

# 数据结构

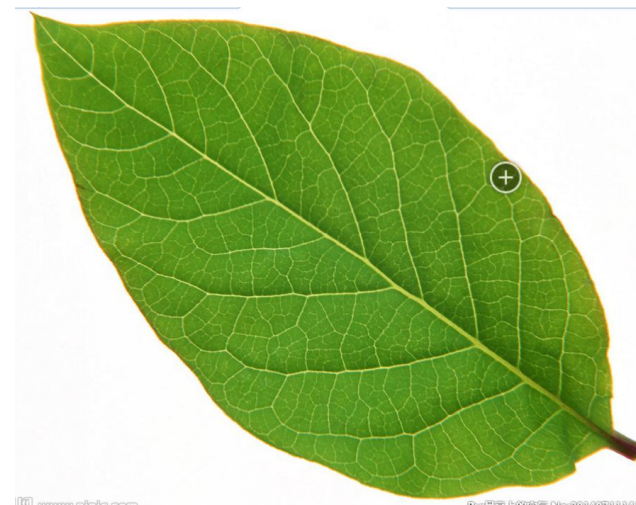
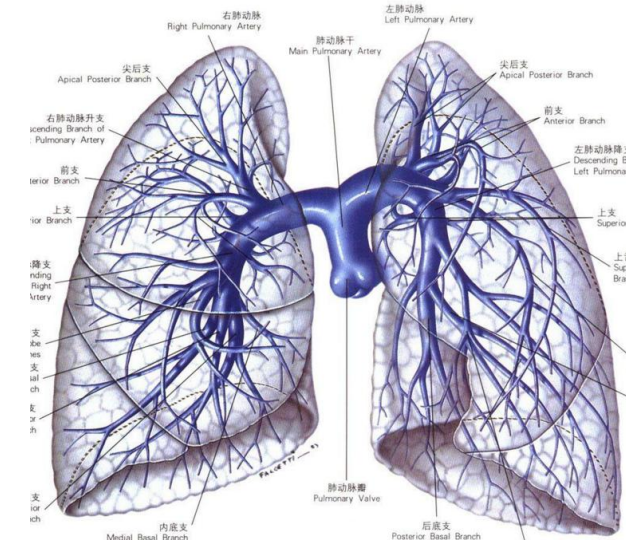
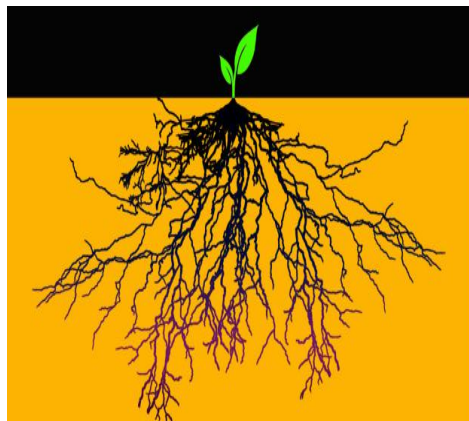
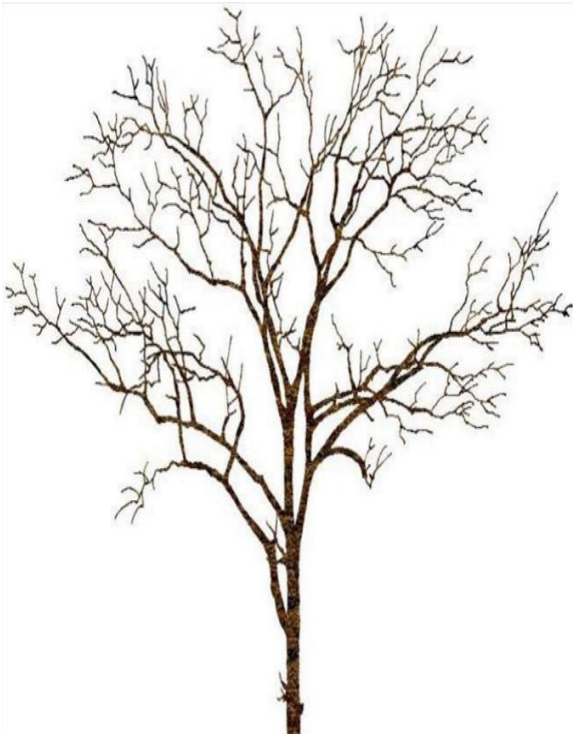
# Data Structure

2017年秋季学期

刘鹏远

助教：韩越，卢梦依

树



# Huffman树及Huffman编码

## 编码及传输问题

数据传输中，需要将数据中出现的每个字符进行二进制编码。在设计编码时需要遵守两个原则：

（1）发送方传输的二进制编码，到接收方解码后必须具有**唯一性**，即解码结果与发送方发送的电文完全一样；

（2）发送的二进制编码**尽可能地短**。

良好的编码可以取得更短的数据长度。

**（数据压缩也类似）**

# 编码及传输问题

**等长编码**，即每个编码所含的二进制位数相同。如ASCII码。

**等长示例：**对“ABACCDA”，可以将A,B,C,D的编码分别为00,01,10,11。则上述7个字符的长度为14位。

**不等长编码**，即每个编码所含二进制位数不一定相同。

大家觉得应该如何编码呢？使总长尽可能短

**示例：**令上例中A,B,C,D的编码分别为0,00,1,01，则上述长度可以减少为9个字符

为了减小传输长度，可以扫描数据一遍，利用自身结构信息，计算各个字符出现的频率，编码使出现频率最多的编码最短。

**编码优化目标：**使出现频次高的字符的编码尽可能的短

问题：编码后，字符的传输是连续的。

**编码还要使不同的字符编码能在连续出现的时候被正确解码。**

ABACCDA      A,B,C,D---->0,00,1,01

000011010      AAAACCACA?    BBCCDA?      ...?

上述示例中的数据无法在接收端正确翻译。怎么破??

**已知：** 每一个字符及对应的自身频次

**优化目标：**

使频次越高的字符，编码长度越短

（后果：频次越低的，编码长度可能会越长，因为编码的唯一性，使短的编码被频次高的占用）

**突破：** 设计每个字符的编码，为某个点走向每个字符的通路，通路越短，编码越短（对应高频）。

# Huffman及Huffman coding

1951年，哈夫曼和他在MIT信息论的同学需要选择是完成学期报告还是期末考试。导师Robert M. Fano给他们的学期报告的题目是，寻找最有效的二进制编码。

由于无法证明哪个已有编码是最有效的，哈夫曼放弃对已有编码的研究，转向新的探索，最终发现了基于有序频率二叉树编码的想法，并很快证明了这个方法是最有效的。

由于这个算法，学生青出于蓝。



# David Albert Huffman

**David Albert Huffman (August 9, 1925 – October 7, 1999) was a pioneer in computer science, known for his Huffman coding.**

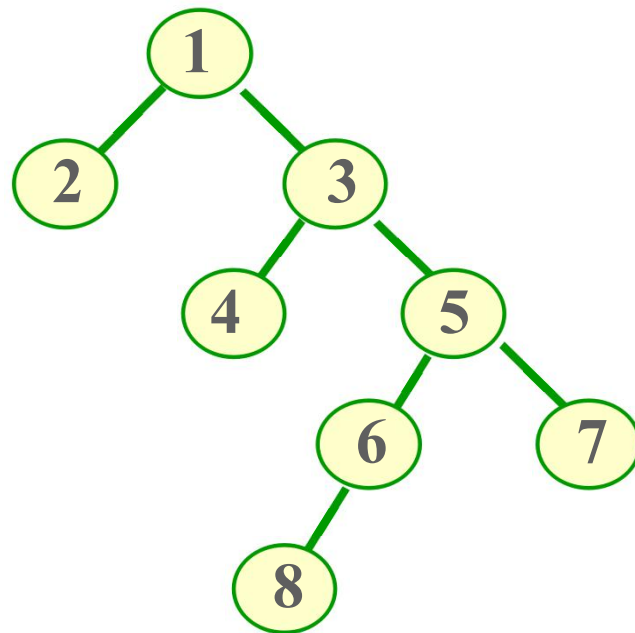
**He was also one of the pioneers in the field of mathematical origami.**

**David Huffman died at the age of 74, ten months after being diagnosed with cancer.**

# 几个基本概念

## 1. 路径和长度 (Path Length)

- 路径为一个节点到另一个节点之间的通路
- 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。右侧树的PL为？

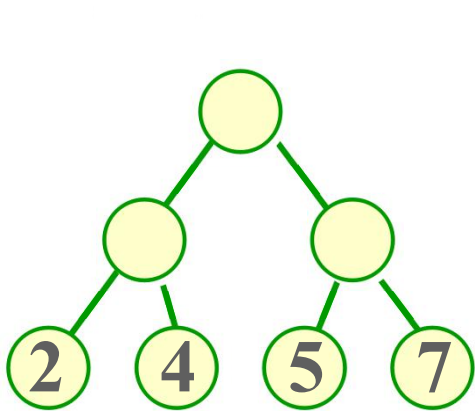


## 2. 树的带权路径长度 (Weighted Path Length of Tree, WPL)

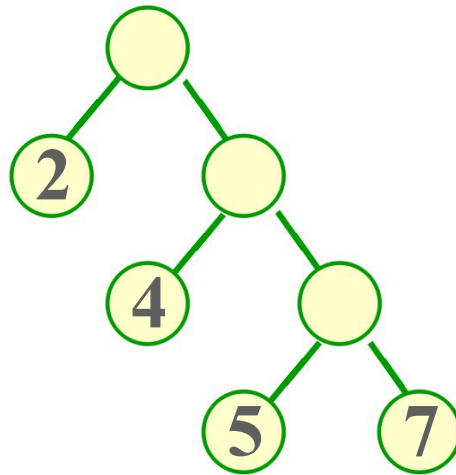
节点的**带权路径长度**为该节点权值与该节点的到根节点间的路径长度之积。**树的WPL为所有叶节点的WPL的和：**

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

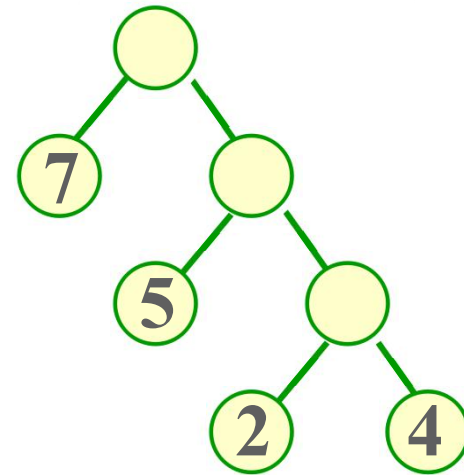
其中， $l_i$  是第  $i$  结点的路径长度， $w_i$  是相应结点的的权值。



**$WPL = ?$**

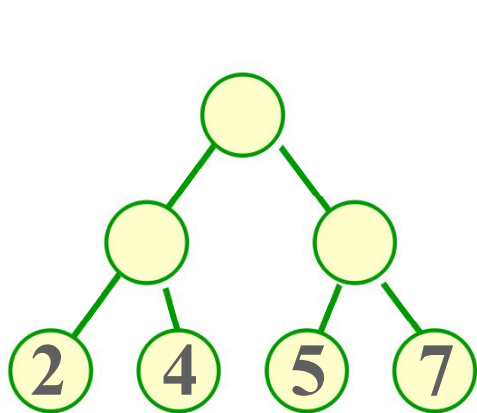


**$WPL = ?$**

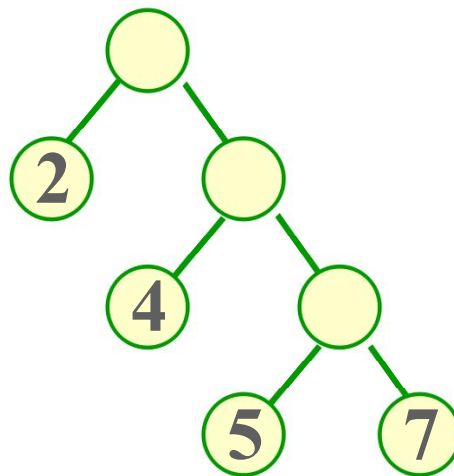


**$WPL = ?$**

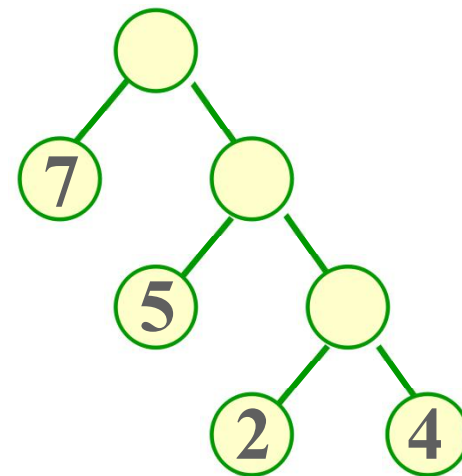
节点内的  
数字  
为权值



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



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



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

- 给定叶节点个数及权值，构造二叉树，其中带权路径长度最小的二叉树被称为**Huffman树**，也即**最优二叉树**。
- 如何构造最优二叉树？(两步)

# Huffman树的构造算法

(0)构造 具有  $n$  棵二叉树的森林  $F = \{ T_0, T_1, T_2, \dots, T_{n-1} \}$ , 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。  $n$  个权值为:  $\{w_0, w_1, w_2, \dots, w_{n-1}\}$

(1)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  $F$ 。

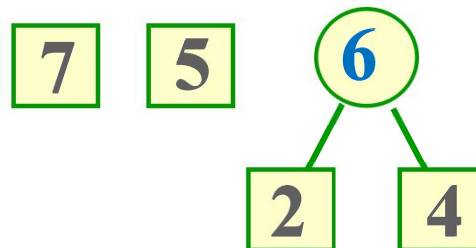
F : {7} {5} {2} {4}



初始

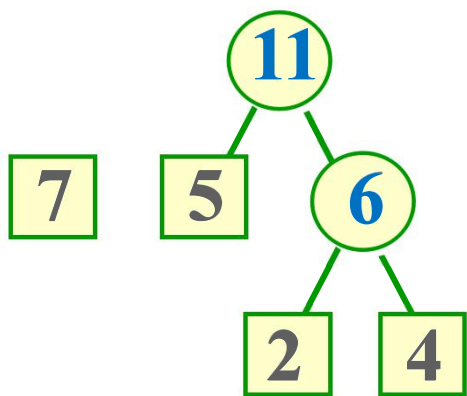


F : {7} {5} {6}



合并{2} {4}

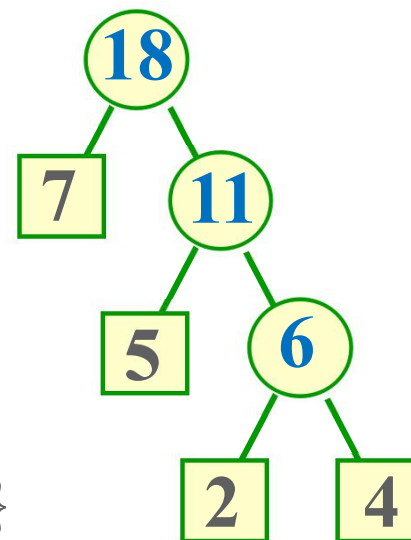
F : {7} {11}



合并{5} {6}



F : {18}



合并{7} {11}

## Huffman树的构造 (1)

自然可想：利用二叉链表。可先将节点的ElemType改为：

`typedef struct {int weight; char ch;} ElemType;` 然后呢？

霍夫曼树的生成过程与符号树的生成是不是有些类似？

区别：符号树是从栈中顺序选取节点，霍夫曼需要每次挑选权值最小的节点。

有一种结构，每次选取的元素按照权值决定，这结构是？

优先队列

基于优先队列的huffman树构造

假设已经实现了优先队列PriorityQueue

```
typedef struct {int weight; char ch;} ElemType;
```

```
typedef struct Node {
```

```
    ElemType data;
```

```
    struct Node *LeftChild, *RightChild;
```

```
} BTNode, *HuffmanTree;
```



```
CreatHuffmanTree (HuffmanTree *T, ElemType w[], int n) { //伪码
    PriorityQueue PQ; HuffmanTree p, q, r, cw[n];
    for(i=0;i<n;i++) {
        p = (BTNode*)malloc(sizeof(BTNode));
        p->data=w[i].data;    //w为字符及权值
        p->LChild=NULL; p->RChild=NULL;
        cw[i]=p; //生成节点p，放入到数组cw中，作为H的输入
    }
    CreatPriorityQueue (&PQ, cw[], n); //创建优先队列
    //初始化完毕，生成了n棵树的森林，在优先队列中
```

```
while(!IsEmpty(PQ)) {  
    DeQueue(&H, &p); QeQueue(&H, &q);  
    r = (BTNode)malloc(sizeof(BTNode));  
    r->data.weight = p->data.weight + q->data.weight;  
    r->LChild=p; r->RChild = q;  
    if(IsEmpty(PQ)) {  
        *T = r;    return;  
    }  
    EnQueue(&PQ, r);  
}  
}
```

## Huffman树的构造（2）

优先队列如何实现？

如不用优先队列，则如何进行构造？

利用一维数组对静态链表的简化版实现来进行  
借鉴带双亲的孩子表示法。

双亲、左孩、右孩，权值信息存入一维数组。

（与教材动态分配数组方法有不同）

# 采用一维数组（静态链表）的Huffman树

```
#define N 100
```

```
typedef struct{
```

```
    int weight;
```

```
    int parent;//char c;节点内容
```

```
    int LChild, RChild;
```

```
}HTNode;
```

```
//索引=-1表示NULL
```

```
先建立一个HT包含所有节点
```

```
typedef struct{
```

```
    HTNode F[2*N-1];
```

```
    int size;//或root根
```

```
}HuffmanTree;
```

为什么 $2*N-1$ ?

因为生成 $N-1$ 个非叶节点。

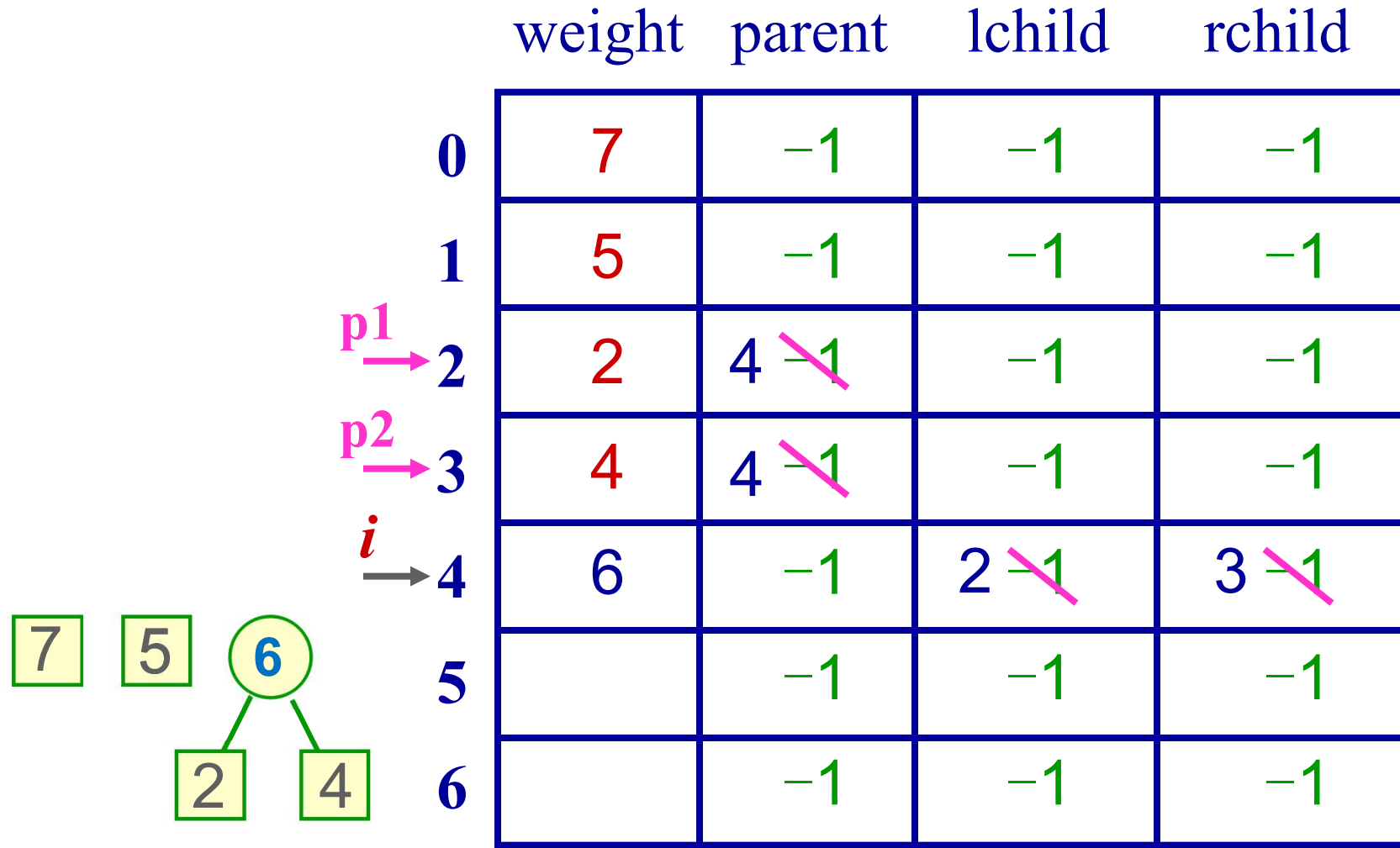
无度为1的节点(严格二叉)

7 5 2 4

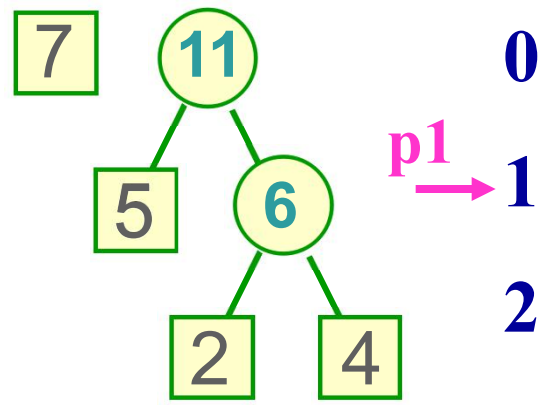
-1表示空

	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

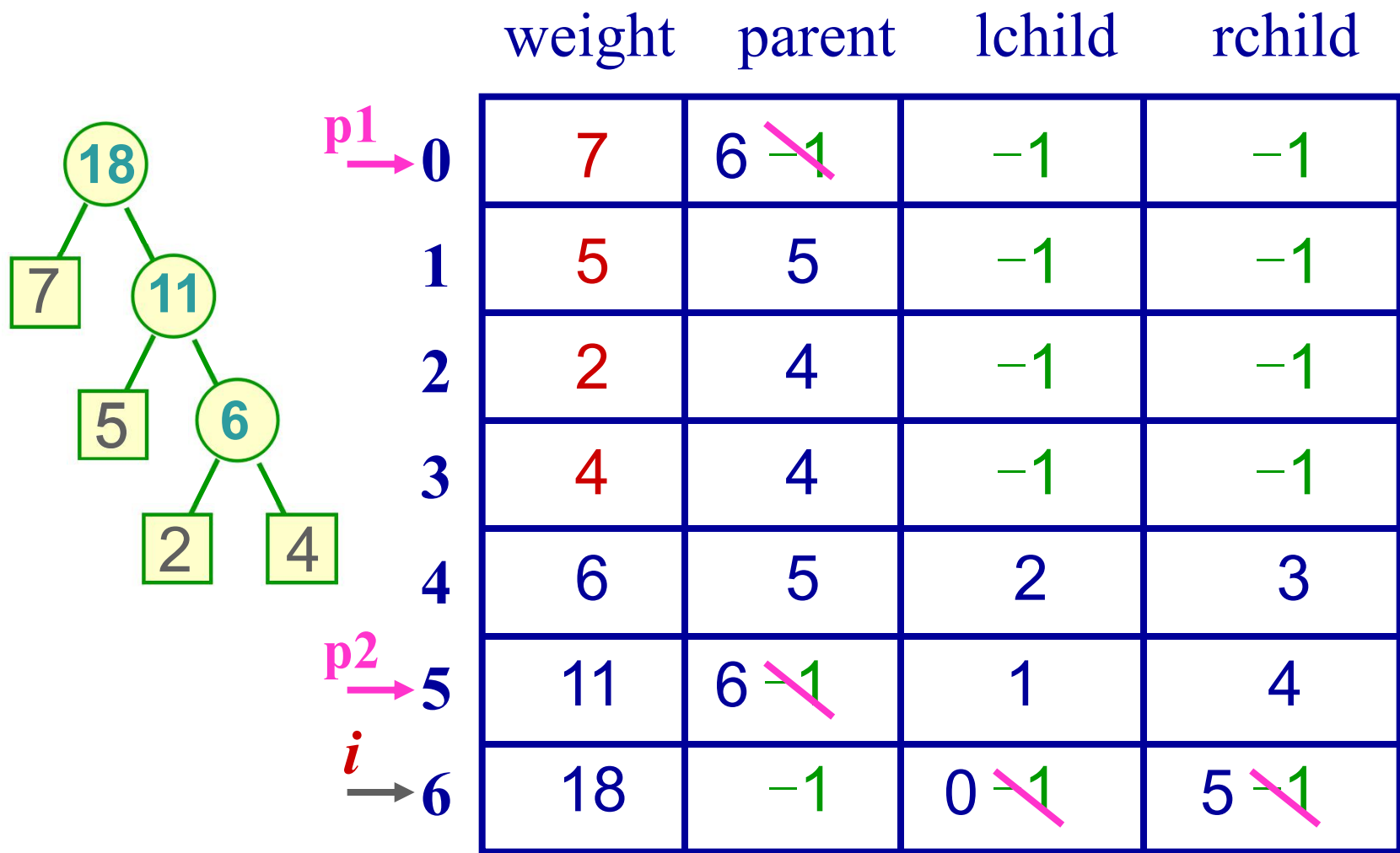
初始形态



总是在父节点为-1的节点里面找两个最小



	weight	parent	lchild	rchild
0	7	-1	-1	-1
1	5	<del>5</del> -1	-1	-1
2	2	4	-1	-1
3	4	4	-1	-1
4	6	<del>5</del> -1	2	3
5	11	-1	<del>1</del> -1	<del>4</del> -1
6		-1	-1	-1



整个过程循环多少次？ N-1， 因生成N-1个新节点



## 利用静态链表创建huffman树的算法框架：

1、初始化，存入节点，所有节点的父、左右孩为-1

2、重复 $n-1$ 次：

(1)从头扫描，找到父节点为-1的，权值最小的两个节点 $p1, p2$

(2) $p1, p2$ 节点权值相加，权值放入表中当前行，将该行左右孩子赋值为 $p1, p2$

(3)更改 $p1, p2$ 的父节点为当前行索引位置

```
void CreatHuffmanTree (HuffmanTree *T, int w[], int n) {  
    int i, j, p1, p2, min1, min2;           //w为权值数组
```

```
    for (i = 0; i < n; i++)
```

```
        T->F[i].weight = w[i];
```

```
    T->size = n;
```

```
    for (i = 0; i < 2*n-1; i++) {
```

```
        T->F[i].parent = -1;
```

```
        T->F[i].LChild = -1;
```

```
        T->F[i].RChild = -1;
```

```
    } //初始化完毕
```

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

```
for(i=n;i<n*2-1;i++){  
    SelectMin1Min2(T, &p1, &p2); //p1,p2为索引  
    T->F[i].LChild = p1;  T->F[i].RChild = p2;  
    T->F[i].weight = T->F[p1].weight + T->F[p2].weight;  
    T->F[p1].parent = T->F[p2].parent = i;  
    T->size++;  
}  
}
```

```
void SelectMin1Min2(HuffmanTree *T, int *p1, int *p2){  
    int i, min1, min2;           //求最小值及次小值的函数  
    min1=min2=INT_MAX; //最小值为整型数最大值  
    for(i=0;i<T->size;i++){  
        if(T->F[i].parent == -1){  
            if(T->F[i].weight < min1){ //比当前最小的小  
                min2 = min1; //（即原最小是目前次最小）  
                *p2=*p1;  
                min1 = T->F[i].weight; //保存当前最小值  
                *p1 = i; }  
        }
```

```
else if(T->F[i].weight < min2){
```

```
    *p2 = i;
```

```
    min2 = T->F[i].weight;
```

```
}
```

```
}
```

```
}
```

```
}
```

如何编码使编码连续出现时能够被正确解码？

“前缀”指除了最后一个字符以外，一个字符串的全部头部组合。break的前缀为：b, br, bre, brea, break

**前缀编码：**在一个字符集中，任何一个字符的编码都不是另一个字符编码的前缀。

解码时候就可以能够一一对应了。

# Huffman编码

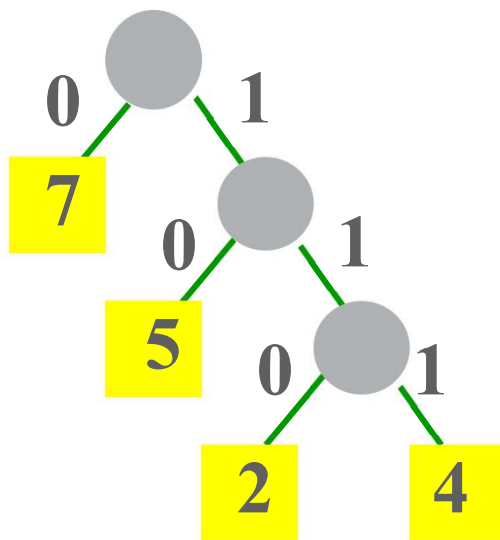
如果将左分支编码为0，右分支编码为1（也可以反过来编码），则huffman树中各节点的路径即可形成**前缀编码**。

7:0

5:10

2:110

4:111



(基于霍夫曼树，可一次性解决编码长度与解码两个问题)

给定字符串：“AATTTTCCSSSCCTTTSS”，其中A,T,C,S各字符出现概率为 $\{ 2/18, 7/18, 4/18, 5/18 \}$ ，直接用 $\{ 2, 7, 4, 5 \}$ 为各叶结点上的权值，建立Huffman树如下图。

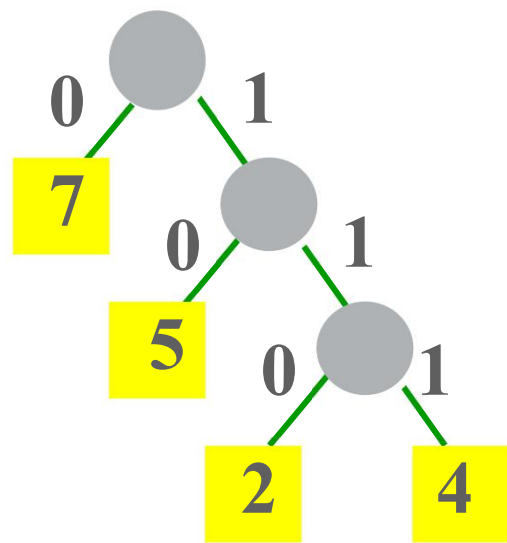
令左分支赋 0，右分支赋 1，可得Huffman编码。

T: 0    S: 10    A: 110    C: 111

它的总编码长度：

$$7*1+5*2+(2+4)*3 = 35。$$

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



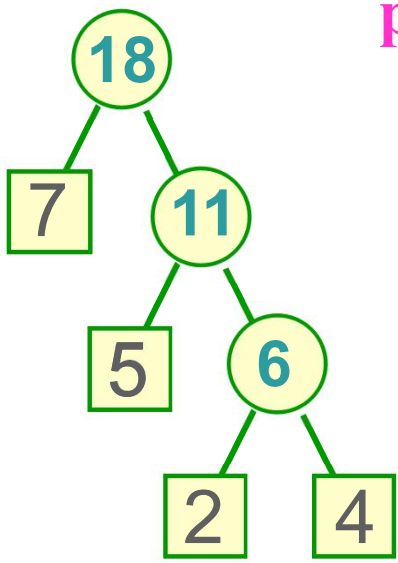


给定一棵huffman树（1维数组中），如何得到各个节点的前缀编码？

节点编码为：  
根节点出发的  
路径编号序列。

可由叶节点出  
发，找根，将  
沿途路径保存。

用什么保存呢？



p1 → 0

p2 → 5

i → 6

weight    parent    lchild    rchild

	7	6 <del>-1</del>	-1	-1
1	5	5	-1	-1
2	2	4	-1	-1
3	4	4	-1	-1
4	6	5	2	3
5	11	6 <del>-1</del>	1	4
6	18	-1	0 <del>-1</del>	5 <del>-1</del>

## 框架(利用栈):

0、得到叶节点数 $n=(1+T.size) / 2$

1、对每个叶节点i循环:

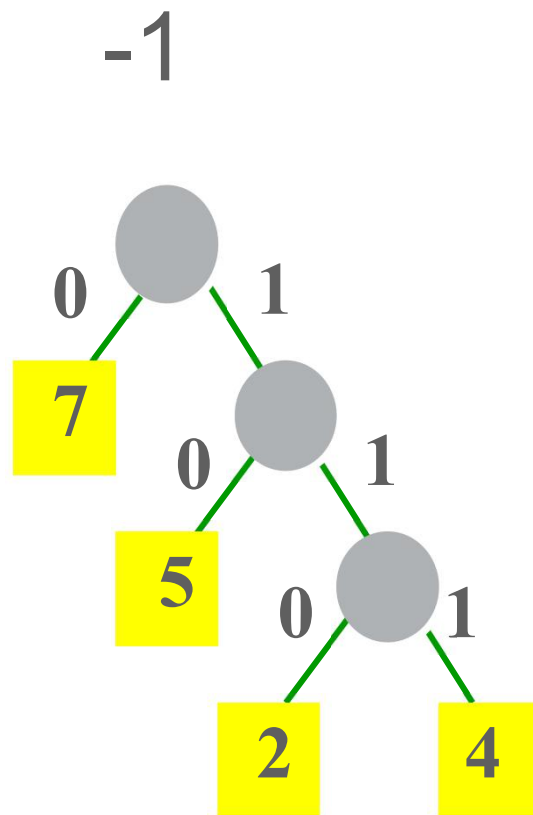
Init(S); R=p的父节点;p为当前节点i;

While (R $\neq$ -1):

if R->left\_child==p: push(0)

else:push(1); 更新p, 更新R;

输出栈内节点(即为编码)



```
void GetPrefixCode(HuffmanTree T){//伪码
```

```
    int i,n=(1+T.size)/2, p, R;
```

```
    LinkStack L;    InitStack(&L);
```

```
    for(i=0;i<n;i++){//对n个节点循环
```

```
        R = T.F[i].parent; p = i;
```

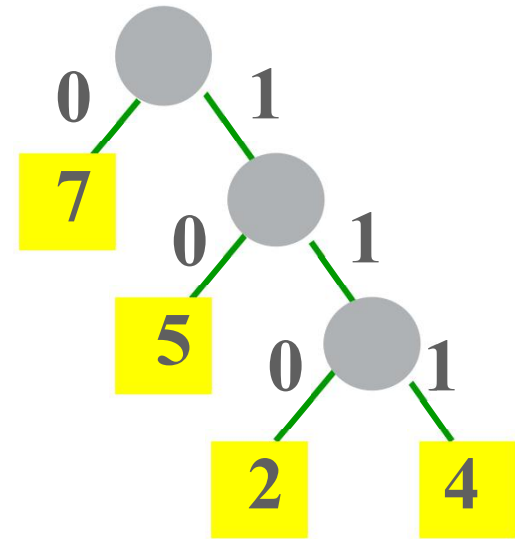
```
        while(R!=-1){
```

```
            if(T.F[R].LChild == p) push(L, 0);
```

```
            else push(L, 1);
```

```
            p=R; R=T.F[R].parent;}
```

```
        //第i节点前缀编码均在L内
```



```
while(!IsEmptyStack(L)) {  
    pop(L, &p);  
    printf("%d", p);  
}
```

```
printf("\n");
```

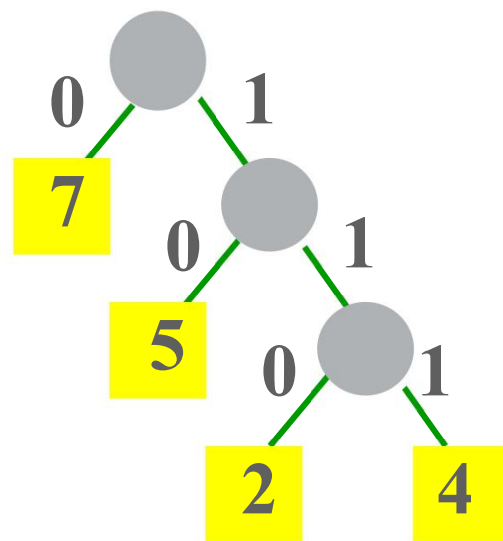
```
//n个节点的字符编码处理完毕
```

```
}
```

给定一棵huffman树（一维数组中），如何得到报文长度？

报文长度也就是树的WPL

- 1、找到每个叶节点到根的路径长度
- 2、该叶节点权值\*路径长度



## 框架：

0、WPL=0；得到叶节点数 $n=(1+T.size) / 2$

1、对所有叶节点循环：

1.1  $R =$  当前 $p$ 的父节点； $length=0$ ;

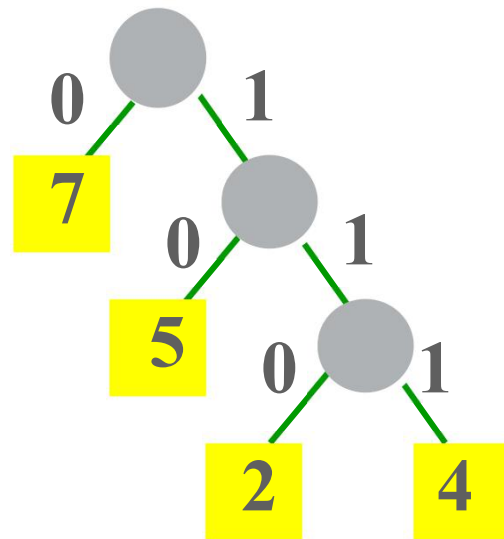
1.2 while  $R \neq -1$ :

$length++$

令 $R$ 为当前节点的父节点；

1.3  $WPL += T.F[i].weight * length$ ;

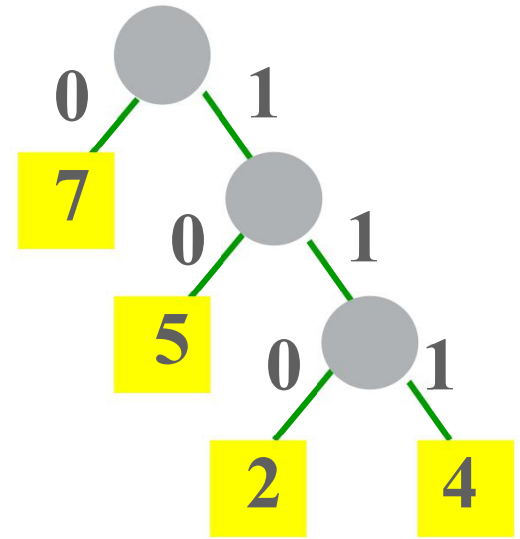
2、返回WPL;



```

float GetHuffmanTreeWPL(HuffmanTree T){//伪码
    int i, n=(1+T.size)/2, length, R;    float WPL=0;
    for(i=0;i<n;i++){//对每个节点循环
        R=T.F[i].parent;        length = 0;
        while(R!=-1){
            length++;
            R=T.F[R].parent;
        }//得到第i节点到根的路径长度
        WPL += length*T.F[i].weight;
    }
    return WPL;}

```



给定一棵huffman树（一维数组中），给定传输过来的报文数据，如何解码得到传输前的原报文？

0101101111

就是根据报文，找叶节点

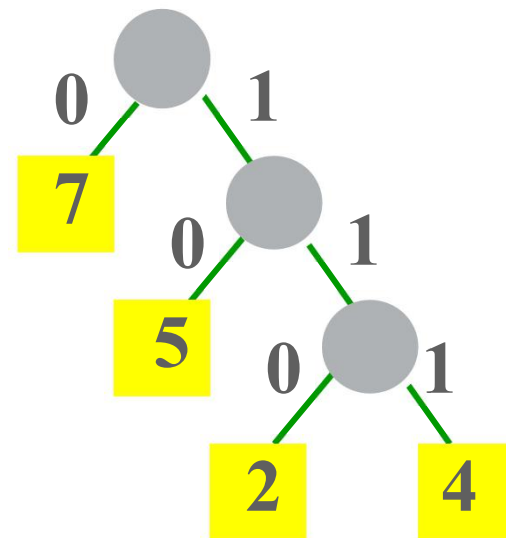
报文控制着从根节点到叶节点的路径

思路：

1、依据报文从霍夫曼树的根节点行进

如果遇到叶节点，则输出叶节点的字符。回到根节点。

2、重复1直至报文结束





框架：给定HT， sequence      0101101111

0、找到根节点编号，设为当前节点j；

1 while sequence[i]!='\0':

    if j是叶节点：

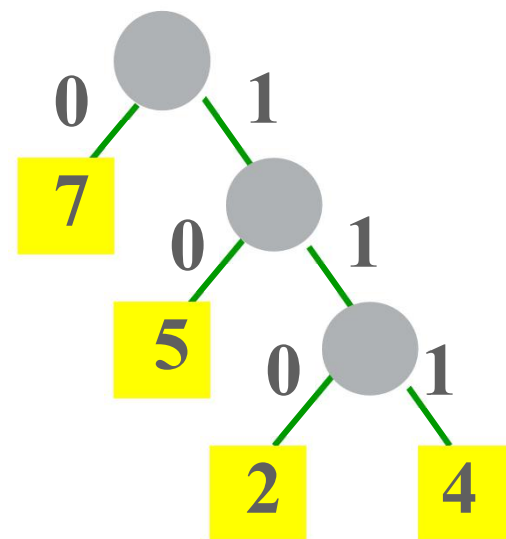
        打印该节点；

        当前节点j回到根节点

    elif sequence[i]=='0': j = j->LChild; 左走

    else: j = j->Rchild; 右走

if j是叶节点：      打印该节点； (最后停留的位置)



```
void DecodeHuffman(HuffmanTree T, char *sequence){//伪码
    int i=0, j=T.size-1;
    while(sequence[i]!='\0')
    {
        if(T.F[j].LChild==-1 && T.F[j].RChild==-1){
            printf("%c", T.F[j].ch);
            j=T.size-1;
        }
    }
}
```

```
else {  
    if(sequence[i]=='0') j=T.F[j].LChild;  
    else j=T.F[j].RChild;  
    } i++;  
}
```

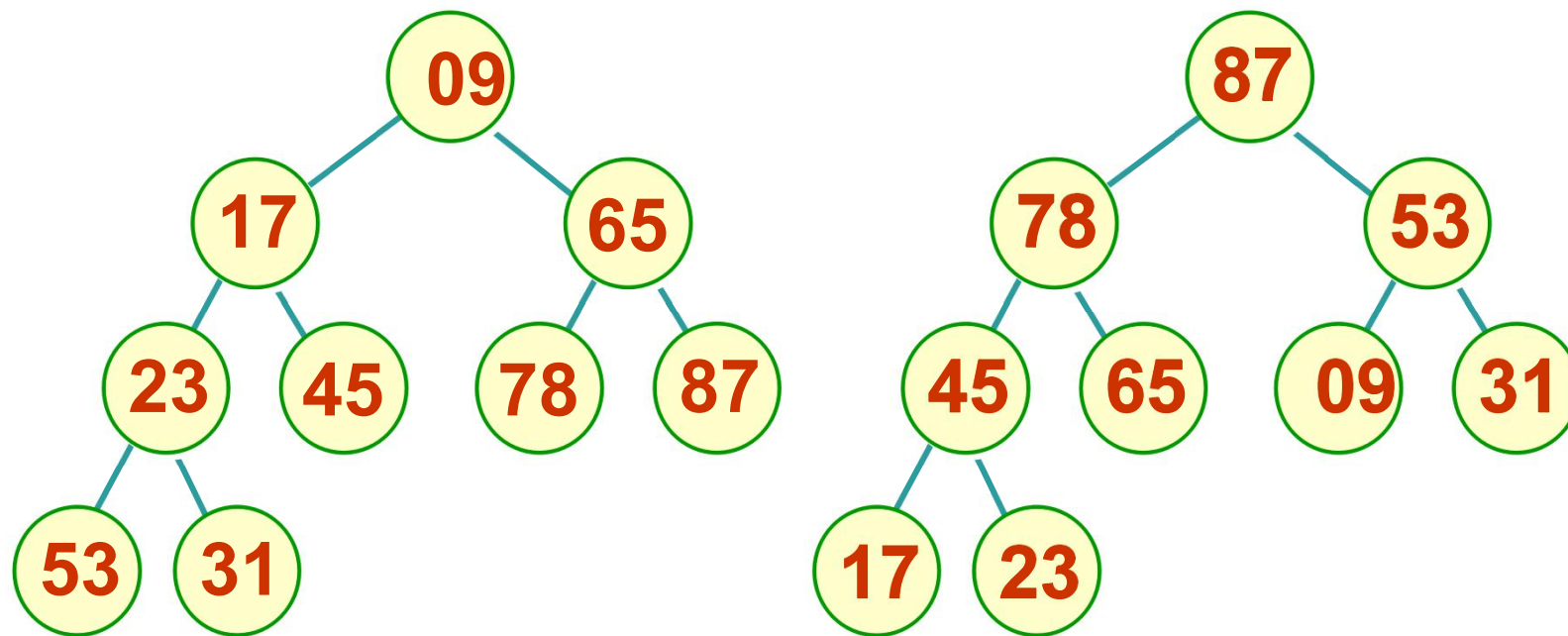
```
if(T.F[j].LChild==-1 && T.F[j].RChild==-1)//最后一个编码  
    printf("%c",T.F[j].ch);
```

}//解码可在节点定义时，多设一个ch域 //或者固定  
A,B,C,D...为索引0,1,2,3...权值也对应

# 如何构造优先队列

- 1、Naive思路：排序，初始 $n$ 个节点时间复杂度 $O(n^2)$ ，每次出队两个节点，入队一个节点，需要重新排序
- 2、改进思路：改进排序算法使之成为 $O(n \log n)$ ，每次出队两个节点，入队一个节点，还需要重新排序 $O(n \log n)$
- 3、改进思路：不重新排序，而是对新生成的节点，在队列中找到位置插入。对顺序表示，需要挪动 $O(n)$ 元素。
- 4、正解：用树状结构---堆结构  
建立为 $O(n \log n)$ ，每次入队出队时间复杂度为 $\log n$

堆



完全二叉树  
顺序表示

$$K_i \leq K_{2i+1} \ \&\& \\ K_i \leq K_{2i+2}$$

$$K_i \geq K_{2i+1} \ \&\& \\ K_i \geq K_{2i+2}$$

## 堆 ( Heap )

设有一个关键字集合，按**完全二叉树**的顺序存储方式存放在一个一维数组中。对它们从根开始，自顶向下，同一层自左向右**从 0 开始**连续编号。若满足

$$K_i \leq K_{2i+1} \ \&\& \ K_i \leq K_{2i+2}$$

或 
$$K_i \geq K_{2i+1} \ \&\& \ K_i \geq K_{2i+2},$$

则称该关键字集合构成一个堆。

前者称为最小堆（小根堆），后者称为最大堆（大根堆）。

下午上机：

实现如下函数（先做1,2,3,4，字符权值数组可以先人工给）：

0、给定字符串统计各个字母频次，输出字符权值数组

1、给定权值数组，构造huffman树

2、给定huffman树，给出报文总编码长度

3、给定huffman树，给出各个字母的huffman编码

4、给定huffman树，给定接收的报文，解码出原报文

测试报文：AABBBBCCCDDEFFAAAABBBCCC

下周:

堆与优先队列、判定树、并查集、二叉搜索树