存储中文语料个人报告

过去已完成情况

1. 经过两次报告提交及代码实现，源代码已能够完成部分功能。
    1. 不小于10M的文本的哈夫曼代码生成与使用。
    2. 多个模式串的读入与检索（量级不大）
    3. 在目标文本中检索与匹配模式串（只允许一段文本，且量级不大）

2. 基本代码框架搭建与确定。
3. 报告书写问题，代码的高亮问题的解决。

当前主要问题与解决

需求

1. 实现10000量级模式串条数读取与检索。
2. 在达到M级的文件中的模式串检索。
3. 解码与检索分离。

程序设计思路

输入输出

输入: 文本文件名（生成哈夫曼编码的参照文件）
　　一个数字n,之后输入n行模式串
　　检索文本的文件名
输出: 所有在文本中出现的模式串，及出现频率。

基本思路

1. 维持原框架：
    - 将表示模式串中文的每个字节编码存储，建立对应编码表，建立字典树，查找模式串。

2. 改变模式串与目标文本存储与读取方式：
    - 模式串直接存储为哈夫曼编码。
    - 目标文本读取方式改为循环队列式读取。
    - 解码放在输出函数中。

3. 更新相应变量统计时间和空间进行对比，并进行效率计算。

个人总结

1. 经过一天的调试与修改，尝试了各种办法来完成需求，效果不太理想
2. 尝试使用循环队列进行存储会出现模式串重复计数的情况。
3. 加了哈夫曼的二叉字典树理论上虽然在构建树时使节点减少，但实际效果并不明显，应该是其他处理耗费时间。
4. 思维不清晰时，不要编程，尽可能想清楚之后再开始。

程序代码

头文件和宏

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define BASE 2
#define CHARMAX 256
#define MAXLINE 200
#define swap(a, b) { \
__typeof(a) temp = a; \
    a = b; b = temp; \
}
```

- 注：其中普通字典树 **BASE** 宏定义为256

结构定义

```
typedef struct HFNode {
    int ch;
    int freq;
    struct HFNode *lchild, *rchild;
} HFNode;

typedef struct TNode {
    int flag;
    struct TNode *child[BASE];
} TNode;
```

全局变量

```
long memory = 0;
int node_cnt = 0;
long time0 = 0;
```

基本通用函数

1. 获取模式串

```
void get_pattern(int n, char **str) {
    for (int i = 0; i < n; ++i) {
        str[i] = (char*)malloc(sizeof(char) * MAXLINE);
        scanf("%s", str[i]);
    }
    return ;
}
```

- 获取节点

```
HFNode *get_HFNode() {
    room += sizeof(HFNode);
    return (HFNode*)calloc(sizeof(HFNode), 1);
}

TNode *get_TNode() {
    room += sizeof(TNode);
    return (TNode*)calloc(sizeof(TNode), 1);
}
```

- 释放空间

```
void clear_trie(TNode *root) {
    if (root == NULL) return ;
    for (int i = 0; i < BASE; ++i) {
        clear_trie(root->child[i]);
    }
    free(root);
    return ;
}

void clear_hf(HFNode *root) {
    if (root == NULL) return ;
    clear_hf(root->lchild);
    clear_hf(root->rchild);
    free(root);
}

void clear(char **str, int n) {
    if (str == NULL) return ;
    for (int i = 0; i < n; ++i) {
        free(str[i]);
    }
    free(str);
    return ;
}
```

普通字典树与二叉字典树关键函数原型

- 建立字典树与查找模式串

```
TNode *insert_trie(char *str, TNode *root);
void search(TNode *root, const char *arm);
```

哈夫曼编码的二叉字典树

1.主函数

```
int main() {
    FILE *fp = NULL;
    char filename[MAXLINE];
    char huffman_code[CHARMAX][100] = {0};
    printf("亲，请输入已存在的初始化文件名：");
    scanf("%s", filename);
    fp = fopen(filename, "r");
```

```
    get_HFcode(fp, huffman_code);
    fclose(fp);

    int n, i = 0, word_cnt = 0;
    char temp[MAXLINE] = {0};
    Pattern*pattern = (Pattern*)malloc(sizeof(Pattern) * MAXLINE);
    printf("亲，请输入已存在的模式串文件名：");
    scanf("%s", filename);
    fp = fopen(filename, "r");
    while (fscanf(fp, "%s", temp) != EOF) {
        word_cnt += strlen(temp);
        pattern[i].str = (unsigned char*)malloc(sizeof(unsigned char) * (strlen(temp) + 1));
        strcpy((char*)pattern[i].str, temp);
        memory += sizeof(char) * strlen((char *)pattern[i].str);
        pattern[i].freq = 0;
        i++;
    }
    n = i;
    memory += (sizeof(Pattern*) * n);
    fclose(fp);

    long time1 = clock();
    char buff[MAXLINE * 10] = {0};
    TNode *root = NULL;
    for (int i = 0; i < n; ++i) {
        memset(buff, 0, sizeof(buff));
        cntocode(pattern[i].str, buff, huffman_code);
        root = insert_trie(root, buff);
    }
    printf("字典树已建立完成！\n");

    long time2 = clock();
    unsigned char arm[MAXLINE * 5];
    printf("亲，请输入已存在的查找母串文件名：");
    scanf("%s", filename);
    long time3 = clock();
    fp = fopen(filename, "r");
    while (fscanf(fp, "%s", arm) != EOF) {
        memset(buff, 0, sizeof(buff));
        cntocode(arm, buff, huffman_code);
        search(root, buff, pattern, n, huffman_code);
    }
    fclose(fp);
    printf("查找结束!\n");
    output(pattern, n);

    clear_trie(root);
    clear(pattern, n);
```

```
    long time4 = clock();
    memory += CHARMAX * 4;

    time0 += time2 - time1 + time4 - time3;
    printf("run time:%ld, used memory:%ld\n", time0, memory);
    printf("Haffman Double storage rate : %lf\n", 1.0 * word_cnt / (1.0 * node_cnt * sizeof
    return 0;
}
```

查找

```
void search(TNode *root, const char *arm, Pattern *pa, int n, char huffman_code[][100]) {
    if (root == NULL) return ;
    TNode *p = root;
    char buff[MAXLINE] = {0}, out[MAXLINE] = {0};
    room += MAXLINE * 2;
    int ind = 0;
    for (int head = 0; arm[head]; ++head) {
        for (int i = head; arm[i]; ++i) {
            if (!p->child[arm[i] - '0']) break;
            buff[ind++] = arm[i];
            buff[ind + 1] = '\0';
            p = p->child[arm[i] - '0'];
        }
        if(p->flag) {
            codetocn(out, buff, huffman_code);
            for (int i = 0; i < n; ++i) {
                if (strcmp(out, (char*)pa[i].str) == 0) {
                    pa[i].freq++;
                }
            }
            memset(out, 0, sizeof(out));
        }
        ind = 0;
        p = root;
    }
    return ;
}
```

输出

```
void output(Pattern*p, int n) {
    printf("已为您查找完成~~\n");
    for (int i = 0; i < n; ++i) {
        printf("%-4d%s 出现的次数:%d\n", i + 1, p[i].str, p[i].freq);
    }
    return ;
}
```

获取哈弗曼编码

```
void get_HFcode(FILE *fp, char (*huffman_code)[100]) {
    int freq[CHARMAX] = {0};
    get_freq(fp, freq);
    for (int i = 0; i < CHARMAX; ++i) {
        if (!freq[i]){
            continue;
        }
        printf("%d %d \n", i, freq[i]);
    }

    HFNode *arr[CHARMAX] = {0};
    char buff[100];
    for (int i = 0; i < CHARMAX; ++i) {
        HFNode *new_node = getNode();
        new_node->ch = (char)i;
        if(freq[i] == 0){
            new_node->freq = 1;
        } else {
            new_node->freq = freq[i];
        }
        arr[i] = new_node;
    }

    build(CHARMAX, arr);
    extract(arr[0], buff, huffman_code, 0);
    for (int i = 0; i < 256; ++i) {
        if (huffman_code[i][0] == 0) continue;
        printf("%d : %s\n", i, huffman_code[i]);
    }
}
```

其他函数原型

```
void get_freq(FILE *fp, int *freq);
void build(int n, HFNode *arr[]);
void extract(HFNode *root, char *buff, char (*huffman_code)[100], int n);
void cntocode(const unsigned char *chinese, char *code, char (*huffman_code)[100]);
void codetocn(char *chinese, char *code, char huffman_code[256][100]);
```

运行效果及对比


1. bash
80 根深叶茂 出现的次数:0
81 骨瘦如柴 出现的次数:0
82 高山流水 出现的次数:0
83 高射炮 出现的次数:0
84 高山族 出现的次数:0
85 公孙 出现的次数:0
86 过甚 出现的次数:0
87 共商国是 出现的次数:0
88 国势 出现的次数:0
89 攻守同盟 出现的次数:0
90 鬼使神差 出现的次数:0
91 公审 出现的次数:0
92 港式 出现的次数:0
93 泔水 出现的次数:0
94 国色 出现的次数:0
95 鸽哨 出现的次数:0
96 观赛 出现的次数:0
97 瓜熟蒂落 出现的次数:0
98 归顺 出现的次数:0
99 高深莫测 出现的次数:0
100 你猜 出现的次数:0
run time:4150, used memory:74290
Haffman Double storage rate : 0.010687
a10.11.5@Daniel:~/Desktop/Daniel/test_pro$
加哈夫曼的二叉字典树