# [KMP算法](https://zh.wikipedia.org/wiki/%E5%85%8B%E5%8A%AA%E6%96%AF-%E8%8E%AB%E9%87%8C%E6%96%AF-%E6%99%AE%E6%8B%89%E7%89%B9%E7%AE%97%E6%B3%95)

KMP算法解决的问题是字符匹配，这个算法把字符匹配的时间复杂度缩小到O(m+n),而空间复杂度也只有O(m),n是target的长度，m是pattern的长度。

* 部分匹配表（Next数组）：表的作用是 **让算法无需多次匹配S中的任何字符**。能够实现线性时间搜索的关键是 **在不错过任何潜在匹配的情况下，我们"预搜索"这个模式串本身并将其译成一个包含所有可能失配的位置对应可以绕过最多无效字符的列表**。
* Next数组（前缀和前缀的比较）：t为模式串，j为下标
* Next[0] = -1
* Next[j] = MAX{ k | 0 < k < j | " t0 t1 ... tk " = "t ( j-k ) t ( j-k+1 ) ... t( j-1 )" }

|  |
| --- |
| i |
| t[i] |
| next[i] |

* NextVal数组：是一种优化后的Next数组，是为了解决类似aaaab这种模式串的匹配，减少重复的比较。 如果t[next[j]]=t[j]：nextval[j]=nextval[next[j]]，否则nextval[j]=next[j]。

|  |
| --- |
| i |
| t |
| next[j] |
| nextval[j] |

在上面的表格中，t[next[4]]=t[4]=b，所以nextval[4]=nextval[next[4]]=0