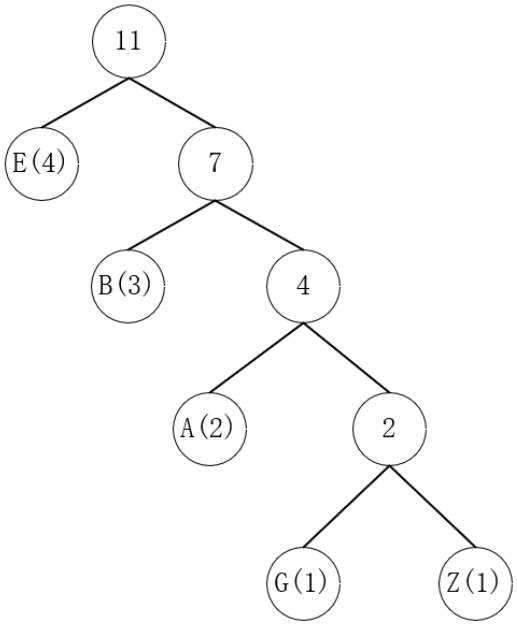
Huffman 编码

SA20225085 朱志儒

实验内容

对字符串进行 01 编码，输出编码后的 01 序列，并比较其相对于定长编码的压缩率。例如对于字符串“AABBBEEEEGZ”,如果使用定长编码，‘A’、‘B’、‘E’、‘G’、‘Z’字符各需要 3 位 01 串编码，编码后的字符长度为 3*11=33 位，如果使用 Huffman 编码，可编码为下图，编码后的字符长度为 2*3+3*2+4*1+4+4=24，压缩率为 24/33=72.73%.



对文件 data.txt 的字符串按照 Huffman 编码方式编码为 01 序列，并输出到 encode.txt 文件，控制台打印压缩率。

程序输入：

字符串文件 data.txt。

程序输出：

压缩后的 01 序列输出到 encode.txt，控制台打印压缩率。

实验目的

编程实现 Huffman 编码问题，并理解其核心思想。

算法设计思路

（1） 统计字符串中每个字符出现的频率；

（2） 将字符和频率存入结点中，所有结点组成一个列表，依据频率从小到大将列表排序；

（3） 从列表中删除前两个结点，生成一个新结点，其频率为删除的两个结点的频率之和，其左右孩子分别是删除的两个结点。将新结点加入列表，再依据频率从小到大将列表排序；

（4） 重复步骤（3）直到列表中只剩下一个结点，该结点就是根结点。

（5） 从根节点先序遍历 Huffman 树，访问左子树的边标为 1，访问右子树的边标为 0。访问到叶子结点的路径就是该叶子结点中字符的 Huffman 编码。

（6） 遍历字符串，将字符替换成对应的 Huffman 编码，计算压缩率。

源码+注释

```
def compare(node1, node2):
    """定义 sort 函数中比较方式"""
    if node1.prob > node2.prob:
        return 1
    elif node1.prob == node2.prob:
        return 0
    else:
        return -1

class Node:
    """定义结点"""
    def __init__(self, value, prob, left, right):
        self.value = value
        self.prob = prob
        self.left = left
        self.right = right

class Huffman:
```

```
19.
20.     def __init__(self):
21.         self.root = None
22.         self.nodeDict = {}
23.
24.     def buildTree(self, dataNode):
25.         """建树"""
26.         while len(dataNode) > 1:
27.             dataNode = sorted(dataNode, key=functools.cmp_to_key(compare))
28.             node1 = dataNode[0]
29.             node2 = dataNode[1]
30.             newNode = Node(None, node1.prob + node2.prob, node1, node2)
31.             del dataNode[0]
32.             del dataNode[0]
33.             dataNode.append(newNode)
34.         self.root = dataNode[0]
35.
36.     def dfs(self, node, str):
37.         """先序遍历树，得到每个字符的编码方式"""
38.         if node.left is None and node.right is None:
39.             self.nodeDict[node.value] = str
40.         if node.left is not None:
41.             self.dfs(node.left, str + '1')
42.         if node.right is not None:
43.             self.dfs(node.right, str + '0')
44.
45.     def encode(self, freqList):
46.         dataNode = []
47.         for item in freqList:
48.             dataNode.append(Node(item[0], item[1], None, None))
49.         self.buildTree(dataNode)
50.         self.dfs(self.root, '')
51.
52. if __name__ == "__main__":
53.     file = open("data.txt")
54.     data = file.readline().strip()
55.     charDict = {}
56.     """统计字符出现的次数"""
57.     for i in range(len(data)):
58.         if data[i] not in charDict.keys():
59.             charDict[data[i]] = 1
60.         else:
61.             charDict[data[i]] += 1
62.     dataList = []
```

```
    for key, value in charDict.items():
        dataList.append([key, value])
    huffman = Huffman()
    huffman.encode(dataList)
    print("字符编码方式:", huffman.nodeDict)

    encodeText = ""
    for i in range(len(data)):
        encodeText += huffman.nodeDict[data[i]]
    print("Huffman 编码:", encodeText)

    length = len(dataList)
    count = 0
    while length > 1:
        length /= 2
        count += 1
    print("压缩率:", len(encodeText) * 1.0 / (count * len(data)))

    with open("encode.txt", "w") as outfile:
        for i in range(len(data)):
            outfile.write(huffman.nodeDict[data[i]])
```

算法测试结果

运行程序，结果显示如下：

```
字符编码方式: {'A': '1', 'B': '011', 'C': '010', 'R': '001', 'S': '0001', 'M': '00001', 'N': '00000'}
Huffman编码: 11000100001101101111100100111011010110100100010010001000100000
压缩率: 0.8055555555555556
```

实验过程中遇到的困难及收获

本次实验实验较为简单，没有遇到特别困难的问题。