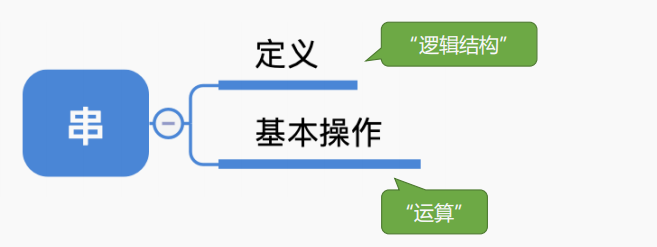
串定义、基本操作



**串**，即**字符串**（String）是由**零个或多个字符**组成的**有限序列**。

一般记为 **S** = ‘**a1a2······an**’ （n ≥0）

其中，S是**串名**，单引号括起来的字符序列是**串的值**；

ai 可以是**字母、数字或其他字符**；

串中字符的个数n称为**串的长度**。

**n = 0**时的串称为**空串**（用**∅**表示）。

**子串**：串中任意个**连续的**字符组成的子序列。

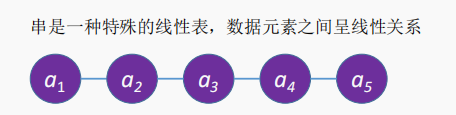
**主串**：包含子串的串。

**字符**在主串中的**位置**：字符在串中的**序号**(位序**从1开始**而不是从0开始)。

**子串**在主串中的**位置**：子串的**第一个字符**在主串中的位置 。

串和线性表的关系：

串是一种特殊的线性表，数据元素之间呈线性关系。



串的**数据对象**限定为**字符集**（如中文字符、英文字符、数字字符、标点字符等）

串的基本操作，如增删改查等通常以**子串**为**操作对象**

串的基本操作：

StrAssign(&T,chars)：**赋值**操作。把串T赋值为chars。

StrCopy(&T,S)：**复制**操作。由串S复制得到串T。

StrEmpty(S)：**判空**操作。若S为空串，则返回TRUE，否则返回FALSE。

StrLength(S)：**求串长**。返回串S的元素个数。

ClearString(&S)：**清空**操作。将S清为空串。

DestroyString(&S)：**销毁串**。将串S销毁（回收存储空间）。

**Concat(&T,S1,S2)**：**串联接**。用T返回由S1和S2联接而成的新串

**SubString(&Sub,S,pos,len)**：**求子串**。用Sub返回串S的第pos个字符起长度为len的子串。

**Index(S,T)**：**定位操作**。若主串S中存在与串T值相同的子串，则返回它在主串S中第一次出现的位置；否则函数值为0。

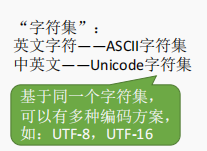
**StrCompare(S,T)**：**比较**操作。若S>T，则返回值>0；若S=T，则返回值=0；若S<T，则返回值<0。（从第一个字符开始往后依次对比，先出现更大字符的串就更大）

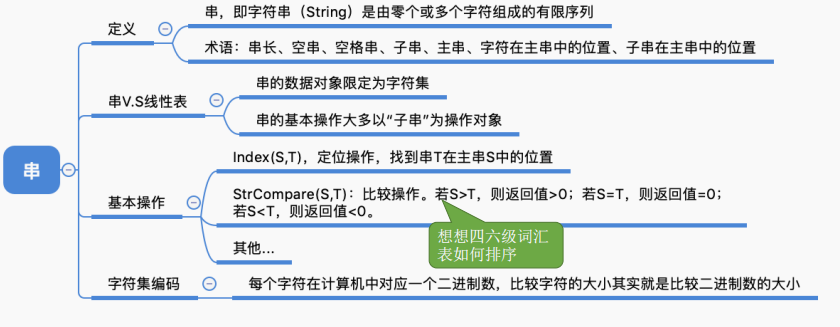
字符集编码：y = f(x)

字符集：函数定义域

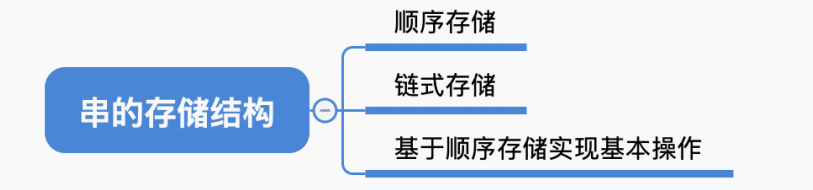
编码：函数映射规则f

y：对应的二进制数

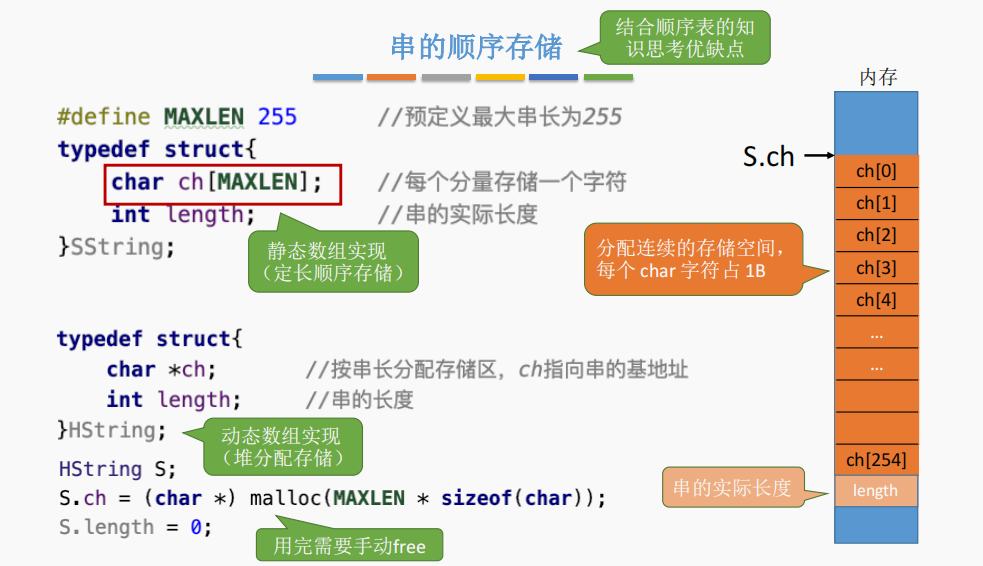




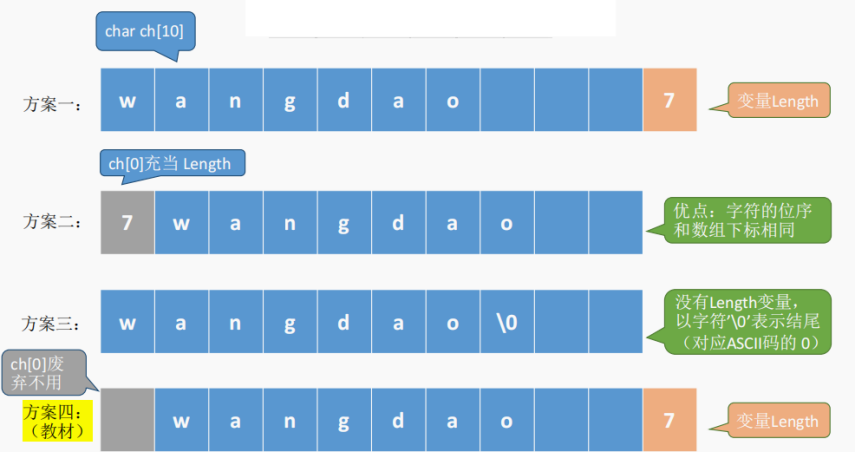
串存储结构



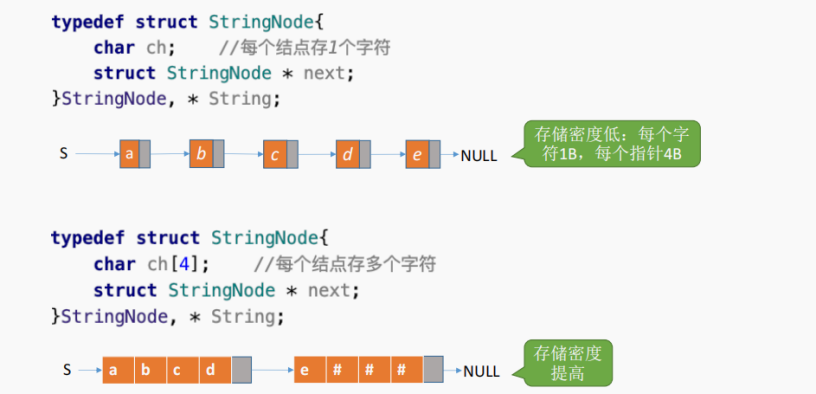
串的顺序存储（动态分配和静态分配）：



串的顺序存储--优化：



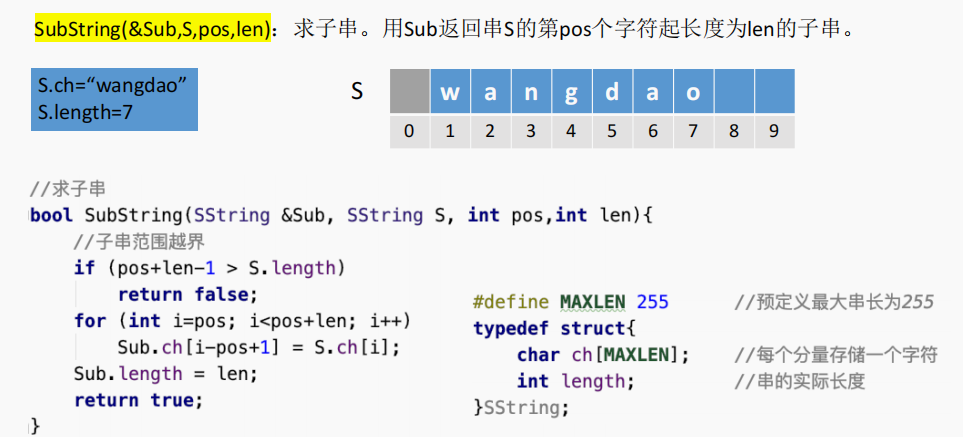
串的链式存储：



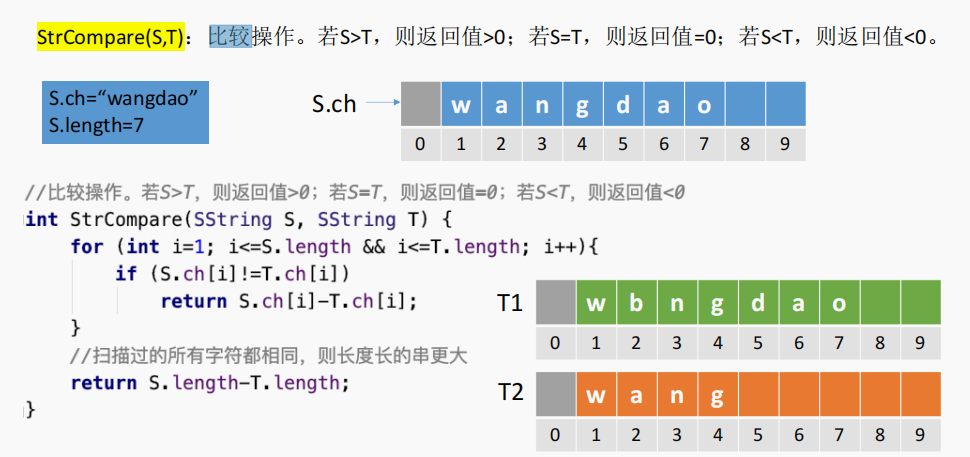
基本操作的实现:



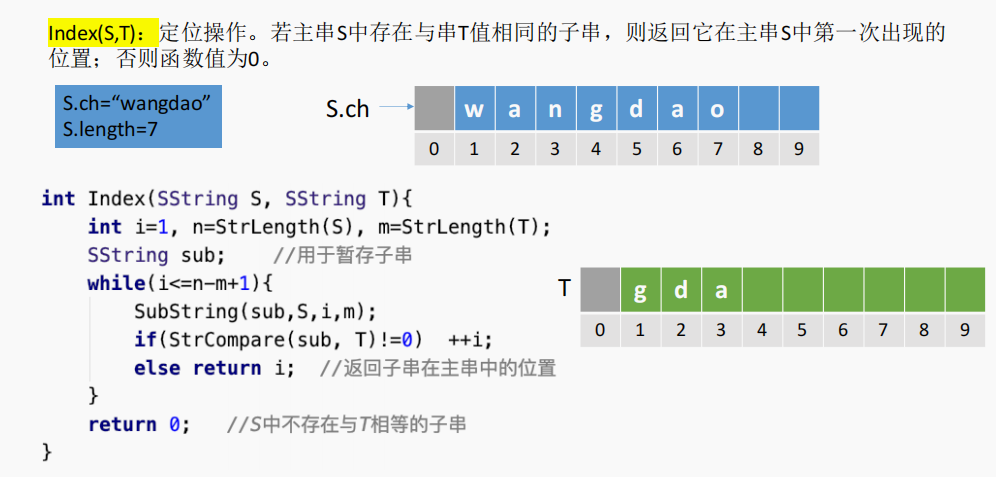
1. 求子串--SubString(&Sub,S,pos,len)：

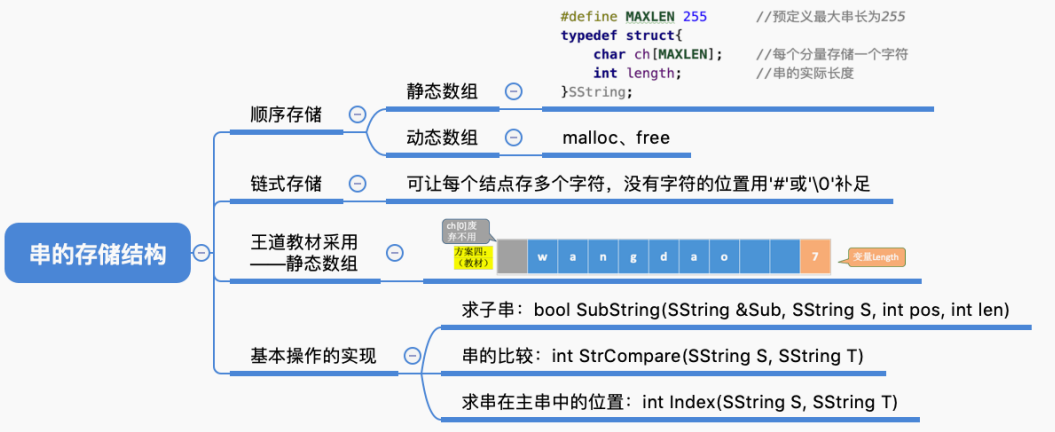


1. 比较--StrCompare():



1. 定位--Index(S,T)：



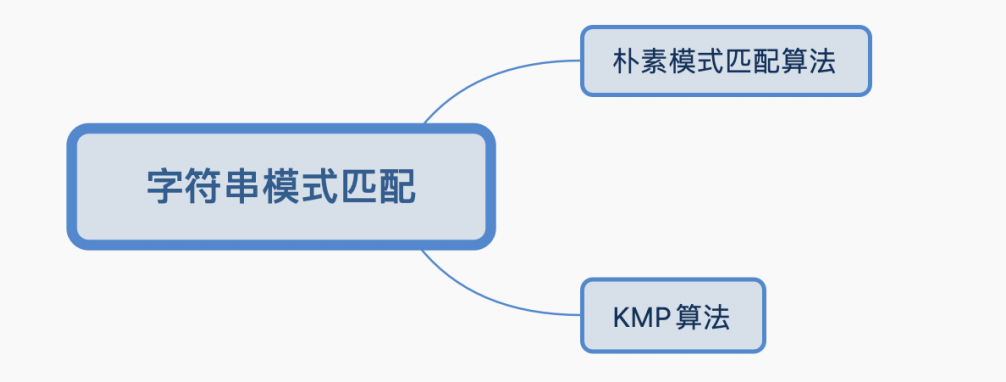


朴素模式匹配算法

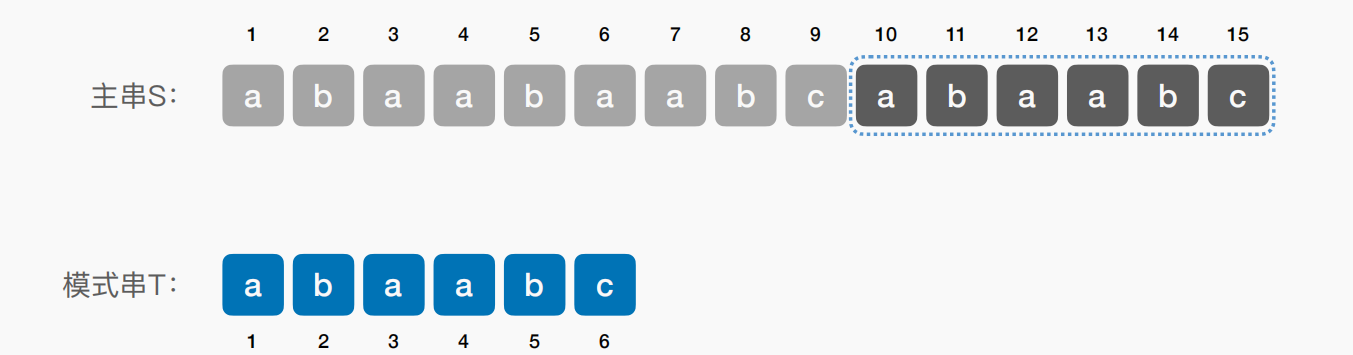
**字符串模式匹配**：在主串中找到与模式串相同的⼦串，并返回其所在位置。

与子串的区别：⼦串——主串的⼀部分，**⼀定存在**

模式串——**不⼀定**能在主串中找到



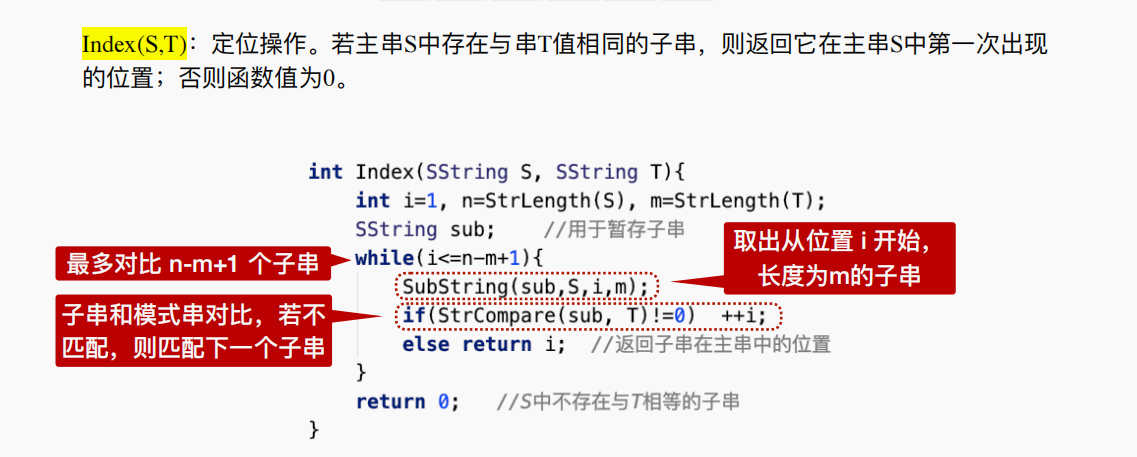
**朴素模式匹配算法**：





算法实现：

（1）方法一--指针



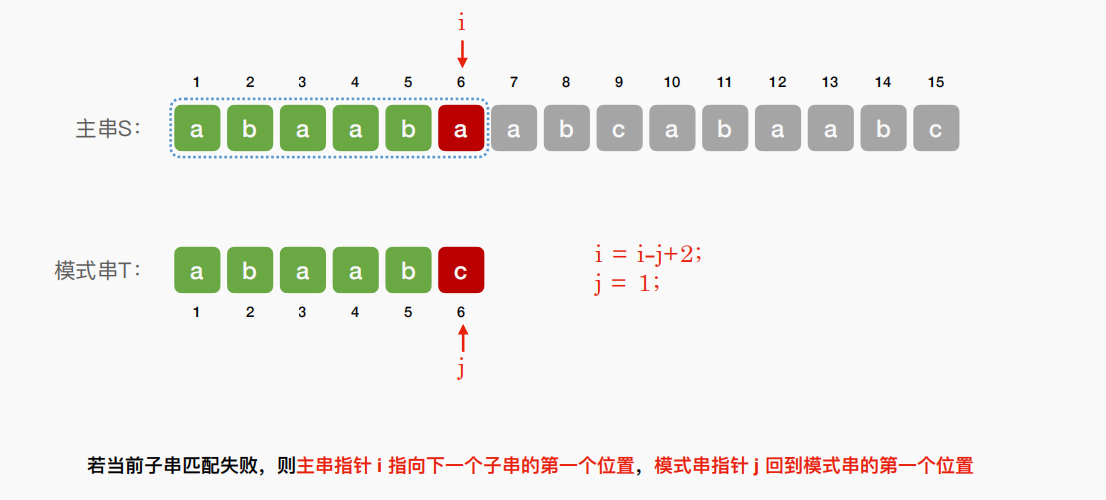
最多对⽐**n-m+1**个⼦串

取出从位置 i 开始，⻓度为m的⼦串

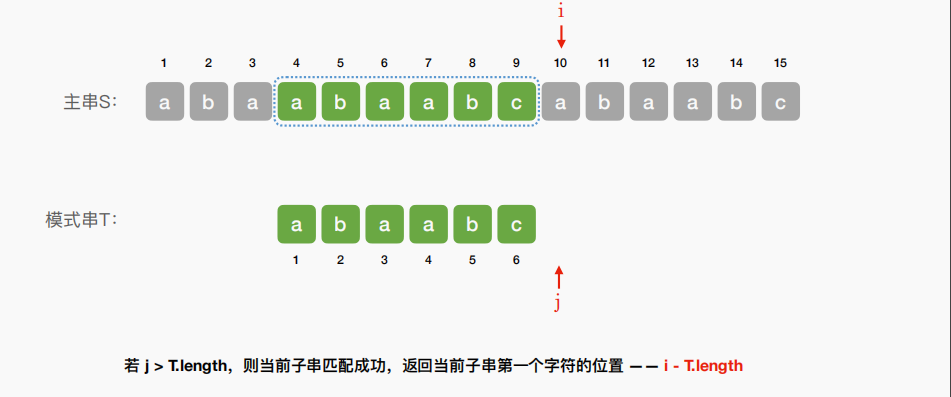
⼦串和模式串**对⽐**，若**不匹配**，则**匹配下⼀个⼦串**

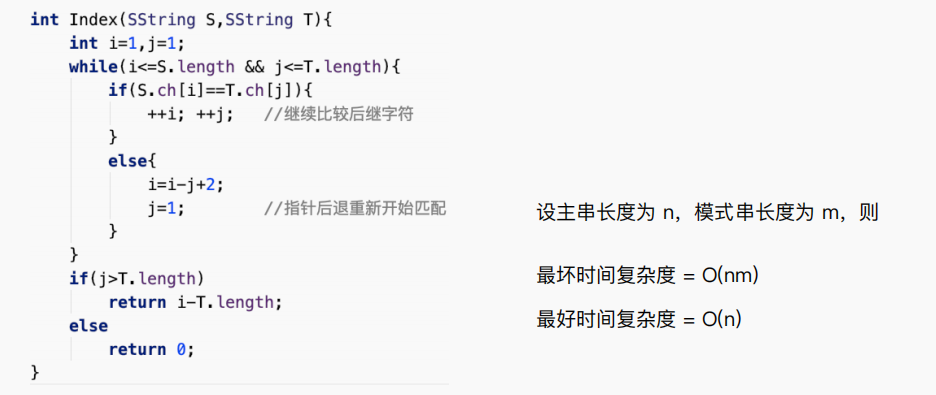
通过数组下标实现朴素模式匹配算法：

（2）方法二--指针：



若当前⼦串**匹配失败**，则**主串指针i指向下⼀个⼦串的第⼀个位置，模式串指针j回到模式串的第⼀个位置**



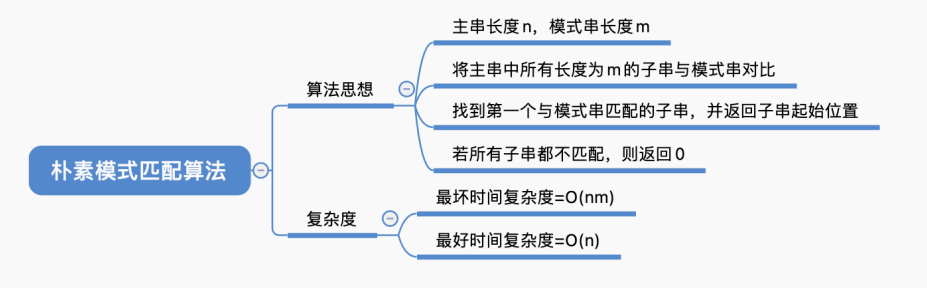


时间复杂度最坏=O(nm)

最好=O(n)：

最好的情况：每个⼦串的第⼀个字符就匹配失败，共 n-m+1 个⼦串

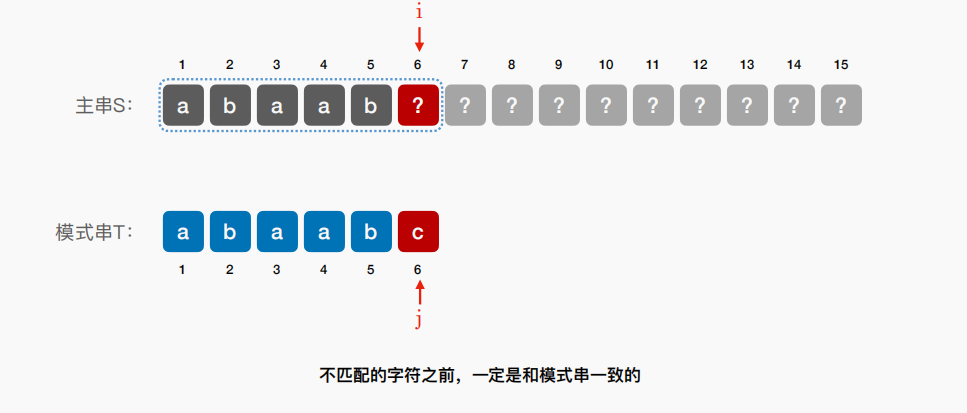
复杂度 = O(n-m+1) = O(n)

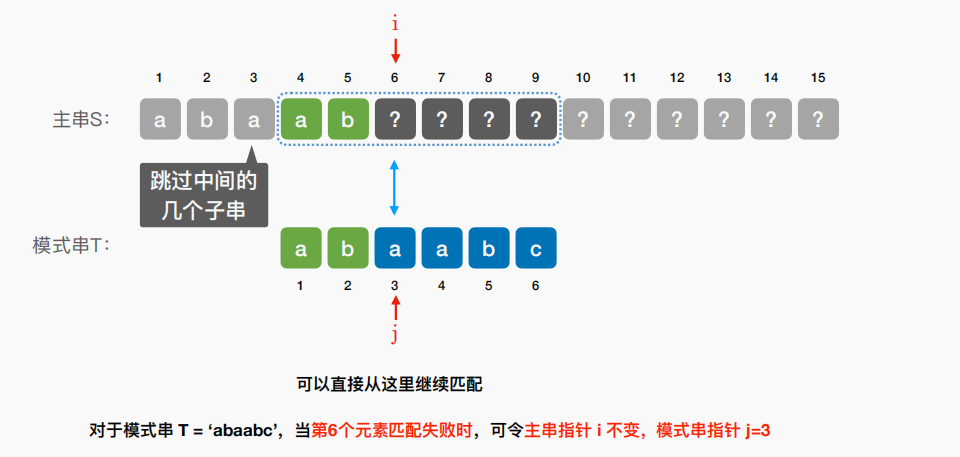


**KMP算法**：

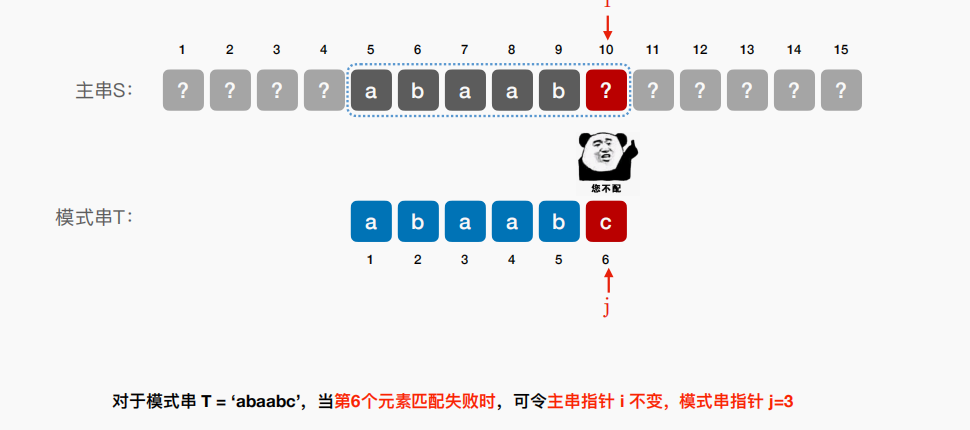
朴素模式匹配算法：⼀旦发现当前这个**⼦串中某个字符不匹配**，就只能转⽽**匹配下⼀个⼦串**（**从头开始**）

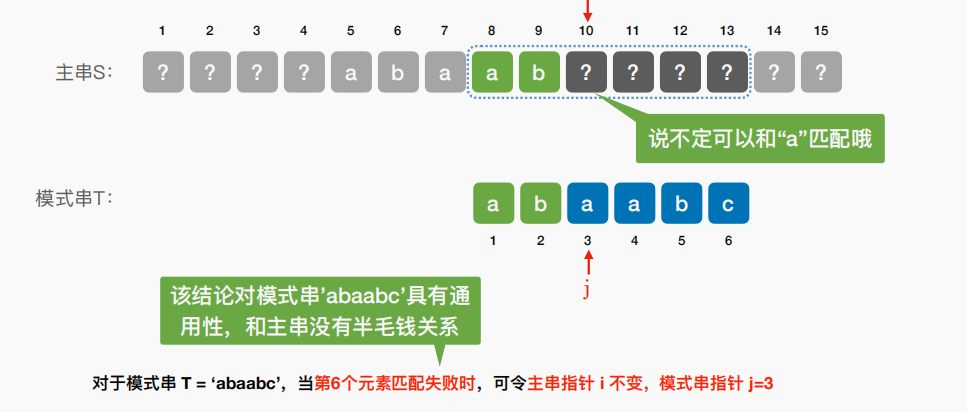
优化：**不匹配的字符之前，⼀定是和模式串⼀致的**



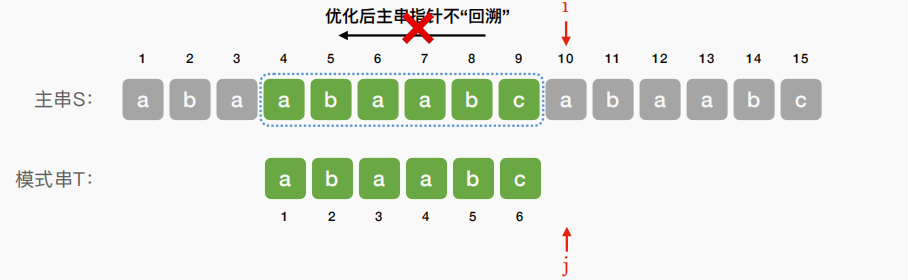


朴素模式匹配算法优化思路：





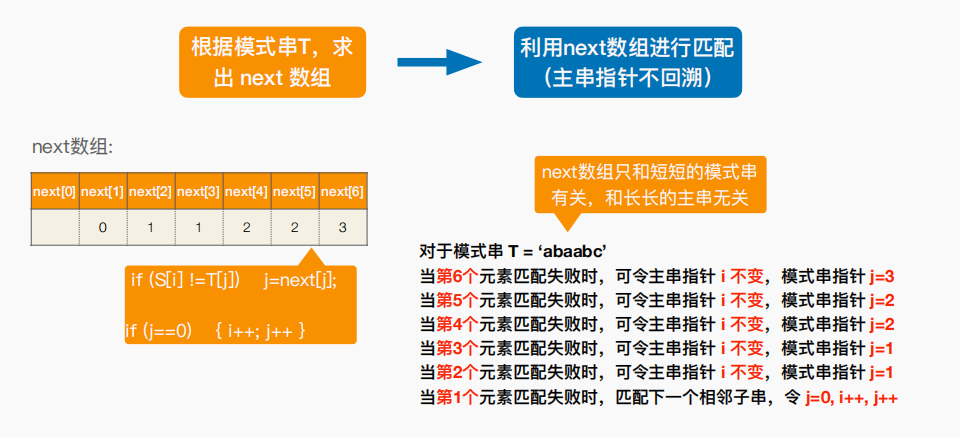


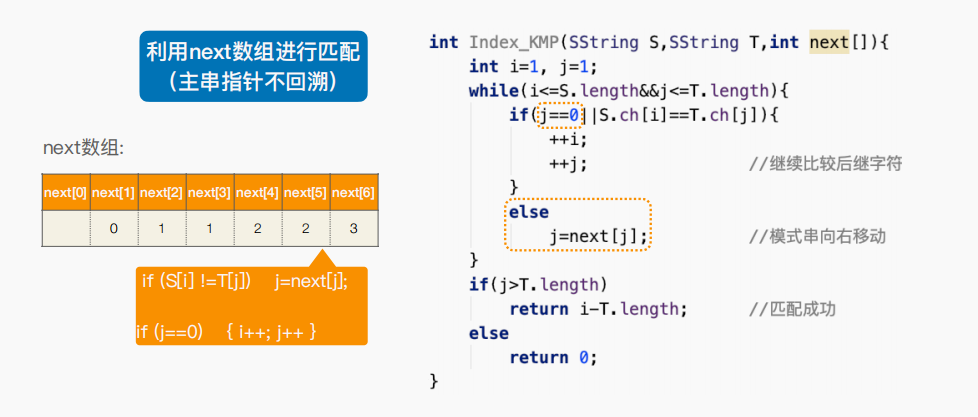


怎么⽤代码实现这个处理逻辑？

next数组只和短短的模式串有关，和⻓⻓的主串⽆关

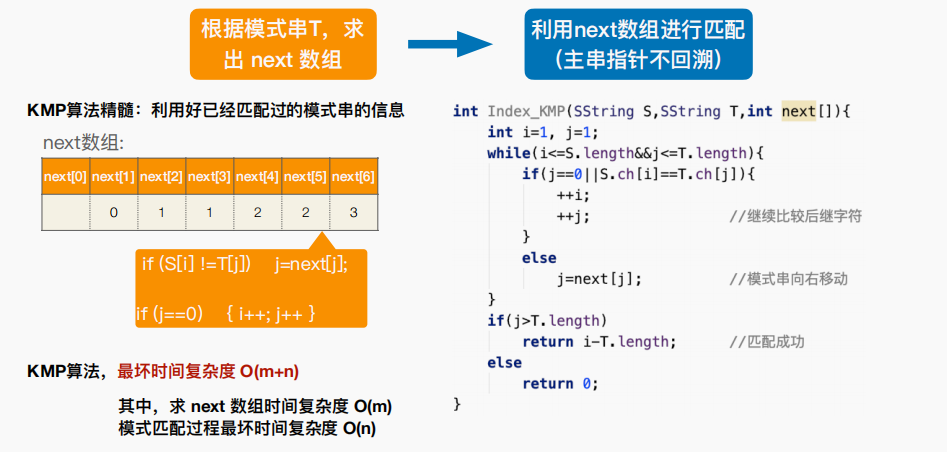








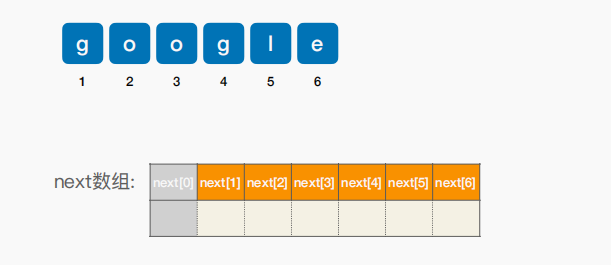
最坏时间复杂度：**O（m+n）**



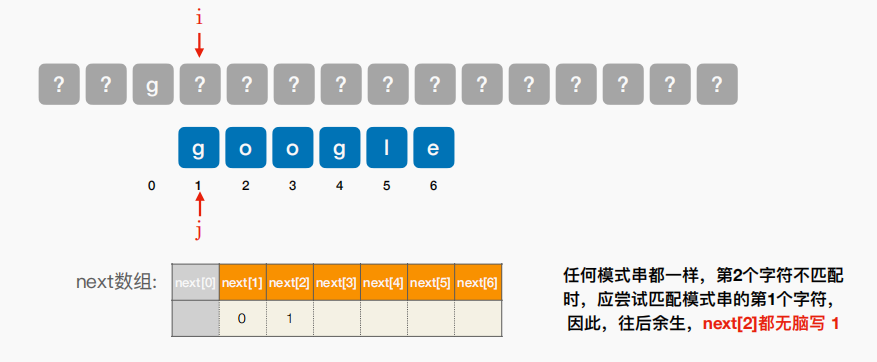
KMP算法求next数组

求模式串的next数组（⼿算练习）:

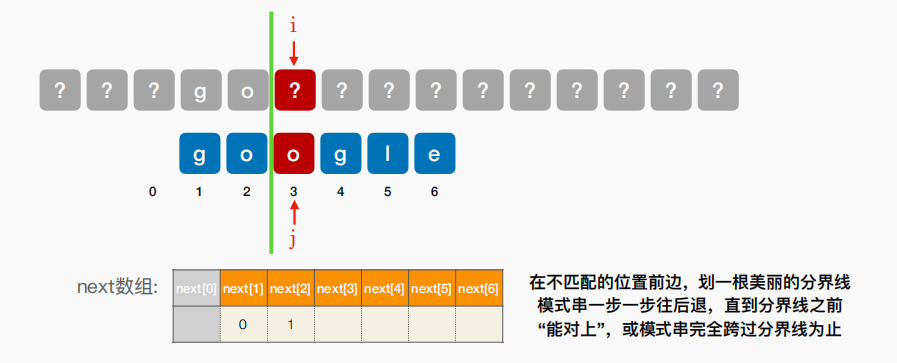
next数组的作⽤：当模式串的第 j 个字符失配时，从模式串的第 next[j] 的继续往后匹配



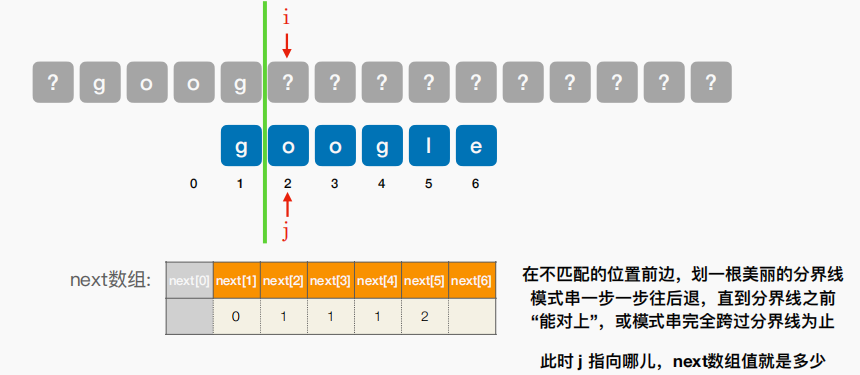


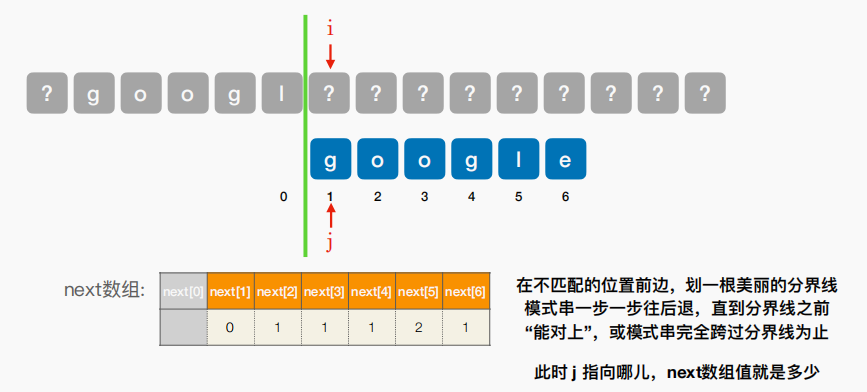


在不匹配的位置前边，划⼀根美丽的**分界线**，**模式串**⼀步⼀步往**后退**，直到**分界线之前**“**能对上**”，或**模式串完全跨过分界线为⽌**，此时 **j 指向哪⼉，next数组值**就是多少

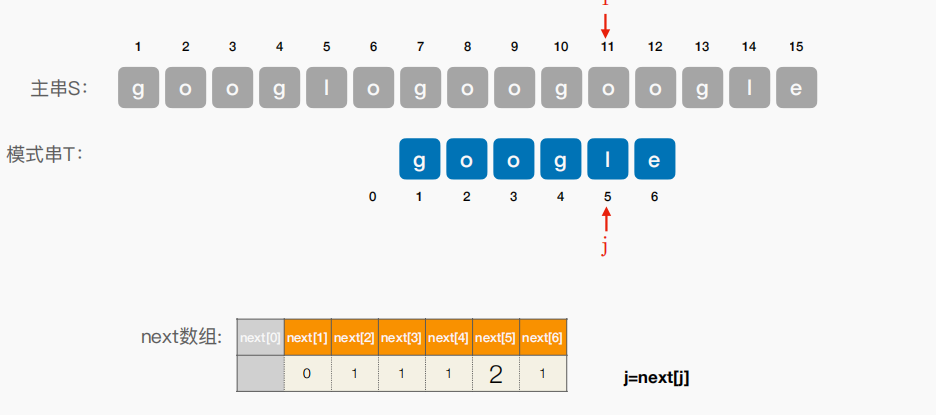


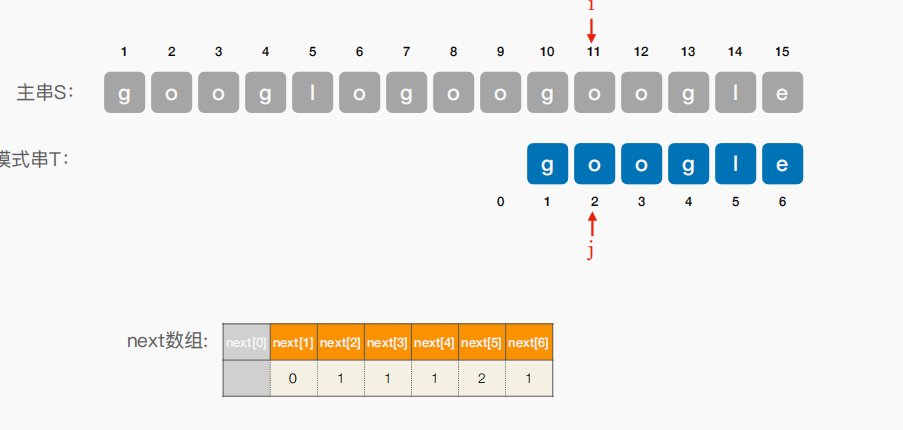


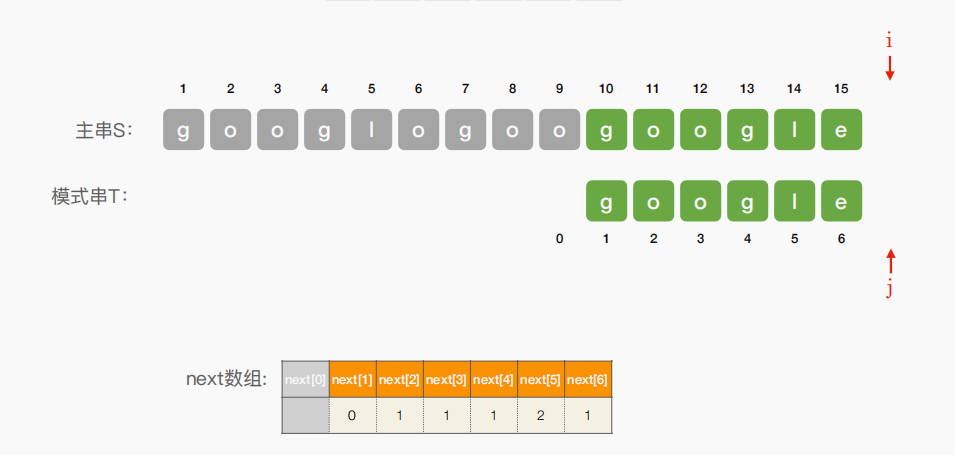


使⽤next数组进⾏模式匹配：

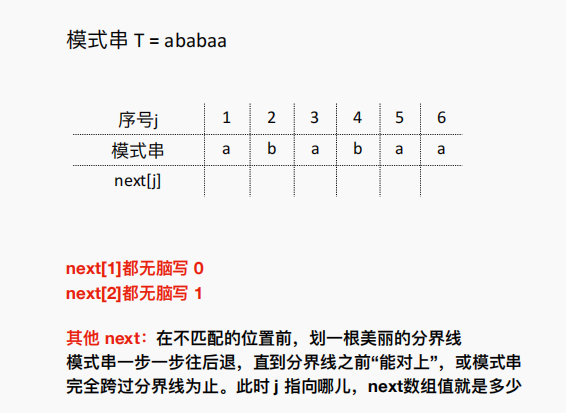
求模式串的next数组（⼿算练习）：

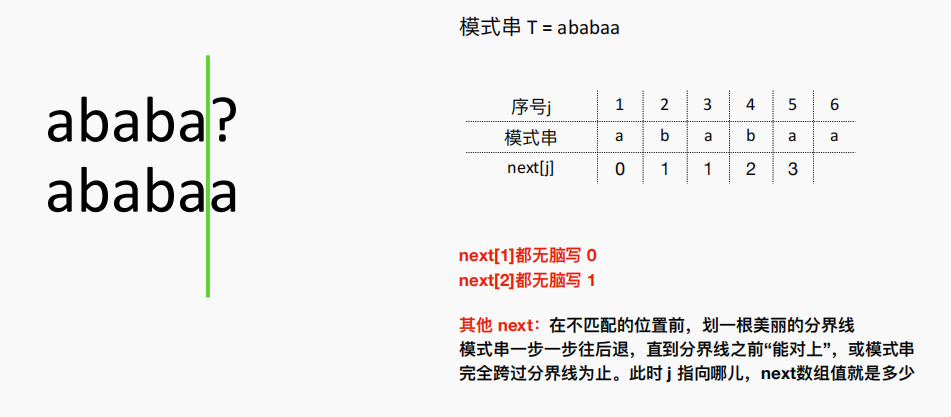






求模式串的next数组（⼿算练习）





总结：

