霍夫曼编码

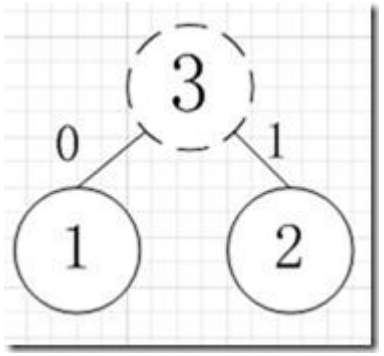
哈夫曼编码(Huffman Coding)，又称霍夫曼编码，是一种编码方式，可变字长编码(VLC)的一种。Huffman于1952年提出一种编码方法，该方法完全依据字符出现的概率来构造异字头的平均长度最短的码字，有时称之为最佳编码，一般就叫做Huffman编码（有时也称为霍夫曼编码）。

哈夫曼编码，主要目的是根据使用频率来最大化节省字符（编码）的存储空间。

霍夫曼编码

霍夫曼编码

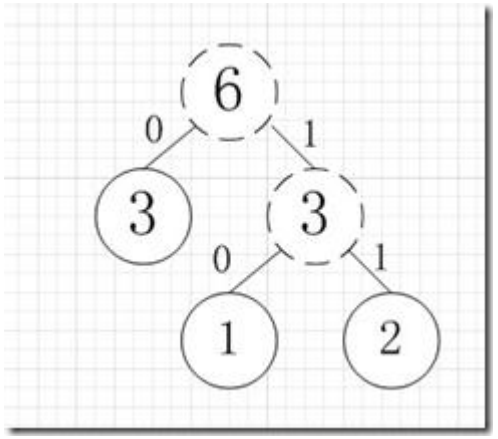
简易的理解就是，假如我有A,B,C,D,E五个字符，出现的频率（即权值）分别为5,4,3,2,1,那么我们第一步先取两个最小权值作为左右子树构造一个新树，即取1，2构成新树，其结点为1+2=3，如图：



霍夫曼编码

霍夫曼编码

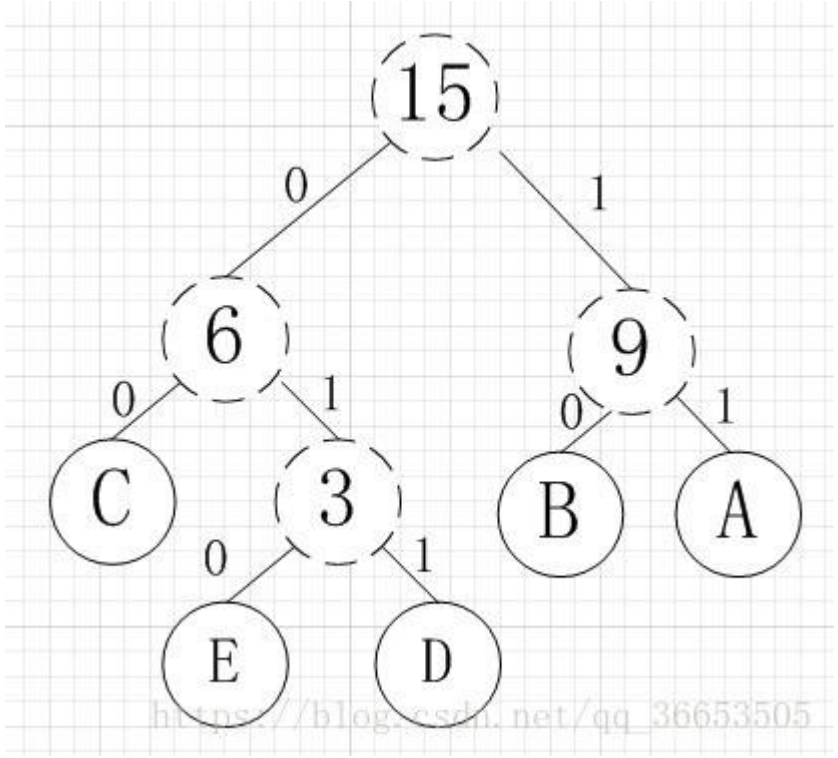
虚线为新生成的结点，第二步再把新生成的权值为3的结点放到剩下的集合中，所以集合变成{5,4,3,3}，再根据第二步，取最小的两个权值构成新树，如图：



霍夫曼编码

霍夫曼编码

再依次建立哈夫曼树，如下图：



霍夫曼编码

所以各字符对应的编码为：A->11,B->10,C->00,D->011,E->010

霍夫曼编码是一种无前缀编码。解码时不会混淆。其主要应用在数据压缩，加密解密等场合。

如果考虑到进一步节省存储空间，就应该将出现概率大（占比多）的字符用尽量少的0-1进行编码，也就是更靠近根（节点少），这也就是最优二叉树-哈夫曼树。

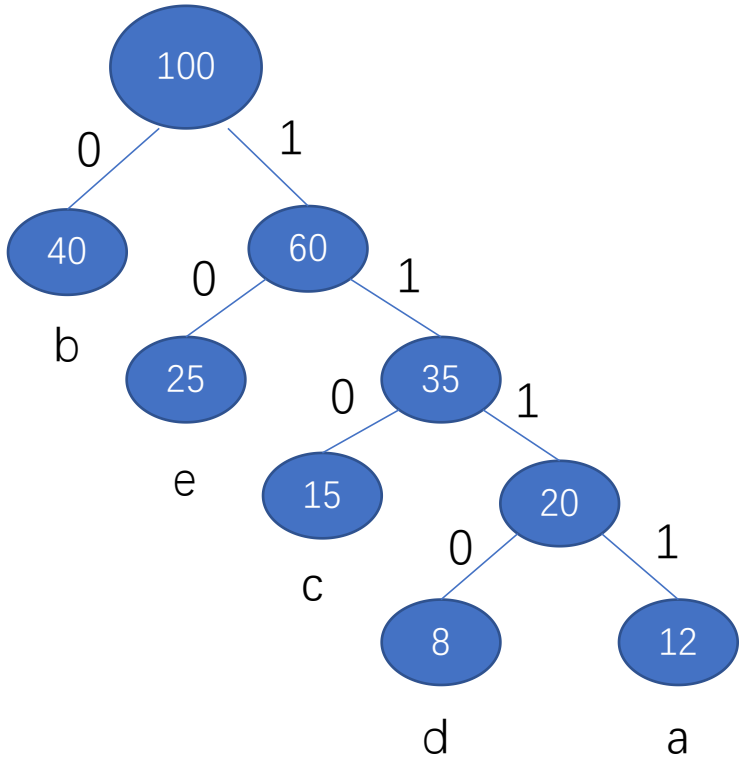
实验3内容算法提示

题目：写一个哈夫曼码的编/译码系统，要求能对要传输的报文进行编码和解码。构造哈夫曼树时，权值小的放左子树，权值大的放右子树，编码时右子树编码为1，左子树编码为0。

总体思路：使用二叉树来表示哈夫曼树，所有的叶子节点分别表示一个符号，从根节点到这个叶子节点，每向左走则标志一个0，向右走就标志一个1，从根节点走到这个叶子节点的路径就可以标志出一个0和1的序列，这个序列从左到右看就可以作为这个叶子节点表示的符号的编码。

哈夫曼编/译码器

实验3内容算法提示



Huffman树构造过程：

1.初始由每种符号来生成一系列的叶子节点，节点中保存这个符号的出现次数或频率，把这个设置成这个节点的权值。

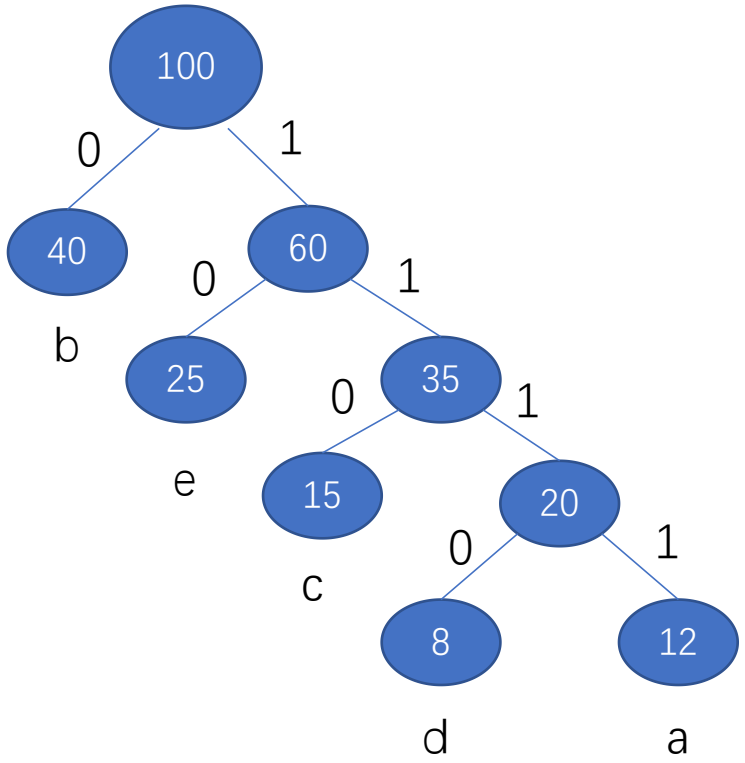
2.在叶子节点集合中选取权值最小的两个叶子节点，再另外生成一个节点，把这两个叶子节点分别作为新生成的节点的左右子节点，并且把新生成的这个节点的权值设置成两个子节点的权值之和，这样就生成了一棵树，把这棵树当作一个叶子节点，把根节点的权值当作其权值，把它加入到剩下的叶子节点的集合中。

3.重复2的过程，直到集合中就只剩下一个节点了，这个节点就是构造的哈夫曼树的根节点。

4.记录从Huffman树根节点出发到所有叶子节点的路劲，左子树为0，右子树为1，得到huffman表。

Char	a	b	c	d	e
Huffman	1111	0	110	1110	10
Weight	12	40	15	8	25

实验3内容改进算法提示



Huffman树构造过程：

1.初始由每种符号来生成一系列的叶子节点，节点中保存这个符号的出现次数或频率，把这个设置成这个节点的权值。

2.将所有节点的权值由小到大进行排序。

3.选取最前面的两个叶子节点，再另外生成一个节点，把这两个叶子节点分别作为新生成的节点的左右子节点，并且把新生成的这个节点的权值设置成两个子节点的权值之和，这样就生成了一棵树，把这棵树当作一个叶子节点，把根节点的权值当作其权值，将其加入到剩下的有序数组中第一个权值比其大的数之前。

4.重复3的过程，直到集合中就只剩下一个节点了，这个节点就是构造的哈夫曼树的根节点。

5.记录从Huffman树根节点出发到所有叶子节点的路劲，左子树为0，右子树为1，得到huffman表。

Char	a	b	c	d	e
Huffman	1111	0	110	1110	10
Weight	12	40	15	8	25

实验3内容参考算法提示

由分析可得Huffman树的数据结构：

```
struct HuffmanNode
{
    int w; //权值;
    char  ch; //结点所表示的字符
    HuffmanNode *left,*right; //左孩子，右孩子
    HuffmanNode(int wight,char c):left(NULL),right(NULL)
   {          w=wight;
              ch=c;      };
};

        struct HuffmanSet
        {
            set<HuffmanNode*> huffmanset;
                    //存储叶子节点的集合
        };
```

实验3内容参考算法提示

Huffman树构造关键代码：

```
while(huffmanset.size()>1)
{
    HuffmanNode* min1=findMinNode();  //寻找集合中最小节点
    HuffmanNode* min2=findMinNode();  //寻找集合中最小节点
    HuffmanNode *newNode=new HuffmanNode(min1->w+min2->w,' '); //新节点不需要表示符号
    newNode->left=min1;
    newNode->right=min2;
    huffmanset.insert(newNode); //插入新节点
    root=newNode;   //根节点为新构建的节点
};
```

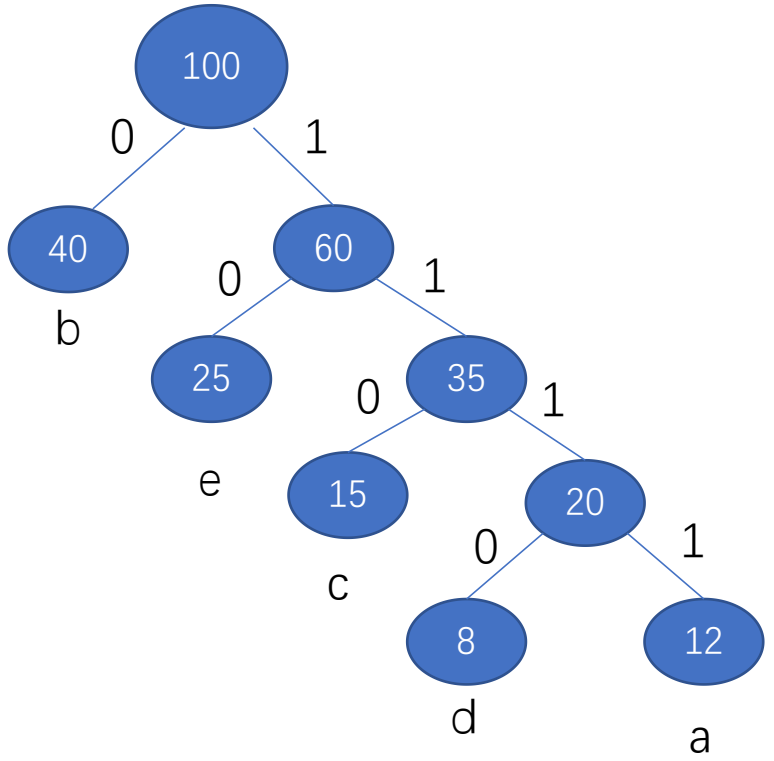
实验3内容算法提示

Huffman树编码

思想：

1.遍历huffman树，从根节点出发，访问左子树，路径加0，访问右子树，路径加个1，若遇到叶子节点，则保存当前路径和当前节点所表示字符的映射，建立映射表。

2.对需要编码的字符串，按位寻找与之对应的huffman编码，将所有编码叠加即为最终结果。



编码字符串：bbbaddeccbbb

映射关系表

Char	a	b	c	d	e
Huffman	1111	0	110	1110	10

编码结果：0+0+0+1111+1110+1110+10+110+110+0+0+0
=000111111101110101101100000

sample_input	5 a b c d e 12 40 15 8 25 bbbaddeccbbb
sample_output	000111111101110101101100000

实验3内容算法提示

Huffman树解码

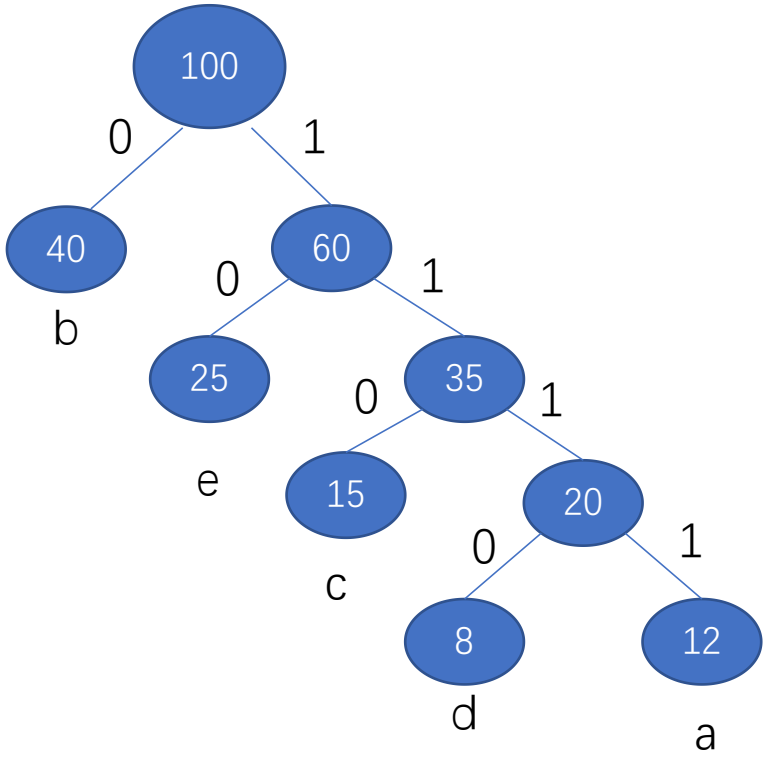
思想：

1.从根节点出发，遍历哈夫曼编码，遇到0则选左子树，1则选右子树。

2.若当前节点为叶子节点，则输出此节点所表示字符，重复1，直到遍历所有编码。

哈夫曼编码： 0001111111011101010110110000

译码结果：b+b+b+a+d+d+e+c+c+b+b+b
=bbbaddeccbbb



解码唯一性：因为所有的符号的编码都是从根节点到某个叶子节点，所以每个符号的编码都不可能是其他符号编码的前缀。如果有x符号的编码是y符号编码的前缀，那么从根节点到y的节点一定经过x的节点，但这在二叉树中是不可能的，叶子节点是一条路径的终点。

sample_input	5 a b c d e 12 40 15 8 25 0001111111011101010110110000
sample_output	bbbaddeccbbb

实验3算法分析

时间复杂度分析

在哈夫曼编码的过程中，需要重复进行排序操作。所以具体要看代码采用何种排序方法。

如果采用冒泡排序、插入排序、选择排序等O(n^2)的排序方法，编码的时间复杂度是O(n^3)；如果采用快速排序，编码的时间复杂度是O(n^2logn)；如采用堆排序方法，编码的时间复杂度是O(n^2logn)

空间复杂度分析

空间消耗主要是：1. 构建新节点，若有n个叶子节点，则需新构建n-1个新节点；
2.字符和编码映射表，映射表也只需n个，所以空间复杂度都为O(n)，整个空间复杂度为O(n) 。