# ac自动机

## 主要数据结构

class acNode {  
public:  
 static int m; //树的阶数  
 static int stateNode; // 状态节点数  
 static int stateTransition;//状态转移次数  
 acNode \*\* ptr; // 下一层的指针ptr[0]表示的是上一层的最后一个叶子节点  
 std::string value; //存储叶子节点的字符串  
 int appearCount; //叶子节点的时候表示当前字符串出现的次数  
 int nodeKind; //0表示分支节点 1 表示叶子节点  
 acNode \* fail; // 失效指针  
 int dep;  
 acNode(int nodeKind);  
 void insert\_node(std::string value,bool flag = false);//将value插入到trie树中，只有root根节点可以调用该函数  
 int query(std::string &value,bool flag = false);//查询树中节点在value中出现的次数  
 int queryWord(std::string &value); //查询一个单词出现的次数，在这之前需要调用过一次query  
 void init(); //root初始化函数 只有root可以调用  
 void buildFail();//使用BFS创建fail指针 只有root调用该函数  
 void close();//只有root可以调用，释放树节点空间  
 void dfsDestoryNode(acNode \*now);// 递归删除节点  
};

创建了一个acNode类，这个类表示的是AC自动机中的每个节点，以及对节点的对应操作，见表1-1和表1-2是该类属性及其方法的含义。

表1-1 acNode属性及其含义

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 含义 |
| m | static int | 树的阶数 |
| stateNode | static | 状态节点数 |
| stateTransition | static int | 状态转移数 |
| ptr | acNode \*\* | 当前节点与孩子节点的连接处 |
| value | string | 表示存储的字符串的内容 |
| appearCount | int | 匹配到的次数 |
| nodeKind | int | 分支节点，叶子节点的标志 |
| fail | acNode\* | 失效指针 |
| dep | int | 节点所在层数（该变量用来调试程序） |

表1-2 acNode方法及其含义

|  |  |
| --- | --- |
| 方法 | 含义 |
| acNode(int nodeKind) | 构造函数，参数表示当前是叶子节点还是内部节点 |
| int query(std::string &value,bool flag=false) | 查询树中节点在value中出现的次数 |
| int queryWord(std::string &value) | 查询一个单词出现的次数，在这之前需要调用过query |
| void init() | root初始化函数，只有root可以调用 |
| void buildFail() | 使用BFS创建fail指针，只有root调用该函数 |
| void close() | 只有root可以调用，释放树节点空间 |
| void dfsDestroyNode(acNode \*now) | 递归删除节点 |
| void insert\_node(std::string value,bool flag=false) | 将value插入到trie树种，只有root根节点可以调用该函数 |

## 主要流程

首先创建root指针，调用root->init()初始化环境并设定阶数m，然后调用insert\_node()函数将关键词插入到AC自动机中。然后调用root->query()将每个content在AC自动机中进行查找。最后调用root->queryWord()查看每个关键词出现的次数，最后进行统计。

insert\_node(value),首先根据静态变量m获得每层存储的二进制数位的个数。遍历vlaue,根据当前其二进制位num，查找到输入哪一个子节点。如果该子节点为空，则开辟分支节点，如果不为空，则遍历到下一层，如果是叶子节点，则开辟分支节点，并将新的分支节点插入到该叶子节点之上，然后遍历到该分支节点。当遍历完value，最后一次遍历，如果子节点位空，则直接创建叶子节点，如果不为空，则创建叶子节点放到下一层的ptr[0]上。

query\_word(value)，根据静态变量m获得每层的二进制位数，遍历value,计算出当前的num,如果对应的子节点为空，则返回-1说明没有找到该内容，如果不为空则继续遍历下一层。当value遍历到最后一位的时候，如果是叶子节点则判断其内容是否是value，是返回其appearCount变量值，否返回-1，如果是分支节点的话，则判断其对应下一层的ptr[0]的内容是否是value，是返回appearCount变量值，不是返回-1表明没有找到。

buildFail()，使用BFS进行创建fail指针，每个fail指针返回其最大前缀的位置。

## 实验过程

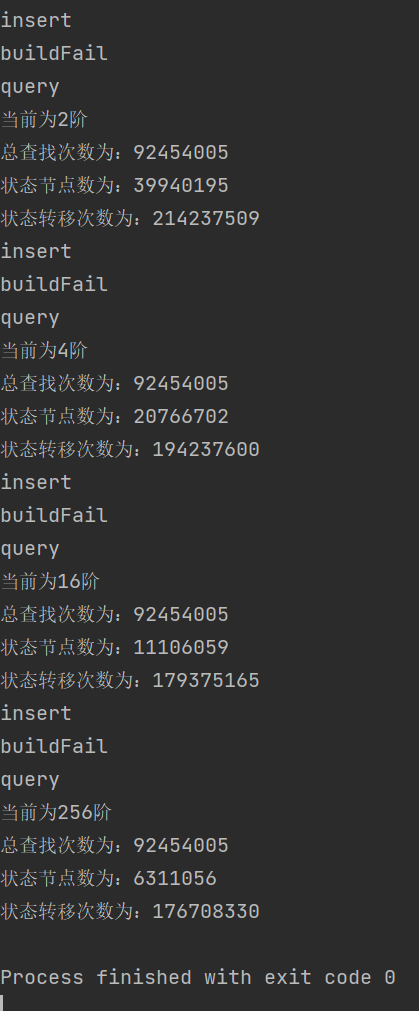


图1 运行结果

如图1所示，是程序运行4中不同阶数的运行结果。

## 技术指标

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 阶数 | 模式串出现总次数 | 状态节点总数 | 状态转移次数 | 内存 |
| 2 | 9245405 | 39940195 | 214237509 | 4265MB |
| 4 | 92454005 | 20766702 | 194237600 | 2510MB |
| 16 | 92454005 | 11106059 | 179375165 | 2248MB |
| 256 | 92454005 | 6311056 | 176708330 | 8524MB |

## 结论和总结

AC自动机对于这种情况的查找极具效果，按照老师给的数据集大概在5~6分钟之间可以使用AC自动机跑完，同时我在python上使用暴力查找的方法进行查找，想寻找一个标准答案时，发现只是将keyword跑完十分之三左右，就已经跑了十几个小时了，由此可以看出对于一个特定问题，一个有效的数据结构和算法对于时间的效率的提升有非常大的作用。