数据结构实验报告
张恒硕　　2212266　　智科 2 班

实验目标:　实现哈夫曼树

哈夫曼树

实验原理:

　　哈夫曼树（Huffman Tree）是一种树形结构，采用无损压缩的编码方式，其原理是将文件中出现频率较高的字符编码为较短的二进制位，而频率较低的字符则使用较长的二进制位。

代码:

```
#include<iostream>
using namespace std;
class Node { //哈夫曼树节点
public:
    int weight; //权值
    int parent; //父亲的数组下标值
    int lchild, rchild; //左、右孩子
};
class Tree { //哈夫曼树
public:
    Node* data;
    int length; //节点数
    Tree(int count, int* pweight) { //构造函数
        length = count;
        data = new Node[2 * length - 1]; //开辟哈夫曼数组，n个叶子节点，2n-1个节点
        for (int i = 0; i < 2 * length - 1; ++i) { //以-1标记未使用节点
            data[i].parent = -1;
            data[i].lchild = -1;
            data[i].rchild = -1;
        }
        for (int i = 0; i < length; ++i) { //为叶子节点赋值
            data[i].weight = pweight[i];
        }
    }
    ~Tree() { //析构函数
        delete[] data;
    }
    int GetLength() { //获取节点数
        return length;
    }
    void preorder(int index) { //前序遍历
        if (index != -1) {
```

```
            cout << data[index].weight << " ";
            preorder(data[index].lchild);
            preorder(data[index].rchild);
        }
    }
    void select(int& index_1, int& index_2) { //选出两个权重最小的节点
        int minvalue1 = 100; //预设一个较大的值
        int minvalue2 = 100;
        for (int i = 0; i < length; ++i) {
            if (data[i].parent == -1) { //父标记未被使用
                if (minvalue1 > data[i].weight) {
                    minvalue1 = data[i].weight; //记录最小值
                    index_1 = i; //记录下标
                }
            }
        }
        for (int i = 0; i < length; ++i) {
            if (data[i].parent == -1 && i != index_1) { //排除第一个找到的
                if (minvalue2 > data[i].weight) {
                    minvalue2 = data[i].weight;
                    index_2 = i;
                }
            }
        }
    }
    void createtree() { //创建哈夫曼树
        int index1 = 0;
        int index2 = 0;
        int length_ = length;
        for (int i = length_; i < 2 * length_ - 1; ++i) {
            select(index1, index2);
            data[i].weight = data[index1].weight + data[index2].weight;
            data[i].lchild = index1;
            data[i].rchild = index2;
            data[index1].parent = data[index2].parent = i;
            length++;
        }
    }
};
int main() {
    int weightlist[] = { 1,1,2,4,8 }; //权值列表
    int size = sizeof(weightlist) / sizeof(weightlist[0]);
    Tree tree1(size, weightlist);
    tree1.createtree();
```

```
    tree1.preorder(tree1.GetLength() - 1); //遍历哈夫曼树，参数是根节点下标
    return 0;
}
```

输入样例:
见代码主函数
1,1,2,4,8

运行结果:

实现操作
构造函数
析构函数
获取节点数
前序遍历
创建哈夫曼树

分析

哈夫曼树是一种有效的编码方法，具有以下优点:

编码效率高: 哈夫曼编码能够将频繁出现的字符编码为较短的二进制串，从而减少了编码的长度，提高了编码效率。

无损压缩: 哈夫曼编码是一种无损压缩算法，它不会丢失任何原始数据，因此可以还原出原始数据。

自适应: 哈夫曼编码的自适应能力很强，可以根据数据的分布情况动态调整编码方案，从而获得最佳的编码效果。

哈夫曼树的时间复杂度和空间复杂度与数据规模有关。以 n 为字符集的大小，建立哈夫曼树的时间复杂度为 O(nlogn)，编码的时间复杂度为 O(logn)，空间复杂度为 O(n)。