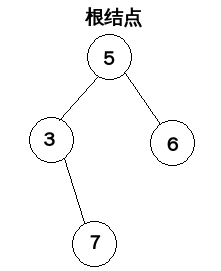
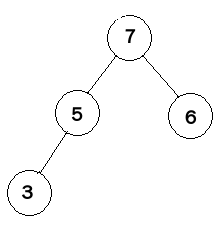
哈夫曼树

哈夫曼编码

带权二叉树

带权路径和值：

3\*1 + 6\*1 + 7\*2



5\*1 + 6\*1 + 3\*2

最优二叉树是带权路径和值最小的二叉树

哈夫曼树（是一种最优二叉树）所有键值在哈夫曼树中，只能是叶子节点。

哈夫曼树的构建方法：

1、先将有全职的叶子节点按照从小到大的顺序排成一个有序数列；

2、每次取头两个权值最小的节点组成一个新的节点，左孩子<右孩子；

3、将新产生的节点放回原有序序列；

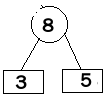
4、重复上述过程，直到哈夫曼树生成。

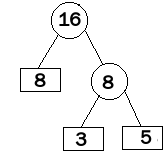
设有如下键值：

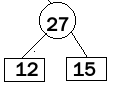
3 5 8 12 15 27 36

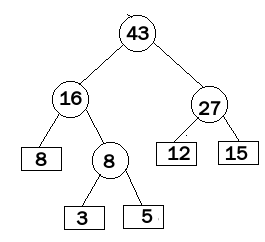
由这些键值构成最优二叉树：（建立过程）

~~3~~ ~~5~~ 8 8 12 15 27 36

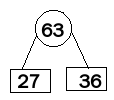
~~8~~ ~~8~~ 12 15 16 27 36

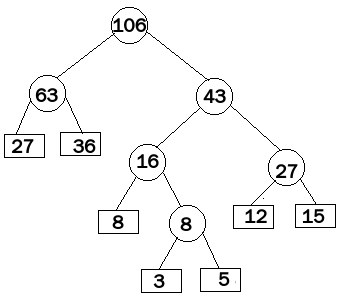
~~12~~ ~~15~~ 16 27 27 36

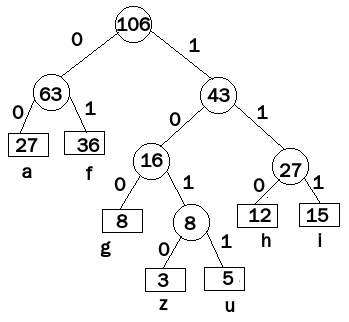
~~16~~ ~~27~~ 27 36 43



~~27~~ ~~36~~ 43 63

~~43~~ ~~63~~ 106



设在某文本文件中出现了如下字符及其频度：

z u g h i a f

3 5 8 12 15 27 36

从根结点出发，到达叶子节点中间所经过的路径，

按顺序排列，则，每个字符对应彝族0、1序列：

z:1010

u:1011

g:100

h:110

i:111

a:00

f:01

从键盘输入一个字符串，编程统计该字符串中出现的字符及其频度。

freq:0 1 2 3 ... 65 66 ... 97 98 99 100 101 102 ... 127

A B a b c d e f

s[i]

freq[s[i]]++;

freq[0] - freq[255] 下标就是该字符ASCII值，该元素的值就是该字符的出现频度

char s[80]

int freq[256] = {0};

gets(s);

for(i=0; s[i]; i++)

freq[s[i]]++;

for(i = 0; i < 256; i++)

if(freq[i] != 0)

i就是该字符的ASCII码，freq[i]是频度

aaaaaaaaaaaaaaaaaaabbbbbbcccccfffggggtt

a:19 b:6 c:5 f:3 g:4 t:2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 下标 | 字符 | 频度 | 左孩子 | 右孩子 | flag | 编码 |
| 0 | a | ~~19~~ | -1 | -1 | 1 |  |
| 1 | b | ~~6~~ | -1 | -1 | 1 |  |
| 2 | c | ~~5~~ | -1 | -1 | 1 |  |
| 3 | f | ~~3~~ | -1 | -1 | 1 |  |
| 4 | g | ~~4~~ | -1 | -1 | 1 |  |
| 5 | t | ~~2~~ | -1 | -1 | 1 |  |
| 6 | # | 2+3=~~5~~ | 5 | 3 | 1 |  |
| 7 | # | 4+5=~~9~~ | 4 | 2 | 1 |  |
| 8 | # | 5+6=~~11~~ | 6 | 1 | 1 |  |
| 9 | # | ~~20~~ | 7 | 8 | 1 |  |
| 10 | # | 39 | 0 | 9 |  |  |

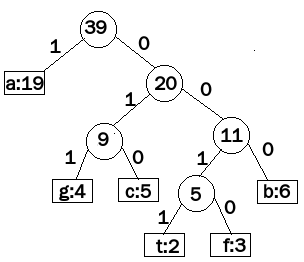
设要处理的字符种类为n个，则，所生成的哈夫曼树的项数（节点数）一定为2\*n-1

有n种字符，则，需要进行n-1次中间节点的生成

|  |
| --- |
| typedef struct HUFFMAN{  char alpha;  int freq;  int flag;  int left;  int right;  }HUFFMAN; |

aaaaaaaaaaaaaaaaaaabbbbbbcccccfffggggtt

a:19 b:6 c:5 f:3 g:4 t:2



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 下标 | 字符 | 频度 | 左孩子 | 右孩子 | flag | 编码 |
| 0 | a | ~~19~~ | -1 | -1 | 1 | 1 |
| 1 | b | ~~6~~ | -1 | -1 | 1 | 000 |
| 2 | c | ~~5~~ | -1 | -1 | 1 | 010 |
| 3 | f | ~~3~~ | -1 | -1 | 1 | 0010 |
| 4 | g | ~~4~~ | -1 | -1 | 1 | 011 |
| 5 | t | ~~2~~ | -1 | -1 | 1 | 0011 |
| 6 | # | 2+3=~~5~~ | 5 | 3 | 1 |  |
| 7 | # | 4+5=~~9~~ | 4 | 2 | 1 |  |
| 8 | # | 5+6=~~11~~ | 6 | 1 | 1 |  |
| 9 | # | ~~20~~ | 7 | 8 | 1 |  |
| 10 | # | 39 | 0 | 9 | 0 |  |

折半查找：

假设有如下数据：待查数据为36

0 1 2 3 4 5 6 7 8 9 10 11

2 6 8 9 13 15 17 22 34 36 40 45

head = 0;

tail = n-1;

while(没找完 && 没找到)(head <= tail && !Found)

{int

mid = (head + tail) /2;

if(key == a[mid])

Found = TRUE;

else if(key > a[mid])

head = mid +1;

else

tail = mid -1;

}

c语言对文件的操作

文件：是一个外存的概念，只在外存中才存在这个概念。

文件是计算机存储表达信息的最小逻辑单位。

文件的组成：文件名和文件内容

文件分为两大类：可执行文件和非可执行文件

设某汉字A区位码为X1 X2

设某汉字B区位码为Y1 Y2

设某汉字C区位码为X1 Y2

AB 中本不存在C，若不注意处理这个问题，那么会误以为AB字符串中存在汉字C

C程序处理文件

先将文件内容调入内存中，才可能被程序所处理。

一个巨大的问题：文件的容量可达可小，小到只有一个字节，大到可能上G。

结论：将文件全部调入内存再处理，明显是不可取的方案；于是，计算机科学家们，打算采用电视机机制，处理文件的内存操作。

FCB:文件控制块

FCB是OS的系统资源，这种资源的数量是有限的！

我们C程序要对文件进行操作，必须先申请这个资源！

与FCB对应，C语言中有一个数据块（结构体）FILE。

站在文件控制块的角度，对文件的操作，应由三步组成：

1、申请系统资源；

2、透过这个系统资源，完成对文件的操作；

3、归还这个系统资源。

这分别对应如下操作：

1、打开文件；本质是申请FCB

2、读写文件；

3、关闭文件；本质是释放FCB

FILE \*fp;

fp = fopen(“文件名”， “打开方式”);

… …

fclose(fp);

打开方式其实是由两个问题组成：

1、识别方式；

2、操作方法。

识别方式：文本文件（字节流）、二进制文件（格式化文件）

操作方法：r只读方法、w写方法（创建）、a追加方法

混搭r+,w+