## 作业1：

右数第二趟所有人，周三之前，交一份报告，用来说明递归版 AC 自动机中为什么不加【if (node->next[i]->fail) continue;】，就会出现段错误。

## 需求

用来说明递归版 AC 自动机中为什么不加【if (node->next[i]->fail) continue;】，就会出现段错误。

## 所需文件

递归版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。

## 推导思路

先正常推导递归版思路

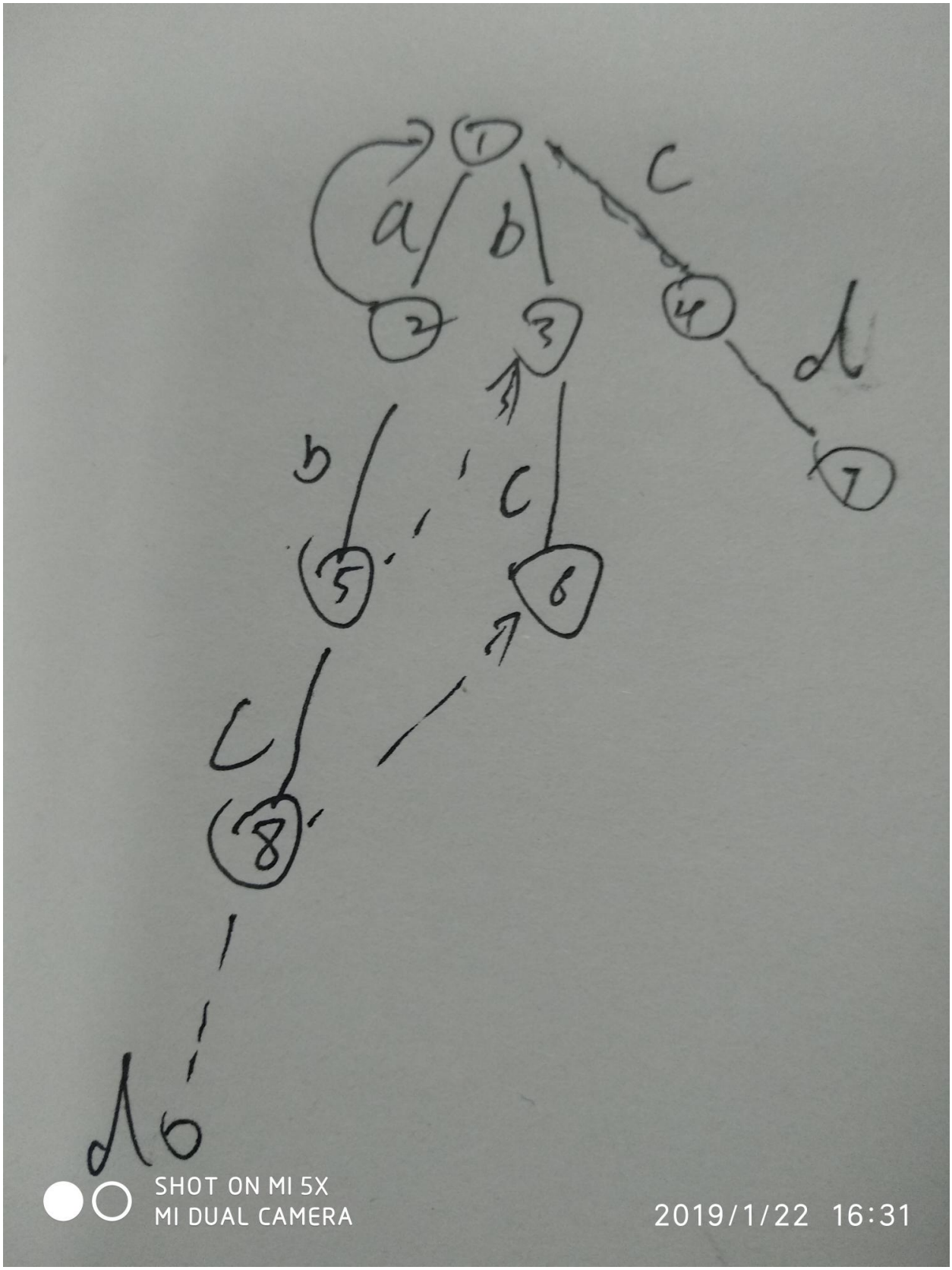
1. 首先我们接受一个节点，就去建立他儿子节点的失败指针，可以得到如下代码

```
void build_automaton(TrieNode *node) {
    if (node == NULL) return ;
    //如果节点为空返回
    for (int i = 0; i < SIZE; i++) {
        if (node->next[i] == NULL) continue;
        //如果他没有这个孩子结束本次循环
        Node *p = node->fail;//他的的失败指针
        while (p && p->next[i] == NULL) {
            //看他失败指针有没有这个孩子，没有就递归沿着失败指针向上找
            p = p->fail;
        }
```

```
            if (p == NULL) p = root;
            //代表没找到则失败指针位跟节点
            else p = p->next[i];
            //找到了他的失败指针就是p相应位置的孩子
            node->next[i]->fail = p;
            //失败指针赋值
            build_automaton(node->next[i]);
            //去建立儿子节点的失败指针
        }
        return ;
}
```

2. 上面一个是基础版本的思路框架，但是会有几个问题？当我们沿着一条路往下走的时候，失败指针会去递归匹配失败指针的失败指针，但是我们只建立了这个分支的失败指针，其他路的失败指针我们还没有进行建立，所以我们需要跳转到那个节点去建立条支路的失败指针（如图），（当我们建立８号节点的失败指针时，６号节点下面是没有ｄ字符的，但是从一个正常自动机的角度来说６号节点的失败指针是４，４下面有ｄ字符）

   又因为建立一个节点的失败指针是根据他的父亲节点的失败指针去建立的所以，我们个结构体添加一个父亲指针，指向他父亲的位置，用来建立连通这条支路，以便建立这条支路的失败指针，实现这些会得到一下代码




```
//上面解释过得代码我们就不进行解释了
void build_automaton(TrieNode *node) {
    if (node == NULL) return ;
    if (node->fail == NULL) build_automaton(node->father);
```

```
    //先看第一步
    //第二步，从第一步跳转而来，因为本条支路的所有的节点的失败指针都没有确定我们就可以
    //通过这句话递归向上，找到根节点，根节点父亲节点为空，上面那个ｉｆ为这个跳转的截止条件
    for (int i = 0; i < SIZE; i++) {
        if (node->next[i] == NULL) continue;
        if (node->next[i]->fail) continue;
        //第三步，因为我们一直跳到了根节点，上一次到根节点的状态与这次到根节点的状态唯一的区别呢就是，又多了一条孩子失败指针已经确立了，为了避免重复确立我们不去搜索已经确立失败指针的子孩子
        //同时如果出现我们上面那个图的情况的时候，每次都遍历第一条支路，到达ｃ节点的时候就会递归向上形成一条环路.
        Node *p = node->fail;
        while (p && p->next[i] == NULL) {
            //第一步，如果他的失败指针的失败指针没有确立我们就确立失败指针的父亲节点
            if (p->fail == NULL) build_automaton(p->father);
            p = p->fail;
        }
        if (p == NULL) p = root;
        else p = p->next[i];
        node->next[i]->fail = p;
        build_automaton(node->next[i]);
    }
    return ;
}
```

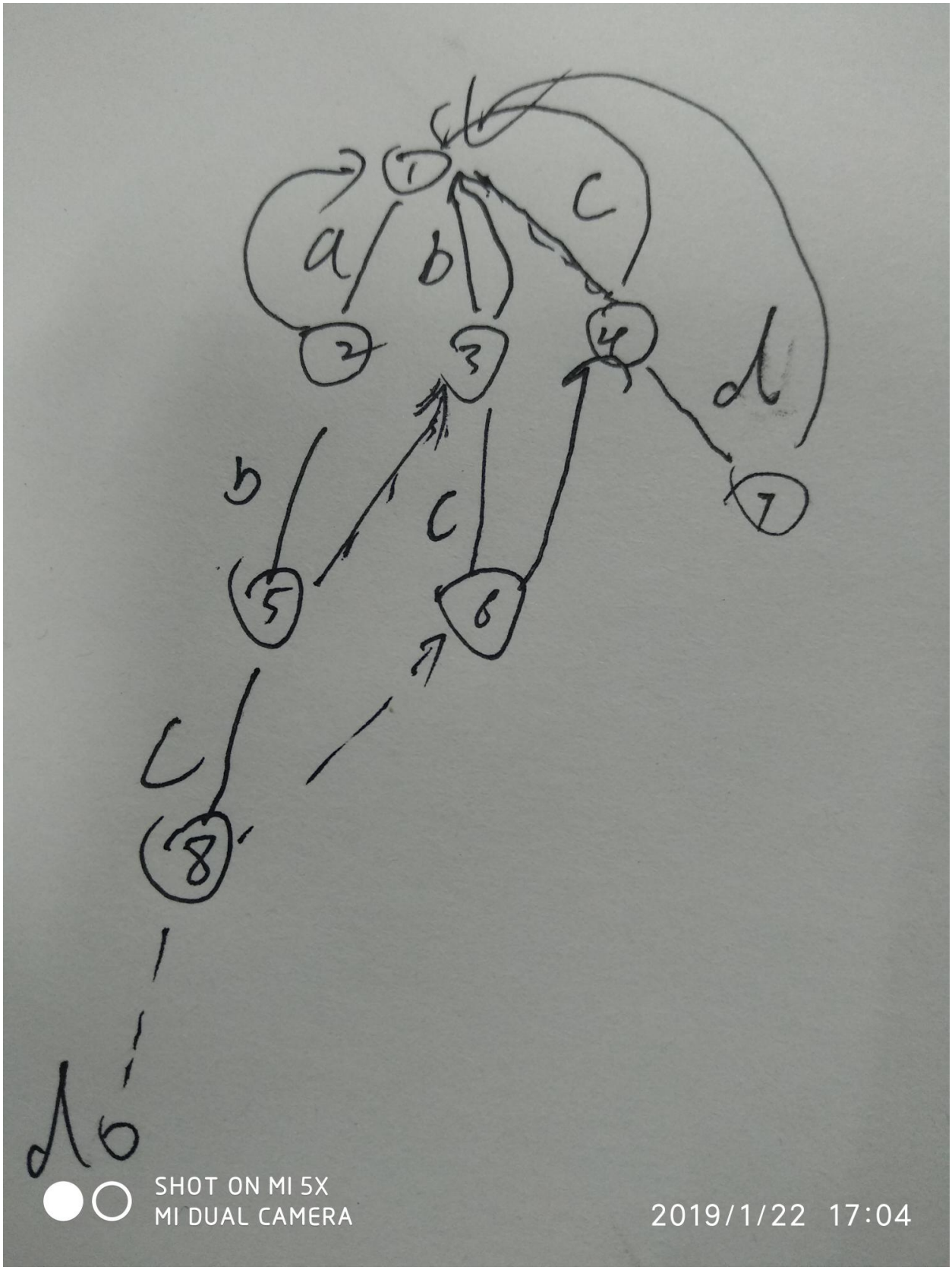
**成环问题请看上面第三步**

详细讨论：我们上图演示的是四层就可以形成一条环路，四层以上可不可以成环呢?

我们可以知道本层父亲节点失败指针一定是指向根节点的．所以只有两种情况一中是根节点下有那个节点，跳出while (p && p->next[i] == NULL)，　二是跟节点下没有这个节点，进行递归，但是因为跟节点没有父亲节点，所以递归进去就知道被第一行的ｉｆ返回了．所以跳出while (p && p->next[i] == NULL)，继续向下进行，不会有成环的可能.

3. 不知道你有没有注意到当节点一直递归到最上层的时候，如果没有匹配到的节点，我们就会将他的失败指针指向根节点，但是我们并没有说根节点是怎么来的，有几种可行的方案，一设立全局变量，二将根节点设置成为函数参数，三，如果匹配不成功，节点ｐ是不断向上跳的，而且每次都去确立那一条支路的失败指针，如图所示，当第一次返回时绝大部分的失败指针都已经确定，然后就变成了非递归版的不断向上找失败时对应节点了

   当完全失败周ｐ等于ＮＵＬＬ，则上一次ｐ指针一定等一ｒｏｏｔ，所以我们可以建立一个指针保存上一次的值




```
void build_automaton(TrieNode *node) {
    if (node == NULL) return ;
    if (node->fail == NULL) build_automaton(node->father);
    for (int i = 0; i < SIZE; i++) {
```

```
        if (node->next[i] == NULL) continue;
        if (node->next[i]->fail) continue;
        Node *p = node->fail, *pre_p = node;
        //保存上一次的值
        while (p && p->next[i] == NULL) {
            if (p->fail == NULL) build_automaton(p->father);
            pre_p = p;
            //保存上一次的值
            p = p->fail;
        }
        if (p == NULL) p = pre_p;
        else p = p->next[i];
        node->next[i]->fail = p;
        build_automaton(node->next[i]);
    }
    return ;
}
```