|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | A | B | D | A | B | A | B | A | E |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  |  |  |  |  |  |  |  |  |

NEXT数组：该值表示当前位置前面字符串的最长相同真前缀和真后缀长度

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | A | B | D | A | B | A | B | A | E |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 0 | 0 | 1 | 2 | 0 | 1 | 2 | 3 | 4 | 3 |

NONE:0

**A**:0

**AB**:0前缀：A；后缀：B

**ABA**：1前缀：A,AB；后缀：BA,A

**ABAB**：2前缀：A,AB,ABA；后缀：BAB,AB,B

**ABABD**：0前缀：A,AB,ABA,ABAB；后缀：BABD,ABD,BD,D

**ABABDA**：1前缀：A,AB,ABA,ABAB,ABABD；后缀：BABDA,ABDA,BDA,DA,A

**ABABDAB**：2前缀：ABABDA,ABABD,ABAB,ABA,AB,A；后缀：BABDAB,ABDAB,BDAB,DAB,AB,A

**ABABDABA**：3前缀：ABABDAB,ABABDA,ABABD,ABAB,ABA,AB,A；

后缀：BABDABA,ABDABA,BDABA,DABA,ABA,BA,A

**ABABDABAB**：4

前缀：ABABDABA,ABABDAB,ABABDA,ABABD,ABAB,ABA,AB,A；

后缀：BABDABAB,ABDABAB,BDABAB,DABAB,ABAB,BAB,AB,B

**ABABDABABA**：3

前缀：

ABABDABAB,ABABDABA,ABABDAB,ABABDA,ABABD,ABAB,ABA,AB,A

后缀：

BABDABABA,ABDABABA,BDABABA,DABABA,ABABA,BABA,ABA,BA,A

**ABABDABABAE**：0

前缀：

ABABDABABA,ABABDABAB,ABABDABA,ABABDAB,ABABDA,ABABD,ABAB, ABA,AB,A

后缀：

BABDABABAE,ABDABABAE,BDABABAE,DABABAE,ABABAE,BABAE,ABAE,BAE,AE,E

NEXT数组存的不是这个位置的最长的相同的真前后缀长度，它存的是这个位置前面的字符串的最长的相同的真前后缀长度。所以看表格，最后E的位置，这时候的字符串是ABABDABABAE其实根本没相同的真前后缀。

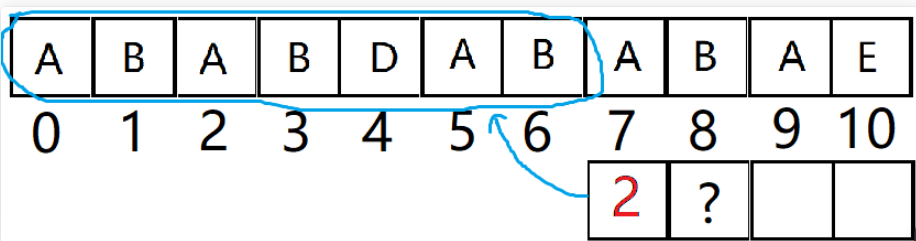
这个3指的是E前面的ABABDABABA最长的相同的真前后缀长度（即ABA）

NEXT数组电脑是怎么求出来呢？仔细想想就会发现，这个NEXT数组其实是一个个按顺序求值的。虽然不知道NEXT[8]对应的NEXT数值是多少，但是前面是已知的，能不能根据前面已知的next数组值来推导？

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | A | B | D | A | B | A | B | A | E |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 0 | 0 | 1 | 2 | 0 | 1 | 2 | 3 | 4 | 3 |

注意到，NEXT[7] = 2, 这并不是指array[7]对应位置的字符串最长的相同的真前后缀长度。前面已经说过这个2指的是array[7]对应位置前面字符相同真前后缀，并不包括array[7]该位置本身。实际上，array[7]是A，对NEXT[7]有影响吗？根本没有。把array[7]换成K这个位置NEXT[7]也是2，这是由前面的字符串决定的，跟array[7]的字符是啥无关。

显然，这个字符A决定的是后面next[8]的值。也就是说，要知道next[8]的值，只需要看array[7]和next[7]即可。



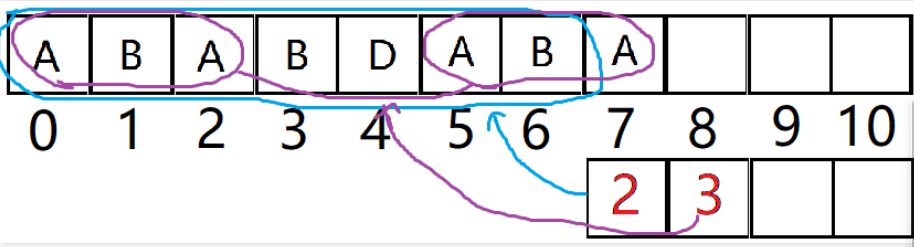
我们再来看这个2，它指的该位置前面的字符串最长的相同的真前后缀长度为2。也就是ABABDAB这个字符串，它们前面两个字符是AB，最后两个字符也是AB。

接下来我们知道，数组都是从0开始计数的。

那么其实array[next[7]] = array[2] = 原数组的第三个。回到原来的例子，我们知道next[7] = 2，也就是array[7]前面两个（array[5]和array[6]）和字符串开头（array[0]和array[1]）两个相同。

显然，如果array[7] = array[2] = A，

那么array[8]前面三个（array[5],array[6],array[7]）和字符串开头（array[0],array[1],array[2]）三个相同。



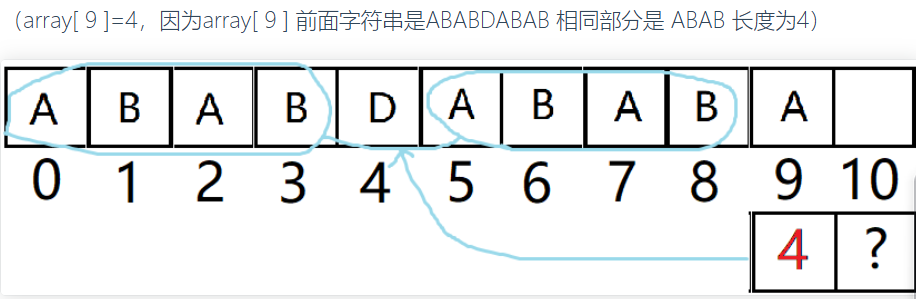
得到结论，如果array[next[m]] = array[m]，那么next[m+1] = next[m] + 1；

比如说next[m] = k，意味着m位置前面字符串的前k个和后k个相同（这里的后k个也就是m位置前k个）。又由于数组从0计数，所以说array[next[m]] = array[k]，指的是数组的第k+1个元素，也就是***第一个不是前缀的位置****。*

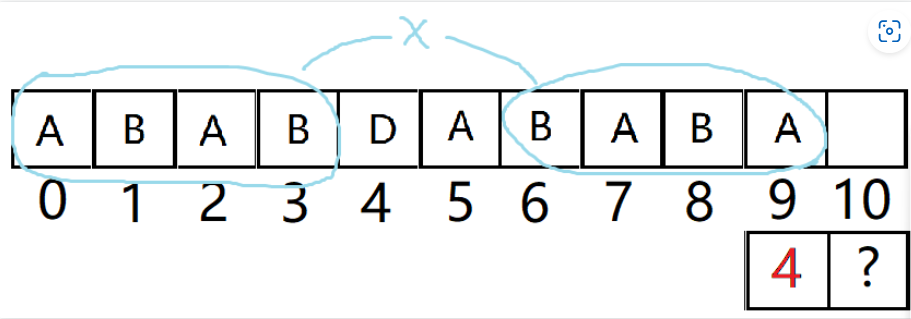
现在m位置前字符串的前k个和后k个相同，而array[]数组的第k+1个元素又等于array[m]，那么m+1个位置的字符串的前k+1个和后k+1个相同。

**接下俩讨论array[next[m]] != array[m]的情况：**

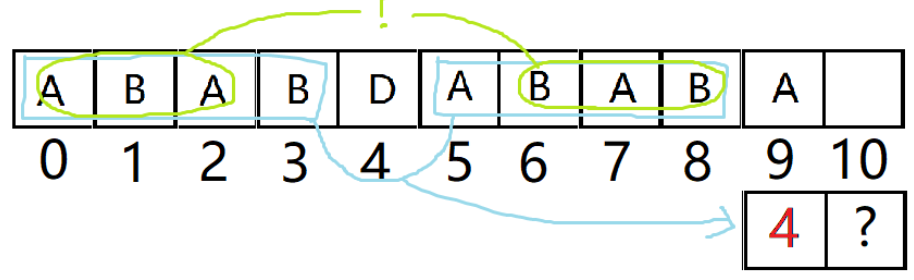
还是这个例子，next[10]应该是多少？



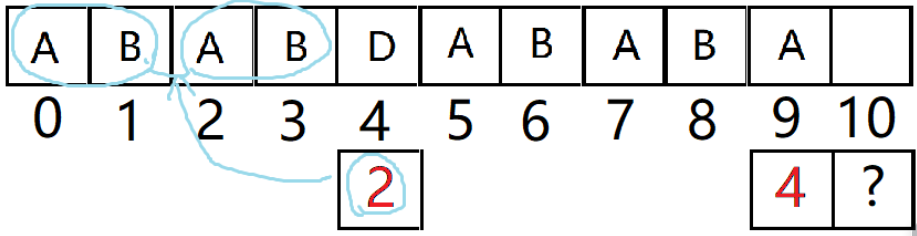
我们之所以能通过只比较两个位置的值等不等就能确定最长前后缀，是因为前面已经求出来相同的地方了。我们只要看array[4]和array[9]不同，是因为前面四个肯定相同（因为next[9] = 4），这样的话，取长度为5的前后缀不同，那么***短一点的前后缀有可能会相同***。我们看看取4个的情况：



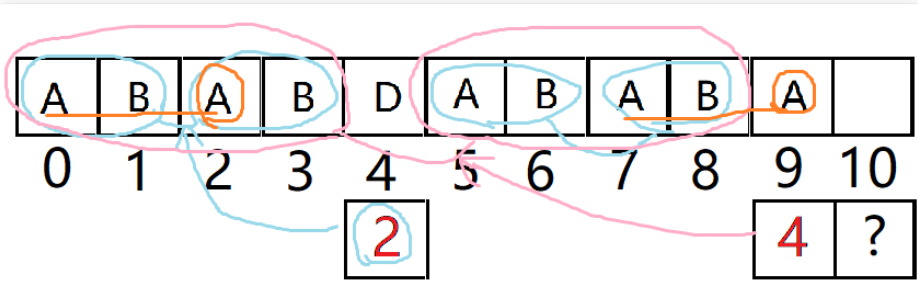
假如4长度的前后缀相同的话，就意味着array[0~3] = array[6~9]，也就意味着array[0~2] = array[6~8]（这里说的跟最长相等前后缀数无关，指的是如果4长度的前后缀相同的话，那么这前后缀的前面三个字符必然也是相同的）



如图，只有绿圈里面相同，我们才有必要比较array[3]是不是等于array[9]，问题是我们怎么知道，或者说电脑怎么知道绿圈里相不相同？我们仔细想一下，蓝圈是已知的next[9] = 4而相同的前后缀，所以前后两个蓝圈的内容是完全相同的。如果绿圈也相同，那不是代表绿圈是蓝色圈中字符串的相同前后缀？而且我们可以知道蓝圈内的相同前后缀是什么，

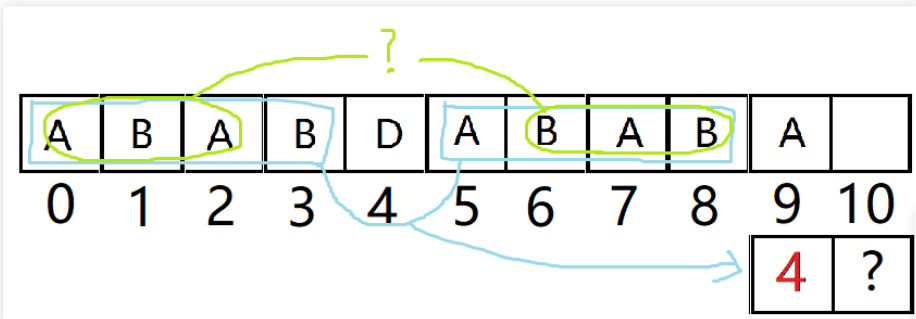


next[4]位置存储的就是前面四个字符的最长相同前后缀。

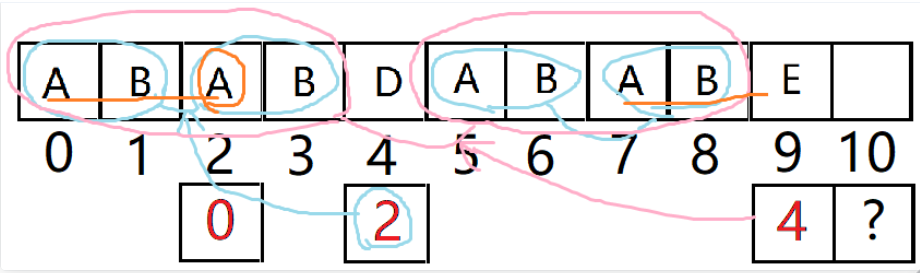


我们看next[9] = 4，就是粉色圈子的两个字符串相同，而next[4] = 2，就是蓝色圈子里的部分，这四个部分都相同。那么现在看橙色部分，只要比较array[9] 与array[2]（因为array[0~1]和array[7~8]肯定相同了），比较发现确实array[9] = array[2]，那么得出结论array[10] = 3；

分析一下，next[9] = 4就表示前九个字符中前四个和后四个字符完全相同，那么，next[next[9]] = next[4]，也就是第一个不属于前缀的位置，而next[4]代表是该位置前面字符串，就是之前的前缀，next[4] = 2，也就是之前的前缀的相同前后缀长度。这也就是为什么上面第一种假设肯定不行了，因为next[next[9]] = next[4] = 2，取前三个前缀后缀肯定不等（如果相等，那么应该next[4]=3）。因此，第一种情况中array[9]和array[3]就没有比较的必要了。



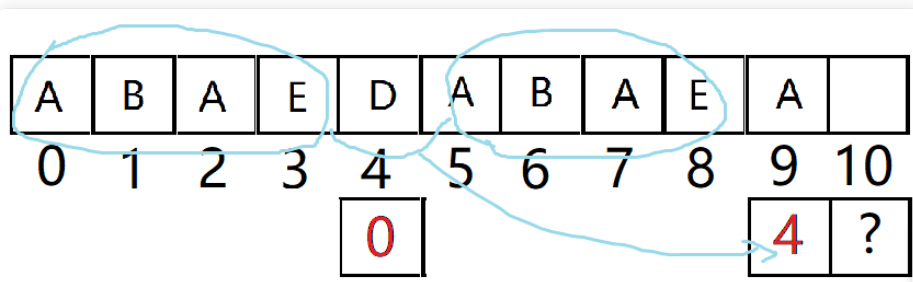
**那如果一直不等呢？（将array[9]改为E）**

****

明显array[2]不等于array[9]，那么继续按照之前的思路计算：next[next[next[9]]] = next[next[4]] = next[2] = 0，比较array[9]和array[0]，还是不等，但是next[0] = 0，继续嵌套还是next[0]，所以将next[0] = -1，这样当检测到-1的时候就没必要嵌套了，该位置就不存在相同前后缀了。

比如上面的例子next[next[next[next[9]]]] = next[next[next[4]]] = next[next[2]] = next[0] = -1

**再举个例子（将之前的array[3]和array[8]改成E）**

****

**那array[9]等于array[next[9]]吗？**

**（等于的话，next[10] = next[9]+1）**

**答：**

**不等于，array[next[9]] = array[4] 是D，而array[9]是A**

**继续比较array[9]与array[next[next[9]]] = array[next[4]] = array[0]**

**array[0]和array[9]都是A，所以next[10] = next[4] + 1 = 1**

**综上所述：**

**如果array[next[m]] = array[m] 则next[m+1] = next[m] + 1**

**如果 array[next[m]] != array[m] 则**

**令t = next[m]**

**比较array[t]和array[next[t]]**

**如果相等，则next[m] = next[t] + 1**

**如果不相等，继续令t = next[t]迭代，直到当t = 0时，此时next[t] = -1，停止迭代**

**备注：**

**对于kmp算法，所有资料给出了next数组的两种不同的解释，但是终其目标是一致的。有的next数组给出的值是该字符串的最长的相同前后缀长度，然后将结果后移一位，最前面一位补“-1”；还有一种解释是，next数组给出的是该字符之前的字符串的最长的相同前后缀长度，且next[0]为-1。道理是一样的。具体操作也是一样的。**

**对于为什么next数组的第一位是-1而不是0呢，在做完整个算法的运算之后，会发现，当不满足array[m] = array[next[m]]的时候，将next[m]第一次迭代为t，若仍不满足，继续将next[t]迭代为t，最终结果会趋于next[0]值，但若next[0] = 0，则程序会一直将t = next[0]迭代下去，所以，将next[0]的值设置为-1，迭代到此时，直接终止，并返回该字符（串）最长的相同前后缀长度为0.**