KMP的个人理解和解析

题外话：刚刚学习KMP算法，一脸懵逼，不过经过一天的思考、查找资料、手动画图模拟啥的终于算是搞清楚了（可能吧，其实我心里也没底），在此写一篇解析整理一下思路。

一、什么是KMP算法及一些基本概念

首先，什么是KMP算法。这是一个字符串匹配算法，对暴力的那种一一比对的方法进行了优化，使时间复杂度大大降低（我不会算时间复杂度。。。，目前也只能这么理解，还有KMP是取的三个发明人的名字首字母组成的名字）。

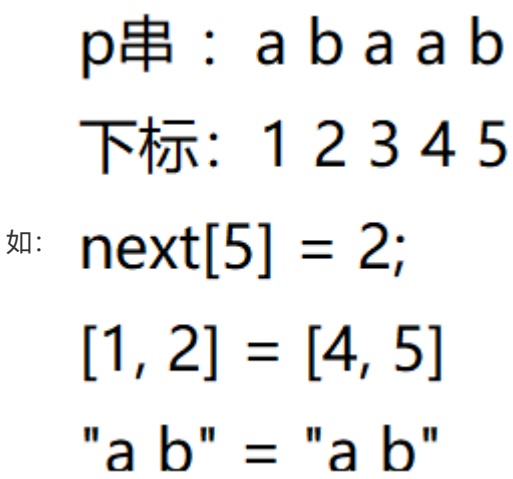
然后是一些基本概念：

1、s[ ]是*模式串*，即比较长的字符串。
2、p[ ]是*模板串*，即比较短的字符串。（这样可能不严谨。。。）
3、“非平凡前缀”：指除了最后一个字符以外，一个字符串的全部头部组合。
4、“非平凡后缀”：指除了第一个字符以外，一个字符串的全部尾部组合。（后面会有例子，均简称为前/后缀）
5、“部分匹配值”：*前缀*和*后缀*的**最长共有**元素的**长度**。
6、next[ ]是“部分匹配值表”，即next数组，它存储的是每一个下标对应的“部分匹配值”，是KMP算法的**核心**。（后面作详细讲解）。

核心思想：在每次失配时，不是把p串往后移一位，而是把p串往后移动至下一次可以和前面部分匹配的位置，这样就可以跳过大多数的失配步骤。而每次p串移动的步数就是通过查找next[ ]数组确定的。

二、next数组的含义及手动模拟（具体求法和代码在后面）

然后来说明一下next数组的含义：对next[j]，是p[1, j]串中*前缀*和*后缀*相同的最大长度（部分匹配值），即 p[1, next[j]] = p[j - next[j] + 1, j]。



手动模拟求next数组：对 p = “abcab”

p	a	b	c	a	b
下标	1	2	3	4	5
next[ ]	0	0	0	1	2

对next[1]： 前缀 = 空集————————后缀 = 空集————————next[1] = 0;

对next[2]： 前缀 = { a }————————后缀 = { b }————————next[2] = 0;

对next[3]： 前缀 = { a , ab }————————后缀 = { c , bc}————————next[3] = 0;

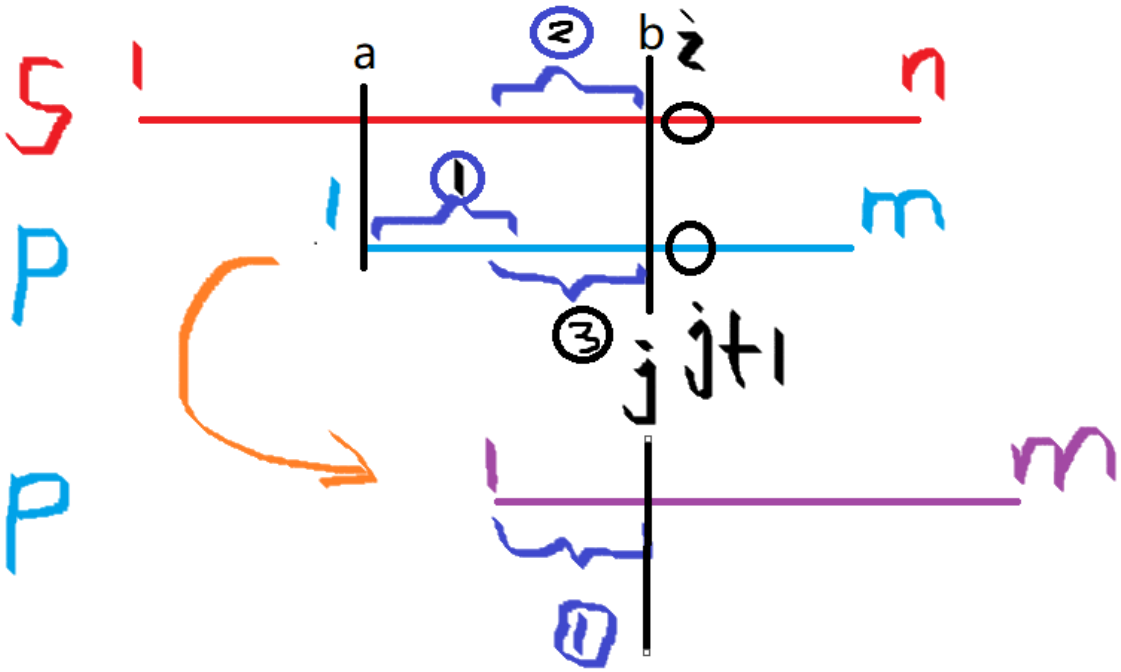
对next[4]： 前缀 = { **a** , ab , abc }————————后缀 = { **a** . ca , bca }————————next[4] = 1;

对next[5]： 前缀 = { a , **ab** , abc , abca }————————后缀 = { b , **ab** , cab , bcab}————————next[5] = 2;

三、匹配思路和实现代码

KMP主要分两步：**求next数组**、**匹配字符串**。个人觉得匹配操作容易懂一些，疑惑我一整天的是求next数组的思想。所以先把匹配字符串讲一下。

s串 和 p串都是从1开始的。i 从1开始，j 从0开始，每次s[i] 和p[j + 1]比较



当匹配过程到上图所示时，

s[a , b] = p[1, j] **&&** s[i] != p[j + 1] 此时要移动p串（不是移动1格，而是直接移动到下次能匹配的位置）

其中1串为[1, next[j]]，3串为[j - next[j] + 1 , j]。由匹配可知 **1串等于3串**，**3串等于2串**。所以直接移动p串使1到3的位置即可。这个操作可由**j = next[j]**直接完成。 如此往复下去，当 j == m时匹配成功。

代码如下

```
for(int i = 1, j = 0; i <= n; i++)
{
    while(j && s[i] != p[j+1]) j = ne[j];
    //如果j有对应p串的元素， 且s[i] != p[j+1], 则失配， 移动p串
    //用while是由于移动后可能仍然失配，所以要继续移动直到匹配或整个p串移到后面（j = 0）

    if(s[i] == p[j+1]) j++;
    //当前元素匹配，j移向p串下一位
    if(j == m)
    {
```

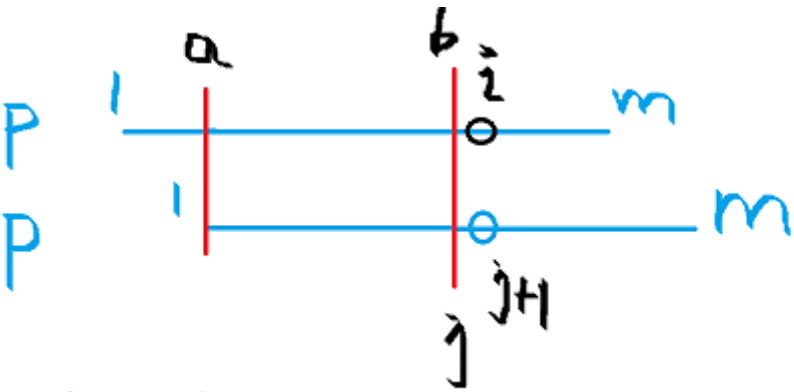
```
            //匹配成功，进行相关操作
            j = next[j];  //继续匹配下一个子串
        }
    }
```

注：采用上述的匹配方法（ i 与 j+1 比较）我不清楚（其实是想不清楚）为什么要这样。。。脑子有点不好使。而不推荐下标从0开始的原因***我认为***是：若下标从0开始的话，next[ ]数组的值都会相应-1，这就会导致它的实际含义与其定义的意思不符（部分匹配值和next数组值相差1），思维上有点违和，容易出错。

（看了习题课，在实际操作上下标从0开始代码会多很多东西，比从1开始复杂一些，嗯。。。确实

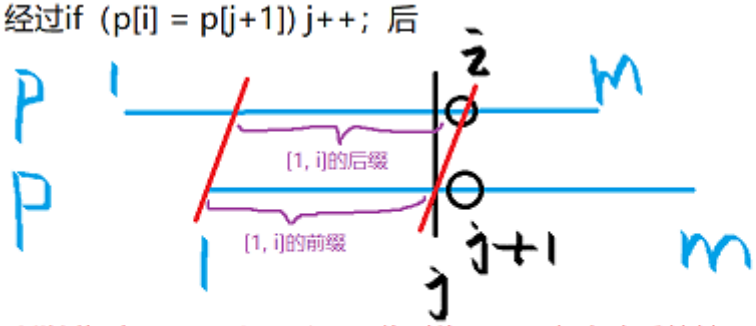
四、求next数组的思路和实现代码

next数组的求法是通过模板串自己与自己进行匹配操作得出来的（代码和匹配操作几乎一样）。



p[a, b] = p[1, j]

经过if (p[i] = p[j+1]) j++; 后



所以此时，next[i] = j。（此时为下面p串移动后的情况，即代码中if后面的。len(前缀) = len(后缀) = j

代码如下

```
for(int i = 2, j = 0; i <= m; i++)
{
    while(j && p[i] != p[j+1]) j = next[j];

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

    next[i] = j;
}
```

代码和匹配操作的代码几乎一样，关键在于每次移动 i 前，将 i 前面已经匹配的长度记录到next数组中。

五、完整代码

```cpp
// 注：这不是题目的AC代码，是一个最基本的模板代码
#include <iostream>

using namespace std;

const int N = 100010, M = 10010; //N为模式串长度，M匹配串长度

int n, m;
int ne[M]; //next[]数组，避免和头文件next冲突
char s[N], p[M];  //s为模式串，  p为匹配串

int main()
{
    cin >> n >> s+1 >> m >> p+1;  //下标从1开始

    //求next[]数组
    for(int i = 2, j = 0; i <= m; i++)
    {
        while(j && p[i] != p[j+1]) j = ne[j];
        if(p[i] == p[j+1]) j++;
        ne[i] = j;
    }
    //匹配操作
    for(int i = 1, j = 0; i <= n; i++)
    {
        while(j && s[i] != p[j+1]) j = ne[j];
        if(s[i] == p[j+1]) j++;
        if(j == m)  //满足匹配条件，打印开头下标，从0开始
        {
            //匹配完成后的具体操作
            //如：输出以0开始的匹配子串的首字母下标
            //printf("%d ", i - m); (若从1开始，加1)
            j = ne[j];              //再次继续匹配
        }
    }

    return 0;
}
```

六、参考

1、 AcWing算法基础课。

2、字符串匹配的KMP算法——前缀和后缀的详解， 作者：阮一峰

https://blog.csdn.net/maotianwang/article/details/34466483