二叉字典树

```c
/*************************************************************************
    > File Name: 2cha.c
    > Author: snowflake
    > Mail: ︿(￣︶￣)︿
    > Created Time: 2018 年08月08日 星期三 21时00分59秒
 ************************************************************************/

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

#define BASE 2
#define BL '0'

unsigned char buff[256][10] = {0};
typedef struct Node {
    char *str;
    int flag;
    struct Node *next[BASE];
    struct Node *fail;
} Node;

int node_cnt = 0;
int search_times = 0;

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

void clear(Node *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(Node *root, const unsigned char *pattern) {
    if (root == NULL) root = get_new_node();
    Node *p = root;
    int len = strlen((char *)pattern);
    for (int i = 0; i < len - 2; i++) {
        char temp[10] = {0};
        strncpy(temp, (char *)buff[pattern[i]], strlen((char *)buff[pattern[i]]));
        //printf("%s\n", temp);
        for (int j = 0; temp[j]; j++) {
            int ind = temp[j] - BL;

            if (p->next[ind] == NULL) p->next[ind] = get_new_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(Node *root, const unsigned char *text) {
    Node *p = root;
    int len = strlen((char *)text);
    printf("%d\n", len * 10);
    unsigned char *text_temp = (unsigned char *)calloc(sizeof(unsigned char), len *
10);
    for (int i = 0; text[i]; i++) {
        strncat((char *)text_temp, (char *)buff[text[i]], strlen((char
*)buff[text[i]]));
    }
    //printf("%s\n", text_temp);
    for (int i = 0; text_temp[i]; i++) {
        int j = i;
        p = root;
        while (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++;
        }
    }
    return ;
}

char *bit(int n) {
    char *bit = (char *)calloc(sizeof(char), 10);
    int i = 0;
    while (n) {
        bit[i++] = n % 2 + '0';
        n /= 2;
    }
    return bit;
}

int main() {
    for (int i = 0; i < 256; i++) {
        strcpy((char *)buff[i], bit(i));
    }
    int total_count = 0;

    FILE *fin = fopen("text", "r");
    if (fin == NULL) perror("fopen\n");
    unsigned char pattern[10000] = {0};
    Node *root = NULL;
    while (fgets((char *)pattern, 10000, fin) != NULL) {
        int len = strlen((char *)pattern);
        total_count += len;
        root = insert(root, pattern);
    }
    unsigned char text[1000000] = {0};
    FILE *fp = fopen("t", "r");
    if (fp == NULL) perror("fopen\n");
    fscanf(fp, "%[^\n]s", text);
    search(root, text);
    printf("storage rate : %lf\n" , 1.0 * total_count / (1.0 * node_cnt *
sizeof(Node)));
    printf("search_times : %lf\n" , 1.0 * total_count / (1.0 * search_times));
    return 0;
}
```

深度截图_选择区域_20180812184313

空间效率是普通字典树的10倍，查找效率比256叉字典树慢了50倍。