哈弗曼加二叉字典树

1. 建立哈弗曼表

```c
/*************************************************************************
    > File Name: HF_Init.c
    > Author: snowflake
    > Mail: ＼(￣▽￣)／
    > Created Time: 2018年08月09日 星期四 20时03分34秒
 ************************************************************************/

#include "HF_Trie.h"

HFNode *get_hfNode() {
    return (HFNode*)calloc(sizeof(HFNode), 1);
}

void build(HFNode **arr) {
    for (int times = 0; times < BASE - 1; times++) {
        HFNode *minNode = arr[0];
        int ind = 0;
        for (int i = 0; i < BASE - times; i++) {
            if (arr[i]->freq >= minNode->freq) continue;
            minNode = arr[i];
            ind = i;
        }
        swap(arr[ind], arr[BASE - times - 1]);
        minNode = arr[0];
        ind = 0;
        for (int i = 1; i < BASE - times - 1; i++) {
            if (arr[i]->freq >= minNode->freq) continue;
            minNode = arr[i];
            ind = i;
        }
        swap(arr[ind], arr[BASE - times - 2]);
        HFNode *new_node = get_hfNode();
        new_node->lchild = arr[BASE - times - 1];
        new_node->rchild = arr[BASE - times - 2];
        new_node->freq = arr[BASE - times - 1]->freq + arr[BASE - times - 2]->freq;
        arr[BASE - times - 2] = new_node;
    }
    return ;
}

int *get_word_freq() {
    int *word_freq = (int*)calloc(sizeof(int), BASE);
    FILE *fp = NULL;
    fp = fopen("./text.txt", "r");
    if (fp == NULL) perror("fopen:");
    unsigned char word[10000] = {0};
    while (fscanf(fp, "%s", word) != EOF) {
        for (int i = 0; word[i]; i++) {
            ++word_freq[(int)word[i]];
        }
        memset(word, 0, sizeof(word));
    }

    for (int i = 0; i < BASE; i++) {
```

```
        word_freq[i] += (word_freq[i] == 0);
    }
    return word_freq;
}

void extract(HFNode *root, unsigned char *buff, int n) {
    buff[n] = 0;
    if (root->rchild == NULL && root->lchild == NULL) {
        //printf("%d : %s\n ", root->ch, buff);
        for (int i = 0; buff[i]; i++) {
            hftable[root->ch][i] = buff[i];
        }
        return ;
    }
    buff[n] = '0';
    extract(root->lchild, buff, n + 1);
    buff[n] = '1';
    extract(root->rchild, buff, n + 1);
    return ;
}

void hf_init() {
    int *word_freq = get_word_freq();
    HFNode *freq_arr[BASE];
    int count = 0;
    for (int i = 0; i < BASE; i++) {
        HFNode *new_node = get_hfNode();
        new_node->ch = i;
        new_node->freq = word_freq[i];
        freq_arr[count++] = new_node;
    }
    build(freq_arr);
    unsigned char buff[BASE] = {0};
    extract(freq_arr[0], buff, 0);
}

void output() {
    for (int i = 0; i < BASE; i++) {
        printf("%d %s\n", i, hftable[i]);
    }
}
```

2. 建立字典树

```c
/*************************************************************************
    > File Name: trie.c
    > Author: snowflake
    > Mail: ＼(￣▽￣)／
    > Created Time: 2018年08月09日 星期四 15时05分22秒
 ************************************************************************/

#include "HF_Trie.h"

Node *get_trie_node() {
    Node *p = (Node *)calloc(sizeof(Node), 1);
    node_cnt += 1;
    return p;
}

void clear(Trie node) {
    if (node == NULL) return ;
    for (int i = 0; i < BASE; i++) {
        if (node->next[i] == NULL) continue;
        clear(node->next[i]);
    }
    if (node->flag) free(node->str);
    free(node);
    return ;
}

Node *insert(Trie root, const unsigned char *pattern) {
    if (root == NULL) root = get_trie_node();
    Node *p = root;
    int len = strlen((char *)pattern);
    for (int i = 0; i < len - 2; i++) {
        char temp[100] = {0};
        strncpy(temp, (char *)hftable[pattern[i]], strlen((char
*)hftable[pattern[i]]));
        for (int j = 0; temp[j]; j++) {
            int ind = temp[j] - BL;
            if (p->next[ind] == NULL) p->next[ind] = get_trie_node();
            p = p->next[ind];
        }
        memset(temp, 0, 10);
    }
    p->flag = 1;
    p->str = strdup((char *)pattern);
    printf("%s", p->str);
    return root;
}

void search(Trie root, const unsigned char *text) {
    Node *p = root;
    int len = strlen((char *)text);
    unsigned char *text_temp = (unsigned char *)calloc(sizeof(unsigned char), len *
10);
    for (int i = 0; text[i]; i++) {
```

```
        strncat((char *)text_temp, (char *)hftable[text[i]], strlen((char
*)hftable[text[i]]));
    }
    for (int i = 0; text_temp[i]; i++) {
        int j = i;
        p = root;
        while (text_temp[j] && p && p->next[text_temp[j] - BL]) {
            search_times += 1;
            p = p->next[text_temp[j] - BL];
            if (p->flag == 1) {
                printf("find word : %s", p->str);
                break;
            }
            j++;
        }
    }
    free(text_temp);
    return ;
}
```

3. 主函数

```c
/*************************************************************************
    > File Name: HF_Trie.c
    > Author: snowflake
    > Mail: ＼(￣▽￣)／
    > Created Time: 2018年08月09日 星期四 15时50分41秒
 ************************************************************************/

#include "HF_Trie.h"
#include "HF_Init.c"
#include "Trie.c"

int main() {
    hf_init();
    output();
    Trie root = NULL;
    int word_cnt = 0;
    unsigned char pattern[11000] = {0};
    FILE *fin = fopen("text", "r");
    while (fgets((char *)pattern, 10000, fin) != NULL) {
        word_cnt += strlen((char *)pattern);
        root = insert(root, pattern);
    }
    unsigned char text[100000] = {0};
    FILE *fp = fopen("t", "r");
    fscanf(fp, "%[^\n]s", text);
    search(root, text);
    int nodes_size = sizeof(Node) * node_cnt;
    printf("storage rate : %lf\n", efficiency(word_cnt, nodes_size));
    printf("time rate : %lf\n", efficiency(search_times, word_cnt));
    return 0;
}
```

4. 头文件

```c
/*************************************************************************
    > File Name: hf_trie.h
    > Author: snowflake
    > Mail: ＼(￣▽￣)／
    > Created Time: 2018年08月09日 星期四 11时51分33秒
 ************************************************************************/

#ifndef _HF_TRIE_H
#define _HF_TRIE_H

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

#define BASE 256
#define BL '0'
#define swap(a, b) {\
    __typeof(a) swap_temp;\
    swap_temp = a, a = b, b = swap_temp;\
}
#define efficiency(a, b) (1.0 * a / (1.0 * b))

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

typedef struct Node {
    int flag;
    char *str;
    struct Node *next[2];//l = 0, r = 1;
} Node, *Trie;

unsigned char hftable[BASE][100];
int node_cnt = 0;
int search_times = 0;

//HF
HFNode *get_hfNode();
void hf_init();//get hftable
int *get_word_freq();  //return *freq_arr
void build(HFNode **hf_arr);
void extract(HFNode *root, unsigned char *buff, int n);//(hf_arr[0], empty arr ,  0);

//Trie
Node *get_trie_node();
Node *insert(Trie root, const unsigned char *pattern);// return root
void search(Trie root, const unsigned char *text);
void clear(Trie root);
#endif
```

5. 哈弗曼表

0 01111000110110011001010000
1 01111000110110011001011101
2 01111000110110011001011111
3 01111000110110011001100011
4 01111000110110011001100101
5 01111000110110011001100111
6 01111000110110011001110001
7 01111000110110011001110011
8 01111000110110011001111111
9 01111000110110011010000011
10 011110001101100110100011 1
11 01111000110110011010010011
12 01111000110110011010010111
13 01111000110110011010011011
14 01111000110110011010011111
15 01111000110110011010100011
16 01111000110110011010100101
17 01111000110110011010100111
18 01111000110110011010101011
19 01111000110110011010101111
20 01111000110110011010110011
21 01111000110110011010110111
22 01111000110110011010111011
23 01111000110110011010111111
24 01111000110110011011000000
25 01111000110110011011000011
26 01111000110110011011000101
27 01111000110110011011000111
28 01111000110110011011001011
29 01111000110110011011001111
30 01111000110110011011010011
31 01111000110110011011010111
32 01111000110110011011011011
33 01111000100010011
34 01111000111101111011001
35 01111000101011
36 01111000110110011011010000
37 01111000111101110110101
38 01111000101110
39 01111000110110011011010011
40 01111000101100101
41 01111000100111110
42 01111000100000
43 01111000110110011011010101
44 01111000110110010
45 011110000
46 01111000111011
47 01111000111001101
48 01111000110011
49 01111000111111
50 01111000110111
51 01111000110100
52 011110001110000

```
53 01111000101010
54 01111000110101
55 011110001101101
56 0111100010100
57 011110001111010
58 0111100010011111
59 01111000101101
60 01111000100010010
61 011110001111010111011011
62 01111000111001110
63 011110001111011101010
64 011110001101100110110101011
65 011110001101100110110101100
66 0111100011011001110
67 011110001101100110110101101
68 011110001101100110110101111
69 011110001101100110111010001
70 011110001101100110111010011
71 011110001101100110111010101
72 011110001101100110111010111
73 011110001101100110111111001
74 011110001101100110111111011
75 011110001101100110111111101
76 011110001101100110111111111
77 011110001111101110100000001
78 011110001111101110100000011
79 011110001111101110100000101
80 011110001111101110100000111
81 011110001111101110100010001
82 011110001111101110100010011
83 01111000111101110100011O
84 011110001111101110100101O1
85 011110001111101110100101l0
86 01111000111101110110100O
87 011110001111101110100101l1
88 011110001111101110110000000
89 011110001111101110110000001
90 011110001111101110110000010
91 0111100011O110011000
92 011110001111101110110000011
93 01111000111101110111O
94 011110001001001
95 01111000111001111
96 011110001111101110100100000
97 011110001011111
98 011110001001000
99 011110001000110
100 0111100011011000
101 011110001110010
102 0111100010111101
103 011110001001110
104 011110001011000

105 01111000110000
```

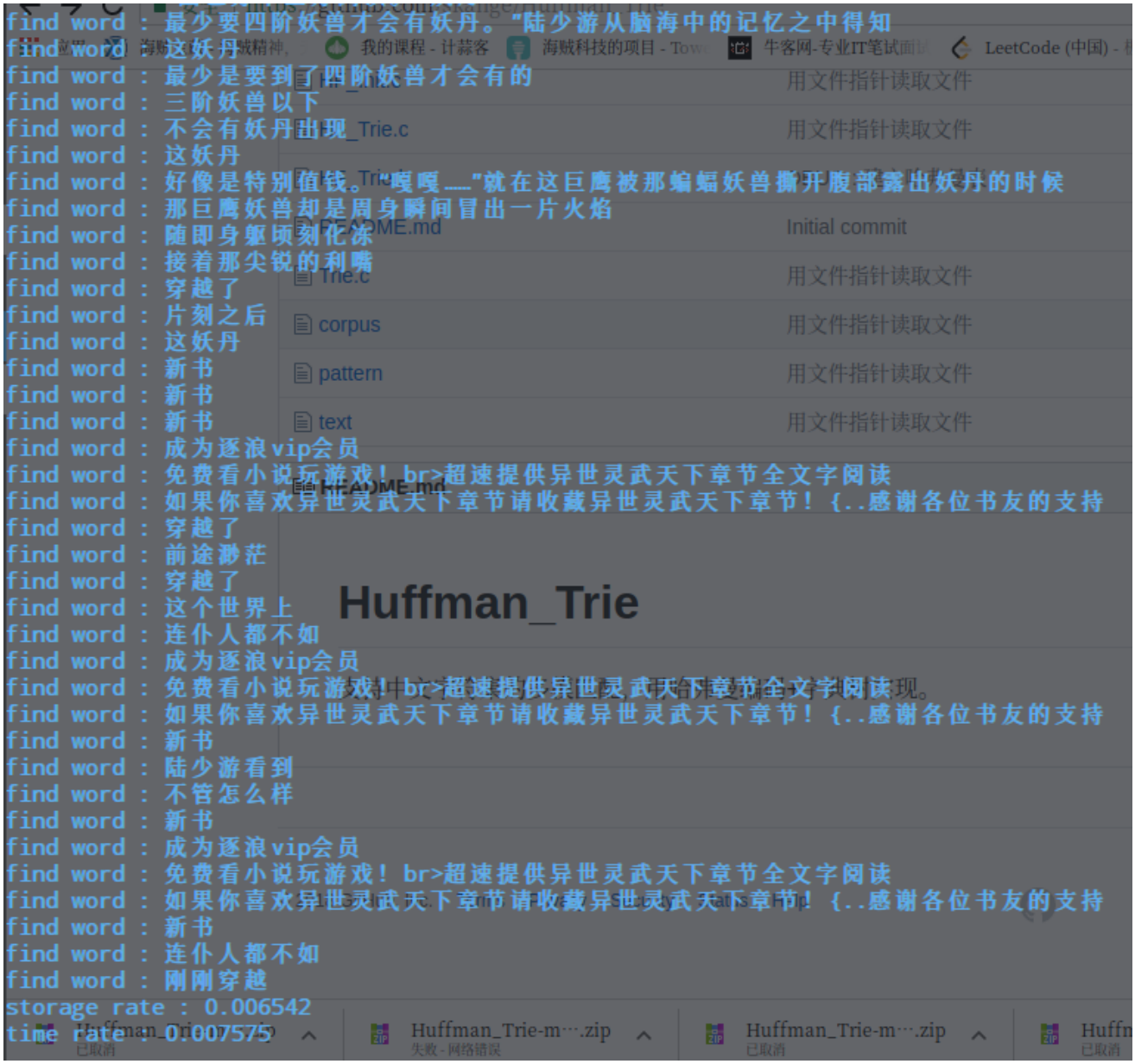
106 01111000111110000
107 01111000110110011111
108 01111000111110001
109 0111100010110011
110 011110001111101
111 011110001000111
112 01111000111001100
113 01111000111101111
114 0111100011110110
115 01111000110001
116 01111000101111000
117 0111100010001000
118 01111000101111001
119 0111100011111001
120 01111000111100
121 011110001000101
122 0111100011110111100
123 01111000100101
124 011110001111011101010010001
125 01111000100110
126 0111100011110111101011
127 011110001111011101010010011
128 10110
129 010010
130 010011
131 0111101
132 011011
133 1011100
134 011000
135 1100111
136 1110011
137 001101
138 1011101
139 1100110
140 10010
141 1111110
142 1010101
143 1100101
144 0011100
145 1110000
146 01111001
147 11101101
148 11100101
149 11100011
150 10111111
151 1110111
152 000010
153 000000
154 100011
155 1100100
156 100111
157 001100

158 10111110

```
159 1010100
160 11111111
161 0000111
162 10101111
163 11100100
164 1111101
165 0111110
166 0001111
167 0011110
168 1000101
169 0011101
170 11101010
171 11111110
172 0000110
173 1011110
174 1000100
175 011001
176 100110
177 0011111
178 10101110
179 11100010
180 1010110
181 11101100
182 0110101
183 0001110
184 10000
185 000110
186 00010
187 1110100
188 10100
189 1111100
190 11101011
191 000001
192 011110001101100110100010 1
193 011110001101100110100000 1
194 011110001101100110011110 1
195 01111000111000 1
196 011110001111011101111
197 011110001101100110011000 1
198 011110001101100110010101 1
199 01111000111101110110001
200 011110001101100110010101 0
201 011110001101100110011000 0
202 011110001101100110011110 0
203 011110001101100110100000 0
204 011110001101100110100010 0
205 011110001111011101001001 0
206 01111000111101110110100 1
207 0111100011110111010011
208 011110001101100110010 0
209 011110001111011101000111
210 011110001111011101001010 0

211 011110001111011101000101 0
```

```
212 01111000111101110100010000
213 01111000111101110100001100
214 01111000111101110100001000
215 01111000111101110100000100
216 01111000111101110100000000
217 01111000110110011011111110
218 01111000110110011011111100
219 01111000110110011011111010
220 01111000110110011011111000
221 01111000110110011011110110
222 01111000110110011011110100
223 01111000110110011011110010
224 01111000110110011011110000
225 01111000110110011011101110
226 0111111
227 0110100
228 0010
229 1101
230 0101
231 11110
232 11000
233 01110
234 01111000110110011010111100
235 01111000110110011010110100
236 01111000110110011010110000
237 01111000110110011010101100
238 01111000110110011010101000
239 01000
240 01111000110110011010100000
241 01111000110110011010011100
242 01111000110110011010011000
243 01111000110110011010010100
244 01111000110110011010010000
245 01111000110110011010001100
246 01111000110110011010000100
247 01111000110110011001111100
248 01111000110110011001110100
249 01111000110110011001110000
250 01111000110110011001101100
251 01111000110110011001101000
252 01111000110110011001100100
253 01111000110110011001011100
254 01111000110110011001011000
255 01111000110110011001010010
```



空间效率：千分之6

查找效率：千分之7