# 重叠的最长子串 （扩展KMP算法）（滚动哈希算法--Rabin-Karp算法）

[字符串匹配](http://blog.csdn.net/tag/details.html?tag=%e5%ad%97%e7%ac%a6%e4%b8%b2%e5%8c%b9%e9%85%8d" \t "https://blog.csdn.net/pi9nc/article/details/_blank)[前后缀匹配](http://blog.csdn.net/tag/details.html?tag=%e5%89%8d%e5%90%8e%e7%bc%80%e5%8c%b9%e9%85%8d" \t "https://blog.csdn.net/pi9nc/article/details/_blank)

****题目描述：****

给定两个字符串，求它们前后重叠的最长子串的长度，比如"abcde"和“cdefg”是"cde"，长度为3。

****输入：****

输入可能包含多个测试案例。  
对于每个测试案例只有一行， 包含两个字符串。字符串长度不超过1000000，仅包含字符'a'-'z'。

****输出：****

对应每个测试案例，输出它们前后重叠的最长子串的长度。

****样例输入：****

abcde cdefg

****样例输出：****

3

思路：扩展KMP，用extend[i]保存 主串 S[i.....n-1]与 模式串 T的最长公共前缀的长度，其中n是S的长度。

然后扫描一遍 extend[] 如 extend[i] == n-i 那么这个后缀的长度就是我们要求的值。

关于扩展KMP，可以去看论文：《求最长回文子串与最长重复子串》何林 的集训队论文

代码：

#include <iostream>

#include <cstring>

#include <cstddef>

#include <cstdio>

**using** **namespace** std;

**const** **int** MAXN = 1000001;

**int** next[MAXN];

**int** extend[MAXN];

**char** S[MAXN],T[MAXN];

**void** getnext(**char** \*T)

{

**int** k = 0;

**int** Tlen = strlen(T);

next[0] = Tlen;

**while**(k < Tlen-1 && T[k] == T[k+1])

k++;

next[1] = k;

k = 1;

**for**(**int** i = 2; i < Tlen; i++)

{

**int** p=k+next[k]-1,L=next[i-k];//p:=已经匹配到的最远的位置

// L :=

**if**((i-1)+L >= p)

{

**int** j=(p-i+1)>0? p-i+1:0;

**while**(i+j < Tlen && T[k+j]==T[j])

j++;

next[i] = j;

k = i;

}

**else** next[k]=L;

}

}

**void** getextend(**char** \*S,**char** \*T)

{

**int** a = 0;

getnext(T);

**int** Slen = strlen(S);

**int** Tlen = strlen(T);

**int** len = Slen<Tlen ? Slen:Tlen;

**while**(a < len && S[a] == T[a]) a++;

extend[0] = a;

a = 0;

**for**(**int** i = 1; i < Slen; i++)

{

**int** p = a + extend[a]-1, L = next[i-a];

**if**(i+L-1 >= p)

{

**int** j = (p-i+1)>0? p-i+1:0;

**while**(i+j < Slen && j<Tlen && S[i+j] == T[j])

j++;

extend[i] = j;

a = i;

}

**else** extend[i] = L;

}

}

**int** main()

{

**while**(scanf("%s%s",&S,&T) != EOF)

{

getextend(S,T);

**int** n = strlen(S);

**int** res = 0;

**for**(**int** i = 0; i < n; i++)

{

**if**(extend[i] == n-i)

{

res = extend[i];

**break**;

}

}

cout<<res<<endl;

}

**return** 0;

}