**KMP算法总结**

**1.查找两个字符串匹配位置**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define N 100

void cal\_next( char \* str, int \* next, int len ){

int i, j;

next[0] = -1;

for( i = 1; i < len; i++ )

{

j = next[ i - 1 ];

while( str[ j + 1 ] != str[ i ] && ( j >= 0 ) )

{

j = next[ j ];

}

if( str[ i ] == str[ j + 1 ] )

{

next[ i ] = j + 1;

}

else

{

next[ i ] = -1;

}

}} **//计算next数组**

int KMP( char \* str, int slen, char \* ptr, int plen, int \* next ){

int s\_i = 0, p\_i = 0;

while( s\_i < slen && p\_i < plen )

{

if( str[ s\_i ] == ptr[ p\_i ] )

{

s\_i++;

p\_i++;

}

else

{

if( p\_i == 0 )

{

s\_i++;

}

else

{

p\_i = next[ p\_i - 1 ] + 1;

}

}

}

return ( p\_i == plen ) ? ( s\_i - plen ) : -1;}**//KMP算法**

int main(){

char str[ N ] = {0};

char ptr[ N ] = {0};

int slen, plen;

int next[ N ];

while( scanf( "%s%s", str, ptr ) )

{

slen = strlen( str );

plen = strlen( ptr );

cal\_next( ptr, next, plen );

printf( "%d\n", KMP( str, slen, ptr, plen, next ) );

}

return 0;

}**//主函数**

1. **下面给出一段计算next[]的代码：**

这样求出来的next数组其实是从下标1开始的，因为下标0之前是个空串，下标1则对应着M串的第0个字符。我们设next[0]=-1，仅仅是个标志而已，没有什么特殊的含义。

#include<string.h>

#include <iostream>

using namespace std;

void compute\_next(const string& pattern)

{

const int pattern\_length = pattern.size();

int \*next\_function = new int[pattern\_length];

int index;

next\_function[0] = -1;

for(int i=1;i<pattern\_length;++i)

{

index = next\_function[i-1];

//store previous fail position k to index;

while(index>=0 && pattern[i]!=pattern[index+1])

{

index = next\_function[index];

}

if(pattern[i]==pattern[index+1])

{

next\_function[i] = index + 1;

}

else

{

next\_function[i] = -1;

}

}

for(int i=0;i<pattern\_length;++i)

{

cout<<next\_function[i]<<endl;

}

delete[] next\_function;

}

int main()

{

string pattern = "abaabcaba";

compute\_next(pattern);

return 0;

}

**运行结果为：**

-1

-1

0

0

1

-1

0

1

2

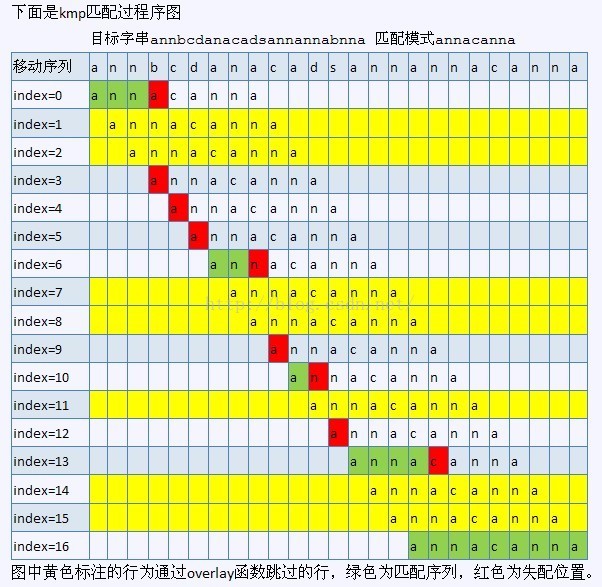
1. **改进型KMP算法:**
2. 若引入f(j)作为媒介，对 f(j) 和 next(j) 重新定义如下：

* f(j)是满足pattern[1...k - 1] = pattern[(j - (k - 1))...j -1](k < j)的k中，k的最大值
* next[j]是所有满足pattern[1...k - 1] = pattern[(j - (k - 1))...j -1](k < j)，且pattern[k] != pattern[j]的k中，k的最大值

根据定义，f(j)与next[j]的有如下递推公式：

* 如果 pattern[j] != pattern[f(j)]，next[j] = f(j);
* 如果 pattern[j] = pattern[f(j)]，next[j] = next[f(j)];

可以看出，本篇介绍的next(i) 其实就是f(j), 并不是最优跳转表。通过f(j)可以进一步计算最优的跳转表，最优跳转表对有多个重复字符的pattern[]，会表现出非常高的性能！



**B，Prefix Substring，给两个串a,b，问多少个a的子串是b的前缀。**

**去长春那天刚好讲了KMP。用next数组的性质水过。**

#include <iostream>

#include <stdio.h>

#include <string.h>

using namespace std;

typedef long long LL;

const int maxn = 1e5 + 5;

char a[maxn], b[maxn];

int next[maxn], f[maxn];

void prekmp(char \*b, int \*next)

{

memset(next,-1,sizeof next);

memset(f,0,sizeof f);

int j = -1;

for (int i=1; b[i]; i++)

{

while (j!=-1 && b[i]!=b[j+1])

j = next[j];

if (b[i] == b[j+1]) j++;

next[i] = j;

}

for (int i=0; b[i]; i++)

{

int tmp = i;

while (tmp != -1)

{

f[i]++;

tmp = next[tmp];

}

}

}

LL kmp(char \*a, char \*b, int \*next)

{

int j = -1;

LL ret = 0;

for (int i=0; a[i]; i++)

{

while (j!=-1 && a[i]!=b[j+1])

j = next[j];

if (a[i] == b[j+1]) ret += f[++j];

}

return ret;

}

int main()

{

scanf("%s%s",a,b);

prekmp(b, next);

printf("%lld\n",kmp(a,b,next));

return 0;

}

**1715: 字符串匹配**

Time Limit: 0 Sec  Memory Limit: 128 MB  
Submit: 27  Solved: 5  
[[Submit](http://10.5.54.252/oj/submitpage.php?id=1715)][[Status](http://10.5.54.252/oj/problemstatus.php?id=1715)][[Web Board](http://10.5.54.252/oj/bbs.php?pid=1715)]

**Description**

给你两个串A，B，可以得到从A的任意位开始的子串和B匹配的长度。 给定K个询问，对于每个询问给定一个x，求出匹配长度恰为x的位置有多少个。 N，M，K< =200000

**Input**

第一行三个数  N，M，K，表示A的长度、B的长度和询问数。 第二行为串A。 第三行为串B。 接下来K行，每行1个数X。

**Output**

对于每个询问输出一个数。

**Sample Input**

6 2 2

aabcde

ab

0

2

**Sample Output**

4

1

**HINT**

**【分析】**

**想法是，用KMP计算出cnt[i]表示可以达到的长度大于等于i的匹配的个数。**

**那么长度为i的匹配的个数就是cnt[i] – cnt[i + 1]。因为KMP只能求到某个位置的最大匹配。**

**记a的每个位置的最大匹配长度是x。初始化cnt[i]为i这个最长匹配长度的频率。**

**如果a[i]所能达到的最大匹配是x，那么以a[i]为结尾长度为ne[x]的字符串也是一个匹配。因为ne[x] <= x。所以要在cnt[ne[x]]上加上cnt[x]。**

#include <stdio.h>

#define MAXN 200001

char a[MAXN],b[MAXN];

int ne[MAXN],cnt[MAXN];

int m,n,k,x,t;

int main() {

scanf("%d%d%d",&n,&m,&t);

scanf("%s%s",a,b);

for (int i = n;i > 0;--i)

a[i] = a[i - 1];

for (int i = m;i > 0;--i)

b[i] = b[i - 1];

int k = 0;

for (int i = 2;i <= m;++i)

{

while ((k > 0) && (b[k + 1] != b[i]))

k = ne[k];

if (b[k + 1] == b[i])

++k;

ne[i] = k;

}

k = 0;

for (int i = 1;i <= n;++i)

{

while ((k > 0) && (b[k + 1] != a[i]))

k = ne[k];

if (b[k + 1] == a[i])

++k;

++cnt[k];

}

for(int i = n;i > 0;--i)

cnt[ne[i]] += cnt[i];

for(int i = 1;i <= t;++i) {

scanf("%d",&x);

printf("%d\n",cnt[x] - cnt[x + 1]);

}

return 0;

}

B-View Code

<http://blog.csdn.net/yearn520/article/details/6729426>

<http://blog.csdn.net/wangbaochu/article/details/50687160>

<http://blog.csdn.net/joylnwang/article/details/6778316#comments>

<http://www.tuicool.com/articles/e2Qbyyf>