AC自动机

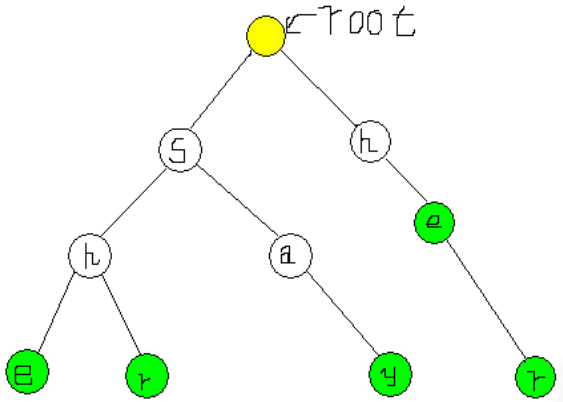
主讲人：邓哲也

Trie

将若干个字符串插入 Trie 树。

从根节点走到任意一个叶子的路径都对应一个字符串。

下图就是插入 {her, say, she, shr} 的 Trie 树。



KMP

KMP 是当用一个串去匹配另一个串的算法。

处理出的 next 数组，可以使得匹配失败的时候，跳过冗余的匹配，减少多余的回溯。

bacbababaabcbab

　　ababaca

bacbababaabcbab

　　ababaca

AC 自动机

当有多个模式串用来匹配时，我们可以把这些模式串都插入 Trie，然后在 Trie 上做匹配。

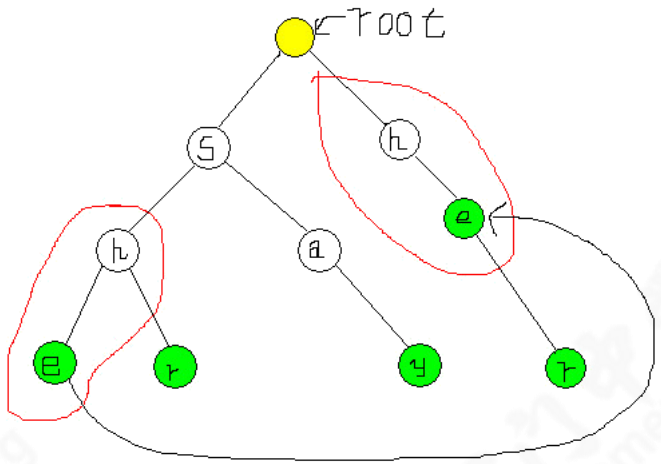
每次枚举文本串的第 i 位，从根开始走，如果有路就走下去，没路就说明不能匹配上。然后从第 i + 1位继续。

这样的算法也会有很多冗余的匹配过程，我们可以引入 KMP 中的 next 数组，也就是对于 Trie 中的每个节点，找出如果在这个点匹配失败的话，应该跳到的下一个进行匹配的节点，减少冗余匹配。

AC 自动机

比如这颗 Trie 中，she 中的 e 的 next 指针应该指向 her 中的 e。

假设文本串是 sher，在匹配到 she 之后，可以跳到 next 指针，继续匹配到 her。过程中没有任何冗余匹配。

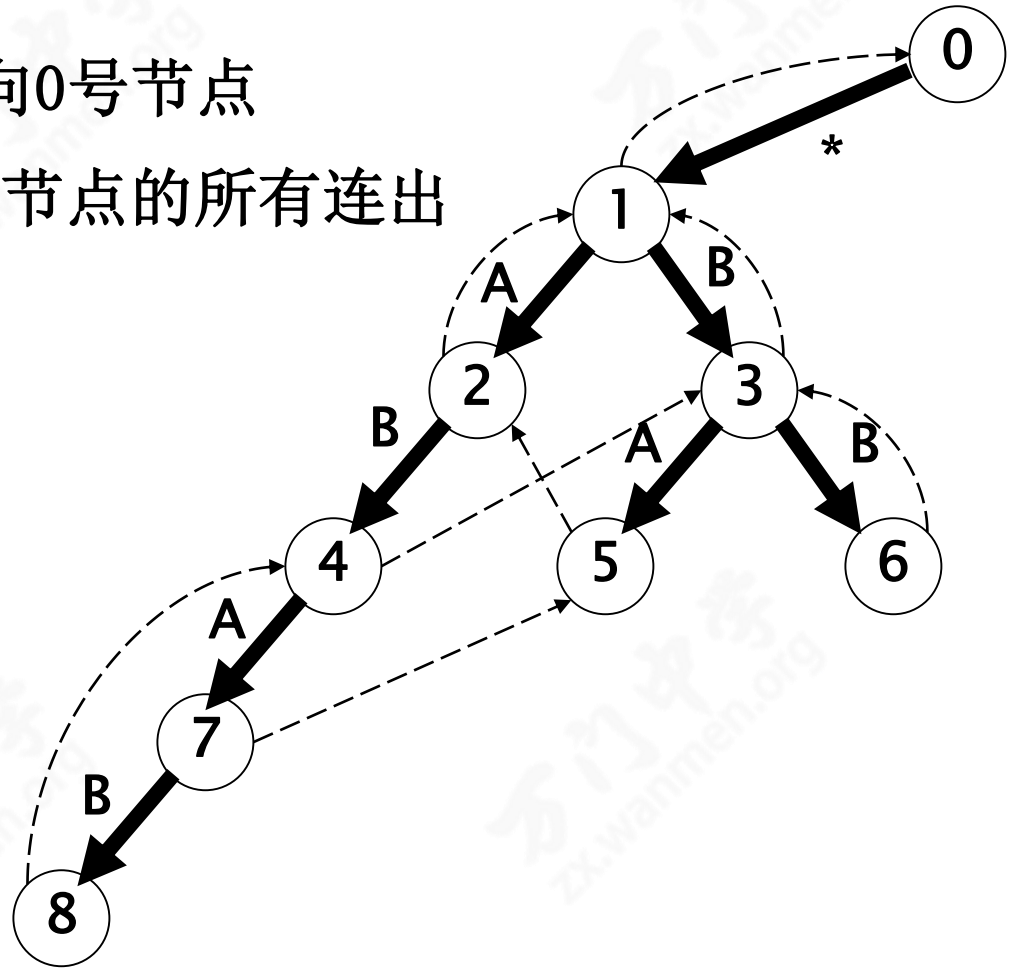


AC 自动机

接下来按照BFS顺序构造每个节点的前缀指针

ROOT1号节点的前缀指针指向0号节点

定义虚拟节点0号节点，0号节点的所有连出的字边都连向ROOT1号节点

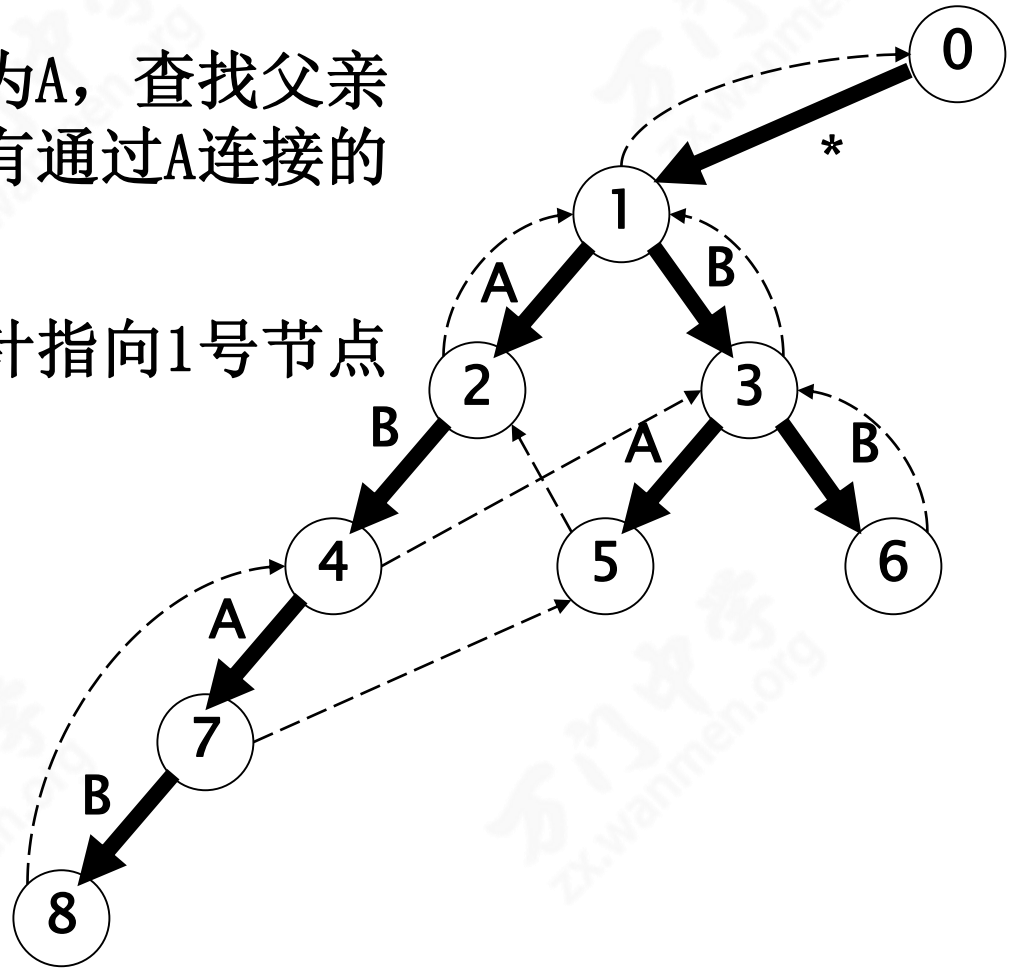


AC 自动机

2号节点：

父亲是1号节点，连接字符为A，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2号节点的前缀指针指向1号节点

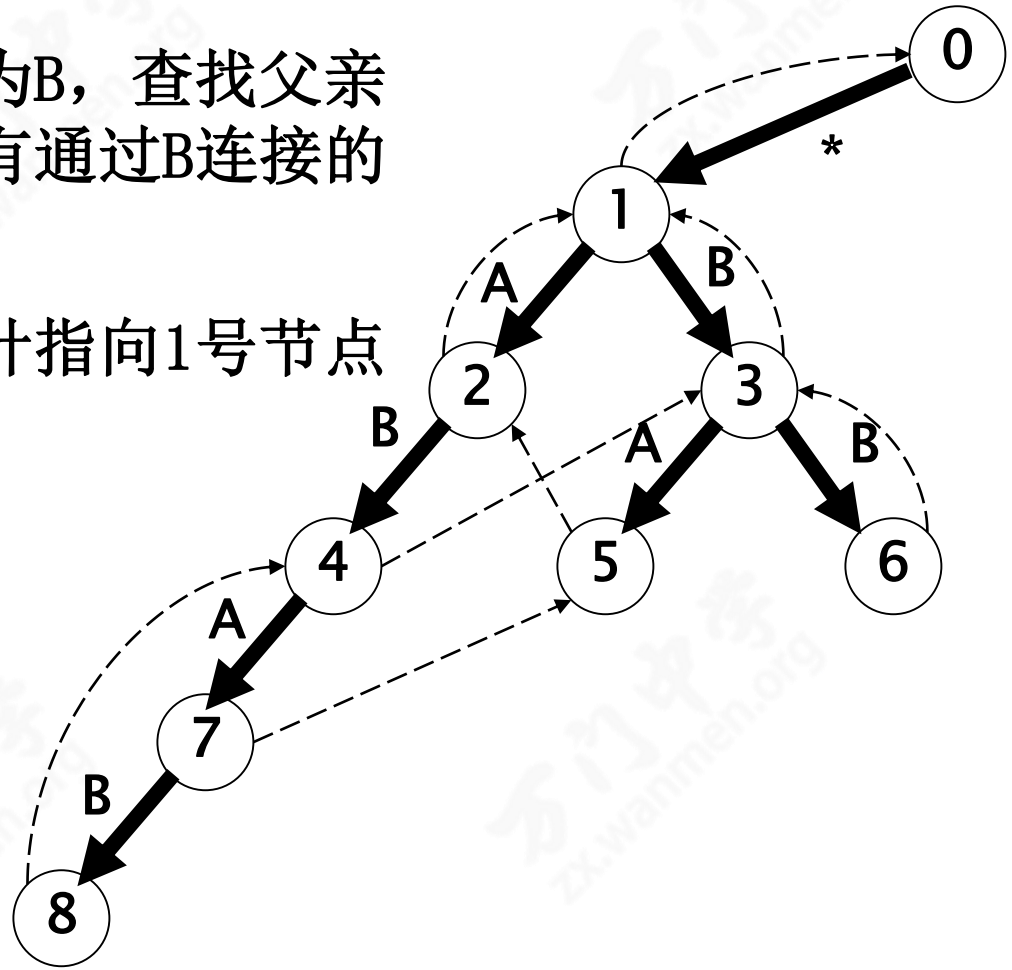


AC 自动机

3号节点：

父亲是1号节点，连接字符为B，查找父亲的前缀指针0号节点，是否有通过B连接的儿子。

有！于是3号节点的前缀指针指向1号节点

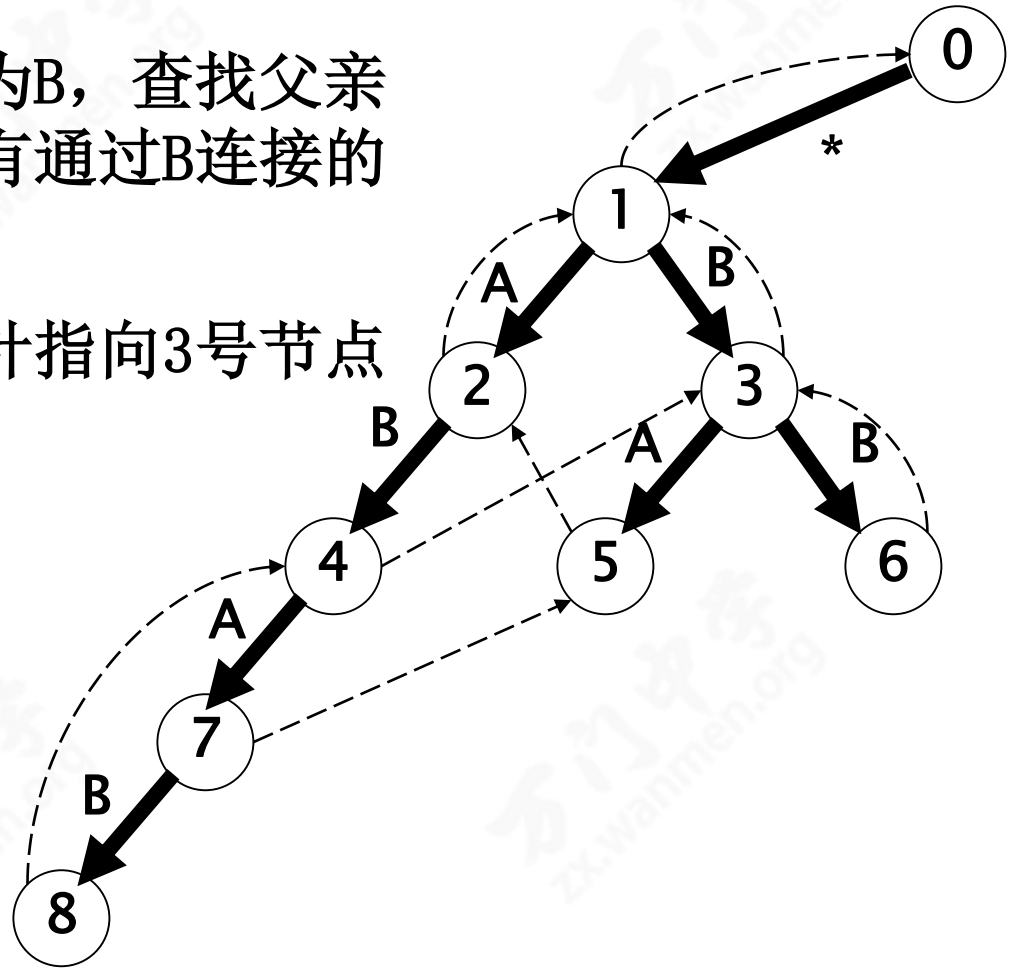


AC 自动机

4号节点：

父亲是2号节点，连接字符为B，查找父亲的前缀指针1号节点，是否有通过B连接的儿子。

有！于是4号节点的前缀指针指向3号节点

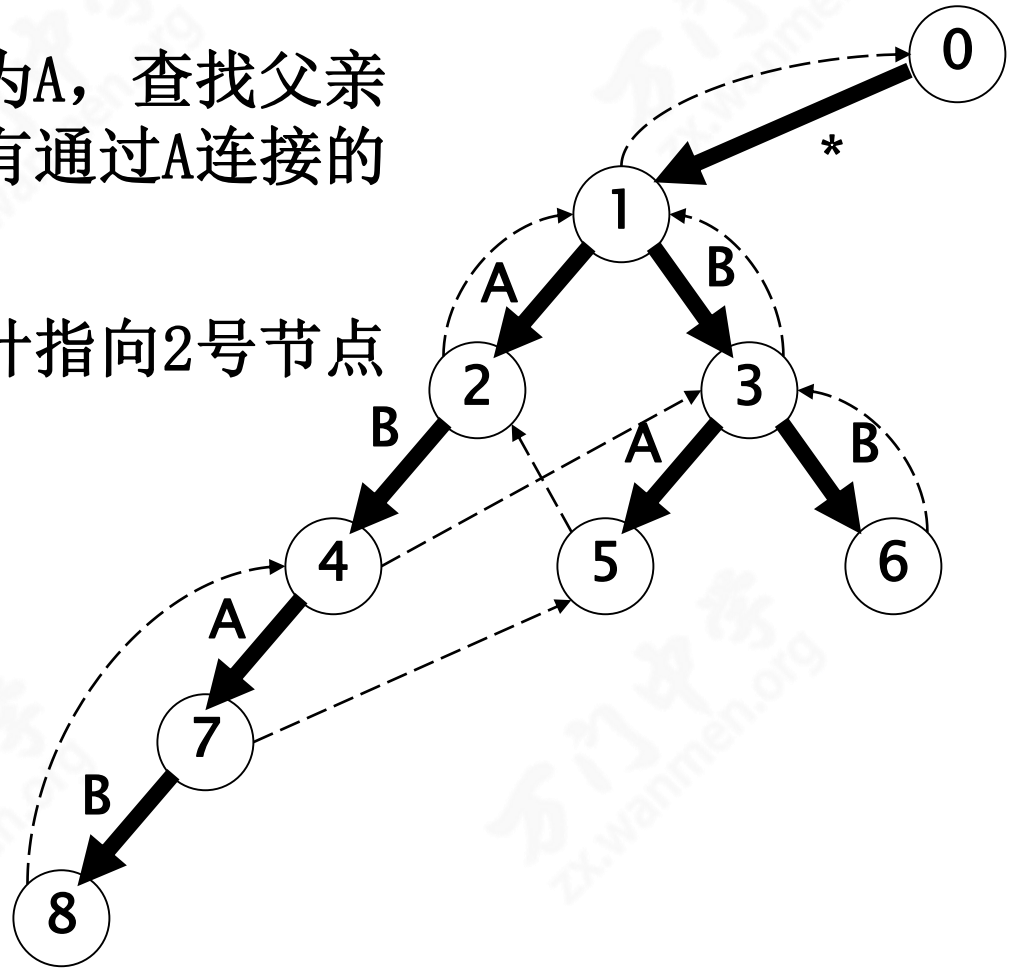


AC 自动机

5号节点：

父亲是3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。

有！于是5号节点的前缀指针指向2号节点

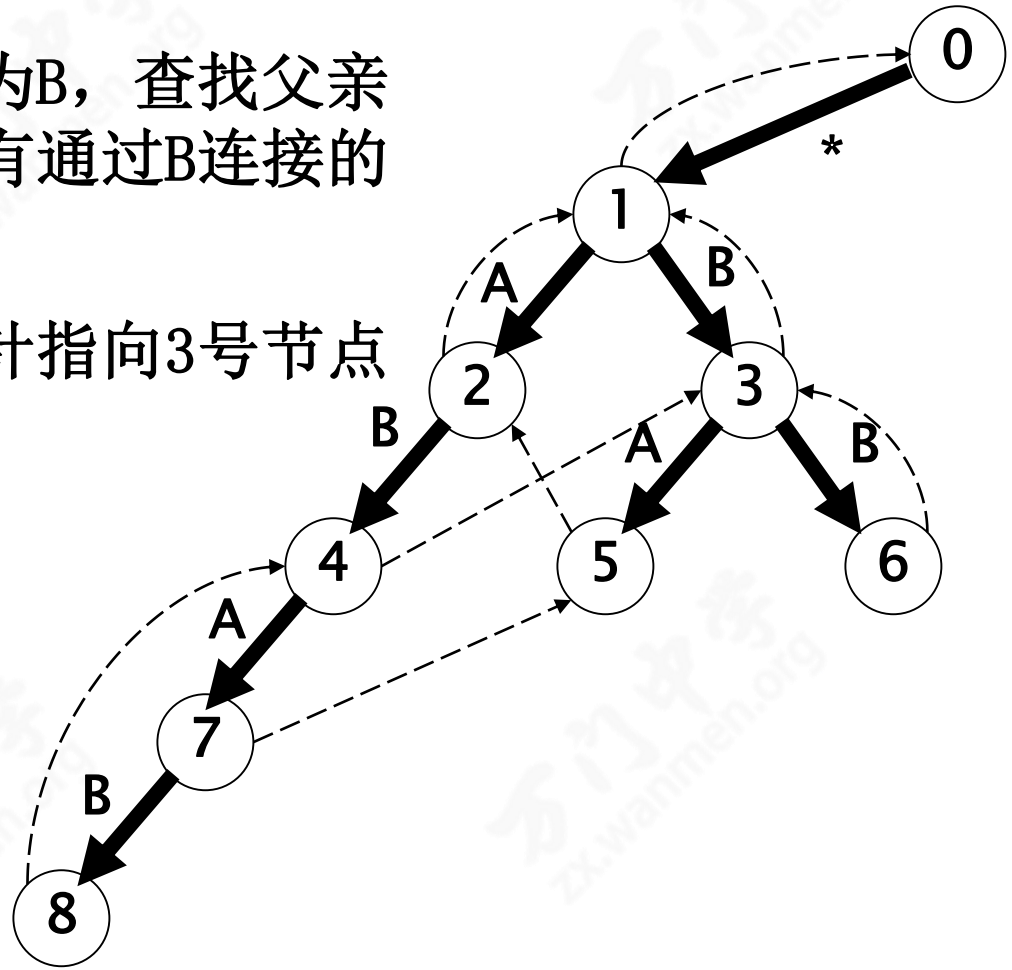


AC 自动机

6号节点：

父亲是3号节点，连接字符为B，查找父亲的前缀指针1号节点，是否有通过B连接的儿子。

有！于是6号节点的前缀指针指向3号节点

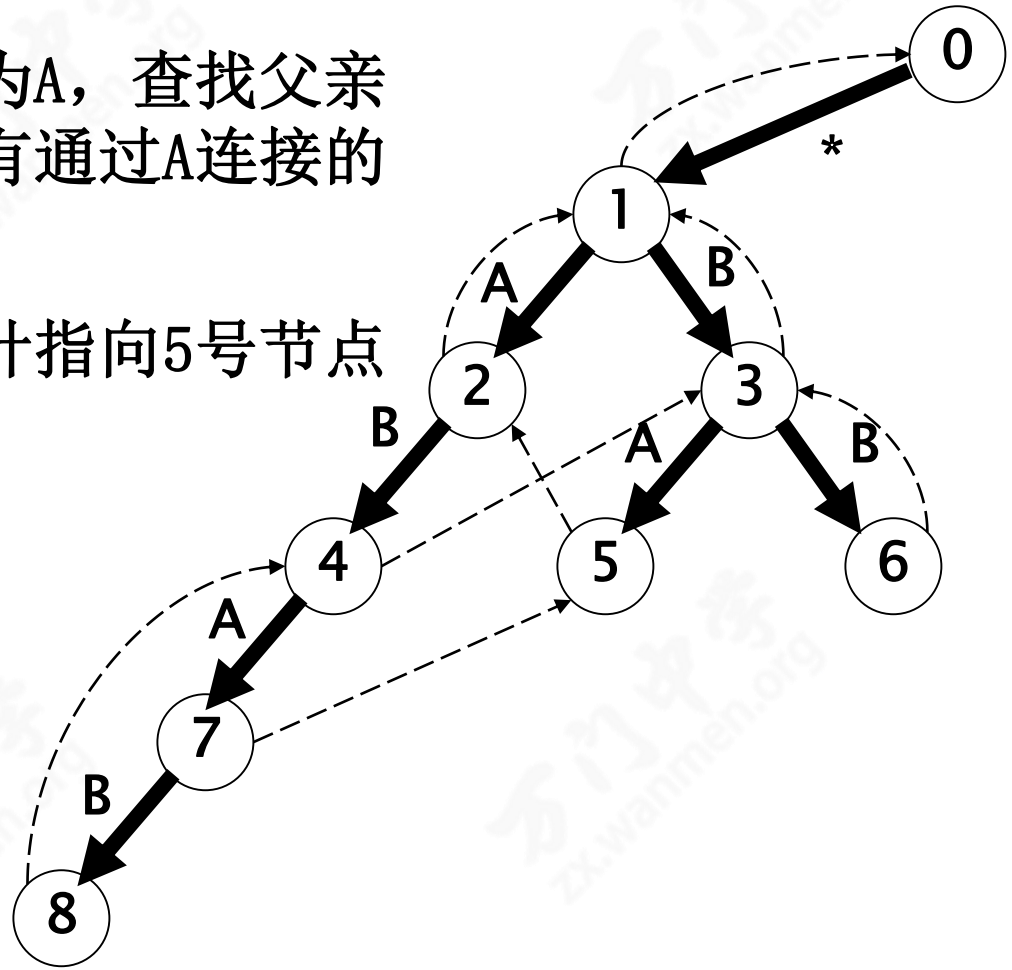


AC 自动机

7号节点：

父亲是4号节点，连接字符为A，查找父亲的前缀指针3号节点，是否有通过A连接的儿子。

有！于是7号节点的前缀指针指向5号节点



AC 自动机

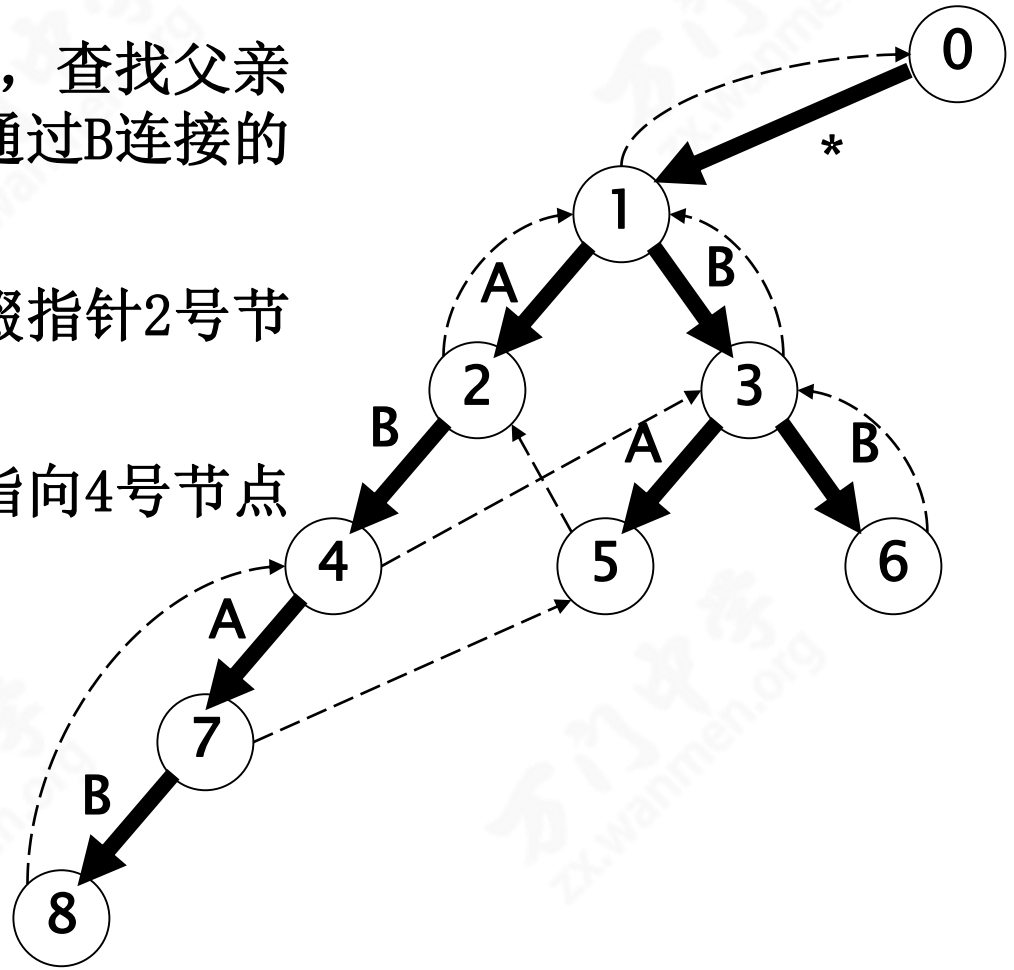
8号节点：

父亲是7号节点，连接字符为B，查找父亲的前缀指针5号节点，是否有通过B连接的儿子。

没有，继续查找5号节点的前缀指针2号节点是否有通过B连接的儿子。

有！于是8号节点的前缀指针指向4号节点

至此，这棵树的前缀指针我们就设计完成了！！



AC 自动机代码实现

插入 Trie 树：

```
for (int i = 1;i <= n;i ++){
        scanf(“%s”, s);
        int x = 0;
        for (int j = 0; s[j]; j ++){
             if (!ch[x][s[j] - ‘A’ ]) ch[x][s[j] - ‘A’ ]
= tot ++;
              x = ch[x][s[j] - ‘A’ ];
        }
        cnt[x] += 1;
}
```

AC 自动机代码实现

BFS 遍历求 next 指针：

```
int s = 0, e = 1;
q[0] = next[0] = 0;
while(s < e){
        int x = q[s ++];
        for (int j = 0;j < 26;j ++)
                if (ch[x][j]){
                        q[e ++] = ch[x][j];
                        next[ch[x][j]] = (!x) ? 0 :
ch[next[x]][j];
                } else ch[x][j] = ch[next[x]][j];
}
```

AC 自动机代码实现

查询一个串s里出现了几次模板串。

```
int x = 0, res = 0;
for (int i = 0; s[i]; i ++)
    res += cnt[x], x = ch[x][s[i] - ‘A’ ];
```

因为 ch[x] 已经把所有的26种可能的转移都处理好了。

HDU 2896 病毒侵袭

有 n 个病毒，每个病毒是一个字符串。

有 m 个网站，每个网站是一个字符串。

输出对于每个网站出现了那几种病毒以及有几个网站含有病毒。

字符集是所有可见字符（ascii 码 < 128）

n ≤ 500, m ≤ 1000

Sample Input

```
3
aaa
bbb
ccc
2
aaabbbccc
bbaacc
```

Sample Output

```
web 1: 1 2 3
total: 1
```

HDU 2896 病毒侵袭

把 n 个病毒字符串都插入 Trie，建成 AC 自动机，每个串终止的位置打上标记。

然后对于 m 个网站，用这 m 个字符串在 AC 自动机上遍历。

如果碰到了有标记的点，说明含有这个病毒。

下节课再见