Huffman树及Huffman编码的算法实现

姓名：岳宇轩 班级：2019级慧与卓越工程师班 指导老师：纪筱鹏

日期：2021年4月16日

# 实验题目

Huffman 树及 Huffman 编码的算法实现

# 实验目的

* 1. 了解该树的应用实例，熟悉掌握 Huffman 树的构造方法及 Huffman 编码的应用，
  2. 了解 Huffman 树在通信、编码领域的应用过程。

# 实验要求

* 1. 输入一段100—200字的英文短文，存入一文件 中。



a

* 1. 写函数统计短文出现的字母个数 及每个字母的出现次数
  2. 写函数以字母出现次数作权值，建 Huffman 树（ 个叶子），给出每个字母的

Huffman 编码。

* 1. 用每个字母编码对原短文进行编码，码文存入文件 中。



b



b



c

* 1. 用 Huffman 树对

中码文进行译码，结果存入文件

中，比较 a 、 c 是否一

致，以检验编码、译码的正确性。

# 实验内容和实验步骤

## 需求分析

陈述程序设计的任务，强调程序要做什么，明确规定：

* + 1. 输入的形式和输入值的范围；
    2. 输出的形式；
    3. 程序所能实现的功能；

## 概要设计

### 数据结构定义

Huffman 树定义

1 typedef struct { 2 // 字 符

1. char letter;
2. // 权 重
3. unsigned int weight;
4. // 父节点、左孩子、右孩子下标
5. unsigned int parent, left, right;
6. } Node, \*Tree;

Huffman 编码定义

1 typedef char \*Code;

### 主程序流程

开始

输入英文短文

构造 Huffman 树

HuffmanTree

获取字符 Huffman 编码

HuffmanTree

HuffmanCode

短文编码并写入文件中

译码并写入文件中

结束

原文与解码后短文比较

### 各程序模块之间的调用关系

main

CreateHuffmanCode

InputAndSave

|  |  |  |
| --- | --- | --- |
| CreateHuffmanTree |  | Select |
|  |

Compare

Decode

Encode

* 1. 详细设计
     1. 主程序入口

1

2

3

4

5

6

7

8

9

10

11

12

13

14

int main() {

// 输入英文文章并保存到 a.txt 中

InputAndSave("a.txt");

// 构 造 Huffman 树

CreateHuffmanTree("a.txt", ht, n);

// 获取 Huffman 编码

CreateHuffmanCode(hc, ht, n);

// 短文编码

Encode("a.txt", "b.txt", hc);

// 短文解码

15

16

17

18

19

20

21

22

23

24

Decode("b.txt", "c.txt", ht, n);

// 原文与解码后短文比较

printf("a.txt is %s to c.txt",

Compare("a.txt", "c.txt") ? "equal" : "NOT equal");

// 释放空间free(ht); return 0;

}

* + 1. 文章读入

1. void InputAndSave(const char \*filename) {
2. // 新建 / 打开文件
3. FILE \*fp = fopen(filename, "w");
4. printf("Please input an essay, end with an enter: ");
5. while (true) {
6. // 从 stdin 中读取一个字符
7. char ch = getchar();
8. // 若字符为'\n'，退出循环

9 if (ch == '\n')

1. break;
2. // 写入文件中
3. fputc(ch, fp); 13 }
4. // 关闭文件
5. fclose(fp); 16 }
   * 1. 构造 **Huffman** 树

以 为例

Hello, World!

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **index** | **letter** | **weight** | **parent** | **left** | **right** |
| 1 | 空格 | 1 | 11 | 0 | 0 |
| 2 | ! | 1 | 11 | 0 | 0 |
| 3 | , | 1 | 12 | 0 | 0 |
| 4 | H | 1 | 12 | 0 | 0 |
| 5 | W | 1 | 13 | 0 | 0 |
| 6 | d | 1 | 13 | 0 | 0 |
| 7 | e | 1 | 14 | 0 | 0 |
| 8 | l | 3 | 17 | 0 | 0 |
| 9 | o | 2 | 15 | 0 | 0 |
| 10 | r | 1 | 14 | 0 | 0 |
| 11 |  | 2 | 15 | 2 | 1 |
| 12 |  | 2 | 16 | 4 | 3 |
| 13 |  | 2 | 16 | 6 | 5 |
| 14 |  | 2 | 17 | 10 | 7 |
| 15 |  | 4 | 18 | 11 | 9 |
| 16 |  | 4 | 18 | 13 | 12 |
| 17 |  | 5 | 19 | 14 | 8 |
| 18 |  | 8 | 19 | 16 | 15 |
| 19 |  | 13 | 0 | 17 | 18 |

1. void CreateHuffmanTree(const char \*filename, Tree &ht, int &n)

{

1. // 从文件中逐个读取字符并计数
2. FILE \*fp = fopen(filename, "r"); 4 int count[128] = {0};

5 n = 0;

1. while (true) {
2. char ch = fgetc(fp);
3. if (ch == EOF)
4. break;
5. if (count[ch] == 0)

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

n += 1;

count[ch] += 1;

}

fclose(fp);

// 申请 Huffman 树空间

ht = (Tree)malloc(2 \* n \* sizeof(Node));

// 不使用 0 号单元

// 将字符及权重存入树中

// 置 parent, left, right 为 0

Tree p = ht + 1;

for (int i = 0; i < 128; i++) {

// ......

}

// 构 建 Huffman 树

for (int i = n + 1; i < 2 \* n; i++) {

// 在 ht[1..i-1] 中选择 parent=0 且 weight 最小的两个结点

int s1, s2;

Select(ht, i - 1, s1, s2);

// 更新子节点与父节点的信息

// ......

}

}

void Select(Tree ht, int n, int &s1, int &s2) {

// ......

}

* + 1. 获取 **Huffman** 编码

1

2

3

4

5

6

7

8

9

10

11

void CreateHuffmanCode(Code hc[], Tree ht, int n) {

// 编码临时存储空间

char \*cd = (char \*)malloc(n \* sizeof(char));

cd[n - 1] = '\0';

// 从叶子节点逆向求编码

// ......

// 释放临时存储空间

free(cd);

}

### 短文编码

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

void Encode(const char \*src, const char \*dst, Code hc[]) {

FILE \*fsrc = fopen(src, "r"); FILE \*fdst = fopen(dst, "w");

while (true) {

// 从 src 中 读 取 字 符 char ch = fgetc(fsrc); if (ch == EOF)

break;

// 查询编码后写入 dst

// ......

}

fclose(fsrc); fclose(fdst);

}

* + 1. 短文解码

1

2

3

4

5

6

7

8

9

10

11

void Decode(const char \*src, const char \*dst, Tree ht, int n)

{

FILE \*fsrc = fopen(src, "r"); FILE \*fdst = fopen(dst, "w");

// 寻找叶子节点

// 到达叶子结点后将对应字符写入 dst 中

// ......

fclose(fsrc); fclose(fdst);

}

* + 1. 原文与解码后短文比较

1

2

3

4

5

6

7

8

bool Compare(const char \*first, const char \*second) {

FILE \*f1 = fopen(first, "r"); FILE \*f2 = fopen(second, "r");

// 逐字符比较两个文件

while (!feof(f1) && !feof(f2)) { char c1 = fgetc(f1);

char c2 = fgetc(f2);

9

10

11

12

13

14

15

16

17

18

19

20

21

22

// 字符不相同，跳出循环

if (c1 != c2)

break;

}

// 两个文件未同时到达末尾

int res = true;

if (!feof(f1) || !feof(f2)) res = false;

fclose(f1); fclose(f2); return res;

}

* 1. 调试分析
     1. 调试过程中所遇到的问题及解决方法
     2. 算法的时空分析

|  |  |
| --- | --- |
| 时间复杂度 | 函数名 |
|  |  |
|  |  |
|  |  |

# 实验用测试数据和相关结果分析

## 实验结果

列出对于给定的输入所产生的输出结果。若可能，测试随输入规模的增长所用算法的实际运行时间的变化。

## 实验总结

有关实验过程中的感悟和体会、经验和教训等。