

# 8.6应用AC自动机进行多 模式匹配的实验范例

吴永辉

ICPC Asia Programming Contest 1st Training Committee – Chair

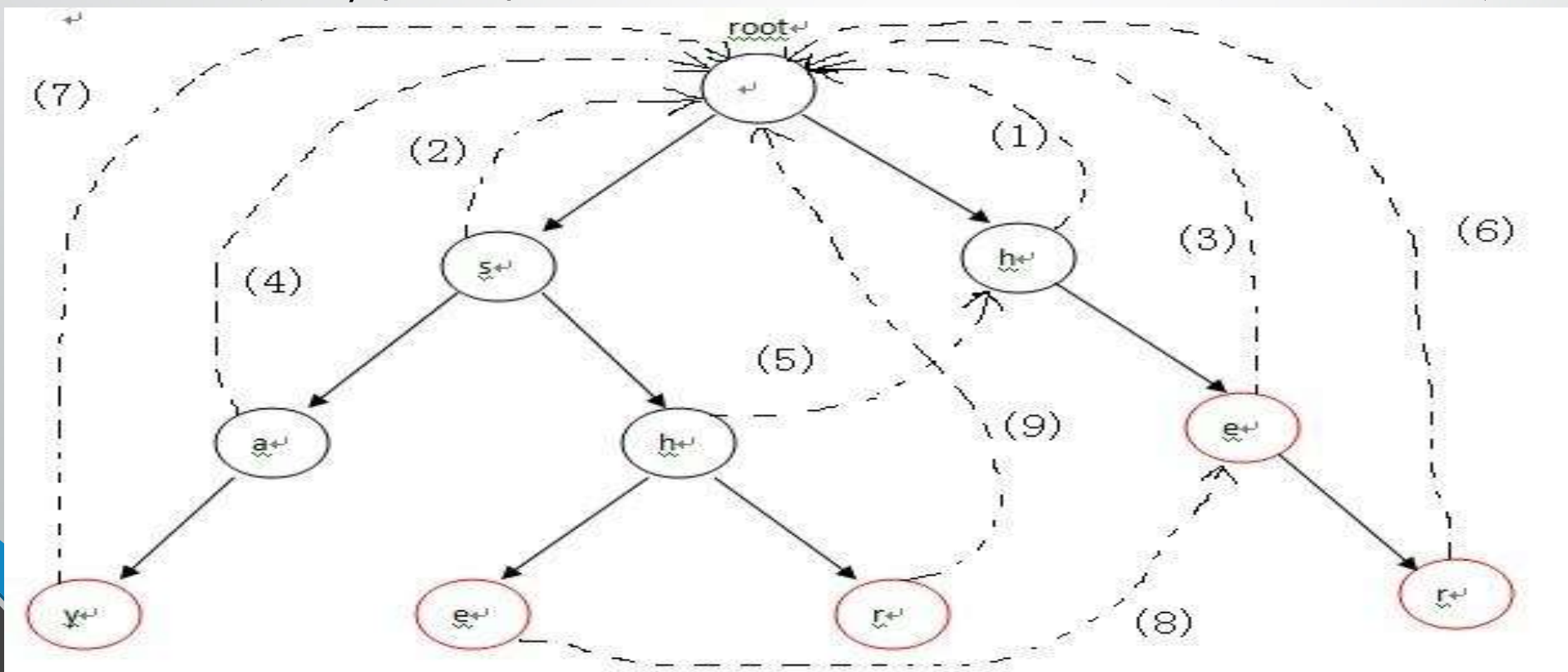
[yhwu@fudan.edu.cn](mailto:yhwu@fudan.edu.cn)

- AC自动机 (Aho-Corasick automaton) 是一种多模式匹配算法, 给出一个目标 $T$ 和多个模式 $P_1, P_2, \dots, P_n$ , 问你有多少个模式在 $T$ 中出现过, 并给出在 $T$ 中匹配的位置。
- 如果对每个模式 $P_i$  ( $1 \leq i \leq n$ ) 和 $T$ , 采用KMP算法, 时间复杂度会比较高, 当模式的个数比较多并且目标很长的情况下, 就不能有效地解决模式匹配的问题。如果用AC自动机算法来解决多模式匹配, 时间复杂度就可以优化到 $O(n)$ , 其中 $n$ 是目标的长度。

# AC自动机算法是建立在Trie树和KMP算法的基础之上，算法步骤如下：

- 步骤1：构造一棵Trie树，作为AC自动机算法的数据结构。构造过程是将多个模式插入Trie树。
- 这棵Trie树不仅有此前介绍的Trie树的性质，而且节点增加一个*fail*指针，如果当前点匹配失败，则将指向当前匹配的字符的指针转移到*fail*指针指向的地方，使得当前匹配的模式串的后缀和*fail*指针指向的模式串的前缀相同，这样就可以继续匹配下去了。例如，有模式"abce"和"bcd"，在目标*T*中有子串"abc"，但下一个字符不是'e'，则由*fail*指针跳到"bcd"中的'c'处，然后看*T*的下一个字符是不是'd'。
- Trie树的节点的结构体类型定义如下：
  - struct node{
  - node \*next[26]; //后继指针，其中*next[i]*为序数值为*i*的子节点
  - node \*fail;    //匹配失败后，当前字符应与*fail*指针指向的字符匹配
  - int sum;       //匹配完成标志（-1）以及匹配单词数
  - };

- 步骤2：通过BFS构造 $fail$ 指针。
- 以字符串"say", "she", "shr"和"her"构造的Trie树为例说明，如图8.6-1所示。



- 首先初始化，Trie树的 $root$ 入队；然后 $root$ 出队；因为 $root$ 是Trie树的入口，不包含字符，所以 $root$ 的孩子的 $fail$ 指针都指向 $root$ 。 $root$ 的孩子入队。即包含字符'h'和's'的节点的 $fail$ 指针都指向 $root$ ，如图8.6-1中的虚线(1)(2)所示；同时这两个包含'h'和's'的节点入队。
- 接下来，包含字符'h'的节点出队，构造该节点的孩子节点（即包含字符'e'的节点）的 $fail$ 指针如下：字符'h'对应的节点的 $fail$ 指针指向 $root$ ，而且 $root$ 没有包含字符'e'的孩子，所以包含字符'e'的节点的 $fail$ 指针指向 $root$ ，如图8.6-1中的虚线(3)所示；并且包含'e'的节点入队。

- 然后，包含字符's'的节点出队，同样，构造该节点的两个孩子（即包含字符'a'和'h'的节点）的*fail*指针，同理，因为字符's'的节点的*fail*指针指向*root*，而且*root*没有包含字符'a'的孩子，所以包含字符'a'的节点的*fail*指针指向*root*，如图8.6-1中的虚线(4)所示；包含'a'的节点入队；而对于包含字符'h'的节点，因为*root*包含字符'h'的孩子，所以该节点的*fail*指针指向Trie树第二层的包含字符'h'的节点，如图8.6-1中的虚线(5)所示；同时该节点入队。



- 此时，队列中有包含'e'、'a'和'h'的3个节点。包含字符'e'的节点先出队，对于该节点的孩子（包含字符'r'）的*fail*指针，因为字符'e'的节点的*fail*指针指向*root*，而且*root*没有包含字符'r'的孩子，所以包含字符'r'的节点的*fail*指针也指向了*root*，如图8.6-1中的虚线(6)所示；该节点进队。然后包含字符'a'的节点出队，同样地，字符'a'的节点的*fail*指针指向了*root*，而且*root*没有包含字符'y'的孩子，所以字符'a'的节点的孩子节点（包含字符'y'）的*fail*指针指向*root*，如图8.6-1中的虚线(7)所示；并且该节点进队。然后包含字符'h'的节点出队，该节点的*fail*指针指向Trie树第二层的包含字符'h'的节点，而这个被指向的节点又有包含字符'e'的孩子节点；所以，该节点包含字符'e'的孩子节点的*fail*指针指向在Trie树中第三层的那个包含字符'e'的节点，如图8.6-1中的虚线(8)所示；并且该节点进队。对于另外一个包含字符'r'的节点，由于那个在第二层的包含字符'h'的节点没有包含'r'的孩子节点，则沿*fail*指针继续找，则指向了*root*，而且*root*没有包含字符'r'的孩子，所以，最后的包含'r'的节点的*fail*指针指向了*root*，如图8.6-1中的虚线(9)所示。


- 
- 基于BFS、队列构造 $fail$ 指针的过程（见讲义）



- 步骤3：扫描目标进行匹配。
- 构造好Trie树和 $fail$ 指针后，就可以对目标进行扫描，这个过程和KMP算法很类似，但是也有一定的区别，主要是因为AC自动机处理的是多模式匹配。为了避免遗漏匹配，引入 $temp$ 指针。

- AC自动机多模式匹配过程分两种情况：
  - (1) 当前的模式和目标字符匹配，表示沿着Trie树的边有一条路径可以到达当前匹配的字符（节点），则从该节点出发，沿Trie树的边走向下一个节点，目标字符串指针移向下一个字符，继续匹配；
  - (2) 当前字符不匹配，则模式指针转移到当前节点的父节点的*fail*指针所指向的节点，目标指针前移一位，新的模式和目标继续匹配；匹配过程随着指针指向*root*结束。
- 重复上述两个过程，直到目标串指针指向结尾为止。

- 例如，Trie树如图8.6-1，模式为"say", "she", "shr", "he"和"her"。目标为字符串"yasherhs"，AC自动机多模式匹配过程如下：
- 设 $p$ 为Trie树的指针， $i$ 为字符串"yasherhs"的指针。
- 当 $i=0, 1$ ，目标串在Trie树中没有对应的路径，故不做任何操作； $i=2, 3, 4$ 时，指针 $p$ 指向Trie树最下层的包含'e'的节点。因为该包含'e'的节点的count值为1，所以 $cnt++$ ，并且将该节点的count值设置为-1，表示改单词已经出现过了，防止重复计数；然后，为避免遗漏， $temp$ 指向包含'e'的节点的fail指针所指向的节点继续查找，并以此类推，直到 $temp$ 指向root，而在这个过程中 $cnt$ 增加了2，表示找到了2个单词"she"和"he"。当 $i=5$ 时， $p$ 则指向包含'e'的节点的fail指针所指向的节点，也就是Trie树中第二层右边那个包含'e'的节点，随后， $p$ 指向该节点的包含'r'的孩子节点，由于该节点的count值为1，从而 $cnt++$ ；然后，循环直到 $temp$ 指向root为止。最后 $i=6, 7$ 时，找不到任何匹配，匹配过程结束。

- 
- 利用fail指针进行多模式匹配的过程（见课件）

## 8.6.1 Keywords Search

- 在线测试: HDOJ 2222

- 现在，搜索引擎已经走进了每个人的生活，比如，大家使用Google、百度，等等。
- Wiskey希望将搜索引擎引入到他的图像检索系统中。
- 每个图像都有一个很长的文字描述，当用户键入一些关键字来查找图像时，系统会将关键字与图像的文字描述进行匹配，并显示出匹配关键字最多的图像。
- 本题要求，给出一个图像的文字描述和一些关键字，请您计算有多少个关键字匹配。



## 试题解析

- 本题是AC自动机入门题和模板题。本题给出 $N$ 个模式串（长度不超过50）和一个目标串（长度不超过1000000），求出有多少个模式串在这个文本串中出现过。
- 按AC自动机算法，首先将 $N$ 个模式串插入Trie树；然后采用BFS算法设置 $fail$ 指针；最后扫描目标串，进行多模式匹配。

- AC自动机首先是一个多模式匹配算法，简单来说就是有多个模式串，模式串之间可以互相重叠，要求查询的主串中有多少个模式串。实际上，AC自动机就是在一棵Trie树上添加了一个fail指针，这个指针和KMP中的Next数组的作用是一样的：代表着失配后应该转移的位置。如果fail指针指向了root，那么说明在trie树中的前缀没有出现在主串的后缀中了。

- 如果我们把trie树上的节点之间的连接和fail指针当成边，那么AC自动机所创建的实际上是一张状态图。我们可以在AC自动机上进行DP，即状态可以在这张图上进行转移。
- 一般来说，在AC自动机上DP需要进行状态压缩。在设计状态转移时，主要考虑的是AC自动机上的连接状态，即当前状态的下一个可能转移到的状态一定是在AC自动机上进行的。在这一指导思想下，按照题意和方便计算的要求设计DP状态。例如，给每个节点创建一个状态矩阵，代表有多少个节点能到达目标。这种在AC自动机上进行DP的方法，在求诸如“不包含某些串的串有多少个”的问题上，相对比较方便了。

