AC自动机

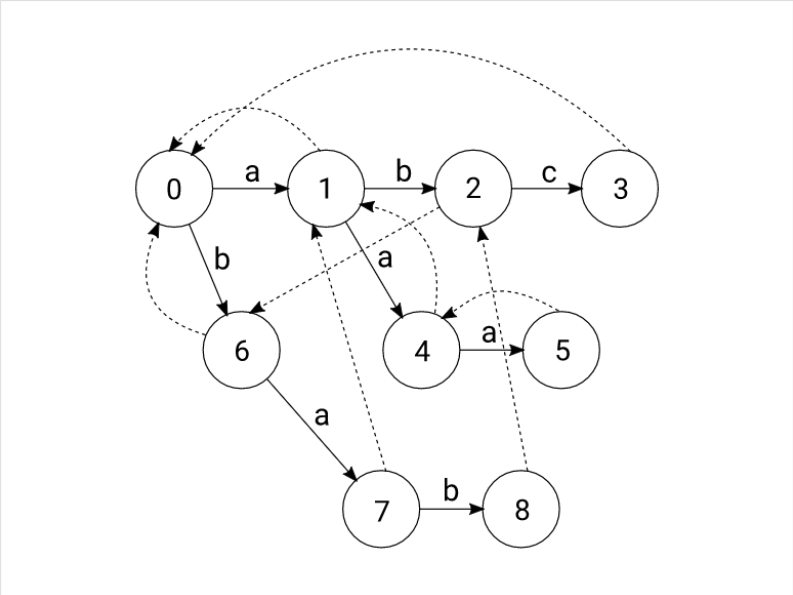
引入

AC 自动机是一种多模式串匹配算法，一般用于解决对于在文本串中匹配一系列模式串（例：给一个文本串和一系列模式串，问模式串在文本串中一共出现了多少次）

构造

具体的构造方法我们可以参考 KMP，在每次匹配失败了之后，则需要从 i 回到 $fail(i)$，即 $fail(i)$ 位置的前缀的是 i 这个位置的前缀的后缀。

而 AC 自动机则是在 $trie$ 上实现这样的操作。



如图所示

设 i 的父亲为 i'，指向i点的边上的字母为c

显然，当 $fail(i')$ 有字母 c的出边时，该出边的指向的点即为 $fail(i)$，图中 $fail(7)=1,fail(8)=2$，

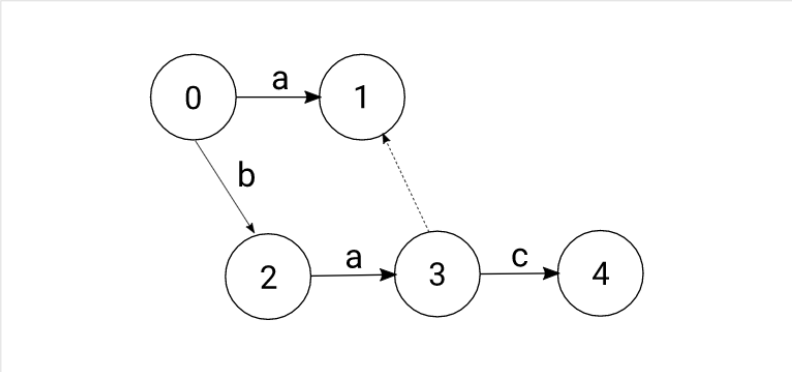
否则，我们就应当沿着 $fail$ 函数一直向上寻找，直到找到为止，如果找不到一个符合条件的点，则 $fail(i)$ 为根。（图中fail(3)=0，

匹配

有了之前的构造之后我们的匹配较为简单，设当前在i点，每次新加入字符c,都检查i点有没有c的出边，如果有，则转移到该点，否则沿着$fail$去寻找这样的点（没有就会回到根结点）

如果到了一个单词结点上，则代表该单词被匹配了（可能会有i点不是单词但$fail(i)$是单词的情况）。

如下图 3 到 1 的情况。



为了解决此类问题，我们又可以引入后缀链接□$nxt(i)$ 表示从i沿着失配边转移，能够到达的第一个单词结点。

后缀链接可以在失配指针之后求出，如果 $fail(i)$ 为单词结点，则 $nxt(i)=fail(i)$,否则 $nxt(i)=nxt(fail(i))$

优化

由于每次失配时需要用到失配指针，每次加入字符时经过节点数不确定，复杂度可能退化，但对于一个状态，添加一个字符后，转移到的状态是确定的，这也意味着我们可以预处理每一个状态可能装一道的所有状态。

对于节点 i ,如果它有字符 c的出边,则加入 c 时，它可以直接转移到该边指向结点，否则应该转移到fail(i)加入对应字符转移到的点上，我们可以用递推的方式求出这些转移方式，加入这些边，得到$Trie$图

模板题

P3808 □模板□AC自动机（简单版）

ps:本题由于只记录串出现次数，可以通过标记来优化复杂度。

```
#include<iostream>
#include<iomanip>
#include<cstdio>
#include<algorithm>
#include<map>
#include<stack>
#include<queue>
#include<complex>
#include<cmath>
#include<cstring>
using namespace std;
const int maxn=1000005;
char s[maxn];
int cnt,ch[maxn][26],Count[maxn],fail[maxn];
int nxt[maxn];
void Insert(){
```

```
    scanf("%s",s+1);
    int len=strlen(s+1);
    int rt=0;
    for(int i=1;i<=len;++i){
        int t=s[i]-'a';
        if(!ch[rt][t])ch[rt][t]=++cnt;
        rt=ch[rt][t];
    }
    ++Count[rt];
}
queue<int> Q;
void Build_AC(){
    Q.push(0);
    while(!Q.empty()){
        int x=Q.front();Q.pop();
        for(int i=0;i<26;++i)
            if(ch[x][i]){
                if(x)fail[ch[x][i]]=ch[fail[x]][i];
                Q.push(ch[x][i]);
                if(Count[ch[fail[x]][i]])nxt[ch[x][i]]=ch[fail[x]][i];
                else nxt[ch[x][i]]=nxt[ch[fail[x]][i]];
            }
            else ch[x][i]=ch[fail[x]][i];
    }
}
void Query(){
    scanf("%s",s+1);
    int len=strlen(s+1),rt=0;
    int ans=0;
    for(int i=1;i<=len;++i){
        int t=s[i]-'a';
        int a=ch[rt][t];
        while(a){
            if(Count[a]==-1)break;
            ans+=Count[a];
            Count[a]=-1;
            a=nxt[a];
        }
        rt=ch[rt][t];
    }
    printf("%d",ans);
}
int n;
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i)Insert();
    Build_AC();
    Query();
    return 0;
}
```