2018 .08 .09 哈夫曼编码 + 二叉字典树 储存中文

一:字典树

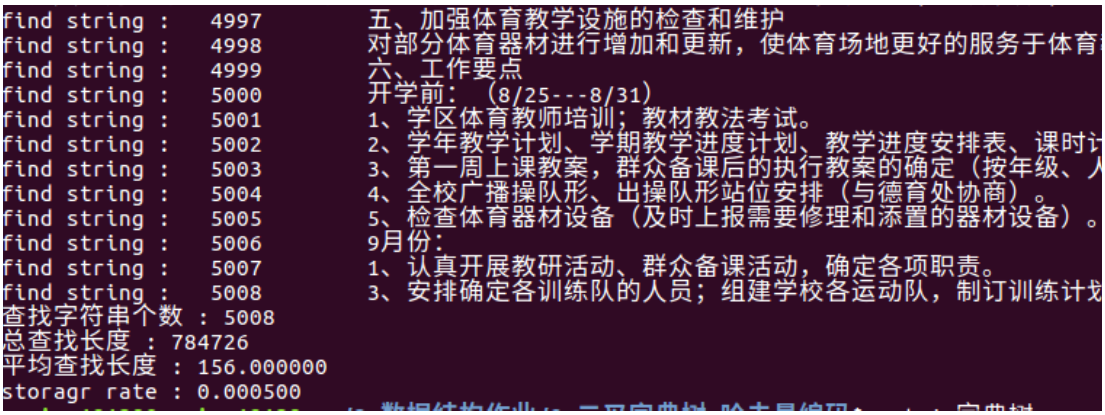
1:程序设计思路

> 开一个256叉的字典树每一叉储存一个字节的信息，然后查找在树上按照每一字节进行查找。其他操作类似TRIE;

2:程序代码

文章最后给出

3 效果演示



二：二叉字典树

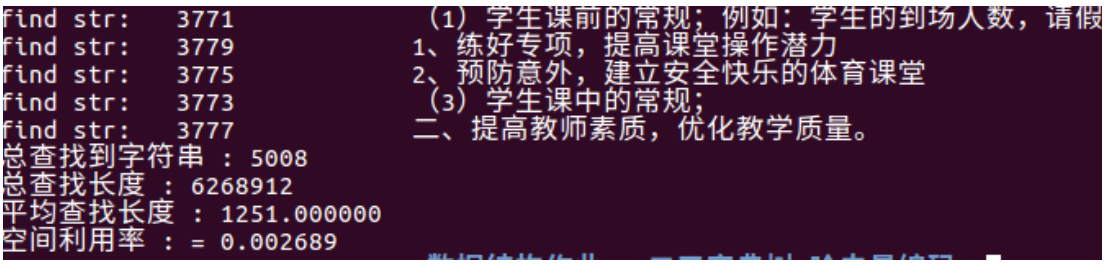
1:程序设计思路

> 把输入的每个汉字当作3个字节，把每个字节转化成8位2进制(01)，然后把这样的二进制01插入到二叉字典树中.

2:程序代码

文章最后给出

3 效果演示



此处发现一个问题，二叉字典树不能进行字典序输出，但是256叉的字典树可以

三：二叉字典树 + 哈夫曼

1:程序设计思路

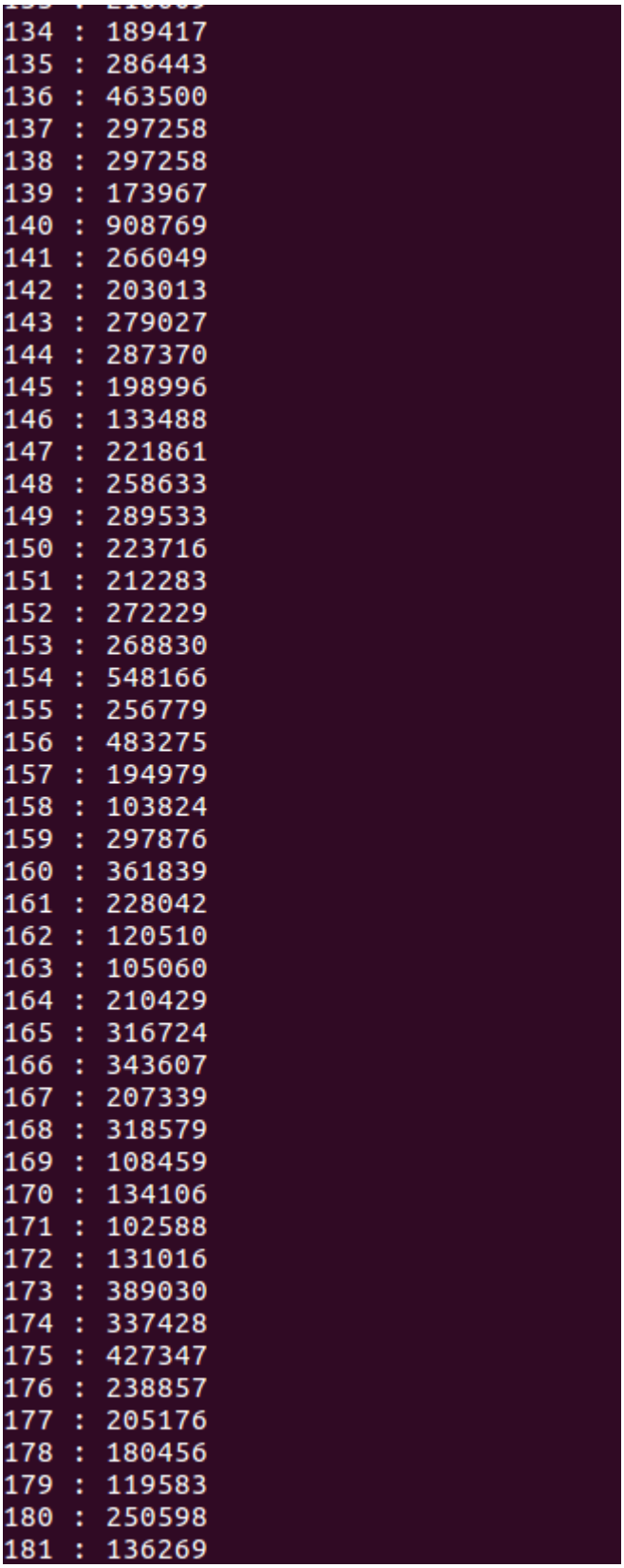
> 把很大的中文语料库中每个汉字当作3个字节，对每个字节统计频率。根据字节频率维护一个小根堆 ,每次取堆顶两个元素，构造哈夫曼树，然后提取huff编码。后面输入字符串会根据子前的huff曼编码构造字典树。对此字典树操作类似Trie.

2:程序代码

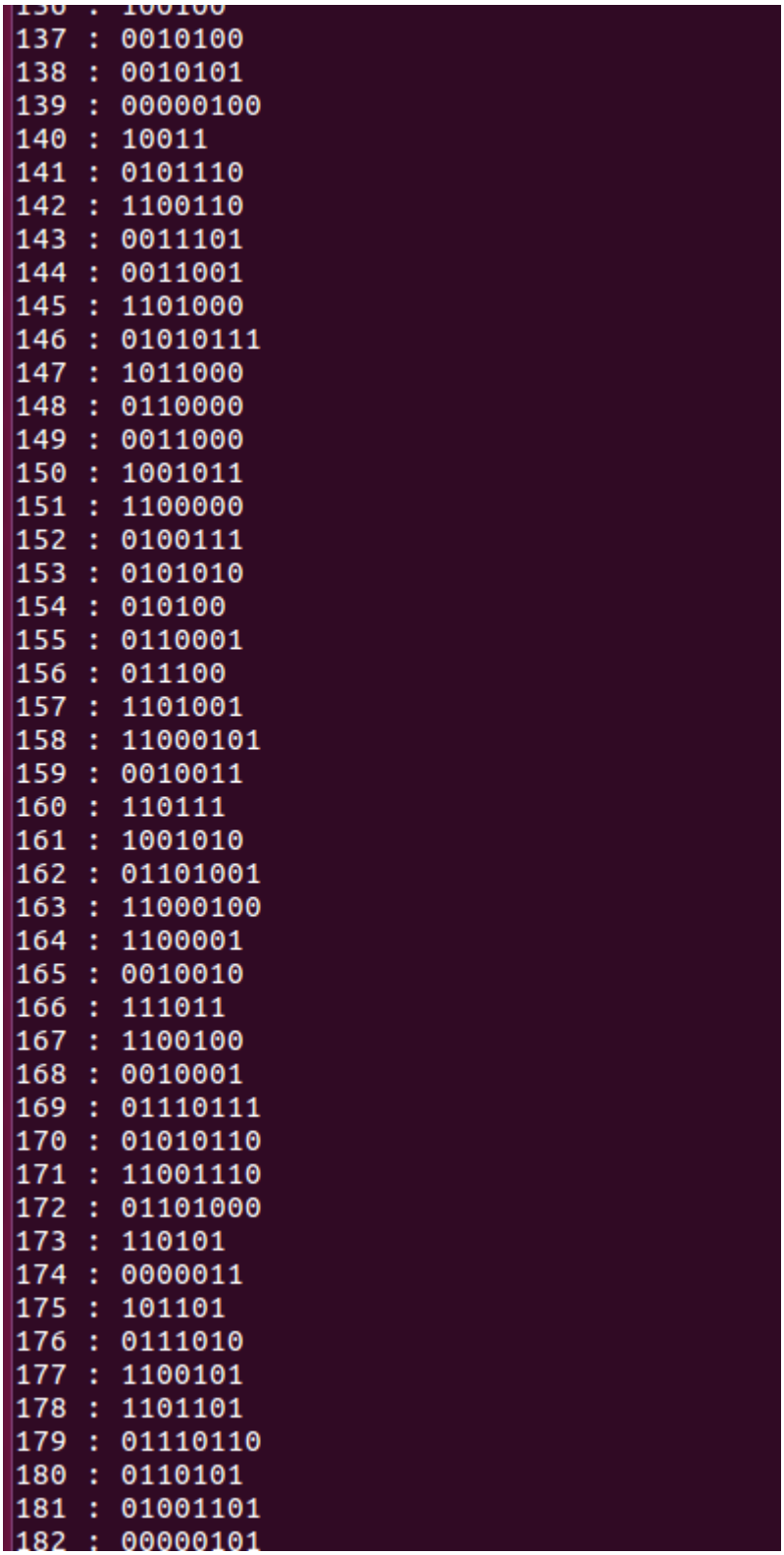
文章最后给出

3 效果演示

3.1 :字符统计 22M语料


134 : 189417
135 : 286443
136 : 463500
137 : 297258
138 : 297258
139 : 173967
140 : 908769
141 : 266049
142 : 203013
143 : 279027
144 : 287370
145 : 198996
146 : 133488
147 : 221861
148 : 258633
149 : 289533
150 : 223716
151 : 212283
152 : 272229
153 : 268830
154 : 548166
155 : 256779
156 : 483275
157 : 194979
158 : 103824
159 : 297876
160 : 361839
161 : 228042
162 : 120510
163 : 105060
164 : 210429
165 : 316724
166 : 343607
167 : 207339
168 : 318579
169 : 108459
170 : 134106
171 : 102588
172 : 131016
173 : 389030
174 : 337428
175 : 427347
176 : 238857
177 : 205176
178 : 180456
179 : 119583
180 : 250598
181 : 136269

3.2 :编码



3.3 :插入字符串 个数5008个

```
5000      开学前：（8/25---8/31）
5001      1、学区体育教师培训；教材教法考试。
5002      2、学年教学计划、学期教学进度计划、教学进度安排表、课时计划的准备。
5003      3、第一周上课教案，群众备课后的执行教案的确定（按年级、人员分工进行
5004      4、全校广播操队形、出操队形站位安排（与德育处协商）。
5005      5、检查体育器材设备（及时上报需要修理和添置的器材设备）。
5006      9月份：
5007      1、认真开展教研活动、群众备课活动，确定各项职责。
5008      3、安排确定各训练队的人员；组建学校各运动队，制订训练计划，建立运
当前总插入字符串 ： 5008个
插入总长度为 ： 4668654步
平均长度 ： 932.000000步
```

3.4: 查找字符串

```
find str :   3334          1、制定体育工作计划。
find str :   3335          2、校运动队开始训练。
find str :   3331          6、组织体育老师定期对现有的体育器材、设施进行安全检查，对
find str :    333          4、体育委员会议；筹备校运动会。
find str :   3338          5、开展多种体育活动。
find str :   3336          3、参加区中小学生排球、足球比赛。
find str :   3330          5、加强学校体育的教科研工作。鼓励体育教师用心参与教科研工
find str :   3332          7、抓好《体质健康》工作，注重新教材的培训学习，注重领会新
find str :   3337          4、搞好学生体质健康测试及数据上报工作
find str :   3333          三、主要工作安排：
查找到字符串个数 ： 5008
查找到字符串总步数 ： 4474196
平均步数 ： 893.000000
storagr rate : 0.007536
```

三: 性能对比

1: 理论分析

> 普通字典树利用效率会很低，因为256叉只用到几个叉，其他完全没有用到，但是查找时会相对较快。
> 二叉字典树利用效率回相对高一点，因为没有太多浪费， 但是因为高度很高，查找时步数会很长。
> 加上huff的二叉字典树效率会更高，因为每个编码都是最优的， 但是查找和二叉字典树有一样的问题。

2: 实验结果

储存效率 : count为插入的字节个数，cnt 为申请的节点个数, space为一个节点占用

的空间,用rate来表示储存效率 ,则有:

$$rate = count/(cnt * space)$$

平均查找长度 : cnt为查找到的字符串数量，length为查找花费总长度(查找所走的步数走0或走1)加起来的总和, average表示平均每个字符串查找长度,则有:

$$average = length/cnt$$

2.1:字典树存储效率和平均查找长度

```
查找字符串个数 : 5008
总查找长度 : 784726
平均查找长度 : 156.000000
storagr rate : 0.000500
```

2.2:二叉字典树储存效率

平均效率

```
总查找到字符串 : 5008
总查找长度 : 6268912
平均查找长度 : 1251.000000
空间利用率 : = 0.002689
```

个别效率

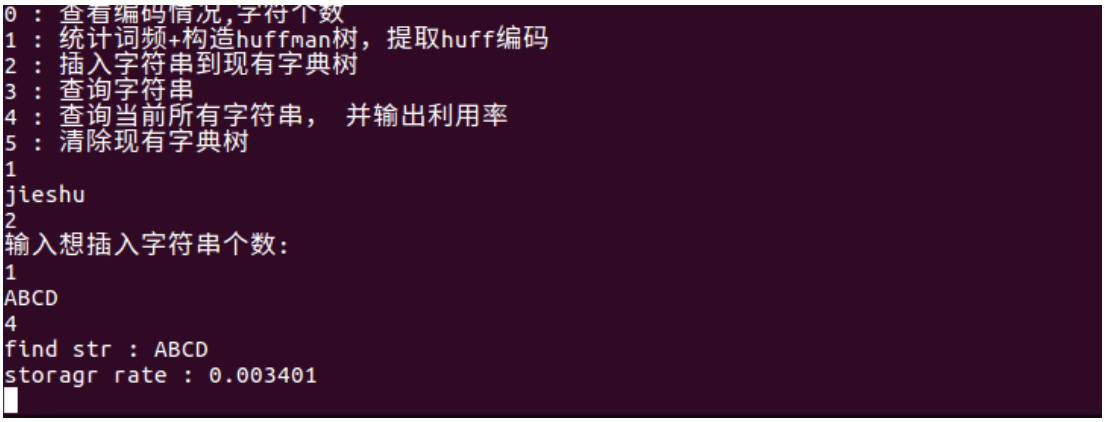
```
ddwt@ddwt:~/文档/HZ/数据结构/作业/0809$ ./a.out
插入字符串个数
1
ABCD
percent = 0.005051
请输入查找字符串
A
ret = 0
^Z
[34]+  已停止               ./a.out
ddwt@ddwt:~/文档/HZ/数据结构/作业/0809$
```

2.3:huff二叉字典树储存效率

平均效率

```
find str :   3333
查找到字符串个数 : 5008
查找到字符串总步数 : 4474196
平均步数 : 893.000000
storagr rate : 0.007536
```

个别效率低下



四: 结论

> 本实验采用同样的文本（5008行大文本）来构建字典树，控制输入相同，得出结果。
> 1: 由试验结果图可以看出普通字典树的效率是万分之5，平均查找长度156, 可以按照字典序输出。
>
> 2：二叉字典树的平均储存效率千分之2.6，平均查找长度是1251，可以看出虽然空间利用率较普通字典树高出5倍，但是查找长度提高了8倍之多，典型的时间换取空间。不可按照字典序输出
>
> 3：二叉字典树+哈夫曼的平均效率为千分之7.53，平均查找长度为893步，由数据可以看出储存效率较二叉字典树提高3倍，查找效率也提高了1.3倍，整体进行了优化。但是较普通字典树而言，空间利用率提高了15倍， 但是查找效率也就是时间提高了5.72倍。而且也无法按照字典序输出.
>
> 4：而且在实验过程中发现因为有部分huffcode长度大于8，当插入字符为这些大于 8的字符时，而且样本(特殊样本)很少时，加上huff曼的二叉字典树储存效率会降低到千分之3,特殊情况会降低效率。

五：代码

普通字典树 代码

```
/*************************************************************
*************
    > File Name: 1_字典树.c
    > Author: moying
```

```
    > Mail: 996941856@qq.com
    > Created Time: 2018年08月08日 星期三 15时29分02秒
 ************************************************************
************/

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define SIZE 256

typedef struct Node {
    int flag;
    unsigned char *str;
    struct Node *next[SIZE];
}Node;
int node_cnt = 0;
Node *get_node() {
    node_cnt++;
    return calloc(sizeof(Node), 1);
}


void clear(Node *root) {
    if(!root) return;
    for(int i = 0; i < SIZE; ++i) {
        if(!root->next[i]) continue;
        clear(root->next[i]);
    }
    if(root->flag) free(root->str);
    free(root);
    return ;
}
int insert_length = 0, insert_cnt = 0;
Node *insert(Node *root, unsigned char *str) {
    int cnt = 0;
    if(!root) root = get_node();
    Node *p = root;
    insert_length += strlen(str);
    for(int i = 0; str[i]; ++i) {
        if(!p->next[str[i]]) p->next[str[i]] = get_node();
        p = p->next[str[i]];
    }
    p->str = strdup(str);
```

```
    if(!p->flag) insert_cnt++;
    p->flag = 1;

    return root;
}

int search(Node *root, char *str) {
    if(!root) return 0;
    Node *p = root;
    int i = 0;
    while(p && str[i]){
        if(str[i] > 0) continue;
        int ind = str[i] * -1;
        p = p->next[ind];
        i++;
    }
    if(p && p->flag == 1) return 1;
    return 0;
}

int search_cnt = 0, search_length = 0;
void output(Node *root) {
    if(!root) return ;
    if(root->flag) {
        search_cnt++;
        printf("find string : %s\n", root->str);
    }
    for(int i = 0; i < SIZE; ++i) {
        if(!root->next[i]) continue;
        search_length++;
        output(root->next[i]);
    }
}

int main() {
    unsigned char str[100000];
    Node *root = NULL;
    FILE *fp = fopen("./input_str", "r");
    while(fgets(str, 10000, fp)) {
        str[strlen(str) - 1] = '\0';
        root = insert(root, str);
    }
    output(root);
```

```
    printf("查找字符串个数 : %d\n总查找长度 : %d\n平均查找长度 : %l
f\n", search_cnt, search_length, 1.0 * (search_length / searc
h_cnt));
    printf("storagr rate : %lf\n", 1.0 * insert_length / (1.0
* node_cnt * sizeof(Node)));
    clear(root);



    return 0;
}
```

二叉字典树 代码

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
typedef struct Node {
    int flag;
    struct Node *next[2];
    unsigned char *str;
} Node, *Trie;
int num[256][8] = {0};
void get_num(int (*num)[8]) {
    for (int i = 0; i < 256; i++) {
        int temp = i;
        for (int j = 0; j < 8; j++) {
            num[i][j] = temp % 2;
            temp /= 2;
        }
    }
}
int node_cnt = 0;
Node *get_new_node() {
    node_cnt++;
    return (Node *)calloc(sizeof(Node), 1);
}
int insert_length = 0, insert_cnt = 0, word_cnt = 0;
Node *insert(Trie root, unsigned char *str) {
```

```
    if(!root) root = get_new_node();
    Node *p = root;
    insert_cnt++;
    for (int i = 0; str[i]; i++) {
        insert_length += 8;
        for (int j = 0; j < 8; j++) {
            if (p->next[num[str[i]][j]] == NULL) {
                p->next[num[str[i]][j]] = get_new_node();
                node_cnt += 1;
            }
            p = p->next[num[str[i]][j]];
        }
    }
    if(!p->flag) word_cnt += strlen(str);
    p->flag = 1;
    p->str = strdup(str);
    return root;
}
int search(Trie root, unsigned char *str) {
    Node *p = root;
    for (int i = 0; str[i]; i++) {
        for (int j = 0; j < 8; j++) {
            if (p->next[num[str[i]][j]] == NULL) return 0;
            p = p->next[num[str[i]][j]];
        }
    }
    return p->flag;
}
int search_length = 0, search_cnt = 0;
void output(Node *p) {
    if(!p) return ;
    for(int i = 0; i < 2; ++i) {
        if(!p->next[i]) continue;
        if(p->next[i]->flag) {
            search_cnt++;
            //printf("find str: %s\n", p->next[i]->str);
        }
        search_length++;
        output(p->next[i]);
    }
    return ;
}
int main() {
```

```
    get_num(num);
    Node *root = NULL;
    unsigned char str[10000];
    FILE *fp = fopen("./input_str", "r");
    while(fgets(str, 10000, fp)) {
        str[strlen(str) - 1] = '\0';
        root = insert(root, str);
    }

    printf("总插入字符串 : %d\n总插入长度 : %d\n平均插入长度 : %lf
\n", insert_cnt, insert_length, 1.0 * (insert_length / insert
_cnt));
    getchar();
    output(root);

    printf("总查找到字符串 : %d\n总查找长度 : %d\n平均查找长度 : %l
f\n", search_cnt, search_length, 1.0 * (search_length / searc
h_cnt));
    printf("空间利用率 : = %lf\n", 1.0 * (word_cnt) / (node_cnt
* (sizeof(Node) - sizeof(char *))));
    fclose(fp);
    return 0;
}
```

二叉字典树+哈夫曼 代码

```
/*************************************************************
*************
    > File Name: 2_二叉字典树.c
    > Author: moying
    > Mail: 996941856@qq.com
    > Created Time: 2018年08月08日 星期三 10时14分40秒
 *************************************************************
************/

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

```
#define MAX_N 100000
#define SIZE 260
#define swap(a, b) {\
    typeof(a) (__temp) = (a);\
    (a) = (b);\
    (b) = (__temp);\
}
typedef struct Node{    // huff结点
    int freq;
    unsigned char ch;
    struct Node *lchild, *rchild;
}Node;

typedef struct Heap{    //堆
    Node **data;
    int count, size;
}Heap;

typedef struct D_Node{    //字典树
    int flag;
    char *str;
    struct D_Node *next[2];
}D_Node;
int d_cnt = 0;
D_Node *get_DNode() {
    d_cnt++;
    return calloc(sizeof(D_Node), 1);
}

void clear(D_Node *root) {
    if(!root) return ;
    for(int i = 0; i < 2; ++i) {
        if(!root->next[i]) continue;
        clear(root->next[i]);
    }
    if(root->flag) free(root->str);
    free(root);
    return ;
}
int insert_length = 0;
int insert_cnt = 0;
D_Node *insert(D_Node *root, unsigned char *str, unsigned cha
r (*huff_code)[SIZE]) {
```

```
    if(!root) root = get_DNode();
    D_Node *p = root;
    for(int i = 0; str[i]; ++i) {
        int ind = str[i];
        insert_length += strlen(huff_code[ind]);
        for(int j = 0; huff_code[ind][j]; ++j) {
            int x = huff_code[ind][j] - '0';
            if(!p->next[x]) p->next[x] = get_DNode();
            p = p->next[x];
        }
    }
    insert_cnt++;
    if(!p->flag)p->str = strdup(str);
    p->flag += 1;
    return root;
}
int search_cnt = 0, search_length = 0;
int match_search(D_Node *root, unsigned char *str, unsigned c
har (*huff_code)[SIZE]) {
    #define RETURN(a, b, c) {\
        printf("当前查找字符串个数 : %d\n总查找步数 : %d\n平均查找
步数为 : %lf\n", b, c, 1.0 * (c/b));\
        printf("find %s : res : %d\n", str, a);\
        return a;\
    }

    if(!root) RETURN(0, search_cnt, search_length);
    search_cnt++;
    D_Node *p = root;
    for(int i = 0; str[i]; ++i) {
        int ind = str[i];
        for(int j = 0; huff_code[ind][j]; ++j) {
            int x = huff_code[ind][j] - '0';
            if(!p->next[x]) RETURN(0, search_cnt, search_leng
th);
            search_length++;
            p = p->next[x];
        }
    }
    if(p->flag) RETURN(1, search_cnt, search_length);
    RETURN(0, search_cnt, search_length);
    #undef RETURN
}
```

```
int find_cnt = 0, find_length = 0;

void find_str_all(D_Node *root) {

    for(int i = 0; i < 2; ++i) {
        if(!root->next[i]) continue;
        if(root->next[i]->flag){
            find_cnt++;
            printf("find str : %s\n", root->next[i]->str);
        }
        find_length++;
        find_str_all(root->next[i]);
    }

    return ;
}


Heap *Get_Heap(int n) {
    Heap *p = (Heap *)malloc(sizeof(Heap));
    p->data = calloc(sizeof(Node *), n);
    p->count = 0;
    p->size = n;
    return p;
}

int cmp_node(Node *p, Node *q) {
    return p->freq > q->freq;
}

void push(Heap *h, Node *p) {
    if(h->count >= h->size) return ;
    int current = h->count;
    int father = (current - 1) / 2;
    h->data[current] = p;
    while(father >= 0 && cmp_node(h->data[father], h->data[cu
rrent])) {
        swap(h->data[father], h->data[current]);
        current = father;
        father = (current - 1) / 2;
    }
    h->count++;
    return ;
```

```
}

void update_heap(Heap *h, int pos, int n) {
    int l = pos * 2 + 1, r = pos * 2 + 2;
    int min = pos;
    if(l < n && cmp_node(h->data[min], h->data[l])) min = l;
    if(r < n && cmp_node(h->data[min], h->data[r])) min = r;
    if(min != pos) {
        swap(h->data[min], h->data[pos]);
        update_heap(h, min, n);
    }
    return ;
}

void pop(Heap *h) {
    if(!h->count) return;
    h->count--;
    swap(h->data[0], h->data[h->count]);
    update_heap(h, 0, h->count);
    return ;
}

Node *top(Heap *h) {
    if(h->count == 0) return NULL;
    return h->data[0];
}
void clear_heap(Heap *h) {
    if(!h) return ;
    free(h->data);
    free(h);
    return ;
}

int node_cnt = 0;
Node *get_node(int freq, unsigned char ch) {
    Node *p =  calloc(sizeof(Node), 1);
    p->ch = ch;
    p->freq = freq;
    node_cnt++;
    return p;
}

Node *build_huff(Heap *h) {
```

```
    while(h->count > 1) {
        Node *p = top(h);pop(h);
        Node *q = top(h);pop(h);
        Node *new_node = get_node(p->freq + q->freq, 'a');
        new_node->lchild = p;
        new_node->rchild = q;
        push(h, new_node);
    }
    return h->data[0];
}

void extract_huff(Node *root, unsigned char *huff, unsigned c
har (*huff_code)[SIZE], int n) {
    huff[n] = '\0';
    if(!root->lchild && !root->rchild) {
        strcpy(huff_code[root->ch], huff);
        return ;
    }
    huff[n] = '1';
    extract_huff(root->lchild, huff, huff_code, n + 1);
    huff[n] = '0';
    extract_huff(root->rchild, huff, huff_code, n + 1);
    return ;
}

void clear_Node (Node *root) {
    if(!root) return;
    if(root->lchild) clear_Node(root->lchild);
    if(root->rchild) clear_Node(root->rchild);
    free(root);
    return ;
}

void word_count(int *arr) {          //22M的串输入来统计字节频率
    unsigned char str[100000];
    FILE *fp = fopen("./input", "r");
    while(fread(str, 999, 1, fp) > 0) {
        //printf("%s\n", str);
        for(int i = 0; str[i]; ++i){
            if(str[i] == ' ' || str[i] == '\n') continue;
            arr[str[i]]++;
        }
    }
```

```
    fclose(fp);
    printf("jieshu\n");
}
D_Node* insert_input_to_droot(D_Node *d_root, unsigned char (
*huff_code)[SIZE], int *word_cnt) {
    unsigned char str[100000];
    FILE *fp = fopen("./input_str", "r");
    while(fgets(str, 100000, fp) != NULL){
        printf("%s\n", str);
        int cnt = strlen(str);
        str[cnt - 1] = '\0';
        *word_cnt += cnt - 1;
        d_root = insert(d_root, str, huff_code);
    }
    fclose(fp);
    printf("当前总插入字符串 : %d个\n插入总长度为 : %d步\n平均长度
: %lf步\n", insert_cnt, insert_length, 1.0 * (insert_length /
insert_cnt));
    return d_root;
}
D_Node* insert_str_droot(D_Node *d_root, unsigned char (*huff
_code)[SIZE], int *word_cnt) {
    printf("输入想插入字符串个数:\n");
    int n;
    fflush(stdin);
    scanf("%d", &n);
    unsigned char str[100000];
    for(int i = 0; i < n; ++i) {
        scanf("%s", str);
        *word_cnt += strlen(str);
        d_root = insert(d_root, str, huff_code);
    }
    return d_root;
}

int IQ_cnt = 0;
void clear_droot_main(D_Node *d_root, int *cnt) {
    if(!d_root) {
        IQ_cnt++;
        if(IQ_cnt >= 5) {
            while(1) printf("操作失误5次，你好像个zz\n");
        }
        printf("当前字典为空不需要清空 剩余机会%d\n", 5 - IQ_cnt);
```

```
        return ;
    }
    clear(d_root);
    printf("清除完毕\n");
    insert_cnt = 0;
    insert_length = 0;
    *cnt = 0;
}

void find_str_droot_main(D_Node *d_root, unsigned char (*huff
_code)[SIZE]) {
    if(!d_root) {
        IQ_cnt++;
        printf("请你先建造一颗字典树,我才能查找\n");
        return ;
    }
    char ch;
    while((ch = getchar()) != '\n' && ch != EOF);
    unsigned char str[1000];
    printf("输入查找字符串\n");
    fgets(str, 1000, stdin);
    str[strlen(str) - 1] = '\0';

    match_search(d_root, str, huff_code);
}
Node *word_cnt_build_huff(Node *root, int *arr_count, unsigne
d char (*huff_code)[SIZE]) {
    clear_Node(root);
    root = NULL;
    memset(arr_count, 0, sizeof(arr_count));
    memset(huff_code, '\0', sizeof(huff_code));
    word_count(arr_count);          //统计词频
    Heap *h = Get_Heap(SIZE);
    unsigned char huff[SIZE] = {0};
    for(int i = 0; i < SIZE; ++i) {
        if(arr_count[i]){
            Node *p = get_node(arr_count[i],  i);
            push(h, p);                                  //入堆
        }
    }
    root = build_huff(h);                  //构造huffman树
    extract_huff(root, huff, huff_code, 0);     //从huff树上提
取huffman编码存到huffcode中  010101
```

```
    clear_heap(h);

    return root;
}
void show_code(int *arr, unsigned char (*huff_code)[SIZE]) {
    for(int i = 0; i < SIZE; ++i) {
        if(!arr[i]) continue;
        printf("%d : %d\n", i, arr[i]);
    }
    for(int i = 0; i < SIZE; ++i) {
        if(!huff_code[i][0]) continue;
        printf("%d : %s\n", i, huff_code[i]);
    }
}
int main() {
    char input_str[1000];
    unsigned char huff_code[SIZE][SIZE] = {0};
    int arr_count[SIZE] = {0};
    Node *root = NULL;
    D_Node *d_root = NULL;
    int n;
    int word_cnt = 0;
    printf("0 : 查看编码情况，字符个数\n1 : 统计词频+构造huffman树
，提取huff编码\n2 : 插入字符串到现有字典树\n");
    printf("3 : 从文件中读取字符串并插入字典树\n4 : 查询字符串\n5 :
查询当前所有字符串， 并输出利用率\n6 : 清除现有字典树\n");
    while(1) {
        scanf("%d", &n);
        switch(n) {
            case 0: {
                show_code(arr_count, huff_code);
            }break;
            case 1:  {                  //统计词频+构造huffman
树，提取
                root = word_cnt_build_huff(root, arr_count, h
uff_code);
                fflush(stdin);
            }break;
            case 2: {              //插入字符串到现有字典树
                d_root = insert_str_droot(d_root, huff_code,
&word_cnt);
            }break;
            case 3:{                    //从文件中读取字符串并插入
```

字典树5000行不重复长文本，输出平均插入长度

```
                d_root = insert_input_to_droot(d_root, huff_c
ode, &word_cnt);
            }break;
            case 4: {              //查询字符串，输出当前查找长度，
和平均查找长度
                find_str_droot_main(d_root, huff_code);
            }break;
            case 5: {               //查询当前所有字符串， 并输出利
用率
                find_cnt = 0,find_length = 0;
                find_str_all(d_root);
                printf("查找到字符串个数 : %d\n查找到字符串总步数
: %d\n", find_cnt, find_length);
                printf("平均步数 : %lf\n", 1.0 * (find_length /
find_cnt));
                printf("storagr rate : %lf \n", 1.0 * word_cn
t / (1.0 * d_cnt *(sizeof(D_Node) - sizeof(char *))));
            }break;
            case 6: {              //清除现有字典树
                clear_droot_main(d_root, &word_cnt);
                d_root = NULL;
            }break;
            default: {
                IQ_cnt++;
                printf("请输0 - 6\n");
                printf("0 : 查看编码情况，字符个数\n1 : 统计词频+
构造huffman树，提取huff编码\n2 : 插入字符串到现有字典树\n");
                printf("3 : 从文件中读取字符串并插入字典树\n4 : 查
询字符串\n5 : 查询当前所有字符串， 并输出利用率\n6 : 清除现有字典树\n
");
                break;
            }break;
        }
        char ch;
        while((ch = getchar()) != '\n' && (ch != EOF));
    }



    return 0;
}
```