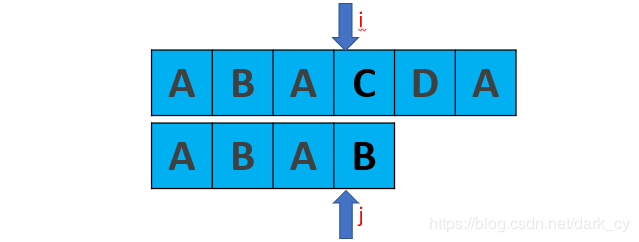
KMP算法

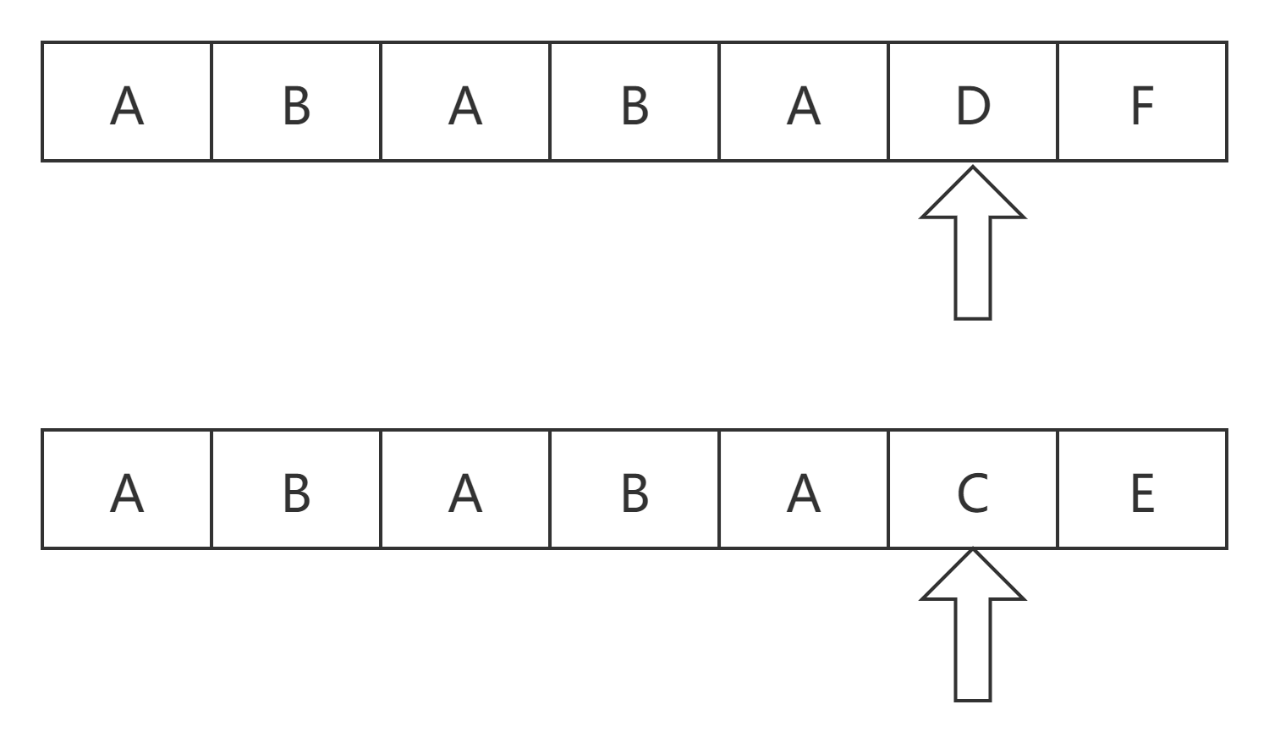
1. 概述

KMP算法由D.E.Knuth，J.H.Morris和V.R.Pratt提出，相比于BF（暴力）算法，它减少了无效的字符串回溯造成的浪费。

1. 原理

如上图，BF算法的话下一步应该是将i移动到i[2]，j移动到j[1]，然后进行比较，但是显然两者不等；继续向下，i移动到i[3]，j移动到j[1]，进行比较，此时两者相同。

那么，在刚刚这个过程中，实际上是存在一个无效的比较的，就是j[1]和i[2]的比较，我们已经提前知道i[2]是和j[2]相同的，而j[2]和j[1]是不一样的，所以j[1]和i[2]必然不一样，所以这一步比较是无效的，而KMP算法就是为了解决这种无效的比较。

具体怎么样呢，我们还是在原来的例子上探讨。可以看到，在j[4]和i[4]不匹配之后，我们看一下j[4]前面的字符串，发现它的前缀j[1]和后缀j[3]是一样的，我们可以直接将j[1]移动到i[3]的位置匹配，那么j就可以只移动到j[2],而i则不需要移动，减少了回溯。但是，为什么呢？为什么直接这样移动不会遗漏呢？这里其实是因为这个j[4]前的字符串的前后缀已经包含了这种遗漏的可能了，换而言之，如果有遗漏，一定是求前后缀时出了问题，前后缀的长度足以容纳这种遗漏。

如图，此时i指向D，j指向C，下一步应该怎么移动呢，我们注意到前面这个字符串的前缀应该为ABA，后缀为ABA，长度为3，所以j应该指向j[4]，由j[4]和D进行比较，而遗漏的话肯定是只考虑了AB这个前缀，将AB和上面的AB对齐，而显然这不是next的值。

1. 具体实现

到这里我们基本的框架已经有了，就是从头开始比较，遇到不一样的就把j指针回溯，用回溯后的值来比较，那么问题的关键就在如何确定这个回溯的位置了，这里我们首先明确这个回溯其实和i字符串没有关系，只和j本身有关，我们使用next[N]数组来记录，现在介绍如何求next数组

首先，第一个字符前面没有字符串，所以我们特殊规定其next值为-1，然后现在我们开始一般化的求next，这里用到了递归的思想。首先我们使用i，j两个指针来指向字符串中的字符，从next[2]开始，显然next[2] = 0；next[3],在前面有两个元素，分别令i=1，j=2，比较两个元素是否相同即可，是的话显然这里next[3]=1，注意，1同时也可以表示成0+1，而j刚好又是0；接下来我们求next[4]，注意此时我们是在next[3]的基础上的，我们如果将i，j都后移一位，如果元素相同，就说明了这个next的值是2了，为什么呢，因为我们前面已经比较了一个元素了，发现他们相等，那后移一位，又是相等，不就说明了一个长度为2的字符串是相等了的吗。而且，我们还注意到，实际上这个j的序号就是指1~j和后面的i-j~i号元素是相等的了，因此我们可以得出第一个结论，next[++i] = ++j。那么如果不相等呢，不相等的话，就意味着1~j这个长度下，不再可能和后面的字符串相等了，必须要想办法摆脱j的影响（毕竟j所指的元素不等于i所指的元素，只有把j往前挪才有可能找到和后缀相等的字符串。那么应该前移多少呢？这里有一个神仙操作，j = next[j]，什么意思呢？注意，字符串A（1~j-1）和字符串B（i-j~i-1）还是一样的呀（不然i，j到不了那里），所以A的前缀=A的后缀=B的前缀=B的后缀，我们将j移到next[j]的位置实际上就使得移动后j指向的元素的前面字符串和i前面的字符串还是一样的，这句话就是这个意思。所以求next实际上是有一个递归的意思在里面的。