数据结构实验报告

**【算法说明】**

**哈夫曼编码：**

**将词频数组里的值输入到哈夫曼树的节点上作为权重，生成一堆小节点**

huffmanNode

像这样：

xn

**Weight**

**Tree**

NULL

NULL

**圈中的数字是权重，即词频**

**箭头表示指向节点的指针**

**在后面的图中字母省略**

E

B

D

C

A

1. **建立最小堆**
2. **建立新节点使新节点左右指针指向堆中最小的两个值**
3. **删除最小两个节点，将新节点推入并重新排序**
4. **建立新节点使新节点左右指针指向堆中最小的两个值**
5. **只剩下一个节点弹出，返回这个节点的tree部分**
6. **遍历哈夫曼树将每一条路径写入数组对应部分**

0

0

1

0

1

1

1

0

E

A

B

D

C

|  |  |
| --- | --- |
| A | 0000 |
| B | 0001 |
| C | 001 |
| D | 01 |
| E | 1 |

**【测试结果】**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **测 试 用 例 表** | | | | | |
|  | 全部用二进制编码 | 词频大于1的用二进制编码 | 词频大于2的用二进制编码 | 词频大于5的用二进制编码 | 原txt文件 |
| 测试输入 | output.txt | | | | |
| 文件长度 |  |  |  |  | 2,365,134 字节 |
| 压缩比例 |  |  |  |  | 1 |
| 错误原因 | 无 | 第二个文件无法读取 | 第二个文件无法读取 | 第二个文件无法读取 | 无 |
| 当前状态 | 良好 | 不报错也不输出结果 | 不报错也不输出结果 | 不报错也不输出结果 | 良好 |

**【分析与报告】**

1. 测试结果分析

第二个文件无法读取，其它分步调试均正常，试过各种方法都读取不了QAQ

1. 探讨更多问题解决的途径

用数组实现哈夫曼树的构造：

建立一个结构体，结构体中含有左右子节点、父节点以及该节点的权重

**【附录】**

#include<iostream>

#include<fstream>

#include<string>

#include<queue>

#include<functional>

using namespace std;

//建立哈希表

struct words

{

string s;

int number;

};

//建立树的节点

struct node

{

int storei;//用于储存哈希值，便于后面生成一个与二进制编码对应的哈希表

struct node\*left;

struct node\*right;

node() { left = right = NULL; }

node(node\*theleft, node\*theright)

{

left = theleft;

right = theright;

}

};

//建立树

class linktree :public node

{

public:

linktree() { root = NULL; }

~linktree() { erase(); }

void erase()

{

root = NULL;

}

void maketree(int n,linktree& righttree, linktree& lefttree)//连接两棵树的函数

{

root = new node(lefttree.root, righttree.root);

root->storei = n;

lefttree.root = righttree.root = NULL;

}

node\*root;

};

//建立有一个权重一个树的节点

struct huffmanNode

{

linktree\*tree;

int weight;

};

//重新定义堆中>操作

bool operator<(huffmanNode a, huffmanNode b)

{

return a.weight > b.weight;

}

//建立霍夫曼树

linktree\*huffmantree(words weight[], int n)

{

priority\_queue< huffmanNode>minheap;

huffmanNode\*hnode = new huffmanNode[n + 1];

linktree emptytree;

int i = 1;

for (int j = 1; j< 30000; j++)//将词频数组里的值赋给每个节点

{

if (weight[j].number != 0)//数组里有的值为空

{

hnode[i].weight = weight[j].number;

hnode[i].tree = new linktree;

hnode[i].tree->storei = j;

hnode[i].tree->maketree(j,emptytree, emptytree);//将这些叶子节点的左右指针置为空

minheap.push(hnode[i]);//将节点放入最小堆

i++;

}

}

huffmanNode w, x, y;

linktree \*z;

for (int i = 1; i < n; i++)

{

x = minheap.top();

minheap.pop();

y = minheap.top();

minheap.pop();

z = new linktree;

z->maketree(0,\*x.tree, \*y.tree);//将树合并

w.weight = x.weight + y.weight;//将权重赋给新节点

w.tree = z;

minheap.push(w);//将合成的新节点推入堆

delete x.tree;

delete y.tree;

}

return minheap.top().tree;

}

//用函数递归遍历每一个节点，并且求出每个叶子节点到根节点的路径

void makebinarycode(node \*p2, words source[],string s)

{

if (p2->left != NULL)

{

s = s + '1';

makebinarycode(p2->left, source,s);

}

if (p2->right != NULL)

{

s = s + '0';

makebinarycode(p2->right, source,s);

}

if (p2->left == NULL&&p2->right == NULL)

{

int hashnumber;

hashnumber = p2->storei;

source[hashnumber].number = stoi(s);

s = "";

}

}

words source[30000];//在main函数外建立哈希表使s和number被自动初始化

int main()

{

//打开文件

ifstream input;

ofstream output;

input.open("output.txt");//文件输入的形式为：哈希值 单词 词频用空格隔开

output.open("outfile.txt", ofstream::out);

if (!input)

cerr << "unable to open";

if (!output)

cerr << "unable to write";

string str;//用于处理字符

int hashnumber;//哈希值，数组下标

int i = 0;

int j = 0;//标记输入的值是三个值中哪一个

//将上次统计的词频以及对应的哈希值放到一个数组中

while (!input.eof())

{

getline(input, str, ' ');

if (str != "")

{

if (j % 3 == 0)//第一个输入的为哈希值

{

i = stoi(str);

}

else if (j % 3 == 1)//第二个输入为单词

source[i].s = str;

else//第三个输入为词频

source[i].number = stoi(str);

j++;

}

}

//哈夫曼树

string binarycode;

linktree\*p;

p = huffmantree(source, 19229);

makebinarycode(p->root, source,binarycode);//通过迭代得到每个单词的二进制编码

input.open("input.txt");

if (!input)

cerr << "unable to open";

char chr;//储存字符

int lastchr;//用于检测全角的单引号

int first;//第一个字母0-26

int fourth;//第四个字母0-26

int wordlen;//单词长度

while (input.eof())

{

str = "";

getline(input, str, ' ');

while (input.eof())

{

input.get(chr);

if (chr < 65)//65以下的都不是字母

{

if (str != "" && chr == '-')//保留“-”

str = str + chr;

else if (str != "" && (chr == -95 || chr == -81))//全角单引号用两个chr值来检测

{

if (chr == -95)

lastchr = 1;

else if (chr == -81 && lastchr == 1)

str = str + "'";//将全角单引号转换成半角

else

break;//如果只有一半则不是单引号

}

else

break;//其他符号表示这个单词已经结束，跳出循环

}

else

str = str + chr;

}

if (str != "")//避免开头就是符号或者多个符号相连

{

if (str[0] < 96)//大写字母

first = (int)str[0] - 65;

else

first = (int)str[0] - 96;

//第四位字符

wordlen = str.length();

if (wordlen > 3 && str[3] > 64)//判断第四位字符是否为字母

{

if (str[3] > 96)//小写字母

fourth = (int)str[3] - 96;

else

fourth = (int)str[3] - 65;

}

else//当第四位字母不存在或者为连字符或单引号是，将第四位的值看作是0

fourth = 0;

//计算哈希值

hashnumber = first \* 1000 + fourth \*wordlen \* 10 + wordlen;

//哈希值为第一位字符\*1000+第四位\*单词长度\*10+单词长度

//对文本进行编码并输出

if (source[hashnumber].s == str&&source[hashnumber].number > 1)//改变此处改变不进行二进制编码的词频

output << source[hashnumber].number;

else if (source[hashnumber].number < 1)

output << source[hashnumber].s;

else

{

while (source[hashnumber].s != str)

hashnumber++;

output << source[hashnumber].number;

}

}

}

}