洛谷P1420

最长连号

题目描述

输入长度为 n 的一个正整数序列，要求输出序列中最长连号的长度。

连号指在序列中，从小到大的连续自然数。

输入格式

第一行，一个整数 n。

第二行，n 个整数 a_i，之间用空格隔开。

输出格式

一个数，最长连号的个数。

样例 #1

样例输入 #1

```
10
1 5 6 2 3 4 5 6 8 9
```

样例输出 #1

```
5
```

提示

数据规模与约定

对于 100% 的数据，保证 $1 \leq n \leq 10^4$，$1 \leq a_i \leq 10^9$。

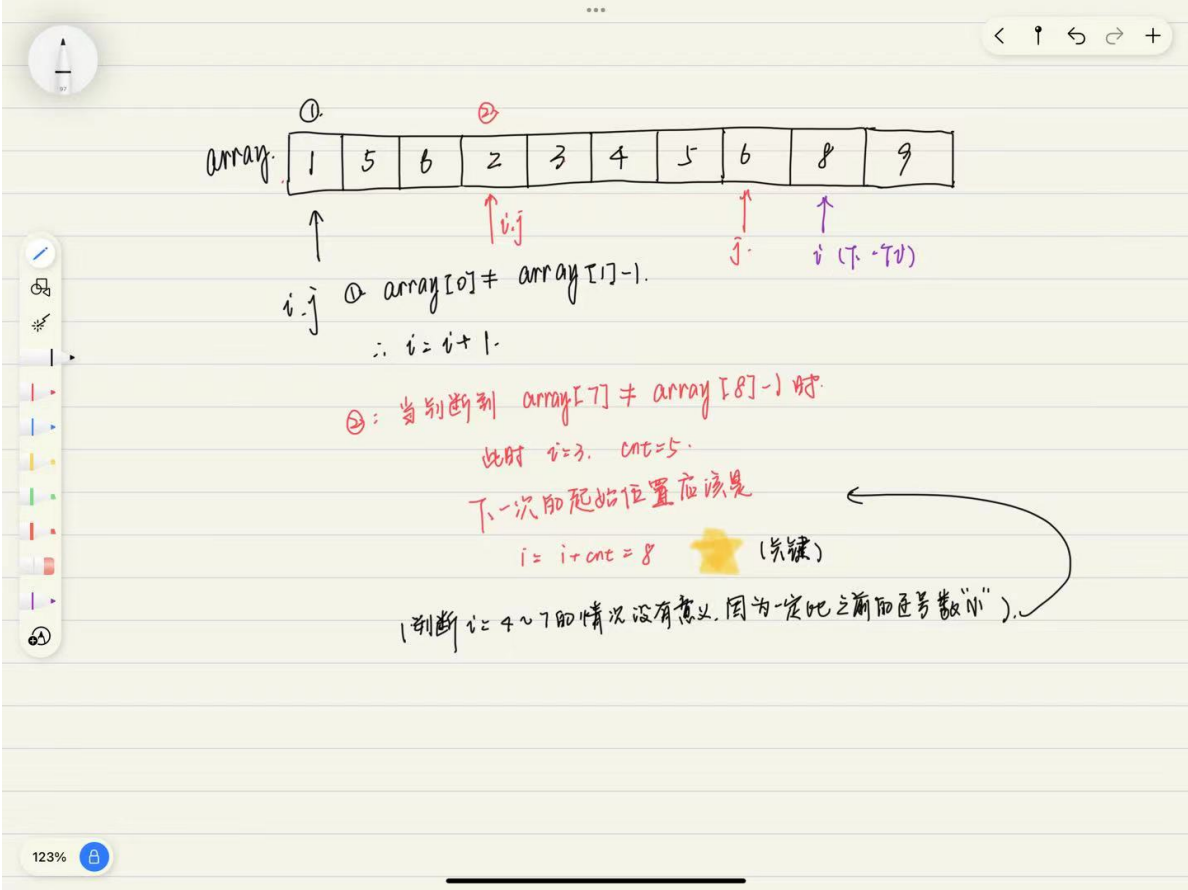
题解

```c
#include <stdio.h>
int array[10005];

int main()
{
    int cnt=1,max=0;    //cnt：记录连号的数字的个数，max：最大连号的个数
    int n;
    int i,j;
    scanf("%d", &n);
```

```
    for(i=0;i<n;i++){
        scanf("%d",&array[i]);
    }

    for(i=0;i<n;){          //i：循环变量
        for(j=i;j<n;j++)    //j：从array[i]往后检查连号的个数
        {
            if(array[j]==array[j+1]-1)
                cnt++;    //检查是否连号
            else break;     //如果不连号就退出循环，更新cnt的值
        }
        if(cnt > max)
            max = cnt;
        i=i+cnt;    //下一次的起点值的选取？（关键）
        cnt = 1;    //每次连号结束，重置连号数为1
    }
    printf("%d",max);
    return 0;
}
```

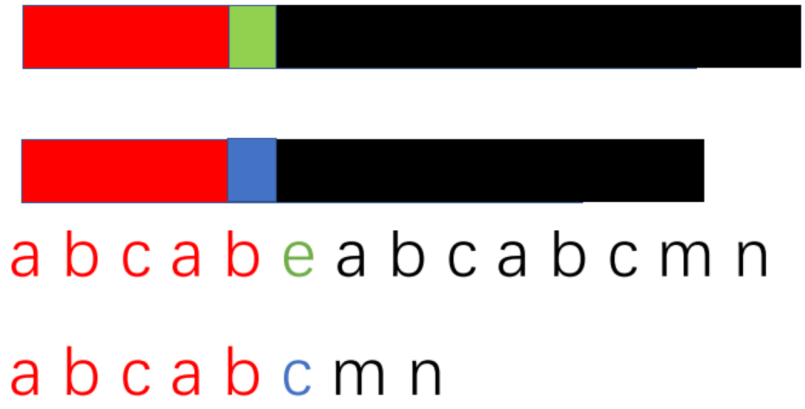


拓展：KMP算法

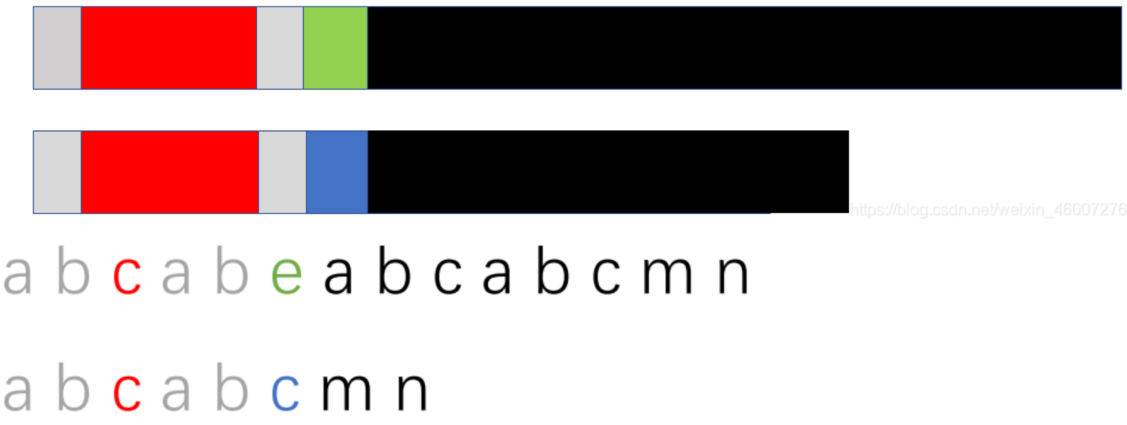
KMP算法作用：在一个已知字符串中查找子串的位置,也叫做串的模式匹配。比如主串s="university",子串t="sit"。现在我们要找到子串t 在主串s 中的位置。

图解KMP

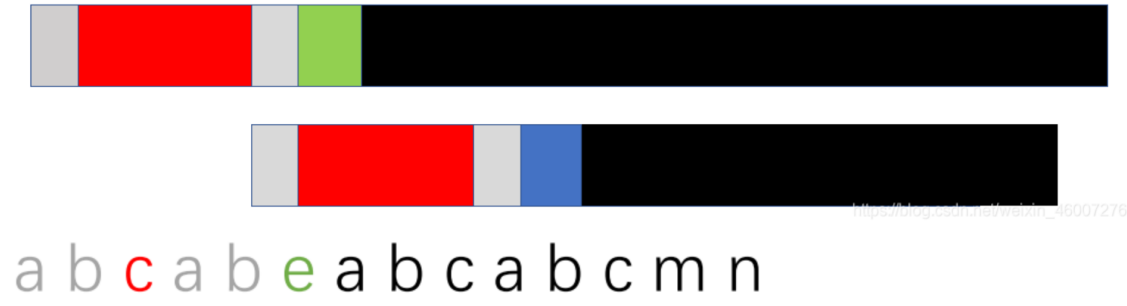
现在我们先看一个图：第一个长条代表主串，第二个长条代表子串。红色部分代表两串中已匹配的部分，
绿色和蓝色部分分别代表主串和子串中不匹配的字符。
再具体一些：这个图代表主串"abcabeabcabcmn"和子串"abcabcmn"。



现在发现了不匹配的地方，根据KMP的思想我们要将子串向后移动，现在解决要移动多少的问题。
之前提到的最长相等前后缀的概念有用处了。因为红色部分也会有最长相等前后缀。如下图：



灰色部分就是红色部分字符串的最长相等前后缀，我们子串移动的结果就是让子串的红色部分最长相等前缀和主串红色部分最长相等后缀对齐。



这一步弄懂了，KMP算法的精髓就差不多掌握了。接下来的流程就是一个循环过程了。事实上，**每一个字符前的字符串都有最长相等前后缀**，而且**最长相等前后缀的长度是我们移位的关键**，所以我们单独用一个next数组存储子串的最长相等前后缀的长度。**而且next数组的数值只与子串本身有关。**

所以next[i]=j,含义是：**下标为i 的字符前的字符串最长相等前后缀的长度为j。**

我们可以算出，子串t= "abcabcmn"的next数组为next[0]=-1(前面没有字符串单独处理)

next[1]=0；next[2]=0；next[3]=0；next[4]=1；next[5]=2；next[6]=3；next[7]=0；

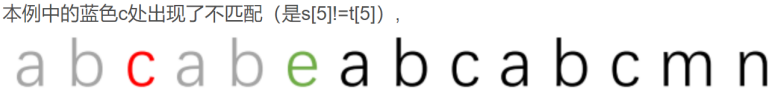
| a | b |c |a |b | c | m |n |

|–|–|–|–|–|–|–|–|–|

|next[0] |next[1] | next[2] | next[3] |next[4] |next[5] | next[6] | next[7] |

|-1 | 0 |0 | 0 | 1 | 2 |3 | 0 |

本例中的蓝色c处出现了不匹配（是s[5]!=t[5]），



我们把子串移动，也就是让s[5]与t[5]前面字符串的最长相等前缀后一个字符再比较，而该字符的位置就是t[？],很明显这里的？是2，就是**不匹配的字符前的字符串 最长相等前后缀的长度。**

也是不匹配的字符处的next数组next[5]应该保存的值，也是子串回溯后应该对应的字符的下标。 所以？=next[5]=2。接下来就是比对是s[5]和t[next[5]]的字符。这里也是最奇妙的地方，也是为什么KMP算法的代码可以那么简洁优雅的关键。

我们可以总结一下，next数组作用有两个：

一是之前提到的，

> next[i]的值表示下标为i的字符前的字符串最长相等前后缀的长度。

二是：

> 表示该处字符不匹配时应该回溯到的字符的下标

next有这两个作用的源头是：之前提到的字符串的最长相等前后缀

想一想是不是觉得这个算法好厉害，从而不得不由衷佩服KMP算法的创始人们。