|  |
| --- |
| 中南大学  《数据结构》课程实验  实验报告  实验题目 Huffman 编码  专业班级 软件工程2005班  学 号 8209200504  姓 名 李均浩  **实验成绩：**  **批阅教师：**  2021年5月5日 |

一、需求分析

1.程序任务

1、根据所提供的字母数据建立一个Huffman树；

2、根据生成的Huffman树的结构，显示输出所有字母的Huffman编码。

3、根据产生的Huffman编码，实现Huffman编/译码器。

2.输入以及输出的形式



图1 程序输入输出形式

3.程序功能

对各个规定的字符进行Huffman编码，对英文原文进行编码或者对Huffman编码进行解码。实现Huffman编码的编码器以及解码器。

4.测试数据

(a) data structure is fantastic

预期输出：010010101001001010001100001011011111011111001011110110110100010011100000001100010110000010010111000010100111111

(b) I LOVE Programing

预期输出：00010010000100001100011111101000011101110101100111101101010100110110011000011110

(c) CSU is Number ONE

预期输出：1111111001111000010011100000100011110001101011111101110100001101000101

(d) 10010000100001100011111101000011100011011110

预期输出：i love you

(e) 1001000100010100111111011101000011000111010000000101000011110100111010100000000110011011001101100001001000011111111100010000100100011111010110000001000101001110000010110000000110011111001111011110111111001000001111101110000011111011101011101

预期输出：i never own a girl friend but i can new an object by cpp

(f) (错误输入) 45456

预期输出：未知编码阻止了解码程序的运行！（随后退出程序）

(g) (错误输入) 01011011101010101011

预期输出：未知编码阻止了解码程序的运行！（随后退出程序）

二、概要设计

1.抽象数据类型定义：

ADT Huffmantree

{

数据对象：D={ai| ai ∈Charset,i=1,2,3,„„n,n≥0}

数据关系：R1＝{< ai-1, ai >| ai-1, ai ∈D, i=2,3,„„n}

基本操作：

Initialization(&HT，&HC,w，n,ch)

操作结果：根据n个字符及其它们的权值w[i],建立Huffman树HT,用字符数组ch[i]作为中间存储变量，最后字符编码存到HC中；

Encodeing(n)

操作结果：根据建好的Huffman树，对文件进行编码，编码结果存入到文件CodeFile中；

Decodeing(HT,n)

操作结果：根据已经编译好的包含n个字符的Huffman树HT，将文件的代码进行翻译，结果存入文件TextFile中。

} ADT Huffmantree

2.主程序的流程



图2 主程序的流程

三、详细设计

1.模块伪码

(1) void GetAlphabetFreq(TreeNode\* node\_array) 开始

输出<< "\*\*\*\*\*\*\*\*\*\*获取字母频度表\*\*\*\*\*\*\*\*\*\*" << endl;

auto sp = (TreeNode)malloc(sizeof(HuffmanTreeNode));

如果(sp == 空指针)

退出程序(OVERFLOW);

sp->letter = ' ';

输出<< "请输入\' 空格 \'的权值：";

输入>> sp->weight;

sp->isDeleted = false;

sp->left = nullptr;

sp->right = nullptr;

sp->pre\_order\_counted\_times = 0;

node\_array[0] = sp;

for (int i = 0; i < 26; ++i) {

auto tn = (TreeNode)malloc(sizeof(HuffmanTreeNode));

if (tn == nullptr)

exit(OVERFLOW);

tn->letter = (char)('a' + i);

输出 << "请输入\' " << tn->letter << " \'的权值：";

输入 >> tn->weight;

tn->isDeleted = false;

tn->left = nullptr;

tn->right = nullptr;

tn->pre\_order\_counted\_times = 0;

node\_array[i + 1] = tn;

}

输出 << endl << "\*\*\*\*\*\*\*\*\*\*获取结束\*\*\*\*\*\*\*\*\*\*" << endl;

array\_length = 27;

结束

(2) TreeNode GetMinWeight(TreeNode\* node\_array) 开始

int min\_index = 10000;

int min\_weight = 10000;

for (int i = 0; i < array\_length; ++i) {

如果 (node\_array[i]->weight < min\_weight && !node\_array[i]->isDeleted) {

min\_weight = node\_array[i]->weight;

min\_index = i;

}

}

node\_array[min\_index]->isDeleted = true;

返回 node\_array[min\_index];

结束

(3) bool isEmpty(TreeNode\* node\_array) 开始

for (int i = 0; i < array\_length; ++i)

如果 (!node\_array[i]->isDeleted)

返回 false;

返回 true;

结束

(4) void SaveNodeToArray(TreeNode new\_node, TreeNode\* node\_array) 开始

node\_array[array\_length] = new\_node;

array\_length++;

结束

(5) void CreateHuffmanTree(HuffmanTree& t, TreeNode\* node\_array) 开始

TreeNode parent\_node;

(无限循环) {

TreeNode node\_1, node\_2;

node\_1 = GetMinWeight(node\_array);

node\_2 = GetMinWeight(node\_array);

parent\_node = 分配类型为(TreeNode)大小为(sizeof(HuffmanTreeNode))的内存空间;

如果 (parent\_node == 空指针)

退出程序(OVERFLOW);

parent\_node->isDeleted = false;

parent\_node->left = node\_1;

parent\_node->right = node\_2;

parent\_node->letter = '#';

parent\_node->weight = node\_1->weight + node\_2->weight;

parent\_node->pre\_order\_counted\_times = 0;

如果 (isEmpty(node\_array))

退出循环;

SaveNodeToArray(parent\_node, node\_array);

}

t.root = parent\_node;

结束

(6) void HuffmanCodeGenerator(HuffmanTree t, string\* huffman\_code\_list) 开始

string s;

s = "";

TreeNode n = t.root;

Stack node\_stack;

初始化栈(node\_stack);

无限循环 {

如果 (n == t.root) {

如果 (n->pre\_order\_counted\_times == 0) {

n->pre\_order\_counted\_times++;

Push(node\_stack, n);

n = n->left;

s += '1';

}

如果 (n->pre\_order\_counted\_times == 1) {

n->pre\_order\_counted\_times++;

入栈(node\_stack, n);

n = n->right;

s += '0';

}

如果 (n->pre\_order\_counted\_times == 2)

break;

}

否则 {

如果 (n->letter == '#') {

如果 (n->pre\_order\_counted\_times == 0) {

n->pre\_order\_counted\_times++;

入栈 (node\_stack, n);

n = n->left;

s += '1';

}

如果 (n->pre\_order\_counted\_times == 1) {

n->pre\_order\_counted\_times++;

入栈 (node\_stack, n);

n = n->right;

s += '0';

}

如果 (n->pre\_order\_counted\_times == 2) {

出栈 (node\_stack, n);

s = s.substr(0, s.length() - 1);

}

}

否则 {

std::cout << '\'' << n->letter << '\'' << "的权值为：" << n->weight << ",Huffman编码为：" << s << endl;

//n->huffman\_code = (string\*)malloc(100);

//\*(n->huffman\_code) = s;

如果 (n->letter != ' ')

huffman\_code\_list[n->letter - 'a' + 1] = s;

如果 (n->letter == ' ')

huffman\_code\_list[0] = s;

否则 (node\_stack, n);

s = s.substr(0, s.length() - 1);

}

}

}

结束

(7) string SearchHuffmanCode(char c, string\* huffman\_code\_list) 开始

返回 huffman\_code\_list[c - 'a' + 1];

结束

(8) string HuffmanEncoder(string plaintext, string\* huffman\_code\_list) 开始

将plaintext全部转为小写;

string ciphertext;

for (char i : plaintext) {

如果 (i == ' ')

ciphertext += huffman\_code\_list[0];

否则

ciphertext += SearchHuffmanCode(i, huffman\_code\_list);

}

返回 ciphertext;

结束

(9) string HuffmanDecoder(const string& ciphertext, string\* huffman\_code\_list) {

string plaintext;

int length = 1; //10010000100001100011111101000011100011011110

int start\_index = 0;

bool isMatch = false;

无限循环 {

如果 (start\_index + length > ciphertext.length())

返回 plaintext;

string temp = ciphertext.substr(start\_index, length);

for (int i = 0; i < 27; ++i) {

如果 (temp == huffman\_code\_list[i]) {

isMatch = true;

如果 (i == 0)

plaintext += ' ';

否则

plaintext += 转为(char)(i + 'a' - 1);

}

}

如果 (isMatch) {

start\_index += length;

length = 1;

isMatch = false;

}

否则{

如果 (start\_index + length >= ciphertext.length()) {

输出 << "未知编码阻止了解码程序的运行！" << endl;

退出程序(错误代码：INVALID\_INPUT);

}

length++;

}

}

结束

(10)string GetBinCode(const string& s) 开始

string bin\_code;

输出 << "\*\*\*\*\*\*\*\*\*\*原文的二进制编码\*\*\*\*\*\*\*\*\*\*" << endl;

for (char i : s)

{

char temp[100];

将第s的第i位转为二进制编码存入temp(i, temp, 2);

输出 << i << '\t' << temp << endl;

bin\_code += temp;

}

返回 bin\_code;

结束

2.函数调用关系图