# KMP步骤说明

## 模式串匹配算法：

模式串匹配算法是指在一个字符串中查找另一个字符串是否存在以及存在位置的算法。在模式串匹配算法中，被查找的字符串称为模式串，查找于的字符串称为主串。

显然暴力匹配时间复杂度是lena\*lenb 太慢，不多说废话了。

## KMP模式串匹配算法：

1.模式串匹配算法的产生：

主串s[]： a b c a c b c a b a c b a

模式串p[]： a c b a

在这次匹配失配后，很明显将模式串移动一个位置后是不可能匹配成功的，因为此时c字符对应的模式串中的a字符的上一个字符是b，b是不可能和c匹配成功的。所以这一次匹配根本不需要去做。

这个栗子表明，如果提前知道模式串或者主串的某两个字符之间能否匹配，就能提高效率。

Kmp大致就是这个思路。步骤是先求一个模式串next数组，它用来匹配时索引模式串移动信息，next数组长度是lenp。求完next数组再，用next数组辅助匹配，具体后面说。

**next数组,有3种版本，分别介绍：**

## next版本一：

next[0]从-1开始(代表第0个字符串的上一个位置)，在后面匹配时，匹配成功返回下标。

对模式串p求next数组，初始化next[0]=-1 两个光标k=0和j=1

参考字符串：  **ABCDABD next:-1 0 0 0 0**

进行循环，每次比对p[k]和p[j]

如果p[k]==p[j] :

令next[j]=next[k] 也就是靠右的等于靠左的

J和k都加一

如果p[k]!=p[j] :

令next[j]=k 也就是靠右的等于靠左光标值

While(k!=-1&& p[k]！=p[j]) (k=next[k]

把左的光标移动到它的next位置,不断移动直到k是-1或者两个位置的字符相等

k==-1 ： 靠左光标值是-1 也就是若k被移到了-1位置，就要k和j都加一：

J和k都加一

上述循环结束条件是 j超出了lenp的长度 也就是靠右的光标超出边界

此版本好处是可以匹配多个，比如在 ABABABC里找 ABA 利用ABAnext数组能找到2个在ABABABC匹配的位置。

void getnext1(int \*nex,char \*word) {

nex[0]=-1;

int k=-1,j=0, num=(int)strlen(word);

while(j<num){

if(k==-1||word[j]==word[k]){

j++;k++;

if(word[j]!=word[k]) nex[j]=k;

else nex[j]=nex[k];

}else k=nex[k];

}

}

下面说怎么匹配：光标k=0代表主串位置， j=0代表模式串位置

比较s[k]和p[j]

如果，

s[k]==p[j] ：

J和k都加一 代表匹配上了，光标移动一

s[k]!=p[j] ：

j =next[j]; 模式串的光标调到next位置，

如果j==-1

J和k都加一

J==lenp: 此时代表匹配上了一个

j=nextt[j] 此时代表匹配上了一个，应该当做没匹配上继续匹配，不想再找也可以退出程序。

以上循环的结束条件是k超出了lens的界限，即k==lens

整个过程k一路向前，没有回头，说明了这个算法主串光标不回溯，时间线性。