编码 ‘0’，右孩子分支为编码 ‘1’
  - 将根结点到叶子结点路径上的分支编码，组合起来，作为该字符的Huffman码，则可得到：

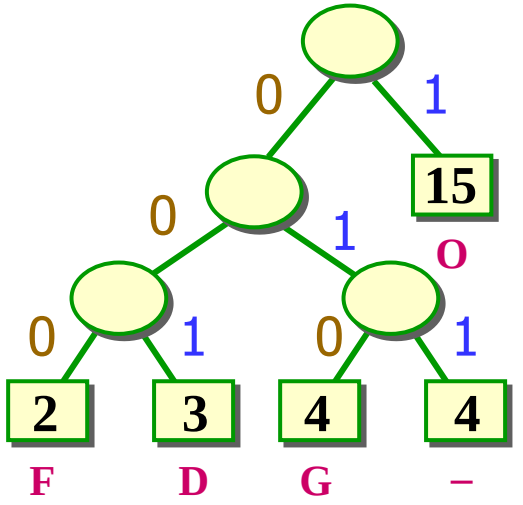
O:1

_:011

G:010

D:001

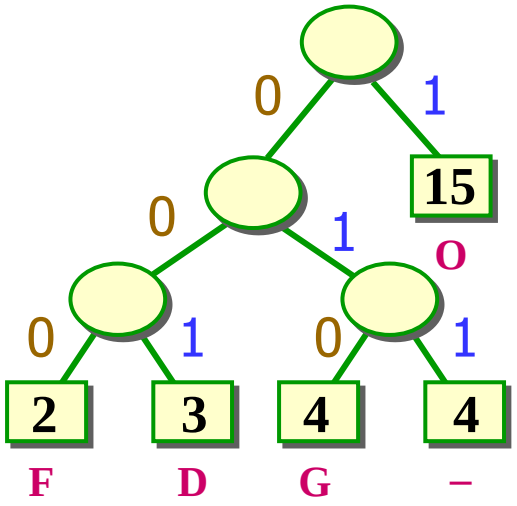
F:000



6.6 赫夫曼树

三. 赫夫曼编码

- 则总编码长度为 15*1+(2+3+4+4)*3 = 54 < 84
- Huffman是一种前缀编码，解码时不会混淆，如GOOD编码为：01011001



6.6 赫夫曼树

三. 赫夫曼编码

- 赫夫曼树的编码

```
Status HuffmanTree::HuffmanCoding(Map<char, string> &HC) {
  //--- 从叶子到根逆向求每个字符的哈夫曼编码 ---
  string code;
  for (i=1; i<=leaves_number; ++i) {    // 逐个字符求哈夫曼编码
    char c = tree[i].data; // 保存一下当前字符
    for (j=i, f=tree[i].parent; f!=0; j=f, f=tree[f].parent)
      // 从叶子到根逆向求编码
      if (tree[f].lchild==j) code = "0" + code;
      else code = "1" + code;
    HC[c] = code;
  }
}
```

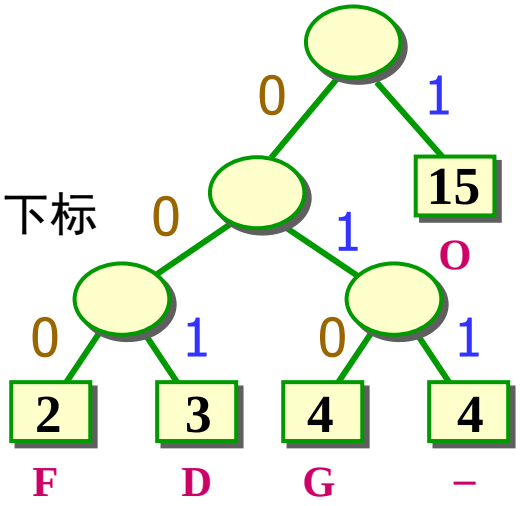
6.6 赫夫曼树

三. 赫夫曼解码

- 算法流程
  - 定义指针p指向赫夫曼树结点，实际是记录结点数组的下标
  - 定义指针i指向编码串，定义ch逐个取编码串的字符
  - 初始化：读入编码串，设置p指向根结点，i为0
  - 执行以下循环：

1. 取编码串的第i个字符放入ch
2. 如果ch是字符0，表示往左孩子移动，则p跳转到左孩子
3. 如果ch是字符1，表示往右孩子移动，则p跳转到右孩子
4. 如果ch非0非1，表示编码串有错误，输出error表示解码失败
5. 检查p指向的结点是否为叶子
   1. 如果是叶子，输出解码，p跳回根节点
   2. 如果不是叶子，设置ch为3
6. 继续循环，一直到编码串末尾

- 循环执行完后，如果ch值为3，输出解码失败，否则成功结束



6.7 回溯法与树遍历

一. 方法

- 树的遍历序列不是预先就建立好的，而是通过函数递归，在遍历过程中生成，因此树的遍历包含了**回溯**的思想
- 许多问题的递归过程实质与树的遍历是相同的，问题求解过程中的状态等于树遍历的中间结点。问题的结果等于树遍历的叶子
- 因此可以把回溯法与树遍历结合起来。

6.7 回溯法与树遍历

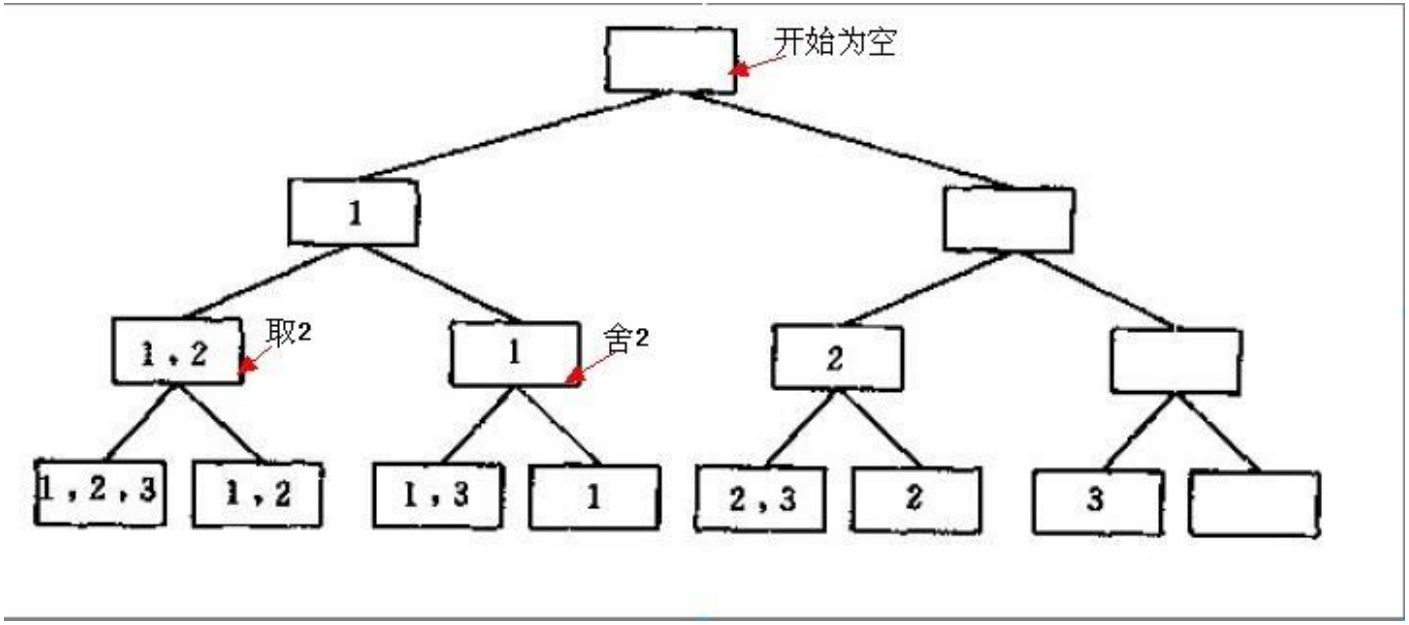
二. 求N个元素集合的幂集

- 集合的幂集是指由集合的所有子集组成的集合
- 例如A = {1, 2, 3}
- 则p(A)={{1, 2, 3} {1, 2} {1, 3} {2, 3} {1} {2} {3} {}}
- 分析：子集的求解实质就是对集合A的元素进行取或舍的过程，因为是取舍两种操作，实质就是二叉树的遍历过程

6.7 回溯法与树遍历

二. 求N个元素集合的幂集

- 用回溯法求解过程就是树的遍历



6.7 回溯法与树遍历

二. 求N个元素集合的幂集

- 程序代码

```
void GetPowerSet(int i, List A, List &B) {
    // 线性表A表示集合A，线性表B表示幂集ρ(A)的一个元素。
    // 局部量k为进入函数时表B的当前长度。
    // 第一次调用本函数时，B为空表，i＝1。
    ElemType x;
    int k;
    if (i > ListLength(A)) Output(B); //B是ρ(A)的一个元素
    else {
        GetElem(A, i, x);          k = ListLength(B);
        ListInsert(B, k+1, x);     GetPowerSet(i+1, A, B);
        ListDelete(B, k+1, x);     GetPowerSet(i+1, A, B);
    }
} // GetPowerSet
```

6.7 回溯法与树遍历

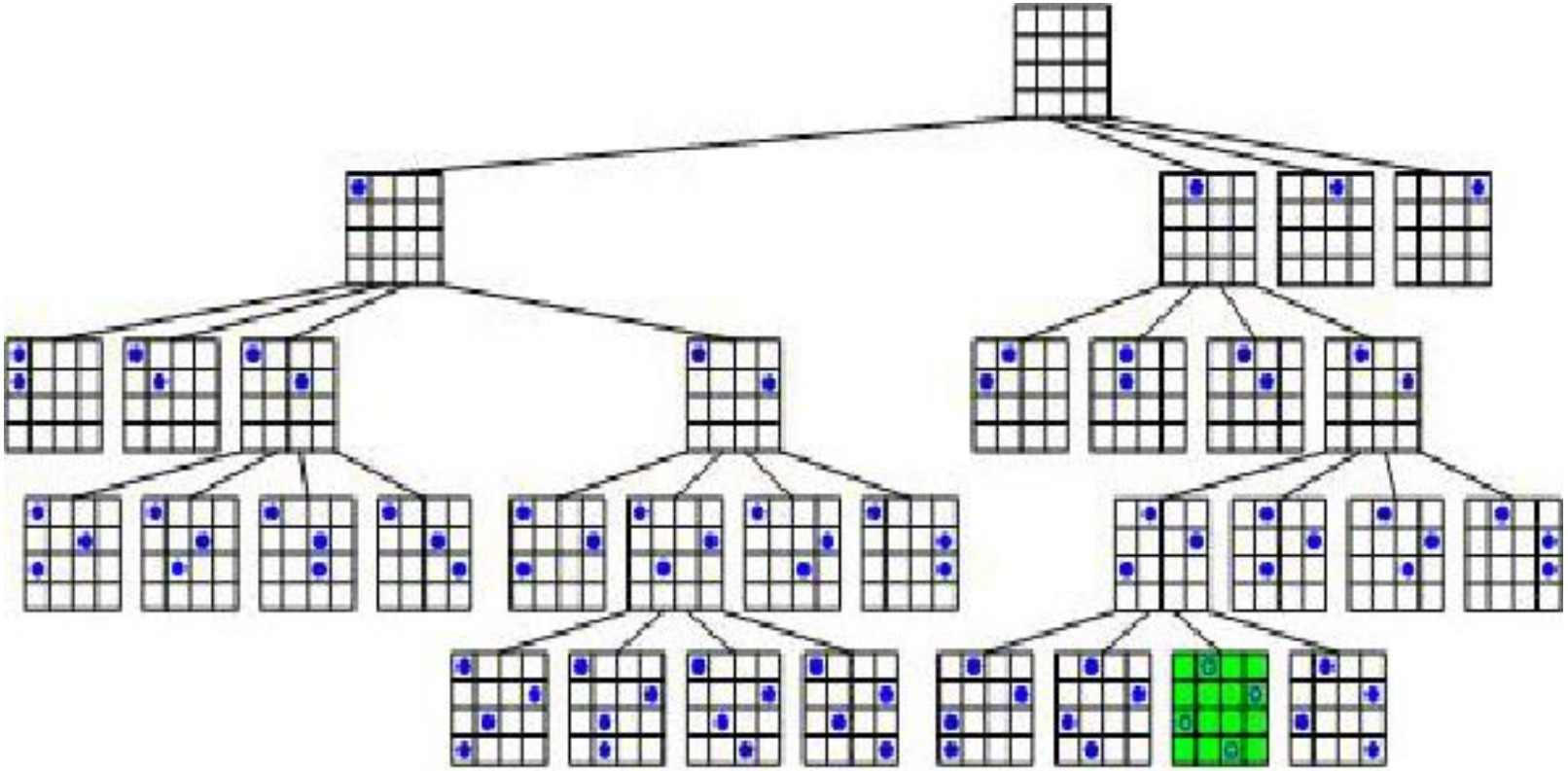
三. 四皇后问题

- 求四皇后的所有合法布局，合法规则：
  - 棋盘为4×4格，即4行4列
  - 共放4个棋子
  - 要求任意两个棋子不能在同一线上，即不能同行、同列、同对角线
- 每放1个棋子就会产生四种状态，所以棋盘的布局过程就是四叉树的遍历过程。在约束条件下，树的一些叶子是非法的

6.7 回溯法与树遍历

三. 四皇后问题

- 求解过程即四叉树遍历



6.7 回溯法与树遍历

三. 四皇后问题

- 遍历分析：当遍历到一个节点时，判断该节点是否已得到一个完整的布局，若是，输出该布局，若不是，依次先根遍历满足约束条件的各棵子树，首先判断该节点布局是否合法，若合法先根遍历其子节点，否则剪去该子树的分支

6.7 回溯法与树遍历

三. 四皇后问题

- 程序流程描述：

```
Void Trial(int i, int n)
{     if (i>n)  output(matrix);
      else for (j=1; j<=n; ++j)
      {  set a chessman in column j (also in row i);
         if (judgelegal) Trial(i+1, n);
         move the chessman out column j;
      }
}
```

6.7 回溯法与树遍历

三. 四皇后问题

- 判断合法的算法描述：
  - 同一行不需要检查，因为算法是每行只放一个棋子，不会出现同一行有2个棋子
  - 当发现第i行第j列有棋子，做以下检查：
    - 检查同一列，如果有超过两个1则非法
    - 检查对角线，分四个方向检查
      - 行列都递减方向检查
      - 行列都递增方向检查
      - 行递增列递减方向检查
      - 往行递减列递增方向检查