假设现在有这样一个问题：有一个文本串S，和一个模式串P，现在要判断S中是否有和P匹配的子串，并查找P在S中的位置，怎么解决呢？

**暴力解决法（bf）：**

如果用暴力匹配的思路，并假设现在文本串S匹配到 i 位置，模式串P匹配到 j 位置，则有：

如果当前字符匹配成功（即S[i] == P[j]），则i++，j++，继续匹配下一个字符；如果匹配失败（即S[i]! = P[j]），令i = i - j + 1，j = 0，即每次匹配失败时，i 回溯到上次开始匹配的下一个位置，j 被置为0。

```
package main

import (
    "fmt"
)

func main() {
    var s string = "abcdefg"
    var p string = "cde"
    index := violentMatch(s, p)
    fmt.Println(index)

}

//一个无序数组中两个数之和等于给定的值sum
func violentMatch(ss string, ps string) (index int) {
    s := []rune(ss)
    p := []rune(ps)
    var i, j int
    i = 0
    j = 0
    for i < len(s) && j < len(p) {
        //①如果当前字符匹配成功（即s[i]==p[j]），则i++，j++
        if s[i] == p[j] {
            i++
```

```
            j++
        } else {
            //②如果失败（即s[i]!=p[j]），令i=i-j+1，j=0
            i = i - j + 1
            j = 0
        }
    }

    if j == len(p) {
        index = i - j
    } else {
        index = -1
    }
    return
}
```

**********************************************

kmp算法：

**首先**，算法需要对模式串P进行预处理，得到一个部分匹配表kt：

```
    i  - 0 1 2 3 4 5 6
  P[i] - a b a b a c a
 kt[i] - 0 0 1 2 3 0 1
```

表中kt[i]等于字符串P[0-i]的前缀和后缀的最长的共有元素的长度。（例如p[0-3] = "abab"，它的前缀有” a “，” ab “，” aba “，后缀有” b “，” ab “，” bab “，共有元素为” ab “，其长度为2，所以kt[3]等于2）。

**然后，**每次开始从左向右匹配，因此此类算法隶属于基于前缀搜索的方法:

若第1位就匹配不成功，则 后移1位。

2.1）若前k位匹配成功，则  后移位数s = 已匹配的字符数k - 部分匹配表对应值kt[k-1]：

```
T - a b c b a b a b a a b c b a b
            | | | | | #
P -         a b a b a c a
-->2            a b a b a c a

∵ kt[5-1]=3
∴ s=5-3=2
```

```go
//KMP算法
package main

import (
    "fmt"
)

//对模式串P进行预处理，得到一个部分匹配表kt：
//表中kt[i]等于字符串P[0-i]的前缀和后缀的最长的共有元素的长度。
//例如p[0-3] = "abab"，它的前缀有”a“，”ab“，”aba“，后缀有”b“，”ab“，”bab“，共有元素为”ab“，其长度为2，所以kt[3]等于2
func PartialMatchTable(ss string) []int {
    p := []rune(ss)
    m := len(p)
    kt := make([]int, m)

    k := 0
    for i := 1; i < m; i++ {
        for k > 0 && p[k] != p[i] {
            k = kt[k]
        }
        if p[k] == p[i] {
            k = k + 1
        }
        kt[i] = k

    }
    return kt
```

```go
}

func Match(ss string, pp string) (index int) {
    index = -1
    s := []rune(ss)
    p := []rune(pp)
    m := len(s)
    n := len(p)
    kt := PartialMatchTable(pp)
    i := 0
    k := 0
    for i < (m - n) {
        for k < n && s[i+k] == p[k] {
            k++
        }
        if k == 0 {
            //若第1位就匹配不成功，则 后移1位。
            i++
        } else {
            if k == n {
                //匹配成功，返回i
                index = i
            }
            //若前k位匹配成功，则后移位数s = 已匹配的字符数k - 部分匹配表对应值kt[k-
1]：
            i = i + k - kt[k-1]

            k = kt[k-1]
        }
    }
    return
}

func main() {
    s := "abababaca"
```

```
    fmt.Printf("%v", Match(s, "bac"))
}
```

*********************************************************

**bm（摩尔算法）：**

```
HERE IS A SIMPLE EXAMPLE
EXAMPLE
```

首先，原字符串和子串左端对齐，但是从尾部开始比较，就是首先比较“S”和“E”，这是一个十分巧妙的做法，如果字符串不匹配的话，只需要这一次比较就可以确定。
在BM算法中，当每次发现当前字符不匹配的时候，我们就需要寻找子串中是否有这个字符；比如当前“S”和“E”不匹配，那我们需要寻找子串当中是否存在“S”。发现子串当中并不存在，那我们将子串整体向后移动到原字符串中“S”的下一个位置

```
HERE IS A SIMPLE EXAMPLE
       EXAMPLE
```

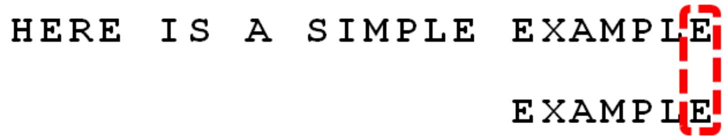
接着，从尾部开始比较，发现“P”和“E”不匹配，那我们查找一下子串当中是否存在“P”，发现存在，那我们就把子串移动到两个“P”对齐的位置：

```
HERE IS A SIMPLE EXAMPLE
         EXAMPLE
```

继续从尾部开始比较，“E”匹配，“L”匹配，“P”匹配，“M”匹配，“I”和“A”不匹配！那我们就接着寻找一下子串当前是否出现了原字符串中的字符，我们发现子串中第一个“E”和原字符串中的字符可以对应，那直接将子串移动到两个“E”对应的位置：

```
HERE IS A SIMPLE EXAMPLE
               EXAMPLE
```

接着从尾部比较，发现“P”和“E”不匹配，那么检查一下子串当中是否出现了“P”，发现存在，那么移动子串到两个“P”对应




从尾部开始，逐个匹配，发现全部能匹配上，匹配成功~
时间复杂度：最差情况O（MN），最好情况O（N）

首先先明确两个规则：坏字符规则、好后缀规则

1、坏字符规则

后移位数 = 坏字符的位置 - 模式串中的坏字符上一次出现位置


因为"P"与"E"不匹配，所以"P"被称为"坏字符"，它出现在模式串（模式串就是EXAMPLE）的第6位（从0开始编号），在模式串中的上一次出现位置为4，所以后移 6 - 4 = 2位

2、好后缀规则

后移位数 = 好后缀的位置 - 模式串中的上一次出现位置

举例来说，如果模式串"ABCDAB"的后一个"AB"是"好后缀"。那么它的位置是5（从0开始计算，取最后的"B"的值），在模式串中的上一次出现位置是1（第一个"B"的位置），所以后移 5 - 1 = 4位，前一个"AB"移到后一个"AB"的位置。
再举一个例子，如果模式串"ABCDEF"的"EF"是好后缀，则"EF"的位置是5 ，上一次出现的位置是 -1（即未出现），所以后移 5 - (-1) = 6位，即整个字符串移到"F"的后一位。