Tire树

文章分类:C++编程

Tire树:

先简单解释一下Trie。“Trie”这个单词来自于"retrieve"，可见它的用途主要是字符串查询。Trie不发tree的音，而发try的音。

应用

题目大意:

统计子串出现的次数,规定子串的长度不超过8.

http://acm.scs.bupt.cn/onlinejudge/showproblem.php?problem\_id=1302

trie算法的思路大概如下：

对输入的长串S 想将所有长度位1-8的子串存入trie树，并在子串末节点上记录下字串出现的次数，然后针对每个输入的查询串s 只要在这颗树上查找 一遍就能找到得到该子串出现的次数。

仍段代码先:

C++代码

#include <iostream>

using namespace std ;

class trie

{

private :

int count[26] ;

trie \* child[26] ;

public :

trie ()

{

int i ;

for ( i = 0 ; i < 26 ; i++ )

{

count[i] = 0 ;

child[i] = NULL ;

}

}

~trie ()

{

int i ;

for ( i = 0 ; i < 26 ; i++ )

{

delete child[i] ;

}

}

void insert ( const char \*key , const int & leng ) //生成了字典树,这次操作统计的次数全部以\*key开头

{

if ( leng == 0 )

return ;

int index = \*key - 'a' ;

if ( child[index] == NULL ) child[index] = new trie ;

count[index]++ ;

child[index] -> insert ( key + 1 , leng - 1 ) ;

}

int quary ( const char \*key ) //返回子串出现的次数

{

if ( \* ( key + 1 ) == '\0' ) //最后一个字符

return count[\*key - 'a'] ;

if ( child[\*key - 'a'] == NULL ) //只有遇到不存在的部分子串,必定返回0.

return 0 ;

return child[\*key - 'a'] ->quary ( key + 1 ) ; //递归寻找下一个字符

}

} ;

char s[32768 + 10] ;

int main ()

{

int stringLength , quaryNumber , left , right , substringLength , test = 1 ;;

while ( scanf ( "%s" , s ) != EOF )

{

trie stringTrie ;

substringLength = 8 ;

stringLength = strlen ( s ) ;

for ( left = 0 ; left + substringLength <= stringLength ; left++ )

stringTrie.insert ( s + left , substringLength ) ; //生成了字典树

for ( substringLength = 7 ; left < stringLength ; left ++ , substringLength-- )

stringTrie.insert ( s + left , substringLength ) ; //添加最后7个开头子串的统计数

scanf ( "%d" , &quaryNumber ) ;

printf ( "Case %d:\n" , test++ ) ;

while ( quaryNumber -- )

{

scanf ( "%s" , s ) ;

printf ( "%d\n" , stringTrie.quary ( s ) ) ;

}

}

return 0 ;

}

这个题目也可以用hash做.

基本思路:

这里采用的最简单的26进制的函数对子串进行hash，由于26^8 会超出int的范围所以在计算hash值时需要64位的整数。在算出每个子串的 hash值后就将相应的slot标记好，并记录下这个slot被标记过几次，然后对各查询子串也用相同的hash函数来处理，然后查询子串所在的slot 的计数器就行了。

C++代码

#include<iostream>

using namespace std ;

typedef long long \_\_int64 ;

char s[1<<16] ;

const int HASH\_SIZE = 1<<18 ;

\_\_int64 mark[HASH\_SIZE] ;

int counter[HASH\_SIZE] ;

int getHashVal ( \_\_int64 tmp )

{

return tmp&(HASH\_SIZE-1) ;

}

void insert ( \_\_int64 tmp )

{

int val = getHashVal ( tmp ) ;

int index = val ;

//index代表下标和tmp的值不是对应的,因此每次都需要需要tmp对应的index.

while ( mark[index] != -1 && mark[index] != tmp )

{

index++ ;

if ( index == HASH\_SIZE ) index = 0 ;

}

counter[index]++ ;

mark[index] = tmp ;

}

int query ( \_\_int64 tmp )

{

int val = getHashVal ( tmp ) ;

int index = val ;

while ( mark[index] != -1 && mark[index] != tmp )

{

index++ ;

if ( index == HASH\_SIZE ) index = 0 ;

}

return counter[index] ;

}

int main()

{

\_\_int64 tmp ;

int t , Q ;

for ( t = 1 ; scanf ( "%s" , s ) != EOF ; ++t )

{

memset ( mark , -1 , sizeof ( mark ) ) ;

memset ( counter , 0 , sizeof ( counter ) ) ;

printf ( "Case %d:\n" , t ) ;

for ( int i = 0 ; s[i] != '\0' ; ++i ) //s[i] != '\0',因此所有的子串都可以遍历到.

{

tmp = 0 ;

for ( int k = 0 ; k < 8 && s[i + k] != '\0'; ++k ) //转化为27进制的一个数

{

tmp = tmp \* 27 + s[i + k] - 'a' + 1 ;

insert ( tmp ) ;

}

}

scanf ( "%d" , &Q ) ;

while ( Q-- )

{

scanf ( "%s" , s ) ;

tmp = 0 ;

for ( int i = 0 ; s[i] != '\0' ; ++i )

{

tmp = tmp \* 27 + s[i] - 'a' + 1 ;

}

printf ( "%d\n" , query ( tmp ) ) ;

}

}

return 0;

}