|  |
| --- |
| **多串匹配-AC自动机** |
|  |
|  |
| |  |  |  | | --- | --- | --- | | AC自动机即 Aho-Corasick automation，该算法在1975年产生于贝尔实验室。AC自动机是用来处理多串匹配问题的，即给你很多串，再给你一篇文章，让你在文章中找这些串是否出现过，在哪出现。  AC自动机思想简单来讲就是在 Trie 上进行 KMP 匹配，所以先要知道 [Trie数据结构](http://www.cnblogs.com/dngc/archive/2010/03/17/1688045.html) 和 [KMP算法](http://www.cnblogs.com/dngc/archive/2010/03/11/1683688.html)。    AC自动机先将所有模式串构建成单词树，如有模式串 { she, he, say, shr, her, ayd }，我们先构建成如下单词树：  http://blogimg.chinaunix.net/blog/upfile2/100412150449.jpg  假设我们现在要对串 yshersayd 进行匹配，找出该串的所有模式串。  一般的做法就是从一个指针 i 指向串的开始匹配位置，  首先 i== 0 这是用串 [i, len(s)] 即 'yshersayd' 进行匹配，没有匹配，i++;  这时用串 [1,len(s)]即 'shersayd' 在单词树中匹配，得到得到匹配 'she'，再 i++;  这时用串 [2,len(s)]即 'hersayd' 在单词树中匹配，得到匹配 'he' 和 'her' 再 i++;  依次进行，易知算法最坏复杂度为 O(nm)  n为主串的长度，m 为模式串平均长度。    实际上我们可以通过构造 失败指针 来优化匹配，从而使算法复杂度达到 O(n)。失败指针类似 KMP 算法的 next[] 数组值，KMP 算法中，设 next[i]= k，则 k 为满足 S[0,k-1]== S[i- k, i-1]最大的值，KMP 算法中求 next[] 只有一个串。而失败指针是在所有模式串或其前缀中找一个最大的那个 K，即对于串 S1，我们在其它模式串或其前缀中找一个串 S2，使得  S1[len(S1)- k, len(S1)]= S2[0, k] 其中 k 最大，则 S1[ len(S1) ] 的失败指针为 S2[ len(s2) ]。也可理解为当我们匹配失配时，利用已经匹配的结果，尽可能的将指针 i 往后移。如图，当我们用 'shersayd' 匹配时，匹配到 'e' 时以后的字符失配，这时我们不是用 'hersayd' 继续从头开始匹配。利用匹配的结果，我们可以只用 'sayd' 在红圈的另外那个 'e' 开始匹配。失败指针就是在匹配失败时转移，使得能够继续匹配。  如上图：我们构建失败指针后图变为  http://blogimg.chinaunix.net/blog/upfile2/100412150517.jpg  上图中，粗红线表示失败指针，没标明失败指针的结点的失败指针都指向根结点。构建了失败指针后，匹配是如果不能匹配就从失败指针走，再匹配。如我们匹配刚才那个字符串  ’yshersayd'，首先是字母 'y' ，没有匹配，走向失败指针根结点，然后字母 's'，匹配，走向 's'。然后字母 'h'，匹配，走向 'h'。然后字母 'e' 走向 'e'，得到模式串 'she'。然后 'r' ，这时 'r' 失配，我们走向 'e' 的失配指针，粗红线指向的另一个 'e'，继续匹配，得到模式串 'her'。依次进行。可知，匹配过程就是在一个图中走动，图中某一个结点标记了匹配了某个模式串。    接下来一个问题就是如何构建 失败指针。  构建失败指针可以用一个BFS过程来构建。  伪码为：   |  | | --- | | Q.push(root); while( !Q.empty() )     p= Q.top(); Q.pop();     for( each child t of p )         tp= p.fail         while( root && tp.child[t]= null ) tp= tp.fail;         if( tp== root ) p.next[t].fail= root;         else p.next[t].fail= tp.child[t];         Q.push( p.child[t] ); |   HDU 2222 Keywords Search (基本的AC自动机)  代码：   |  | | --- | | #include <stdio.h> #include <stdlib.h>  int const N= 500010; struct Trie{     int flag; // 标记是否为某一模式串的结尾       int fail; // 失败指针       int next[26];     void init(){         flag= 0; fail= -1;         for( int i= 0; i< 26; ++i ) next[i]= 0; } }tb[N];  int cnt= 0, que[N], n; char str[1000010];  void inline insert( char\* s ){     int rt= 0;     while( \*s ){         int t= \*s- 'a';         if( !tb[rt].next[t] ){             tb[++cnt].init();             tb[rt].next[t]= cnt;         }         rt= tb[rt].next[t]; s++;     }     tb[rt].flag++; }  void bfs(){     int head= 0, tail= 0, p, q;     que[0]= 0;     while( head<= tail ){         int now= que[head++];         for( int t= 0; t< 26; ++t )         if( tb[now].next[t] ){             p= tb[now].fail, q= tb[now].next[t];             while( -1 && !tb[p].next[t] ) p= tb[p].fail;             if( p== -1 ) tb[q].fail= 0;             else tb[q].fail= tb[p].next[t];             que[++tail]= q;         }     } }  void Match( char\* s ){     int ans= 0, rt= 0, t, p;     while( \*s ){         t= \*s- 'a';         if( tb[rt].next[t] ) rt= tb[rt].next[t];         else{             p= tb[rt].fail;             while( -1 && !tb[p].next[t] ) p= tb[p].fail;             if( p== -1 ) rt= 0;             else rt= tb[p].next[t];         }         p= rt;         while( 0 && tb[p].flag ){             if( tb[p].flag ){                 ans+= tb[p].flag; tb[p].flag= 0; }             p= tb[p].fail;         }         s++;     }     printf("%d\n", ans ); }  int main(){     int test;     scanf("%d",&test );      while( test-- ){         scanf("%d\n",&n );          cnt= 0; tb[0].init();         while( n-- ){             gets(str);             insert( str );         }         bfs();         gets(str);         Match( str );     }          return 0; } | | |