**《数据结构》**

**课程设计报告**

|  |  |
| --- | --- |
| 课程名称： | 《数据结构》课程设计 |
| 课程设计题目： | 哈夫曼树 |
| 姓 名： | 易子顺 |
| 院 系： | 计算机学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 23052314 |
| 学 号： | 23050411 |
| 指导教师： | 葛瑞泉 |

2019年12月25日

1. **需求分析**

#### 功能需求：

设计并实现一个写一个哈夫曼码的编/译码系统

#### 界面需求：

（1）I：初始化（Initialization）。从终端读入字符集大小n，以及n个字符和n个权值，建立哈夫曼树，并将它存于文件hfmTree中；

（2）E：编码（Encoding）。利用以建好的哈夫曼树（如不在内存，则从文件hfmTree中读入），对文件ToBeTran中的正文进行编码，然后将结果存入文件CodeFile中；

（3）D：译码（Decoding）。利用已建好的哈夫曼树将文件CodeFile中的代码进行译码，结果存入文件TextFile中；

（4）P：印代码文件（Print）。将文件CodeFile以紧凑格式显示在终端上，每行50个代码。同时将此字符形式的编码文件写入文件CodePrin中；

（5）T：印哈夫曼树（Tree printing）。将已在内存中的哈夫曼树以直观的方式（树或凹入表形式）显示在终端上，同时将此字符形式的哈夫曼树写入文件TreePrint中。

#### 模块层次结构设计

#### parse\_args(argc, argv);

#### 接口设计

int hfm\_init();

void encode();

void decode();

void print\_code();

数据结构设计

typedef struct BNode {

int val;

char c;

struct BNode \*left;

struct BNode \*right;

struct BNode \*parent;

char \*name;

} BNode;

typedef struct Tree {

BNode \*root;

//meta

int depth;

char \*name;

} BTree;

BTree \*btree\_init(char \*name);

void btree\_print(BNode \*, int);

void btree\_fprint(BNode \*root, int depth);

BNode \*bnode\_init(int val, char \*name, char);

int btree\_insert(BTree \*t, BNode \*victim, BNode \*parent, int val, char \*name);

**详细设计**

**#include <hfm.h>**

**/\***

**设计并实现一个写一个哈夫曼码的编/译码系统，系统功能包括：**

**（1）I：初始化（Initialization）。从终端读入字符集大小n，以及n个字符和n个权值，建立哈夫曼树，并将它存于文件hfmTree中；**

**（2）E：编码（Encoding）。利用以建好的哈夫曼树（如不在内存，则从文件hfmTree中读入），对文件ToBeTran中的正文进行编码，然后将结果存入文件CodeFile中；**

**（3）D：译码（Decoding）。利用已建好的哈夫曼树将文件CodeFile中的代码进行译码，结果存入文件TextFile中；**

**（4）P：印代码文件（Print）。将文件CodeFile以紧凑格式显示在终端上，每行50个代码。同时将此字符形式的编码文件写入文件CodePrin中；**

**（5）T：印哈夫曼树（Tree)**

**\*/**

**int parse\_args(int argc, char \*argv[]);**

**int main(int argc, char \*argv[]){**

**parse\_args(argc, argv);**

**return 0;**

**}**

**#include <hfm.h>**

**FILE \*hfmtree;**

**typedef struct character {**

**char c;**

**int val;**

**BNode \*node;**

**bool is\_use;**

**}character;**

**typedef struct chartop{**

**int n;**

**character \*chars;**

**}chartop;**

**BNode \*construct(chartop, int \*);**

**void encode\_tree(BTree \*tree);**

**void generateCode(BNode \*node, char \*code, int depth);**

**int hfm\_init(){**

**//get data**

**chartop ct;**

**printf("n: ");**

**if (scanf("%d", &ct.n) != 1 || ct.n <= 0) {**

**panic("Invalid input for n");**

**}**

**ct.chars = malloc(sizeof(character) \* ct.n);**

**if (ct.chars == NULL) {**

**panic("Memory allocation failed");**

**}**

**for(int i = 0; i < ct.n; i++){**

**printf("char%d: ", i);**

**while (getchar() != '\n');**

**scanf("%c", &ct.chars[i].c);**

**}**

**for(int i = 0; i < ct.n; i++){**

**printf("val%d: ", i);**

**if (scanf("%d", &ct.chars[i].val) != 1) {**

**panic("Invalid input for value");**

**}**

**ct.chars[i].node = NULL;**

**ct.chars[i].is\_use = false;**

**}**

**printf("construct tree\n");**

**//construct tree**

**BTree \*tree = btree\_init("hfm");**

**tree->root = construct(ct, &tree->depth);**

**printf("tree depth: %d\n", tree->depth);**

**btree\_print(tree->root, tree->depth);**

**btree\_fprint(tree->root, tree->depth);**

**encode\_tree(tree);**

**return 0;**

**}**

**void encode\_tree(BTree \*tree){**

**hfmtree = fopen("./file/hfmtree", "w");**

**char codeBuffer[MAX\_CHAR];**

**generateCode(tree->root, codeBuffer, 0);**

**}**

**void generateCode(BNode \*node, char \*code, int depth){**

**if(node == NULL)**

**return;**

**if(node->left == NULL && node->right == NULL){**

**int c = node->c;**

**char \*encode = strdup(code);**

**fprintf(hfmtree, "%c %s\n", c, encode);**

**return;**

**}else{**

**code[depth] = '0';**

**generateCode(node->left, code, depth + 1);**

**code[depth] = '1';**

**generateCode(node->right, code, depth + 1);**

**}**

**}**

**BNode \*construct(chartop ct, int \*depth){**

**int min1=0, min2=1;**

**int min1val = ct.chars[0].val;**

**int min2val = ct.chars[1].val;**

**int some\_not\_use = 0;**

**\*depth += 1;**

**while(1){**

**some\_not\_use = 0;**

**//find min1 and min2**

**for(int i = 0;i < ct.n; i++){**

**if(ct.chars[i].is\_use == false){**

**some\_not\_use ++;**

**if(min1 == -1 || min1val == -1){**

**min1 = i;**

**min1val = ct.chars[i].val;**

**}**

**if(ct.chars[i].val < min1val){**

**min1val = ct.chars[i].val;**

**min1 = i;**

**}**

**}**

**}**

**for(int i = 0;i < ct.n; i++){**

**if(ct.chars[i].is\_use == false && i != min1){**

**if(min2 == -1 || min2val == -1){**

**min2 = i;**

**min2val = ct.chars[i].val;**

**}**

**if(ct.chars[i].val < min2val){**

**min2val = ct.chars[i].val;**

**min2 = i;**

**}**

**}**

**}**

**assert(min1 != min2);**

**//----------**

**if(some\_not\_use == 1){**

**\*depth += 1;**

**break;**

**}else{**

**#ifdef DEBUG**

**printf("min1: %d, min2: %d\n", min1, min2);**

**printf("min1val: %d, min2val: %d\n", min1val, min2val);**

**printf("not\_use : %d\n", some\_not\_use);**

**#endif**

**//construce two nodes and generate a new node**

**BNode \*node1 = ct.chars[min1].node;**

**BNode \*node2 = ct.chars[min2].node;**

**if(node1 == NULL)**

**node1 = bnode\_init(min1val, "lnode", ct.chars[min1].c);**

**if(node2 == NULL)**

**node2 = bnode\_init(min2val, "lnode", ct.chars[min2].c);**

**BNode \*nodep = bnode\_init(min1val + min2val, "node", (char)0);**

**//depth logic**

**if(strcmp(node1->name, "node") == 0 || strcmp(node2->name, "node") == 0){**

**\*depth += 1;**

**}**

**//connect the three nodes**

**node1->parent = nodep;**

**node2->parent = nodep;**

**nodep->left = node1;**

**nodep->right = node2;**

**ct.chars[min1].node = node1;**

**ct.chars[min2].node = node2;**

**ct.chars[min1].is\_use = true;**

**ct.chars[min2].is\_use = true;**

**ct.n ++;**

**ct.chars = realloc(ct.chars, sizeof(character) \* ct.n);**

**ct.chars[ct.n - 1].node = nodep;**

**ct.chars[ct.n - 1].is\_use = false;**

**ct.chars[ct.n - 1].val = min1val + min2val;**

**min1 = -1 ; min2 = -1; min1val = -1; min2val = -1;**

**}**

**}**

**return ct.chars[ct.n - 1].node;**

**}**

**#include <hfm.h>**

**int parse\_args(int argc, char \*argv[]) {**

**const struct option table[] = {**

**{"init"             , no\_argument      , NULL, 'i'},**

**{"encoding"         , no\_argument      , NULL, 'e'},**

**{"decoding"         , no\_argument      , NULL, 'd'},**

**{"print"            , no\_argument      , NULL, 'p'},**

**{"tree"             , no\_argument      , NULL, 't'},**

**{"help"             , no\_argument      , NULL, 'h'},**

**{0                  , 0                , NULL,  0 },**

**};**

**int o;**

**while ( (o = getopt\_long(argc, argv, "-iedpth", table, NULL)) != -1) {**

**switch (o) {**

**case 'i': hfm\_init(); break;**

**case 'e': encode(); break;**

**case 'd': decode(); break;**

**case 'p': print\_code(); break;**

**case 't': printf("pls use -i to generate treeprint"); break;**

**default:**

**printf("Usage: %s [OPTION...]\n\n", argv[0]);**

**printf("\*\*\*\*\*\*\*\* 哈夫曼编码系统 \*\*\*\*\*\*\*\*\n");**

**printf("1. I：初始化（Initialization）\n");**

**printf("   - 从终端读入字符集大小 n，以及 n 个字符和 n 个权值\n");**

**printf("   - 建立哈夫曼树，并将它存于文件 hfmTree 中\n");**

**printf("2. E：编码（Encoding）\n");**

**printf("   - 利用已建好的哈夫曼树（如不在内存，则从文件 hfmTree 中读入）\n");**

**printf("   - 对文件 ToBeTran 中的正文进行编码\n");**

**printf("   - 将结果存入文件 CodeFile 中\n");**

**printf("3. D：译码（Decoding）\n");**

**printf("   - 利用已建好的哈夫曼树将文件 CodeFile 中的代码进行译码\n");**

**printf("   - 结果存入文件 TextFile 中\n");**

**printf("4. P：印代码文件（Print）\n");**

**printf("   - 将文件 CodeFile 以紧凑格式显示在终端上，每行 50 个代码\n");**

**printf("   - 同时将此字符形式的编码文件写入文件 CodePrin 中\n");**

**printf("5. T：印哈夫曼树（Tree）\n");**

**printf("   - 打印哈夫曼树结构\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\n");**

**exit(0);**

**}**

**}**

**return 0;**

**}**

**#include <hfm.h>**

**void encode(){**

**FILE \*target = fopen("./file/ToBeTran", "r");**

**FILE \*hfmtree = fopen("./file/hfmtree", "r");**

**FILE \*encoded = fopen("./file/CodeFile", "w");**

**char \*huffmanCode[MAX\_CHAR];**

**char c, code[128];**

**while(fscanf(hfmtree, "%c %s\n", &c, code) != EOF){**

**huffmanCode[(unsigned char)c] = strdup(code);**

**}**

**while(fscanf(target, "%c", &c) != EOF){**

**if(huffmanCode[(unsigned char)c] == NULL){**

**panic("Invalid character in input file");**

**}**

**fprintf(encoded, "%s", huffmanCode[(unsigned char)c]);**

**}**

**}**

**#include <hfm.h>**

**void decode(){**

**FILE \*encoded = fopen("./file/CodeFile", "r");**

**FILE \*hfmtree = fopen("./file/hfmtree", "r");**

**FILE \*decoded = fopen("./file/TextFile", "w");**

**char \*huffmanCode[MAX\_CHAR];**

**char reverseCode[MAX\_CHAR][MAX\_CHAR];**

**char c, code[128];**

**while(fscanf(hfmtree, "%c %s\n", &c, code) != EOF){**

**huffmanCode[(unsigned char)c] = strdup(code);**

**strcpy(reverseCode[(unsigned char)c], huffmanCode[(unsigned char)c]);**

**}**

**char buffer[MAX\_CHAR] = {0};  // 用于存储当前读取的编码**

**char bit;**

**int index = 0;**

**while ((bit = fgetc(encoded)) != EOF) {**

**buffer[index++] = bit;  // 将当前位加入 buffer**

**buffer[index] = '\0';   // 确保是字符串**

**// 查找反向查找表**

**for (int i = 0; i < MAX\_CHAR; i++) {**

**if (reverseCode[i][0] != '\0' && strcmp(reverseCode[i], buffer) == 0) {**

**fputc(i, decoded);  // 输出对应字符**

**index = 0;          // 重置 buffer**

**break;**

**}**

**}**

**}**

**if(index != 0){**

**panic("decode: error");**

**}**

**}**

**#include <hfm.h>**

**void print\_code()**

**{**

**FILE \*codeFile = fopen("./file/CodeFile", "r");**

**if (!codeFile) {**

**perror("Failed to open ./file/CodeFile");**

**return;**

**}**

**FILE \*codePrin = fopen("./file/CodePrin", "w");**

**if (!codePrin) {**

**perror("Failed to open ./file/CodePrin");**

**fclose(codeFile);**

**return;**

**}**

**char buffer[51];  // 用于存储每行 50 个字符，加 '\0' 保证是字符串**

**int index = 0;**

**char c;**

**while ((c = fgetc(codeFile)) != EOF) {**

**buffer[index++] = c;**

**// 如果缓冲区满了（50 个字符），输出并写入文件**

**if (index == 50) {**

**buffer[index] = '\0';  // 添加字符串结束符**

**printf("%s\n", buffer);**

**fprintf(codePrin, "%s\n", buffer);**

**index = 0;  // 重置缓冲区索引**

**}**

**}**

**// 如果还有剩余字符（不足 50 个），也需要输出**

**if (index > 0) {**

**buffer[index] = '\0';  // 添加字符串结束符**

**printf("%s\n", buffer);**

**fprintf(codePrin, "%s\n", buffer);**

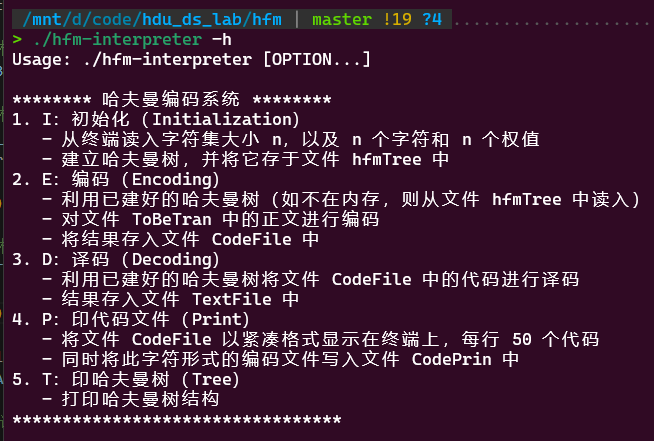
**}**

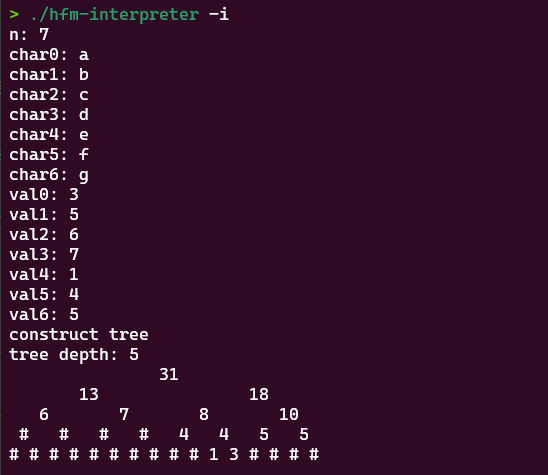
**fclose(codeFile);**

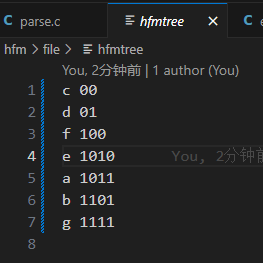
**fclose(codePrin);**

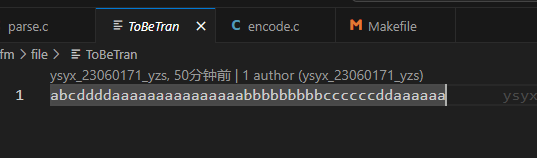
**}**

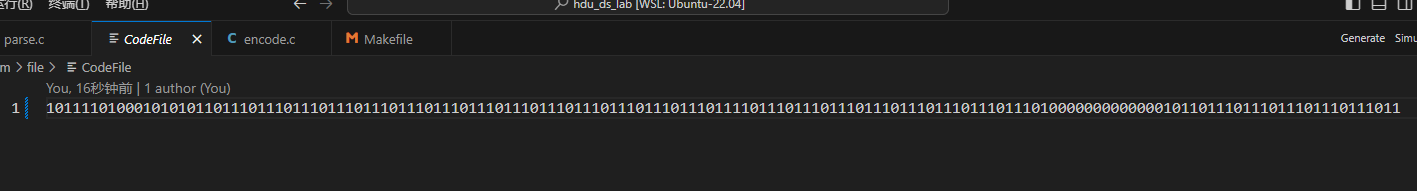
1. **用户手册**

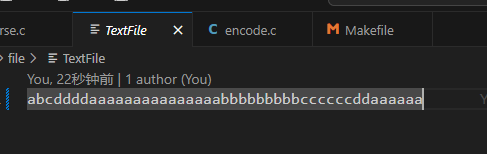


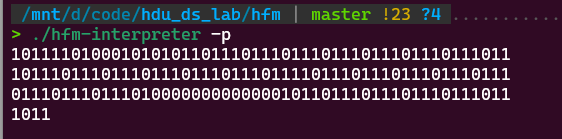












PS：详细的运行结果请打开电梯（final）.exe文件

1. **附录**

源程序文件名清单：电梯（final）.cpp，电梯（final）.exe