KMP 算法三个核心内容

- 主程序的匹配代码
  - 一个指针指向主串
  - 一个指针指向子串
- next 数组
- nextval 数组

next 数组

通常而言有两个版本 { 首元素为 -1 / 首元素为 0

首元素为-1的版本 ⟶ 对应数组下标从0开始的版本

下标	0	1	2	3	4	5
	a	b	a	b	a	a
next	-1	0	0	1	2	3

当子串与主串无法匹配时

子串从 next[j] 对应位置重新匹配

若 next[j] == -1 则 i ++

首元素为0的版本 ⟶ 对应数组下标从1开始的版本

下标	1	2	3	4	5	6
	a	b	a	b	a	a
next	0	1	1	2	3	4

当子串与主串无法匹配时

子串从 next[j] 对应位置重新匹配

若 next[j] == 0 则 i ++

可以发现此版本 next 数组对应位置比上一版本大 1

nextval 数组

可以视作 next 数组优化版，可以进一步在回溯中抛弃不可能的匹配位置

当构建出 nextval 数组后，就只需要用 nextual 确定回溯位置，

由此观之，next 数组更像一个中间产物

```
int KMP_from_minus_one(MS * pms,SS * pss)
{
    //根据字串创建next数组
    /*
    注意:此处我采用首元素为-1的next数组版本

    next[i]的大小表示从序号0至序号i-1的字符串具有的最长公共前后缀的长度,也对应着表示公共前缀的下一个字符的序号

    在进入if判断的时候i对应着后缀的下一个字符,而j对应着前缀的下一个字符,若此时两字符相等,则前缀与后缀的长度+1,
    则此时i对应的字符成为了后缀的一部分,对应着后缀的最后一个字符;j对应的字符成为了前缀的一部分,对应着前缀的最后一个字符

    next数组的用法:在kmp算法中,当子串与主串进行匹配时,主串的指针一直向前移动(不会回溯),而子串的指针则需要回溯
    可以发现,当指向的字串字符与指向的主串字符不相等时,匹配失败,这时候字串指针就需要回溯,而回溯的位置下标恰好是匹配失败字符对应的next数组
值
    因此借助next数组可以实现高效字串指针回溯
    */



    int * next=(int *)calloc(1, sizeof(int)*pss->length);
    next[0]=-1;
    int i=0;
    int j=-1;
    while(i<pss->length-1)
    {
        if (j==-1||*(pss->string+i)==*(pss->string+j))
        {
            i++;
            j++;
            next[i]=j;
        }
        else
        {
            j=next[j];
        }
    }
```

```
//根据next数组构建nextval数组
//nextval是next数组的改良版,可以进一步排除一些不可能成立的回溯点,进一步减少时间损耗
//方式一:额外开辟一段空间创建nextval数组
int t=1;
int * nextval=(int *)calloc(1,sizeof(int)*pss->length);
nextval[0]=-1;
while (t<pss->length)
{
    if (*(pss->string+t)==*(pss->string+next[t]))
    {
        nextval[t]=next[next[t]];
    }
    else
    {
        nextval[t]=next[t];
    }
    t++;
}



//方式二:在next数组的原地修改为nextval(不需要额外开辟空间)
/*
int r=1;        //r从1开始就保证了next[t]一定>=0,因此next[next[r]]下标越界错误的问题
while(r<pss->length)
{
    if(*(pss->string+r)==*(pss->string+next[r]))
    {
        next[r]=next[next[r]];
    }
    r++;
}
*/
```

```
//实现kmp匹配
int u=0;
int v=0;
while(u<pms->length && v<pss->length)
{
    if (*(pms->string+u)==*(pss->string+v))
    {
        u++;
        v++;
    }
    else
    {
        v=nextval[v];
        if (v==-1)
        {
            v++;
            u++;
        }
    }
    if (v==pss->length)
    {
        return u-v;
    }
    else
    {
        return -1;
    }
}
```