**实验题目：哈夫曼编码**

姓名：彭凯龙 学号：16122460

**====================================================**

1. **题目分析：**

HuffmanTree首先用字符集C中的每一字符c的频率f(c)初始化优先队列Q，然后不断地从Q中取出具有最小频率的两颗树x和y，将他们合并为新树z，z的频率是x和y的频率之和。经过n-1次合并后，优先队列中只剩一棵树，即所要求的树T。

1. **程序及运行结果：**

class HuffmanTree

{

protected:

// 哈夫曼树的数据成员:

HuffmanTreeNode<WeightType> \*nodes; // 存储结点信息

CharType \*LeafChars; // 叶结点字符信息

string \*LeafCharCodes; // 叶结点字符编码信息

int num; // 叶结点个数

// 辅助函数:

void Select(int n, int &r1, int &r2); // 在nodes[0 ~ n-1]中选择双亲为0,权值最小的两个结点r1,r2

void CreatHuffmanTree(CharType ch[], WeightType w[], int n);

// 由n个字符和权值创建一棵哈夫曼树

public:

// 哈夫曼树方法声明及重载编译系统默认方法声明:

HuffmanTree(CharType ch[], WeightType w[], int n); // 由n个字符和权值构造哈夫曼树

virtual ~HuffmanTree(); // 析构函数

string Encode(CharType ch); // 编码

string Code(int i);

LinkList<CharType> Decode(String strCode); // 译码

HuffmanTree(const HuffmanTree<CharType, WeightType> &copy); // 复制构造函数

HuffmanTree<CharType, WeightType> &operator=(const HuffmanTree<CharType,

WeightType>& copy); // 赋值运算符重载

};

template <class CharType, class WeightType>

void HuffmanTree<CharType, WeightType>::CreatHuffmanTree(CharType ch[], WeightType w[], int n)

// 操作结果：由n个字符和权值创建一棵哈夫曼树

{

num = n; // 叶结点个数

int m = 2 \* n - 1; // 结点个数

nodes = new HuffmanTreeNode<WeightType>[m];

LeafChars = new CharType[n];

LeafCharCodes = new String[n];

int i, p, q;

for (i = 0; i < n; i++) { // 存储叶结点信息

nodes[i].weight = w[i];

nodes[i].leftChild = -1;

nodes[i].rightChild = -1;

nodes[i].parent = -1;

LeafChars[i] = ch[i];

}

for (i = n; i < m; i++){ // 构造哈夫曼树

int r1, r2;

Select(i, r1, r2);

// 合并以r1,r2为根的树

nodes[r1].parent = nodes[r2].parent = i;// r1,r2的双亲为i

nodes[i].leftChild = r1; // r1为i的左孩子

nodes[i].rightChild = r2; // r2为i的右孩子

nodes[i].parent = -1; // i为新的根结点

nodes[i].weight = nodes[r1].weight + nodes[r2].weight;//i的权为r1,r2的权值之和

}

for (i = 0; i < n; i++) { // 求n个叶结点字符的编码

LinkList<char> charCode; // 暂存叶结点字符编码信息

q = i;

p = nodes[q].parent;

while (p != -1) { // 从叶结点到根结点逆向求编码

if (nodes[p].leftChild == q)

charCode.InsertElem(1, '0');// 左分支编码为'0'

else

charCode.InsertElem(1, '1');// 右分支编码为'1'

q = p;

p = nodes[q].parent;

}

LeafCharCodes[i] = charCode; // 复制字符编码

}

}

template <class CharType, class WeightType>

void HuffmanTree<CharType, WeightType>::Select(int n, int &r1, int &r2)

// 操作结果：在nodes[0 ~ n-1]中选择双亲为0,权值最小的两个结点r1,r2

{

r1 = r2 = -1;

for (int i = n-1; i > -1; i--) // 查找树值最小的两个结点

if (nodes[i].parent == -1) // 只处理双亲不为-1的结点

if (r2 == -1)

r2 = i;

else if (nodes[i].weight < nodes[r2].weight) {

r1 = r2;

r2 = i;

}

else if (r1 == -1 || nodes[i].weight < nodes[r1].weight)

r1 = i;

}

int main(void)

{

int w[50],num;

while(cin>>num){

int n;

for(int i=1;i<=num;i++){

cin>>n;

for(int j=0;j<n;j++) cin>>w[j];

HuffmanTree<int, int> hmTree(w, w, n);

cout<<"Case "<<i<<endl;

for(int t=0;t<n;t++){

String strTmp = hmTree.Code(t);

cout<<w[t]<<" "<<strTmp.CStr()<< endl;

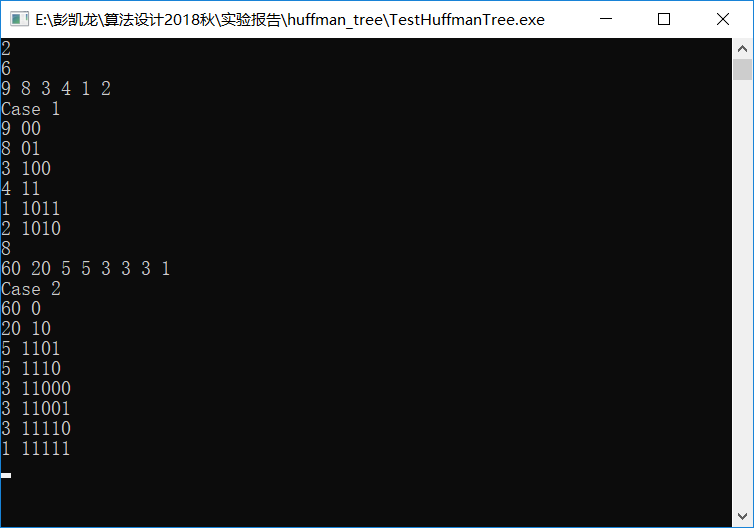
}

}

}

return 0;

}



输入中的开始行上有一个整数T，（0<T<=20），表示有T组测试数据。

接下来是T行测试数据的描述，每组测试数据有2行。测试数据的第1行上是一个正整数n，（n<50），表示序列的长度。第2行是n个字母出现的权值序列w1,w2,…,wn，它们均为正整数，相邻的两个整数之间用空格隔开。

1. **问题以及体会：**

本实验的要点在于利用哈夫曼算法的贪心选择性质构造哈夫曼树，并且根据字符所在的叶节点的位置确定该字符的编码。

通过这次实验，我进一步体会了解了贪心算法的贪心选择性质与最优子结构性质。其实哈夫曼编码并不是唯一的，其编码结构取决于构造哈夫曼树时从优先队列中选取树的方法。特别是在很多书的频率相同时，哈夫曼编码结果会有多种可能。