哈尔滨工业大学

数据科学与大数据技术专业

实验报告

课程名称：计算机算法基础

实验项目：哈夫曼编码

实验题目：哈夫曼编码与译码方法

实验日期：2019.12.11

班级：1534701

学号姓名：1153470114 薛博洋

1153470104 沈孙乐

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 张岩 |

1. **实验目的**

哈夫曼编码是一种以哈夫曼树为基础的基于统计学的变长编码方式。其基本思想是：将使用次数多的代码转换成长度较短的编码，而使用次数少的采用较长的编码，并且保持编码的唯一可解性。在计算机信息处理中，经常应用于数据压缩。是一种一致性编码法，用于数据的无损耗压缩。本实验要求利用贪心算法实现一个完整的哈夫曼编码与译码系统。

**二、实验要求及实验环境**

1. 从文件中读入任意一篇英文文本文件，分别统计英文文本文件中各字符（包

括标点符号和空格）的使用频率；

2. 根据已统计的字符使用频率构造哈夫曼编码树，并给出每个字符的哈夫曼编

码（字符集的哈夫曼编码表）；

3. 将文本文件利用哈夫曼树进行编码，存储成压缩文件（哈夫曼编码文件）；

4. 计算哈夫曼编码文件的压缩率；

5. 将哈夫曼编码文件译码为文本文件，并与原文件进行比较。

以下可以不做，供思考，做了可以适当加分

6. 能否利用堆结构，优化的哈夫曼编码算法。

7. 上述1-5 的编码和译码是基于字符的压缩，考虑基于单词的压缩，完成上述

工作，讨论并比较压缩效果。

8. 上述1-5 的编码是二进制的编码，可以采用K 叉的哈夫曼树完成上述工作，

实现“K 进制”的编码和译码，并与二进制的编码和译码进行比较。

**三、设计的基本思想和算法的原理**

利用哈夫曼编码方法的贪心选择性，得到一棵最优化的前缀树。

贪心思想：循环地选择具有最低频数的两个节点，生成子树，直到形成一棵树。

其中有几个步骤：

1. 读取文本字节，计算出现的频率与出现的字符

2.利用频率数据构造哈夫曼静态三叉列表。

3.利用得到的哈夫曼表，回溯出所有字符的哈夫曼编码。

4.利用编码对文件进行无损的压缩。

5.再利用编码和列表进行译码操作。

**四、实验结果与分析**

最终得到的哈夫曼静态三叉列表

0 10 weight: 0.094540 parent: 66 lchild: -1 rchild: -1

1 32 weight: 13.944694 parent: 119 lchild: -1 rchild: -1

2 34 " weight: 0.189081 parent: 74 lchild: -1 rchild: -1

3 39 ' weight: 0.165446 parent: 73 lchild: -1 rchild: -1

4 40 ( weight: 0.283621 parent: 80 lchild: -1 rchild: -1

5 41 ) weight: 0.283621 parent: 78 lchild: -1 rchild: -1

…… ……

55 116 t weight: 5.932404 parent: 112 lchild: -1 rchild: -1

56 117 u weight: 2.268967 parent: 103 lchild: -1 rchild: -1

57 118 v weight: 0.590877 parent: 90 lchild: -1 rchild: -1

58 119 w weight: 0.921768 parent: 94 lchild: -1 rchild: -1

59 120 x weight: 0.118175 parent: 69 lchild: -1 rchild: -1

60 121 y weight: 0.921768 parent: 95 lchild: -1 rchild: -1

61 122 z weight: 0.236351 parent: 78 lchild: -1 rchild: -1

62 0 weight: 0.070905 parent: 65 lchild: 27 rchild: 33

63 0 weight: 0.094540 parent: 68 lchild: 20 rchild: 52

64 0 weight: 0.141810 parent: 72 lchild: 12 rchild: 28

65 0 weight: 0.141810 parent: 72 lchild: 32 rchild: 62

66 614 f weight: 0.189081 parent: 74 lchild: 0 rchild: 14

67 0 weight: 0.189081 parent: 75 lchild: 16 rchild: 23

68 62 > weight: 0.212716 parent: 77 lchild: 63 rchild: 25

69 0 weight: 0.236351 parent: 77 lchild: 13 rchild: 59

70 0 weight: 0.283621 parent: 80 lchild: 17 rchild: 26

…… ……

116 0 weight: 20.515245 parent: 120 lchild: 109 rchild: 110

117 0 weight: 23.422359 parent: 120 lchild: 111 rchild: 112

118 0 weight: 25.785866 parent: 121 lchild: 113 rchild: 114

119 0 weight: 30.276530 parent: 121 lchild: 1 rchild: 115

120 0 weight: 43.937603 parent: 122 lchild: 116 rchild: 117

121 0 weight: 56.062397 parent: 122 lchild: 118 rchild: 119

122 0 weight: 100.000000 parent: -1 lchild: 120 rchild: 121

可以看到最终所有的节点都汇聚到了最后的节点122 上。

最后的编码结果如下：

: 0011101110

: 110

": 001110110

': 1111111110

(: 111110110

): 111101111

,: 1111100

-: 101110000

.: 11111110

0: 111110100

1: 0011100

2: 00111100

3: 11111100100

4: 1011100110

5: 0011101111

6: 001111111

7: 0011111100

8: 1111101110

9: 11110100

A: 111110101

B: 10111001000

C: 00111010

D: 00111101

E: 0011111101

H: 1111111111

I: 1011100101

M: 1111101111

N: 111111001110

O: 11111100101

P: 1111110000

S: 101110001

T: 1111110001

U: 11111100110

W: 111111001111

[: 11110101

]: 11110110

a: 0100

b: 1011110

c: 10110

d: 01100

e: 1110

f: 111100

g: 00110

h: 00011

i: 1010

j: 111111110

k: 00111110

l: 00010

m: 01101

n: 1000

o: 0101

p: 00001

q: 10111001001

r: 1001

s: 0010

t: 0111

u: 00000

v: 11111101

w: 1011101

x: 1011100111

y: 1011111

z: 111101110

实验选取的原有文本大小为4235byte,如果用哈夫曼编码来表达则应占用2465byte,压缩率约为58.2%。

**五、经验体会与不足**

1、c语言编程能力仍然需要提高，在文件的读取和写入方面仍然有较大优化空间。

2、其实应该有更好的方法来保存大小写字母，因为大小写字母是对于编码很大的浪费，但是可能在本题中会使编码不具备连续性。

3、最终编码的结果应该以二进制保存是最小的但是这里只保存为’010111…’的字符形式，而计算的时候以除以8倍来计算理论压缩值。

4、文本字节处理时，回车号特别容易出错。

**六、附录：源代码（带注释）**

/\* to count how many type of words in the file

\* as well as count how many s in the file

\* count the percentage of letters in the file

\*/

void getTxtData(FILE \*fp, double rate[]){

/\*initiate rate[]\*/

for(int i = 0; i < 127; i++){

rate[i] = 0;

}

/\*if we meet '10', the text would change line and can't get by a single fgets\*/

char buf[S];

/\* use a while loop to read txt with line break!!!

\* though I still don't understand why it could do it

\*/

int sum = 0;

while (fgets(buf, S, fp) != NULL){

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

if(buf[i] == 10){

sum += 1;

rate[buf[i]] += 1;

break;

}

sum += 1;

//printf("%d", buf[i]);

rate[buf[i]] += 1;

}

}

double summary;

/\*calculate the percentage of every letter\*/

for(int i = 0; i < 127; i++){

rate[i] = rate[i] / sum \* 100;

summary += rate[i];

}

}

#include <stdio.h>

#include <string.h>

#include <math.h>

#define N 26

#define S 1500

/\*define the sturct of huffman tree\*/

typedef struct{

double weight;

int lchild;

int rchild;

int parent;

}HTNODE;

/\*print HTNODE H[]\*/

void printH(HTNODE H[], int length, int letter[]){

int len = 2 \* length - 1;

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

printf("%d %d %c weight: %f parent: %d\tlchild: %d\trchild: %d\n"

, i, letter[i], letter[i], H[i].weight, H[i].parent, H[i].lchild, H[i].rchild);

}

printf("\n\n");

}

/\*initiate the huffman tree\*/

void initH(HTNODE H[], int length){

int len = 2 \* length - 1;

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

H[i].weight = 0;

H[i].lchild = H[i].rchild = H[i].parent = -1;

}

}

/\* build the huffman tree \*/

void buildH(HTNODE H[], int length){

int maxNode = length;

for(int i = 0; i < length-1; i++){

/\*find the smallest two number\*/

double smallFirst = 100;

int first = -1;

double smallSecond = 100;

int second = -1;

for(int j = 0; j < maxNode; j++){

if(H[j].parent == -1){

/\*inner compare lines\*/

if(H[j].weight < smallSecond){

if(H[j].weight < smallFirst){

smallFirst = H[j].weight;

first = j;

//printf("%d, %f\n", j, H[j].weight);

}else{

smallSecond = H[j].weight;

second = j;

}

}

}

}

/\* change two smallest's parents

\* add a new node

\* give parent's l&r child

\*/

H[maxNode].weight = smallSecond + smallFirst;

H[maxNode].lchild = first;

H[maxNode].rchild = second;

H[first].parent = H[second].parent = maxNode;

maxNode++;

}

}

/\*define the huffmanCode table\*/

typedef struct{

char ch;

char bits[62+1];

}codeNode;

/\*input: HuffmanTree output: Hufftable\*/

void buildTable(HTNODE H[], codeNode T[], int length, int letter[]){

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

int track = i;

T[i].ch = letter[i];

char code[length + 1];

int count = length;

code[count] = '\0';

int p;

/\*give the code in a reverse way\*/

while(H[track].parent != -1){

count--;

p = H[track].parent;

if(H[p].lchild == track){

code[count] = '0';

}else{

code[count] = '1';

}

track = p;

}

/\*OMG use the strcpy to copy the string right on the situation where loop ends\*/

strcpy(T[i].bits, &code[count]);

}

}

/\*print the huffman table\*/

void printTable(codeNode T[], int length){

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

printf("%c: %s\n", T[i].ch, T[i].bits);

}

}

/\*write data from a plain .txt to a new encoding .txt\*/

void HuffWrite(FILE \*origin, FILE \*target, codeNode T[], int length){

/\*if we meet '10', the text would change line and can't get by a single fgets\*/

char buf[S];

while (fgets(buf, S, origin) != NULL){

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

int j;

for(j = 0; j < length; j++){

/\*again!!! get wrong with '==' and '='\*/

if(T[j].ch == buf[i]){

break;

}

}

if(buf[i] == 10){

fprintf(target, "%s", T[j].bits);

break;

/\* a confusing situation!

\* if

\*/

}else{

fprintf(target, "%s", T[j].bits);

}

}

}

}

/\*transform the encoding file back to text form\*/

void decodeH(FILE \*code, FILE \*restore, codeNode T[], HTNODE H[], int length, int letter[]){

char buf[40 \* S];

int len = 2 \* length - 2;

while(fgets(buf, 40 \* S, code) != NULL){

int parent = len;

for(int i = 0; i < 40\*S; i++){

if(buf[i] == NULL){

return;

}

if(buf[i] == '1'){

parent = H[parent].rchild;

}else if(buf[i] == '0'){

parent = H[parent].lchild;

}

if(H[parent].lchild == -1){

//printf("parent: %c\t", letter[parent]);

fprintf(restore, "%c", letter[parent]);

parent = len;

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* main funtion:

\* open file - distill data - build huffman tree - get huffman code

\* - compress txt - calculate rates – decode & compare

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void){

/\* get the data of the text

\* change the array of letter rates

\*/

FILE \*file = NULL;

file = fopen("test.txt", "r");

double wordsRate[127];

getTxtData(file, wordsRate);

/\* clear the data make sure every line is filled

\* first get sum and make huffman tree

\*/

int length = 0;

for(int i = 0; i < 127; i++){

if(wordsRate[i] != 0.0){

length++;

}

}

/\* struct huffman tree

\* word array records the chars appear in the text

\* HuffmanT get the percentages stored in wordsRate according the sequence of count

\*/

int word[length];

HTNODE HuffmanT[2 \* length-1];

initH(HuffmanT, length);

int count = 0;

double summary;/\*only to test the correctness of distill\*/

for(int i = 0; i < 127; i++){

if(wordsRate[i] != 0.000){

summary += wordsRate[i];

word[count] = i;

HuffmanT[count].weight = wordsRate[i];

count++;

}

}

//printH(HuffmanT, length, word);

/\*then implement huffman algorithm get the huffman table\*/

buildH(HuffmanT, length);

printH(HuffmanT, length, word);

/\*get the huffman code table\*/

codeNode HuffTable[length];

buildTable(HuffmanT, HuffTable, length, word);

printTable(HuffTable, length);

/\*transform the test.txt into a compressed file\*/

fclose(file);

/\*

\* open a new fp to iterate the file!

\*/

FILE \*fp = fopen("test.txt", "r");

FILE \*fpWrite = fopen("comp.txt", "w");

HuffWrite(fp, fpWrite, HuffTable, length);

fclose(fpWrite);

/\*the last part, to decode a encode text and restore the text\*/

FILE \*fpCode = fopen("comp.txt", "r");

FILE \*fpDecode = fopen("restore.txt", "w");

decodeH(fpCode, fpDecode, HuffTable, HuffmanT, length, word);

}