！！！！！！非常重要 KMP算法

该算法的关键在于：在**当前对文本串和模式串检索的过程中，若出现了不匹配，如何充分利用已经匹配的部分**。——这个是算法的核心理解

在说明KMP算法之前 我们先明确 前缀和后缀的含义 以及前后缀表

**前缀**＝不包含末尾的且包含头字符的一个字符串中的所有子串

**后缀**＝不包含头字符且包含尾字符的一个字符串中的所有子串

给出一个例子比如aabaaf

a：前后缀都为0，长度为0；

aa：前后缀都为a，长度为1；

aab：前缀aa，后缀ab，不相同，再取前缀a，后缀b，不相同，长度为0；

aaba：前缀aab，后缀aba，不相同，再取前缀aa，后缀ba，不相同，再取前缀a，后缀a，相同，长度为1；

aabaa：前缀aaba，后缀abaa，不相同，再取前缀aab，后缀baa，不相同，再取前缀aa，后缀aa，相同，长度为2；

aabaaf：前缀aabaa，后缀abaaf，不相同，再取前缀aaba，后缀baaf，不相同，再取前缀aab，后缀aaf，不相同，再取前缀aa，后缀af，不相同，再取前缀a，后缀f，不相同，长度为0.

所以该字符串的前后缀表为010120

如不明白 再给出一个例子 aabaab

a: 前后缀都为0，长度为0；

aa:前后缀都为a,长度为1；

aab；前缀为aa，后缀为ab，不相同，再取前缀a,后缀b，不相同，长度为0；

aaba：前缀为aab，后缀为aba，不相同，再取前缀aa，取后缀ba，不相同，再取前缀a，后缀a，相同，长度为1；

aabaa：前缀aaba，后缀abaa，不相同，再取前缀aab，后缀baa，不相同，再取前缀aa，后缀aa，相同，长度为2；

aabaab：前缀为aabaa，后缀为abaab，不相同，再取前缀aaba，后缀为baab，不相同，再取前缀aab，取后缀aab，相同，长度为3；

所以该字符串的前后缀表为010123；

通过两个例子 应该可以明白前后缀 以及前后缀表

那么如何写出代码呢？

我们先明确i与j的定义为 i代表后缀末尾 j代表前缀末尾

注意j不仅仅代表前缀末尾 还有一层隐形含义是目前所能扩展的最大公共子串的长度

根据上面的例子 逻辑上来说i到循环到最后一位 所以最外层一个for循环

在循环内

{

如果i与j相同那么j与i分别再扩展下一位

如果i与j不相同那么可以使j直接为0吗？

显然不是 由于j是最大公共子串的末尾 当不相同时 那么j要回溯到最大公共子串末尾的前一位 再判断该位是否与i相同 如果相同则扩展j

如果还是不相同则j继续回溯到前一位 直到j回溯到第一位才算放弃

}

如果不是很明白的话可以按着程序走一遍下面这个例子

aabaaab

这个例子可用来解释 为什么在j与i不相同的情况下要写成j=next[j-1] 而不是直接宣判s[i]=0； 从感性上来讲 相当于next作用于他自己

vector<int> getnext(string s)

{

int j = 0;

vector<int> next(s.size());

next[0] = 0;

for (int i = 1; i < s.size(); i++)

{

while (j>0 && s[i]!=s[j])

{

j = next[j - 1];

}

if (s[i]==s[j])

{

j++;

}

next[i] = j;

}

return next;

}

int main()

{

string s = "aabaaab";

for (auto i : getnext(s))

{

cout << i << '\t';

}

}

以下给出完整代码

int strStr\_KMP(string haystack, string needle)

{

if (needle.size()==0)

{

return 0;

}

vector<int> next = getnext(needle);

int j = 0;

for (int i = 0; i < haystack.size(); i++)

{

while (j>0 && haystack[i]!=needle[j])

{

j = next[j - 1];

}

if (haystack[i]==needle[j])

{

++j;

}

if (j==needle.size())

{

return i - needle.size() + 1;

}

}

return -1;

}

int main()

{

string a = "aabaaf";//010120

string b = "aacaabaabaaf";

for (auto i : getnext(a))

{

cout << i << '\t';

}

cout << endl;

cout << strStr\_KMP(b, a);

}