作业二：完成双数组字典树的AC自动机

主要函数：

```
int has_child(DATNode *trie, int p, int i) {
    return abs(trie[trie[p].base + i].check) == p;
}

void build_ac(DATNode *trie, int cnt) {
    if (cnt == 0) return ;
    if (trie[cnt].fail == 0) build_ac(trie, abs(trie[cnt].check));
    for (int i = 0; i < 'a'; ++i) {
if (!has_child(trie, cnt, i)) continue;
        if (trie[trie[cnt].base + i].fail) continue;
        int p = trie[cnt].fail, pre_p = cnt;
        while (p && !has_child(trie, p, i)) {
            if (trie[p].fail == 0) build_ac(trie,
abs(trie[p].check));
            pre_p = p;
            p = trie[p].fail;
        }
        if (p == 0) p = pre_p;
        else p = trie[p].base + i;
        trie[trie[cnt].base + i].fail = p;
        build_ac(trie, trie[cnt].base + i);
    }
    return ;
}
```

整体函数：

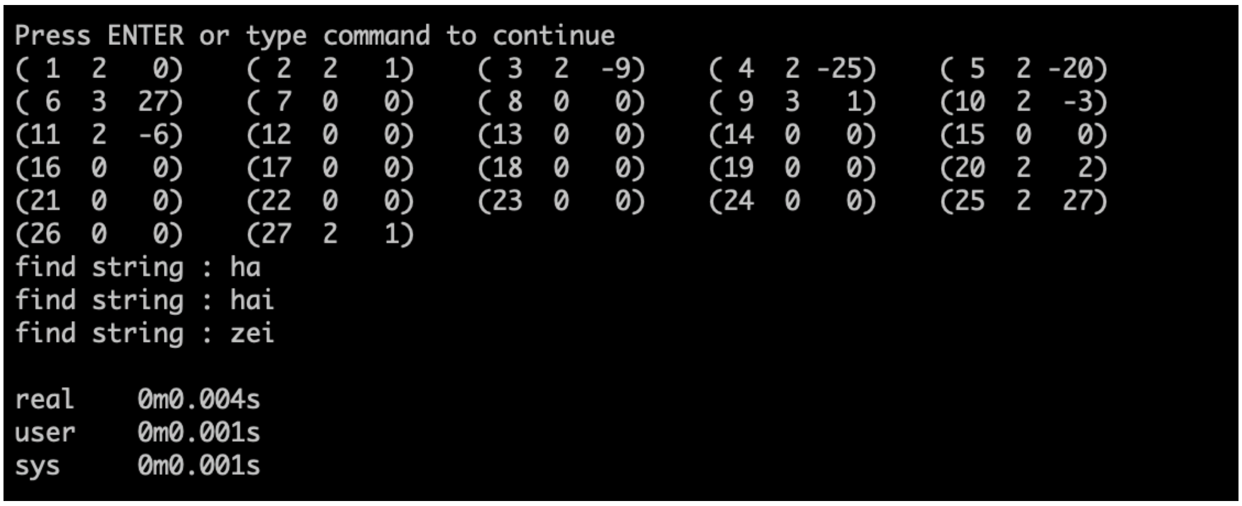
```
/**********************************************************************
******
    > File Name: homework_finish.cpp
    > Author: HaiZeiJoMA
    > Mail: 860007544@qq.com
    > Created Time: 三  1/23 20:26:12 2019
 *********************************************************************
*****/

#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <ctime>
#define max(a, b) ((a) > (b) ? (a) : (b))
```

```
using namespace std;

typedef struct Node {
    int base, check, fail;
    char *str;
} Node;

typedef struct DATNode {
    int flag;
    char *str;
    struct DATNode *next[26];
} DATNode, *Trie;

int node_cnt = 0;

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

void clear(Trie root) {
    if (root == NULL) return ;
    for (int i = 0; i < 26; i++) {
        clear(root->next[i]);
    }
    if (root->flag) free(root->str);
    free(root);
    return ;
}

DATNode *insert(Trie root, const char *str) {
    if (root == NULL) root = get_new_node();
    DATNode *p = root;
    for (int i = 0; str[i]; i++) {
        int ind = str[i] - 'a';
        if (p->next[ind] == NULL) p->next[ind] = get_new_node();
        p = p->next[ind];
    }
    p->flag = 1;
    p->str = strdup(str);
    return root;
}

int get_base(DATNode *node, Node *data) {
    int base = 2, flag = 0;
    while (!flag) {
        flag = 1;
        for (int i = 0; i < 26; i++) {
            if (node->next[i] == NULL) continue;
            if (data[base + i].check == 0) continue;
            flag = 0;
            break;
        }
        base += (!flag);
    }
    return base;
}

int build(DATNode *node, Node *data, int ind) {
    if (node == NULL) return 0;
    if (node->flag) data[ind].check = -data[ind].check,
data[ind].str = node->str;
    int max_ind = ind;
    data[ind].base = get_base(node, data);
    for (int i = 0; i < 26; i++) {
        if (node->next[i] == NULL) continue;
        data[data[ind].base + i].check = ind;
    }
    for (int i = 0; i < 26; i++) {
        if (node->next[i] == NULL) continue;
        int temp = build(node->next[i], data, data[ind].base + i);
        max_ind = max(max_ind, temp);
    }
    return max_ind;
}


int have_child(Node *node, int ind, int next) {
    return abs(node[node[ind].base + next].check) == ind;
}

int has_child(Node *data, int ind, int i) {
    return abs(data[data[ind].base + i].check) == ind;
}

//双数组判断迭代

void build_ac(Node *node, int ind) {
    if (ind == 0) return ;
    if (node[ind].fail == 0) build_ac(node, abs(node[ind].check));
    for (int i = 0; i < 26; ++i) {
        if (!have_child(node, ind, i)) continue;
        if (node[node[ind].base + i].fail) continue;
        int p = node[ind].fail, pre_p = ind;
        while (p && !have_child(node, p, i)) {
            if (node[p].fail == 0) build_ac(node,
abs(node[p].check));
```

```
            pre_p = p;
            p = node[p].fail;
        }
        if (p == 0) p = pre_p;
        else p = node[p].base + i;
        node[node[ind].base + i].fail = p;
        build_ac(node, node[ind].base + i);
    }
    return ;
}

void search_ac(Node *data, const char *str) {
    int p = 1;
    for (int i = 0; str[i]; i++) {
        while (p && !has_child(data, p, str[i] - 'a')) p =
data[p].fail;
        if (p == 0) p = 1;
        else p = data[p].base + str[i] - 'a';
        int q = p;
        while (q) {
            if (data[q].check < 0) printf("find string : %s\n",
data[q].str);
            q = data[q].fail;
        }
    }
    return ;
}

void output_da(Node *data, int n) {
    for (int i = 1; i <= n; i++) {
        if (i - 1 && i % 5 == 1) printf("\n");
        printf("(%2d %2d %3d)    ", i, data[i].base, data[i].check);
    }
    printf("\n");
    return ;
}

int main() {
    Trie root = NULL;
    root = insert(root, "hai");
    root = insert(root, "zei");
    root = insert(root, "ha");
    root = insert(root, "asd");
    root = insert(root, "zxc");
    Node *data = (Node *)calloc(sizeof(Node), (5 * 100));
    int da_cnt = build(root, data, 1);
    build_ac(data, 1);
    //output_da(data, da_cnt);    //展示所有的递归过程
    search_ac(data, "haizei");
    return 0;
}
```

测试结果:

测试字符串为: haizei



```
Press ENTER or type command to continue
( 1  2   0)    ( 2  2   1)    ( 3  2  -9)    ( 4  2 -25)    ( 5  2 -20)
( 6  3  27)    ( 7  0   0)    ( 8  0   0)    ( 9  3   1)    (10  2  -3)
(11  2  -6)    (12  0   0)    (13  0   0)    (14  0   0)    (15  0   0)
(16  0   0)    (17  0   0)    (18  0   0)    (19  0   0)    (20  2   2)
(21  0   0)    (22  0   0)    (23  0   0)    (24  0   0)    (25  2  27)
(26  0   0)    (27  2   1)
find string : ha
find string : hai
find string : zei

real    0m0.004s
user    0m0.001s
sys     0m0.001s
```