作业2：

右数第一趟、右数第四趟所有人，周三之前，交一份代码，将双数组字典树的 AC 自动机的构建过程，改成递归版。

需求：

将双数组字典树ａｃ自动机构建过程改变成递归版本

所需文件

自动机递归版，双数组字典树ａｃ自动机

作业提交方式：

邮箱：hug@haizeix.com 邮件名称：【作业一/二/三】姓名 代码放到 markdown 文档中，加上必要的说明和注释。

作业要求

所有作业，不仅要有代码实现，而且还要有充分的测试，用来说明你的东西是正确的。

如果测试过程不充分，也会被认定为作业不合格。自己设计几个数据，随便测一测那种的，肯定是不合格的。

作业

```
/*************************************************************************
    > File Name: ac.cpp
    > Author: ldc
    > Mail: litesla
    > Created Time: 2019年01月22日 星期二 21时25分33秒
 ************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 26
typedef struct Node {
    int flag;
    struct Node *next[SIZE];
} Node;

typedef struct DATNode {
    int base, check, fail;
} DATNode;
```

```
Node *getNewNode() {
    Node *p = (Node *)calloc(sizeof(Node), 1);
    return p;
}

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

int insert(Node *node, const char *str) {
    Node *p = node;
    int cnt = 0;
    for (int i = 0; str[i]; i++) {
        if (p->next[str[i] - 'a'] == NULL) {
            p->next[str[i] - 'a'] = getNewNode();
            cnt += 1;
        }
        p = p->next[str[i] - 'a'];
    }
    p->flag = 1;
    return cnt;
}

int getBase(Node *node, DATNode *trie) {
    int base = 1, flag = 0;
    while (!flag) {
        flag = 1;
        base += 1;//base是从2开始的
        for (int i = 0; i < SIZE; i++) {
            if (node->next[i] == NULL) continue;
            if (trie[base + i].check == 0) continue;
            flag = 0;
            break;
        }
    }
    return base;
}

int Transform(Node *node, DATNode *trie, int ind) {
    if (node == NULL) return 0;
    if (ind == 1) trie[ind].check = 0;//将一号节点的check设置称为0,不然递归时午饭判断第一次和第二次有什么区别
    if (node->flag) trie[ind].check = -trie[ind].check;
    trie[ind].base = getBase(node, trie);
    for (int i = 0; i < SIZE; i++) {
        if (node->next[i] == NULL) continue;
        trie[trie[ind].base + i].check = ind;
    }
```

```
    int cnt = ind;
    for (int i = 0; i < SIZE; i++) {
        if (node->next[i] == NULL) continue;
        int temp = Transform(node->next[i], trie, trie[ind].base + i);
        if (temp > cnt) cnt = temp;
    }
    return cnt;
}

int search(DATNode *trie, const char *str) {
    int p = 1;
    for (int i = 0; str[i]; i++) {
        int delta = str[i] - 'a';
        int check = abs(trie[trie[p].base + delta].check);
        if (check - p) return 0;
        p = trie[p].base + delta;
    }
    return trie[p].check < 0;
}

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

// 建立编号为ind节点的孩子的失败指针, 前提是编号为ind的节点失败指针已经建立了
void build_acdfs(DATNode *trie,int ind) {
    if(ind == 0) return ;//返回到根节点的失败指针
    if(trie[ind].fail == 0) build_acdfs(trie, abs(trie[ind].check));
    for (int i = 0; i < SIZE; i++) {
        int childind = trie[ind].base + i;
        if (!has_child(trie, ind, i)) continue;
        if (trie[childind].fail) continue;//失败指针已经建立不用再次建立
        int p = trie[ind].fail;
        while (p && !has_child(trie, p, i)) {
            if (trie[p].fail == 0) build_acdfs(trie,abs(trie[p].check));
            p = trie[p].fail;
        }
        if (p == 0) p = 1;
        else p = trie[p].base + i;
        trie[childind].fail = p;
        build_acdfs(trie,childind);
    }
}

int match(DATNode *trie, const char *str) {
    int cnt = 0;
    int p = 1, q;
    while (str[0]) {
        while (p && !has_child(trie, p, str[0] - 'a')) p = trie[p].fail;
        if (p == 0) p = 1;
        else p = trie[p].base + str[0] - 'a';
        q = p;
        while (q) {
```

```
            cnt += (trie[q].check < 0);
            q = trie[q].fail;
        }
        str++;
    }
    return cnt;
}

int main() {
    int n, cnt1 = 1, cnt2 = 0;
    char str[100];
    Node *root = getNewNode();
    scanf("%d", &n);
    while (n--) {
        scanf("%s", str);
        cnt1 += insert(root, str);
    }
    DATNode *trie = (DATNode *)calloc(sizeof(DATNode), cnt1 * 10);
    cnt2 = Transform(root, trie, 1);
    build_acdfs(trie, 1);
    scanf("%s", str);
    printf("matchcount : %d\n", match(trie, str));
    clear(root);
    return 0;
}
```

遇到的bug

1. 根节点的check为1,在递归时,无法判从check判断是否是第一次到这,还是后面的到这,因为buicd传入的参数下标是1,

   解决方案:把跟节点的check变成0,当递归到ind为0时就是递归截止的条件了,

2.
   ```
           while (p && !has_child(trie, p, i)) {
               if (trie[p].fail == 0) build_acdfs(trie,abs(trie[p].check));
               p = trie[p].fail;
           }
   ```

代码在这里面吧`abs(trie[p].check)`写成了`abs(trie[ind].check)`,进行了修改

3. match函数p指针没有后移

   添加了这一行

   ```
           if (p == 0) p = 1;
           else p = trie[p].base + str[0] - 'a';
   ```

测试截图

输入格式

第一行输入一个整数 n（$1 \leq n \leq 1000$），表示蒜头君学习的 n 个单词。

接下来 n 行，每行输入一个字符串，仅由小写字母组成，长度不超过 20。

最后输入一个字符串 S，仅由小写字母组成，长度不超过 10^5，表示蒜头君看的文章。

输出格式

依次输出 n 行，每行输出格式为 `i: num`，i 表示单词的编号（从 0 开始编号），num 为一个整数，表示第 i 个单词在母串 S 中出现了 num 次。

样例输入 复制

```
2
ab
bca
abcabc
```

样例输出 复制

```
0: 2
1: 1
```

main.c — C 语言

```c
}

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

// 建立编号为ind节点的孩子的失败指针，前提是编号为ind的节点失败指针已经建立了
void build_acdfs(DATNode *trie, int ind) {
    if(ind == 0) return ;//返回到根节点的失败指针
    if(trie[ind].fail == 0) build_acdfs(trie, abs(trie[ind].check));
    for (int i = 0; i < SIZE; i++) {
        int childind = trie[ind].base + i;
        if (!has_child(trie, ind, i)) continue;
        if (trie[childind].fail) continue;//失败指针已经建立不用再次建立
        int p = trie[ind].fail;
        while (p && !has_child(trie, p, i)) {
            if (trie[p].fail == 0) build_acdfs(trie,abs(trie[p].check));
            p = trie[p].fail;
        }
        if (p == 0) p = 1;
        else p = trie[p].base + i;
        trie[childind].fail = p;
        build_acdfs(trie,childind);
    }
}

void match(DATNode *trie, const char *str, int *ret) {
    int p = 1, q;
    while (str[0]) {
```

恭喜你满分通过了这道题！ 稍后继续 继续学习