# 字符串匹配

```
/*************************************************************************
    > File Name: 3.string_algrithm.cpp
    > Author: ldc
    > Mail: litesla
    > Created Time: 2018年12月05日 星期三 10时53分25秒
 ************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>


int BF(const char *text, const char *pattern) {
    int len1 = strlen(text),len2 = strlen(pattern);
    for (int i = 0; i < len1 - len2 + 1; i++) {
        int flag = 1;
        for (int j = 0; pattern[j]; j++) {
            if (pattern[j] == text[i + j]) continue;
            flag = 0;
        }
        if (flag) return i;
    }
    return -1;
}
int KMP(const char *text, const char *pattern) {
    int len1 = strlen(text), len2 = strlen(pattern);
    int *next = (int *)malloc(sizeof(int) * len2);
    int j = -1;
    next[0] = -1;
    for (int i = 1; pattern[i]; i++) {
        while (j >= 0 && pattern[j + 1] != pattern[i]) j = next[j];
        if (pattern[j + 1] == pattern[i]) j += 1;
        next[i] = j;
    }
    j = -1;
    for (int i = 0;text[i]; i++) {
        while(j >= 0 && pattern[j + 1] != text[i]) j = next[j];
        if (pattern[j + 1] == text[i]) j += 1;
        if (j + 1 == len2) return i;
    }
    return -1;
}
int Sunday(const char *text, const char *pattern) {
    int len1 = strlen(text), len2 = strlen(pattern);
    int ind[127];
    for (int i = 0; i < 127; i++) ind[i] = len2 + 1;
    for (int i = 0; pattern[i]; i++) ind[pattern[i]] = len2 - i;
    for (int i = 0; i <= len1 - len2;) {
```

```
        int j = 0;
        for (; j < len2; j++) {
            if (pattern[j] != text[i + j]) break;
        }
        if (j == len2) return i;
        i += ind[text[i + len2]];
    }
    return -1;
}


int shift_and(const char *text, const char *pattern) {
    int d[127] = {0},n = 0;
    for (; pattern[n]; n++) {
        d[pattern[n]] |= (1 << n);
    }
    int p = 0;
    for (int i = 0; text[i]; i++) {
        p = (p << 1 | 1) & d[text[i]];
        if (p & (1 << (n - 1))) return i;
    }
    return -1;
}
int main() {
     char text[] = "hello world";
    printf("BF(%s ,%s) = %d\n", text,"wor", BF(text,"wor"));
    printf("BF(%s ,%s) = %d\n", text,"wr", BF(text,"wr"));
    printf("KMP(%s ,%s) = %d\n", text,"worl", KMP(text,"worl"));
    printf("KMP(%s ,%s) = %d\n", text,"wr", KMP(text,"wr"));
    printf("Sunday(%s ,%s) = %d\n", text,"wor", Sunday(text,"wor"));
    printf("Sunday(%s ,%s) = %d\n", "ababc", "abc", Sunday("ababc","abc"));
    printf("shift_and(%s ,%s) = %d\n", text,"worl", shift_and(text,"worl"));
    printf("shift_and(%s ,%s) = %d\n", text,"wr", shift_and(text,"wr"));

    return 0;
}
```

## ac自动机

```
/*************************************************************************
    > File Name: acfuxi.cpp
    > Author: ldc
    > Mail: litesla
    > Created Time: 2019年01月20日 星期日 10时13分23秒
 ************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const int SIZE = 26;
const char BASE = 'a';
```

```c
#define Node TrieNode

typedef struct TrieNode {
    int flag;
    struct TrieNode *fail;
    struct TrieNode **next;
} TrieNode, *Trie;

TrieNode* new_node() {
    Node *p = (Node *)calloc(sizeof(Node), 1);
    p->next = (Node **)calloc(sizeof(Node *), SIZE);
    return p;
}

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

void insert(Trie root, const char *str) {
    if (root == NULL) root = new_node();
    Node *p = root;
    for (int i = 0; str[i]; i++) {
        if (p->next[str[i] - BASE] == NULL) p->next[str[i] - BASE] = new_node();
        p = p->next[str[i] - BASE];
    }
    p->flag += 1;
    return ;
}

void build_automaton(Trie root) {
    if (root == NULL) return ;
    Node **queue = (Node **)malloc(sizeof(Node *) * 100003);
    int head = 0,tail = 0;
    queue[tail++] = root;
    while (head < tail) {
        Node *node = queue[head++];
        for (int i = 0; i < SIZE; i++) {
            if (node->next[i] == NULL) continue;
            Node *p = node->fail;
            while (p && p->next[i] == NULL) p = p->fail;
            if (p == NULL) node->next[i]->fail = root;
            else node->next[i]->fail = p->next[i];
            queue[tail++] = node->next[i];
        }
    }
    return ;
}
```

```
int match_count(Trie root, const char *text) {
    if (root == NULL) return 0;
    int cnt = 0;
    Node *p = root;
    for (int i = 0; text[i]; i++) {
        while (p && p->next[text[i] - BASE] == NULL) p = p->fail;
        if (p == NULL) p = root;
        else p = p->next[text[i] - BASE];
        Node *q = p;
        while (q) {
            if (q->flag) cnt += q->flag;
            q = q->fail;
        }
    }
    return cnt;
}

int main() {
    int n;
    Trie root = new_node();
    char str[100003];
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%s", str);
        insert(root,str);
    }
    scanf("%s",str);
    build_automaton(root);
    printf("%d\n", match_count(root, str));
    return 0;
}
```

线索化

```
/*************************************************************************
    > File Name: acxiansuohua.cpp
    > Author: ldc
    > Mail: litesla
    > Created Time: 2019年01月20日 星期日 11时10分56秒
 ************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const int SIZE = 26;
const char BASE = 'a';
const int MAX_SIZE = 200000;
const int MAX_LEN = 200000;
char str_buffer[200005];

typedef struct TrieNode {
```

```
    int count;
    struct TrieNode** childs;
    struct TrieNode* fail;
} TrieNode, *Trie;

TrieNode* new_node() {
    TrieNode *p = (TrieNode *)malloc(sizeof(TrieNode));
    p->childs = (TrieNode **)malloc(sizeof(TrieNode *) * SIZE);
    for (int i = 0; i < SIZE; i++) {
        p->childs[i] = NULL;
    }
    p->fail = NULL;
    p->count = 0;
    return p;
}

void clear(TrieNode *node) {
    if (node == NULL) return ;
    for (int i = 0; i < SIZE; i++) {
        if (node->childs[i] == NULL) continue;
        clear(node->childs[i]);
    }
    free(node->childs);
    free(node);
    return ;
}

void insert(TrieNode *trie, const char *buffer) {
    TrieNode *p = trie;
    for (int i = 0; i < strlen(buffer); i++) {
        if (p->childs[buffer[i] - BASE] == NULL) {
            p->childs[buffer[i] - BASE] = new_node();
        }
        p = p->childs[buffer[i] - BASE];
    }
    p->count++;
    return ;
}

void build_automaton(TrieNode *node) {
    TrieNode **queue = (TrieNode **)calloc(sizeof(TrieNode *), 200000);
    int head = 0,tail = 0;
    queue[tail] = node;
    while (head < tail) {
        TrieNode *now = queue[head++];
        for (int i = 0; i < SIZE; i++) {
            if (now->childs[i] == NULL) {
                if(now != node) now->childs[i] = now->fail->childs[i];
                continue;
            }
            TrieNode *p = (now->fail ? now->fail->childs[i] : node);
            //while (p && p->childs[i] == NULL) p = p->fail;
            if (p == NULL) p = node;
```

```
            now->childs[i]->fail = p;
            queue[tail++] = now->childs[i];
        }
        /*
        for (int i = 0; i < SIZE && now != node; i++) {
            if (now->childs[i]) continue;
            now->childs[i] = now->fail->childs[i];//?
        }
        */

    }
    free(queue);
}

int match_count(TrieNode *ac_tree, const char *str) {
    int ret = 0;
    TrieNode *p = ac_tree, *q;
    while (str[0]) {
        p = p->childs[str[0] - 'a'];
        //while (p && p->childs[str[0] - 'a'] == NULL) p = p->fail;
        //if (p == NULL) p = ac_tree;
        //else p = p->childs[str[0] - 'a'];
        q = p;
        while (q) ret += q->count, q = q->fail;
        if (p == NULL) p = ac_tree;
        str++;
    }
    return ret;
}

int main() {
    Trie root = new_node();
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        char pattern[MAX_LEN];
        scanf("%s", pattern);
        insert(root, pattern);
    }
    build_automaton(root);
    scanf("%s", str_buffer);
    printf("%d\n", match_count(root, str_buffer));
    //clear(root);
    return 0;
}
```

# 哈希算法

```
/*************************************************************************
    > File Name: 6.hash.cpp
    > Author: ldc
    > Mail: litesla
    > Created Time: 2018年11月06日 星期二 20时41分41秒
 ************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


typedef struct Node{
    char *str;
    struct Node *next;
}Node;

typedef struct HashTable{
    Node **data;
    int size;
}HashTable;

Node *init_node(char *str,Node *head){
    Node *p = (Node *)malloc(sizeof(Node));
    p->str = strdup(str);
    p->next = head;
    return p;
}

HashTable * init_hasttable(int n){
    HashTable * h = (HashTable * )malloc(sizeof(HashTable));
    h->size = n << 1;
    h->data = (Node **)calloc(sizeof(Node *),h->size);
    return h;
}

int BKDRHash(char *str){
    int seed = 31,hash = 0;
    for(int i = 0; str[i]; i++) hash = hash * seed + str[i];
    return hash & 0x7fffffff;
}

int insert(HashTable * h,char *str){
    int hash = BKDRHash(str);
    int ind = hash % h->size;
    h->data[ind] = init_node(str,h->data[ind]);
    return 1;
}

/*
int insert(HashTable * h,char *str){
```

```
    int hash = BKDRHash(str);
    int ind = hash % h->size;
    int times = 0;
    Node *node = init_node(str,NULL);
    while(h->data[ind]){
        times++;
        ind += times * times;
        ind %= h->size;
    }
    h->data[ind] = node;
}*/
int search(HashTable * h, char *str){
    int hash = BKDRHash(str);
    int ind = hash % h->size;
    Node *p = h->data[ind];
    while(p && strcmp(p->str, str)) p->next;
    return p != NULL;
}

void clear_node(Node * node){
    if(node == NULL) return ;
    Node *p = node, *q;
    while(p){
        q = p->next;
        free(p->str);
        free(p);
        p = q;
    }
    return ;
}
void clear_hasttable(HashTable * h){
    if(h == NULL) return ;
    for(int i = 0; i  < h->size; i++) clear_node(h->data[i]);
    free(h->data);
    free(h);
    return;
}


int main(){
    int op;
    char str[100];
    HashTable *h = init_hasttable(100);
    while(scanf("%d%s",&op,str) != EOF){
        switch (op){
            case 0:{
                printf("insert %s to hast tabile\n", str);
                insert(h,str);
            }break;
            case 1:{
                printf("search %s result = %d\n", str,search(h,str));
            }break;
        }
```

```
    }

    return 0;
}
```

## 并查集

```
/*************************************************************************
    > File Name: 4.unionset.cpp
    > Author: ldc
    > Mail: litesla
    > Created Time: 2019年01月20日 星期日 17时10分57秒
 ************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

typedef struct unionset{
    int *fa;
    int size;
}unionset;

unionset *init(int n) {
    unionset *u = (unionset *)calloc(sizeof(unionset), 1);
    u->fa = (int *)malloc(sizeof(int) *n);
    for (int i = 0; i < n; i++) {
        u->fa[i] = i;
    }
    u->size = n;
    return u;
}
int find(unionset *u, int x) {
    if (u->fa[x] != x) return find(u, u->fa[x]);
    return x;
}

int merge(unionset *u, int a, int b) {
    int fa = find(u,a), fb = find(u,b);
    if (fa == fb) return 0;
    u->fa[fa] = fb;
    return 1;
}

void output(unionset *u) {
    for (int i = 0; i < u->size; i++) {
        printf("(%d, %d)\t", i, u->fa[i]);
        if (i + 1 < u->size && i + 1 % 5 == 0) printf("\n");
    }
    printf("\n\n");
    return ;
}
```

```c
void clear(unionset *u) {
    if (u == NULL) return ;
    free(u->fa);
    free(u);
    return ;
}

int main() {
    srand(time(0));
    int op, a, b;
    #define MAX_OP 10
    #define MAX_N 10
    unionset *u = init(MAX_N);
    for (int i = 0; i < MAX_OP; i++) {
        op = rand() % 4;
        a = rand() % MAX_N;
        b = rand() % MAX_N;
        switch(op) {
            case 0: {
                printf("find %d <-> %d = %d\n", a, b,find(u,b));
            }break;
            default: {
                printf("union %d <-> %d = %d\n",a,b,merge(u, a,b));
            }break;
        }
        output(u);
    }

    return 0;
}
```