



数据结构和算法实习

郭 炜

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄



北京大学
PEKING UNIVERSITY

信息科学技术学院

Trie图



北京大学
PEKING UNIVERSITY

信息科学技术学院

Trie图的作用



法国勃朗峰

我们是谁？



Trie图!



我们任务是什么？



多模式串的字符串匹配!



我们有多快？



建好图以后 $O(n)$!



多模式串的字符串匹配问题

- 每个病毒都有特征字符串，可以看作一个模式串(子串)
- 文件包含某个病毒的特征字符串，就认为被这个病毒感染
- 杀毒软件保存了成千上万个病毒的特征字符串
- 如何只扫描文件（母串）一遍，就发现感染该文件的病毒

多模式串的字符串匹配问题

关键:

- 母串当前字符，同时($O(1)$ 时间) 匹配多个模式串当前字符
- 失配时（母串当前字符，不能匹配上任意一个模式串当前字符），母串指针不回溯，调整若干个模式串当前字符到合适位置，继续进行匹配。相当于若干个模式串的指针同时回溯到合适位置 --- KMP 算法思想的扩展



北京大学
PEKING UNIVERSITY

信息科学技术学院

KMP算法回顾




法国勃朗峰

KMP算法

KMP算法是如何避免母串指针回溯的？

母串： aabcdaa**k**g.....
子串： aabcdaa**f**



KMP算法

暴力算法母串指针需要回溯。假设前 $n-1$ 个字符匹配，第 n 个失配：



母串： $a_1 a_2 a_3 \dots a_{n-1} a_n \dots$
子串： $b_1 b_2 b_3 \dots b_{n-1} b_n \dots$

母串指针如果不回溯，直接用 a_n 和 b_1 比，可能会忽略下面的情况：

母串： $a_1 a_2 a_3 a_4 a_5 \dots a_{n-1} a_n \dots$
子串： $b_1 b_2 \dots b_{i-1} b_i \dots b_n \dots$
 $a_4 a_5 \dots a_{n-1} a_n$ 和 $b_1 b_2 \dots b_{i-1} b_i$ 匹配上

KMP算法

若发生了下述情况：

母串： $a_1 a_2 a_3 a_4 a_5 \dots a_{n-1} a_n \dots$

子串： $b_1 b_2 \dots b_{i-1} b_i \dots b_n \dots$

$a_4 a_5 \dots a_{n-1} a_n$ 和 $b_1 b_2 \dots b_{i-1} b_i$ 匹配

则 $b_1 b_2 \dots b_{i-1}$ 是 $a_1 a_2 a_3 a_4 a_5 \dots a_{n-1}$ 的后缀，也是 $a_1 a_2 a_3 a_4 a_5 \dots a_{n-1}$ 的前缀

类似情况可能有多种，考察其中 b_1 位置最靠左的。

则 $b_1 b_2 \dots b_{i-1}$ 就是所有既是 $a_1 \dots a_{n-1}$ 前缀又是 $a_1 \dots a_{n-1}$ 后缀的串中最长的（不包括 $a_1 \dots a_{n-1}$ ）。

KMP算法

发生了下述情况时：

母串： $a_1 a_2 a_3 a_4 a_5 \dots a_{n-1} a_n \dots$

子串： $b_1 b_2 \dots b_{i-1} b_i \dots b_n \dots$

$a_4 a_5 \dots a_{n-1} a_n$ 和 $b_1 b_2 \dots b_{i-1} b_i$ 匹配

接下来要比较的就是 a_n 和 b_i

那么，不如当初母串指针不要回溯，直接比较 a_n 和 b_i -----前提是：

找到了一个既是 $a_1 \dots a_{n-1}$ 前缀又是 $a_1 \dots a_{n-1}$ 后缀的最长的串

$b_1 b_2 \dots b_{i-1}$ 。

KMP算法


KMP算法是如何避免母串指针回溯的？

母串： acabaca[↓]kg.....
子串： acabacaef...

KMP算法

KMP算法是如何避免母串指针回溯的？


母串： acabaca**k**g.....
子串： acab**a**caef...



KMP算法

KMP算法是如何避免母串指针回溯的？


母串： acabaca**k**g.....
子串： a**c**abacaef...



KMP算法

KMP算法是如何避免母串指针回溯的？


母串： acabaca**k**g.....
子串： **a**cabacaef...



KMP算法

KMP算法是如何避免母串指针回溯的？

母串： aabcdaakg.....
子串： acabacaef...





北京大学
PEKING UNIVERSITY

信息科学技术学院

多模式串匹配



美国黄石公园大棱镜温泉

多模式串匹配

如果有多个子串：

母串： abcde**k**g.....
子串1： abcde**g**c
子串2： abcde**u**ae
子串3： cden
子串4： dek
子串5： sdecse

希望母串指针不回溯，要记住已经匹配了哪些子串的前缀。

在一个子串的失配位置，还应该和别的子串进行比较，是有可能匹配上的。

多模式串匹配

如果有多个子串：

母串： abcde**k**g.....
子串1： abcde**g**c
子串2： abcde**u**ae
子串3： cde**n**
子串4： dek
子串5： sdecse

希望母串指针不回溯，要记住已经匹配了哪些子串的前缀。

在一个子串的失配位置，还应该和别的子串进行比较，是有可能匹配上的。

多模式串匹配

如果有多个子串：

母串： abcde**k**g.....
子串1： abcde**g**c
子串2： abcde**u**ae
子串3： cde**n**
子串4： de**k**
子串5： sdecse

希望母串指针不回溯，要记住已经匹配了哪些子串的前缀。

在一个子串的失配位置，还应该和别的子串进行比较，是有可能匹配上的。

多模式串匹配

如果有多个子串：

母串： abcde**k**g.....
子串1： **a**bcdegc
子串2： **a**bcdeuae
子串3： **c**den
子串4： **s**decse

如果都匹配不上，再从每个子串的开始匹配

多模式串匹配

如果有多个子串：

母串： abcde**kg**.....
子串1： abcdegc
子串2： abcdeuae
子串3： cden
子串4： sdecse

母串当前字符和所有子串的头一个字符都不匹配，则移动母串指针到下一个字符



北京大学
PEKING UNIVERSITY

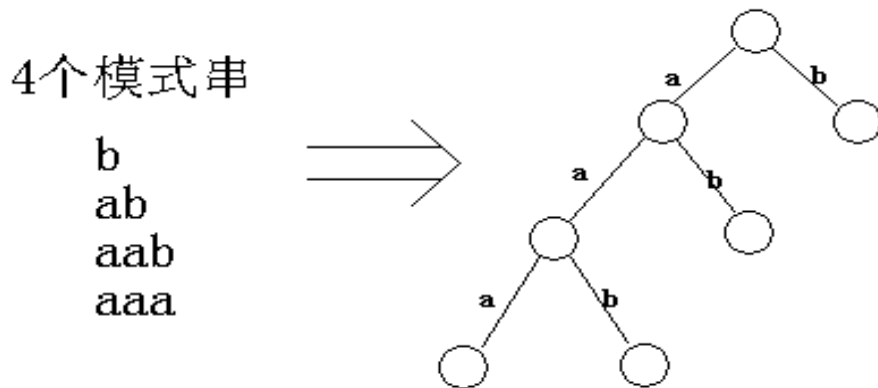
信息科学技术学院

Trie树



黄石公园老忠实泉

Trie树 (字典树, 单词前缀树)



- 由若干模式串构建而成。树中任一节点p，都对应于一个字符串S，S由从根出发走到p所经过的边上的字符构成，S是一个或多个模式串的前缀。
- 任一模式串的前缀，都对应于树中的唯一节点

Trie树 (字典树, 单词前缀树)

将串s插入到根为root的trie树:

```
void build(string s, trienode * root)
{
    trienode* p=root;
    for (int i=0;i<s.size();++i)
    {
        if (p->child[s[i]-'a']== NULL)
            p->child[s[i] -'a'] = new trienode();
        p=p->child[s[i] -'a'];
    }
}
```

```
struct trienode
{
    trienode * child[26] ;
    //假设所有字符就是26个小写字母
    trienode() {
        memset(child,0,
        sizeof(child));
    }
};
```

将n个模式串插入到一棵单词前缀树的时间复杂度为 $O(\sum \text{len}(i))$, 其中 $\text{len}(i)$ 为第i个模式串的长度。



北京大学
PEKING UNIVERSITY

信息科学技术学院

Trie图



北京黄花城水长城

Trie图

有时也称为AC自动机，DFA

- trie图可以由trie树为基础构造出来
- 对于插入的每个模式串，其插入过程中使用的最后一个节点称为终止节点
- 要求一个母串包含哪些模式串，以母串作为Trie图的输入，在Trie图上行走，走到终止节点，就意味着匹配了相应的模式串（没能走到终止节点，并不意味着一定不包含模式串）

Trie图

模式串:

abcd

abc

abe

ae

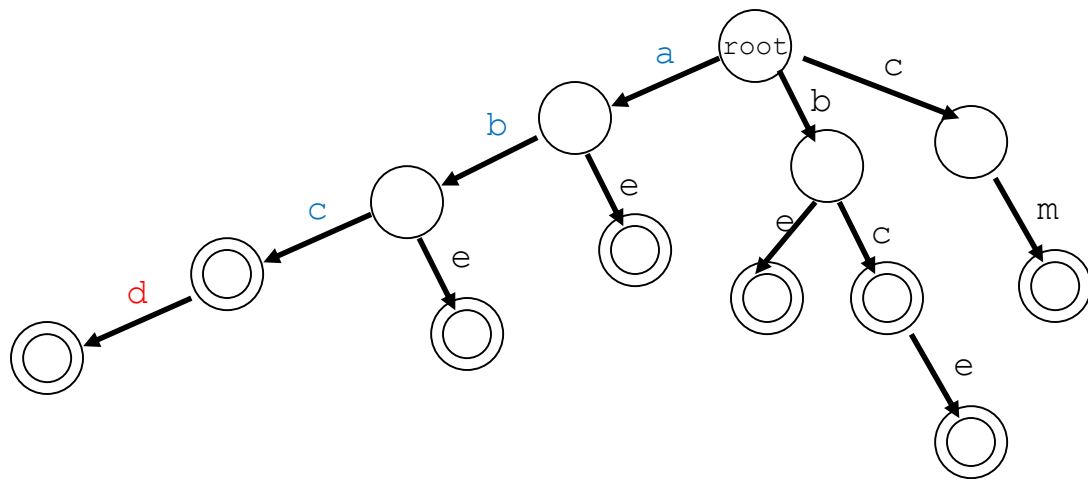
bc

be

bce

cm

母串: kcabcmgh



走不动了就回到root, 从b开始继续在DFA上走。这样, 母串指针就回溯了。如何避免母串指针回溯, 即如何做到只要回到root, 就直接从母串的下一个未访问过的字符开始继续走?

Trie 図

模式串：

abcd

abc

abe

ae

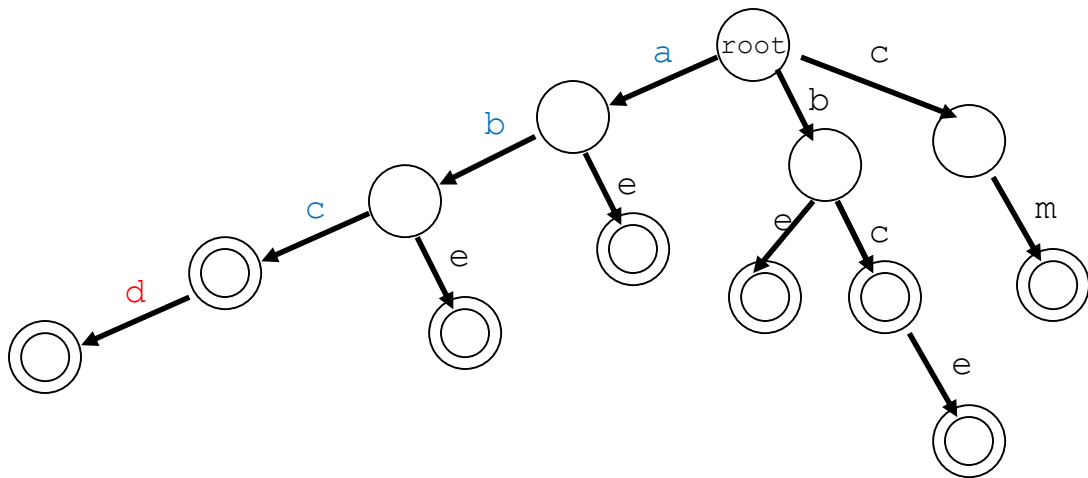
bc

be

bce

cm

母串: kcabcmgh



走不动了就回到root，从b开始继续在DFA上走。这样，母串指针就回溯了。如何避免母串指针回溯，即如何做到只要回到root，就直接从母串的下一个字符开始继续走？

关键：要记住哪些模式串的哪个前缀已经被匹配

Trie图

模式串:

abcd

abc

abe

ae

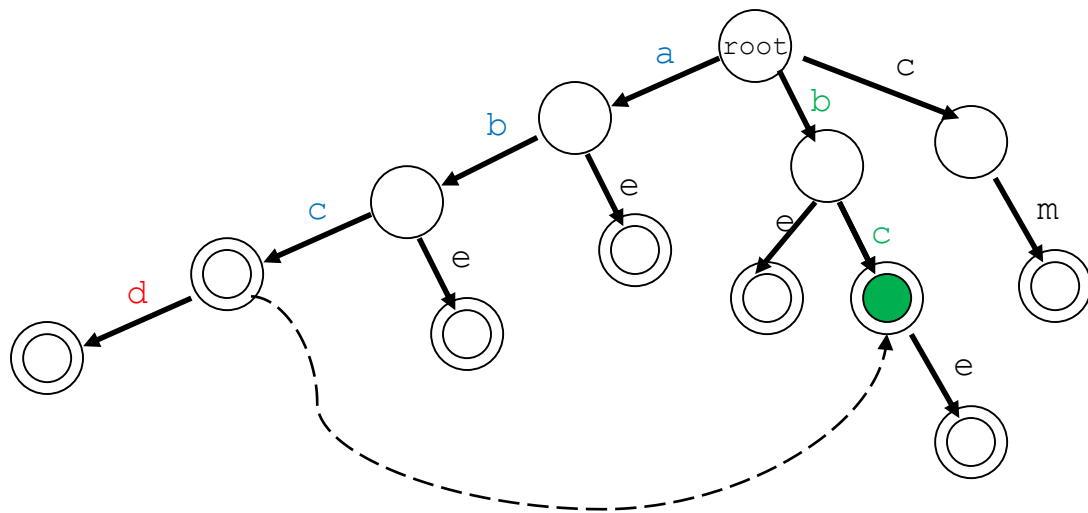
bc

be

bce

cm

母串: kc**abc**mgh



发现绿色节点对应的字符串是已经匹配的母串 `abc` 的后缀 (`bc`一定是某个子串的前缀), 那么就应该让母串中 `abc` 的后一个字符继续和绿色节点出发的字符匹配, 即试图匹配一个前缀为 `bc` 的子串。

Trie图

模式串:

abcd

abc

abe

ae

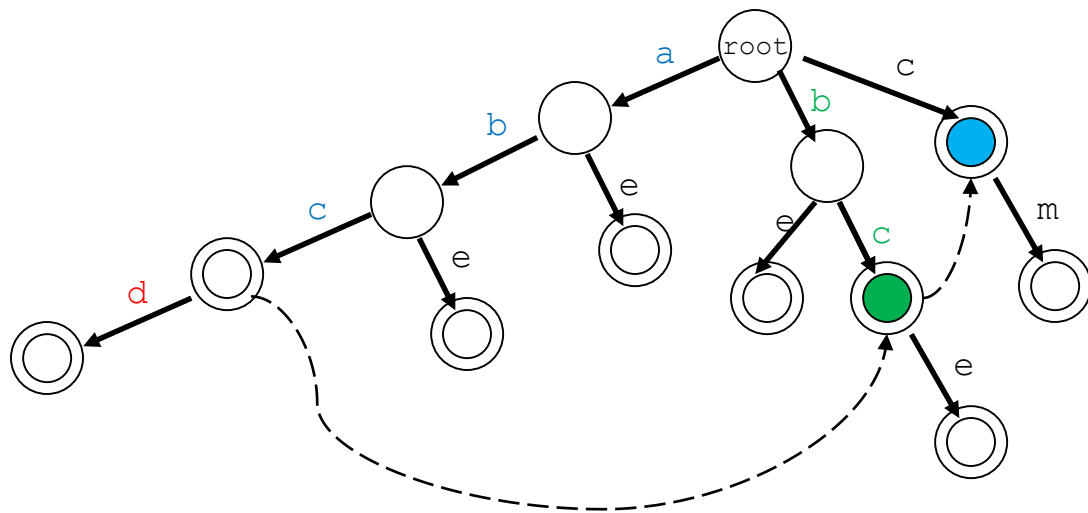
bc

be

bce

cm

母串: kc**a**bc**m**gh



m和e失配。但是发现蓝色节点节点对应的字符串是已经匹配的母串 abc 的后缀c, 那么就应该让母串中 abc 的后一个字符继续和蓝色节点出发的字符匹配

Trie图

模式串:

abcd

abc

abe

ae

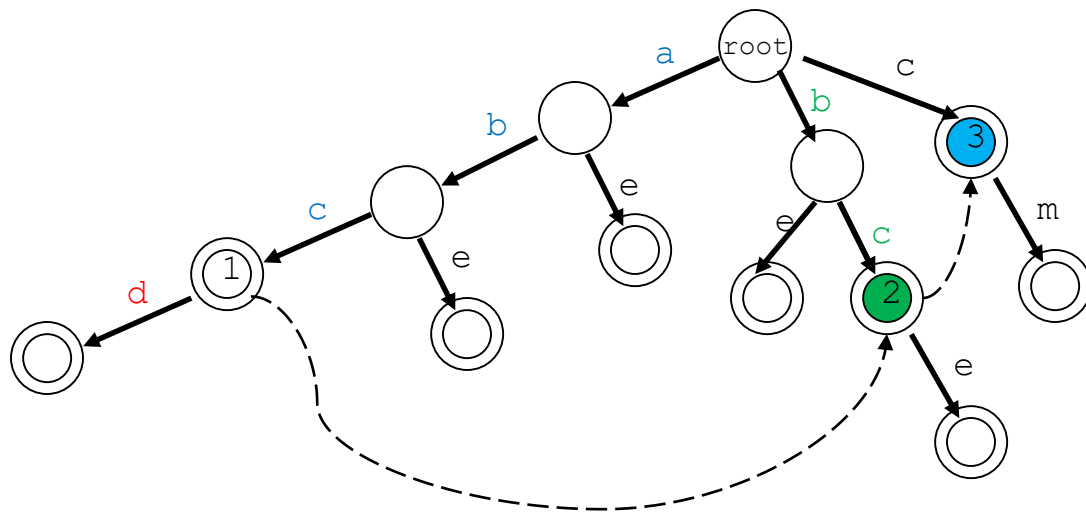
bc

be

bce

cm

母串: kc**a**bc**m**gh




要想办法在Trie图上走时, 可以从 1->2->3

Trie图

如果有多个子串：

母串： abcde**k**g.....
子串1： **a**bcdegc
子串2： **a**bcdeuae
子串3： **c**den
子串4： **s**decse




如果都匹配不上，再从每个子串的开头开始匹配.相当于在trie图上回到root

在trie图上回到root就意味着现在还没有任何一个子串被匹配上任何一个字符（以前即便完整匹配了，也是以前的事情）

Trie图

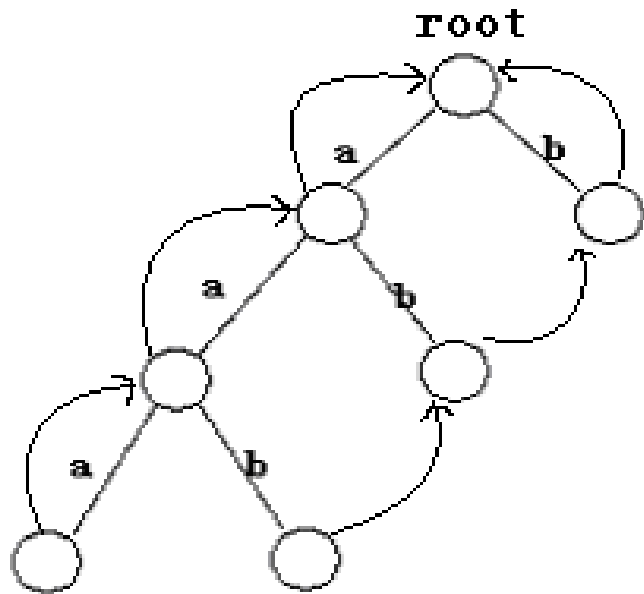
如果有多个子串：

母串： abcdekg.....
子串1： abcdegc
子串2： abcdeuae
子串3： cden
子串4： sdecse



前缀指针

- 对树上的每一个节点建立一个前缀指针
- 前缀指针的定义和KMP算法中的next数组相类似
- 若节点P对应的字符串为S，节点P的前缀指针须指向节点Q，Q对应的字符串为T，则
 - 1) T是S的一个后缀，
 - 2) T是某些模式串的前缀 (废话)
 - 3) T满足1), 2) 的最长的
 - 4) T不可等于S，可以为空串 (此时Q就是root)



在Trie树上添加前缀指针

用BFS按深度由浅到深求每一个节点的前缀指针

对于当前节点 P ，设其父节点与其连边上的字符为 Ch 。若父节点的前缀指针所指向的节点 Q 有儿子 R ，且边 (Q, R) 上的字母是 Ch ，则 P 的前缀指针指向 R 。若无这样的 R ，则通过 Q 的前缀指针继续向上查找，直到到达 $root$ 。

前缀指针总是指向层次更高（浅）的节点

在Trie树上添加前缀指针

接下来我们遍历树构造每个节点的前缀指针！

首先我们遍历根节点0

根节点0的所有连出的

边都指向1号节点

父亲是0号节点，连接字符

为A，查找父亲的前缀指针

1号节点，是否有通过A连接的

儿子。

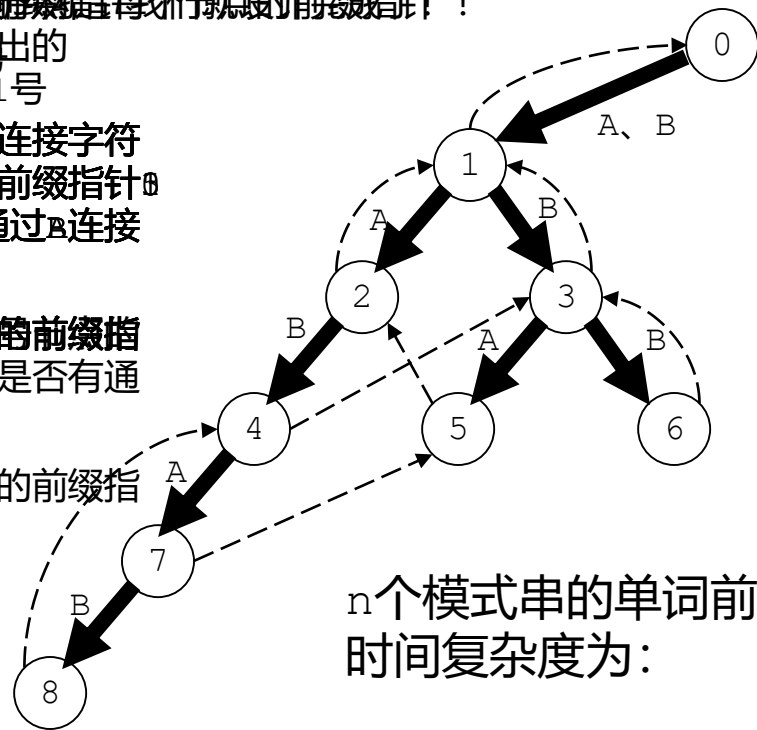
没有！于是继续查找父亲

的前缀指针0号节点是否有通

过A连接的儿子。

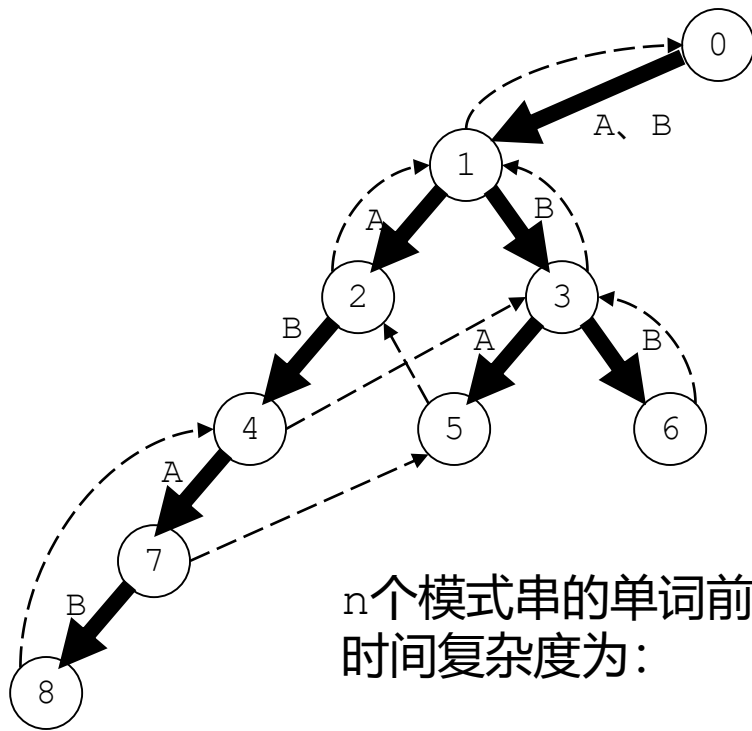
有！于是0号节点的前缀指

针指向1号节点



n个模式串的单词前缀树，添加前缀指针的时间复杂度为：

在Trie树上添加前缀指针



n 个模式串的单词前缀树，添加前缀指针的时间复杂度为：

$$O(\sum \text{len}(i)) \quad (i=1..n)$$

关于复杂度 $O(\sum \text{len}(i))$ ($i=1..n$) 的解释

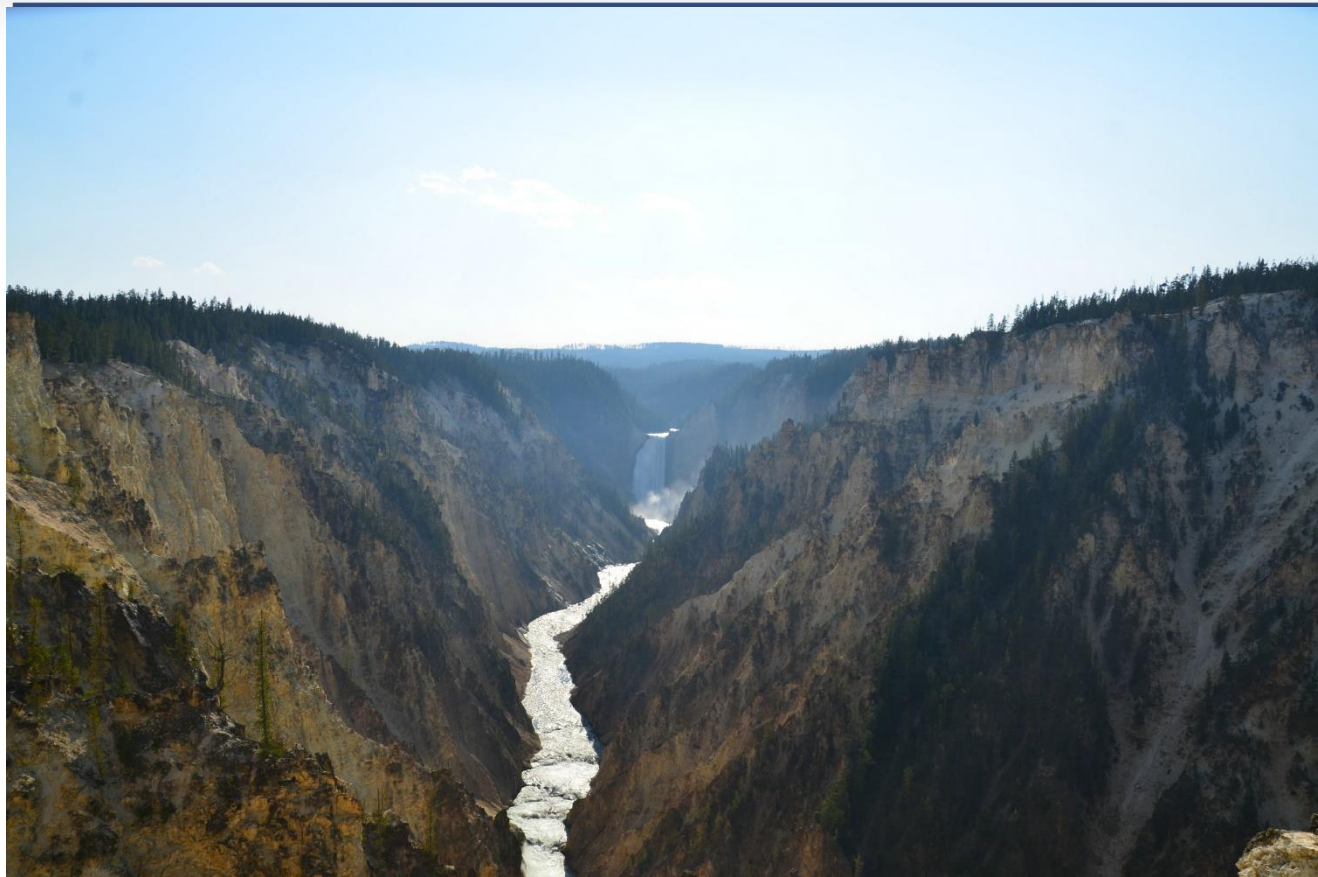
如果为节点 P 添加前缀指针时，沿前缀指针链往上走了很多层，则为 P 的儿子添加前缀指针时，就能迅速到达高层，不会再往上走很多



北京大学
PEKING UNIVERSITY

信息科学技术学院

在Trie图上做多字
字符串匹配



黄石大峡谷

在Trie图上遍历母串

对母串 S ，要将其在Trie图上遍历，看其是否包含某些Trie图中的模式串

危险节点的概念：

- 1) 终止节点是危险节点
- 2) 如果一个节点的前缀指针指向危险节点，那么它也是危险节点。

在Trie图上遍历母串

- 从ROOT出发，按照母串指针指向的字符ch在图上移动。
- 若当前点P不存在通过ch连接的儿子，则考查 P 的前缀指针指向的节点Q，若Q也没有通过ch连接的儿子，再考查Q的前缀指针指向的节点...直到找到通过ch连接的儿子（最终一定能找到，就是root），再母串指针前进，继续遍历。
- 若遍历过程中经过了某个终止节点，则说明s包含该终止节点代表的模式串。
- 若遍历过程中经过了某个非终止节点的危险节点，则可以断定s包含某个模式串。要找出是哪个，沿着危险节点的前缀指针链走，碰到终止节点即可。
- 遍历一个串s的时间复杂度是 $O(\text{len}(S))$

关于复杂度 $O(\text{len}(S))$ 的解释

- 1) 母串每过掉一个字符，不论该字符是匹配上了还是没匹配上，在trie图上最多往下走一层(层的概念来自原trie树)。
- 2) 一个节点的前缀指针总是指向层次更高的节点，所以每沿着前缀指针走一步，节点的层次就会向上一层
- 3) 母串 S 最终被过掉了 $\text{len}(S)$ 个字符，所以最多向下走了 $\text{len}(S)$ 次。

最多向下走 $\text{len}(S)$ 次，那么就不可能向上走超过 $\text{len}(S)$ 次，因此沿前缀指针走的次数，最多不会超过 $\text{len}(S)$



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题1 多模式字符串匹 配模板题



新疆库车大峡谷

多模式字符串匹配模板题

给N个模式串，以及M个句子，判断每个句子里是否包含模式串
句子和模式串都由小写字母组成。N,M ≤ 1000
每个模式串长度不超过20，每个句子长度不超过1000

样例输入

```
3
abc
def
gh
2
abcddddddddddd
akggggg
```

样例输出

```
YES
NO
```

```
#include <iostream>
#include <vector>
#include <queue>
#include <string>
using namespace std;
```

```
int n,m;
const int MAXN = 1010;
char str[MAXN];
const int LETTERS = 26;
int nNodesCount = 0;
struct CNode
{
    CNode * pChilids[LETTERS];
    CNode * pPrev; //前缀指针
    bool bBadNode; //是否是危险节点
    CNode() {
        memset(pChilids,0,sizeof(pChilids));
        bBadNode = false;
        pPrev = NULL;
    }
};
CNode Tree[20000]; //1000个模式串, 每个20个字符, 每个字符一个节点, 也只要
20000个节点
```

```
void Insert( CNode * pRoot, char * s)
{ //将模式串s插入trie树
    for( int i = 0; s[i]; i ++ ) {
        if( pRoot->pChildds[s[i]-'a'] == NULL) {
            pRoot->pChildds[s[i]-'a'] =
                Tree + nNodesCount;
            nNodesCount ++;
        }
        pRoot = pRoot->pChildds[s[i]-'a'];
    }
    pRoot-> bBadNode = true;
}
```

```
void BuildDfa( ) { //在trie树上加前缀指针
    for( int i = 0; i < LETTERS ; i ++ )
        Tree[0].pChilids[i] = Tree + 1;
    Tree[0].pPrev = NULL;
    Tree[1].pPrev = Tree;
    deque<CNode * > q;
    q.push_back(Tree+1);
```

```

while( ! q.empty() ){
    CNode * pRoot = q.front();
    q.pop_front();
    for( int i = 0; i < LETTERS ; i ++ ) {
        CNode * p = pRoot->pChildds[i];
        if( p) {
            CNode * pPrev = pRoot->pPrev;
            while( pPrev->pChildds[i] == NULL)
                pPrev = pPrev->pPrev;
            p->pPrev = pPrev->pChildds[i];
            if( p->pPrev->bBadNode)
                p->bBadNode = true;
            //前缀指针指向的节点是危险节点, 则自己也是危险节点
            q.push_back(p);
        }
    }
} //对应于while( ! q.empty() )
}

```

```
bool SearchDfa(char * s)
{ //返回值为true则说明包含模式串
    CNode * p = Tree + 1;
    for( int i = 0; s[i] ; ++i) {
        while( p->pChilds[s[i]-'a'] == NULL)
            p = p->pPrev;
        p = p->pChilds[s[i]-'a'];
        if( p->bBadNode)
            return true;
    }
    return false;
}
```

```
int main()
{
    nNodesCount = 2;
    scanf("%d", &n);
    for( int i = 0; i < n; ++i ) {
        scanf("%s", str);
        Insert(Tree + 1, str);
    }
    BuildDfa();
    scanf("%d", &m);
    for( int i = 0 ; i < m; i ++ ) {
        scanf("%s", str);
        if( SearchDfa(str) )
            printf("YES\n");
        else
            printf("NO\n");
    }
    return 0;
}
```



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题2 躲不开的病毒



美国太浩湖

例题2: 躲不开的病毒

有若干种病毒，每种病毒的特征代码都是一个01串。
每个程序也都是一个01串。问是否存在不被病毒感染（不包含任何病毒的特征代码）的无限长的程序。

所有的病毒的特征代码总长度不超过30000

例题2: 躲不开的病毒

如果存在无限长的安全母串，在Trie图上，从根节点出发遍历该母串，则可以永远不停地走下去，即走无限步，都不会路过危险节点。

例题2: 躲不开的病毒

如果存在无限长的安全母串，在Trie图上，从根节点出发遍历该母串，则可以永远不停地走下去，即走无限步，都不会路过危险节点。

因此，问题等价于能否在Trie图上走无穷多步而不经危险节点。

例题2: 躲不开的病毒

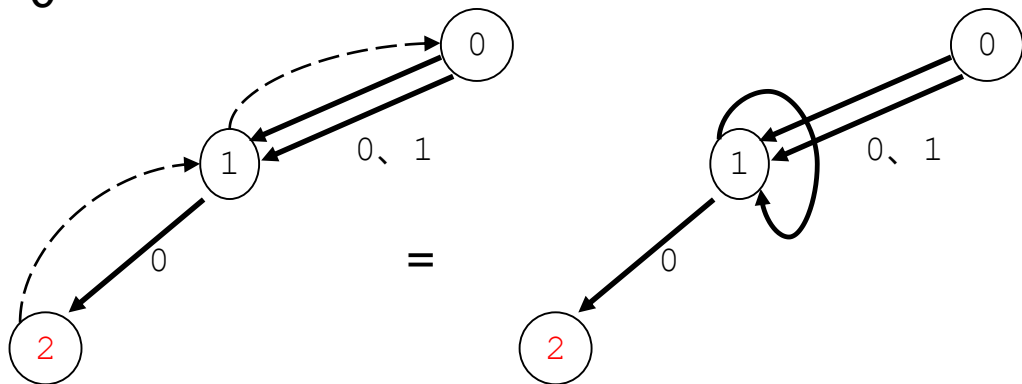
用 DFS 从root出发在Trie图上寻找环:

- 1) 只有字母边才算是有向图的边
- 2) 不能走危险节点
- 3) 从一个节点P出发进行DFS时, 考查所有字母。对于字母ch,若字母边存在, 则沿着该字母边往前走, 若字母边不存在, 则**必须**顺着前缀指针链进行跳跃, 最终沿一条字母边ch到达节点Q ---- **等价于P有字母边ch连到Q。**也可以改图, 直接加上这些字母边, 然后再从root做 DFS。在修改过的图上再次DFS时, 须忽略所有前缀指针。
- 4) 如果一个节点的所有字母边都不为NULL, 则该节点的前缀指针视为不存在, 不能通过它跳跃
- 5) 若能走回到DFS当前路径上的节点 (栈中的节点), 则发现环

例题2: 躲不开的病毒

1个病毒

0

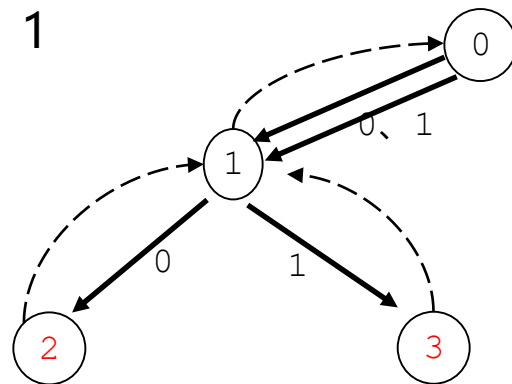


YES

2个病毒：

0

1



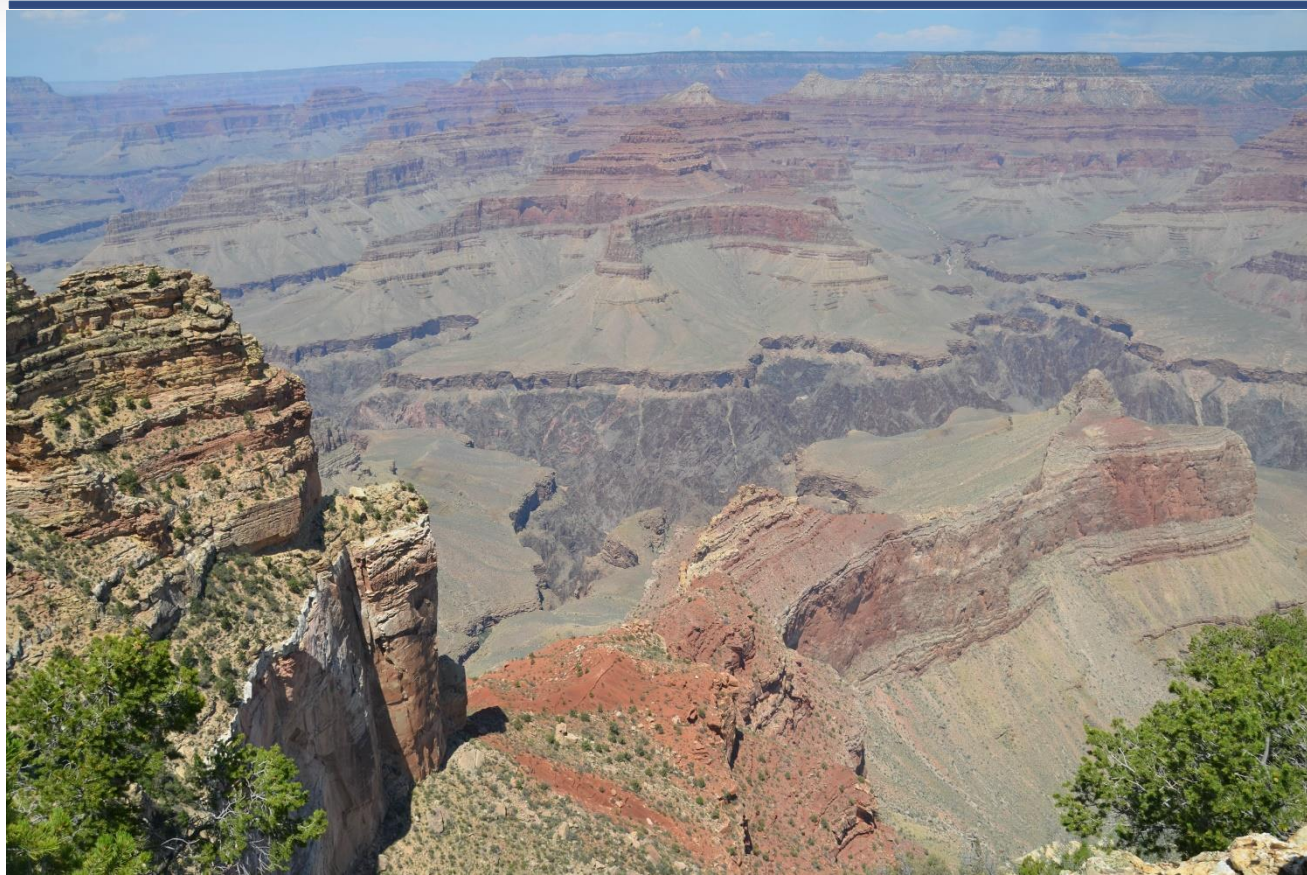
NO



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题3 DNA Repair



美国科罗拉多大峡谷

例题3: POJ 3691 DNA repair

给定由 'A', 'G', 'C', 'T' 四个字母组成的不超过50个模式串, 每个模式串长度不超过20, 再给一个不超过1000个字符的同样由上述字母组成的母串S, 问在S中至少要修改多少个字符, 才能使其不包含任何模式串。

例题3: POJ 3691 DNA repair

Sample Input

```
2
AAA
AAG
AAAG
2
A
TG
TGAATG
4
A
G
C
T
AGT
0
```

Sample Output

```
Case 1: 1
Case 2: 4
Case 3: -1
```


例题3: POJ 3691 DNA repair

$dp[i][j]$ 表示若要用长度为 i 的母串的前缀遍历Trie图, 使之达到节点 j , 至少要修改多少个字符。 j 是安全节点。

初始情况:

$dp[0][1] = 0$ //1是Trie图的 root (0号节点不是root)

其他所有 $dp[i][j]$ 为无穷大

例题3: POJ 3691 DNA repair

问题的答案就是

$\min \{ dp[len][j] \mid j \text{ 是安全节点} \}$

len是母串的长度

例题3: POJ 3691 DNA repair

递推公式:

由 $dp[i][j]$ 可以推出:

$$dp[i+1][son(j)] = \min \{ dp[i+1][son(j)], dp[i][j] + (Char(j,son(j)) \neq str[i]) \}$$

($Char(j,son(j)) \neq str[i]$) 表达式值为0或1

$Char(j,son(j))$ 表示从节点 j 走到节点 $son(j)$ 所经过的字母

str 是母串,下标从0开始算

“我为人人”方式的动规

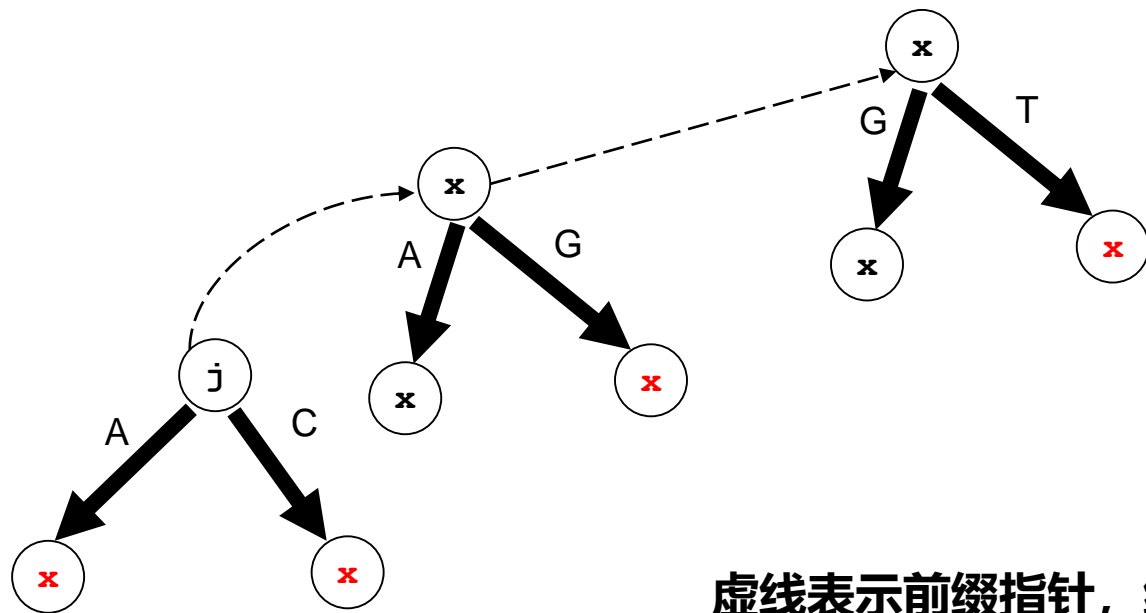
例题3: POJ 3691 DNA repair

$\text{Char}(j, \text{son}(j))$ 表示从 j 到 $\text{son}(j)$ 经过的字母, 假设是 a 。

这里所说的“经过”，不仅指从 j 通过一条字母 a 边直接到达 $\text{son}(j)$, 也可以是通过若干前缀指针后再通过一个字母 a 边到达 $\text{son}(j)$

所以 $\text{son}(j)$ 并不只是 j 的子节点。

例题3: POJ 3691 DNA repair



虚线表示前缀指针，红色节点都是 son(j)
前提是它们都安全



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题4 Censored!



美国科罗拉多大峡谷

例题4: POJ1625 Censored!

给出一个字符集V和P个模式串(长度小于10), 问由这个字符集中字符组成的长度为N的且不包含任意一个模式串的字符串有多少个? (字符集大小, $N \leq 50$, $P \leq 10$)

样例输入:

2 3 1 //N=3 P=1

ab //字符集为ab

bb

样例输出:

5

样例解释:

aaa

aab

aba

baa

bab

以上5个

例题4: POJ1625 Censored!

$dp[i][j]$ 表示长度为 i 且能走到节点 j 的字符串个数 (节点 j 是安全节点)

初始 $dp[0][1] = 1$, 其他 $dp[i][j] = 0$

对每个由 j 一步可以到达的安全节点 $son(j)$, 都应执行:

$$dp[i+1][son(j)] += dp[i][j]$$

因为从根走 i 步到达节点 j 有 n 种走法, 那么走 $i+1$ 步到达 $son(j)$ 的走法就要加 n (j 以外节点也可能通过一步走到 $son(j)$)

整个问题最终的答案就是:

$$\sum \{ dp[N][j] \mid j \text{ 是安全节点} \}$$

此题需要高精度计算。类似题: [POJ2778 DNA Sequence](#)