选25、哈夫曼树与哈夫曼编码

1、什么是哈夫曼树

哈夫曼树也称为最优二叉树。

如果二叉树的每个叶子节点都有对应的权重，这个权重与根节点到叶子节点的路径长度的乘积称为带权路径长度，则如果某个二叉树所有叶子节点的带权路径长度之和最小，则该二叉树称为最优二叉树，或哈夫曼树。

具体一点讲：

在一段文本中，不同的字符的出现次数并不是相同的，如果设计这样的一套编码，使得出现频次高的字符的编码尽可能短，那么最后得到的文本编码也会短。如莫斯电码就是不等长编码。

利用哈夫曼树，就可以来制定这样一套编码体系

2、哈夫曼编码

假设有六个字符，其出现频次为：’A’:3,’B’:2,’C’:10,’D’:2,’E’:4,’F’:6

首先根据频次进行升序排序：

’B’:2,’D’:2,’A’:3,’E’:4,’F’:6,’C’:10

然后取前两个字符组成一组，并放到最后：

’A’:3,’E’:4,’F’:6,’C’:10,[’B’,’D’]:4，其中[’B’,’D’]的4是B与D的频次之和

再进行升序排序：

’A’:3,’E’:4,[’B’,’D’]:4,’F’:6,’C’:10,

再取前两个组成一组：

[’B’,’D’]:4,’F’:6,’C’:10,[’A’,’E’]:7

再排序：

[’B’,’D’]:4,’F’:6,[’A’,’E’]:7,’C’:10

再组合：

[’A’,’E’]:7,’C’:10,[[’B’,’D’],’F]:10,

再排序（顺序不变）

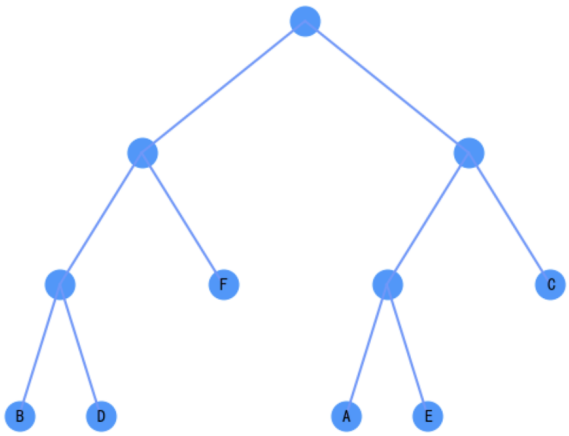
再组合：

[[’B’,’D’],’F]:10,[[’A’,’E’],’C]:17

再来一次，直到把所有项都合并了：

[[[’B’,’D’],’F],[[’A’,’E’],’C]]:27

最后得到的[[[’B’,’D’],’F],[[’A’,’E’],’C]]其实就是一颗二叉树：



从根节点出发，约定访问左子树就记录0，右子树则记录1，那么到B的记录为000，到D的记录为001，到C的记录为11，具体如下：

```
{'A': '100', 'B': '000', 'C': '11', 'D': '001', 'E': '101', 'F': '01'}
```

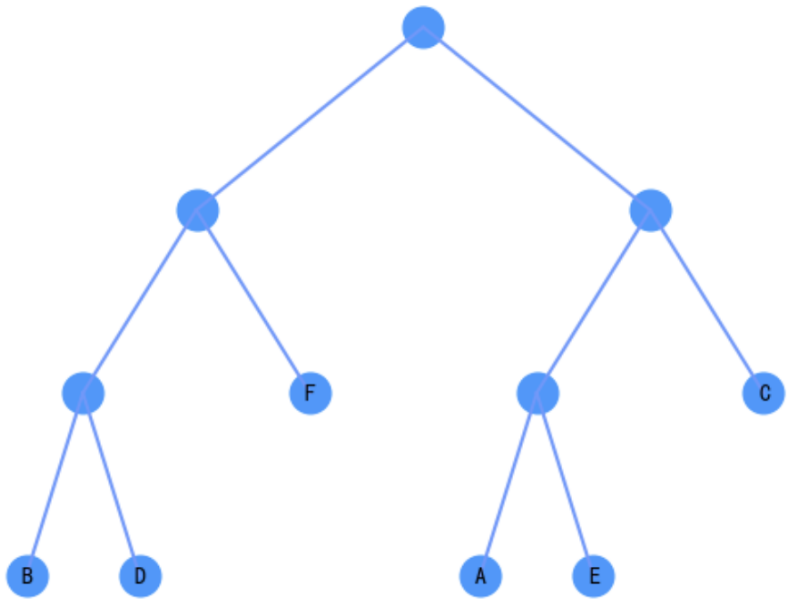
这样就得到了哈夫曼编码，利用该编码体系对文本进行编码，得到的二进制数总长度是最小的。

3、哈夫曼编码的代码实现：

```
class HuffmanTree(BinTree):
    def __init__(self,freq_dict):
        self.array=[]
        for f in freq_dict:
            self.array.append([f,freq_dict[f]])
        while len(self.array)!=1:
            self.array.sort(key=lambda x:x[1])
            a,b=self.array[:2]
            self.array=self.array[2:]
            self.array.append([[a[0],b[0]],a[1]+b[1]])
    def create_tree(self,arr=None):
        if arr==None:
            arr=self.array[0][0]
        bt=BinTree(None)
        if type(arr[0])==list:
            bt.setLeft(self.create_tree(arr[0]))
        else:
            bt.setLeft(BinTree(arr[0]))
        if type(arr[1])==list:
            bt.setRight(self.create_tree(arr[1]))
        else:
            bt.setRight(BinTree(arr[1]))
        return bt
    def show_tree(self):
        bt=self.create_tree()
        bt.drawTree()
    def get_code(self,key,arr=None):
        if arr==None:
            arr=self.array[0][0]
        if arr[0]==key:
            return '0'
        if arr[1]==key:
            return '1'
        bt=self.create_tree(arr)
        lt=bt.getLeft()
        rt=bt.getRight()
        if key in lt.midTraversal():
            return '0'+self.get_code(key,arr=arr[0])
        if key in rt.midTraversal():
            return '1'+self.get_code(key,arr=arr[1])
        return None
```

```
freq_dict={'A':3,'B':2,'C':10,'D':2,'E':4,'F':6}
ht=HuffmanTree(freq_dict)
print(ht.array)
bt=ht.create_tree(ht.array[0][0])
ht.show_tree()
encode={}
for c in freq_dict:
    encode[c]=ht.get_code(c)
print(encode)
decode={}
for enc in encode:
    decode[encode[enc]]=enc
print(decode)
```

```
[[[[['B', 'D'], 'F'], [['A', 'E'], 'C']], 27]]
```




```
{'A': '100', 'B': '000', 'C': '11', 'D': '001', 'E': '101', 'F': '01'}
{'100': 'A', '000': 'B', '11': 'C', '001': 'D', '101': 'E', '01': 'F'}
```

最后得到的encode，decode就可以用来对文本进行编码和解码

有能力的话，写一下编码和解码的函数吧！