串

KMP算法：再改进

13-C6

有顏囯者好學，不遷怒，不貳過

错误和挫折教训了我们，使我们比较地聪明起来了，我们的事情就办得好一些。任何政党，任何个人，错误总是难免的，我们要求犯得少一点。犯了错误则要求改正，改正得越迅速，越彻底，越好

邓俊辉

deng@tsinghua.edu.cn

反例

❖ 在T[3]处

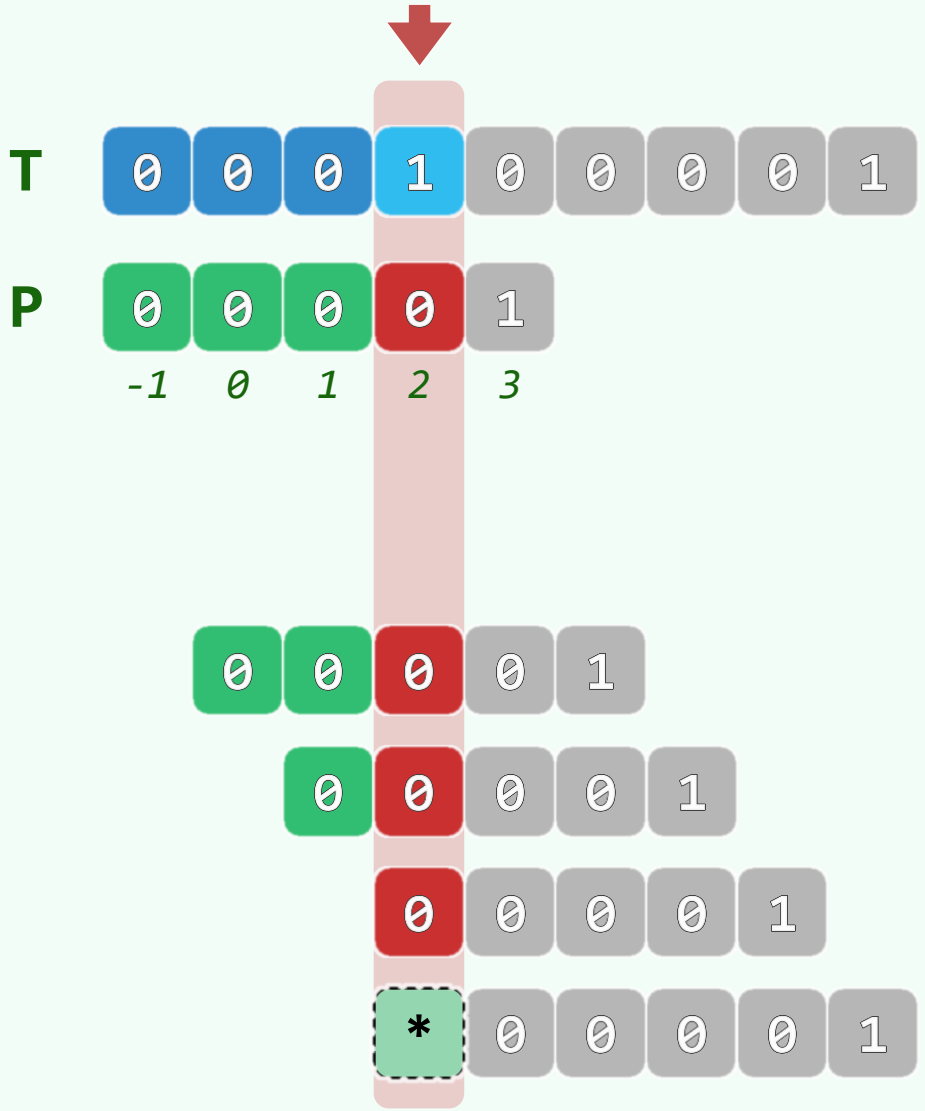
又：与 P[3] 比对，失败

双：与 P[2] = P[next[3]] 比对，失败

叒：与 P[1] = P[next[2]] 比对，失败

叕：与 P[0] = P[next[1]] 比对，失败

最终，才前进到T[4]



根源

❖ 无需T串，即可在事先确定：

P[3] =

P[2] =

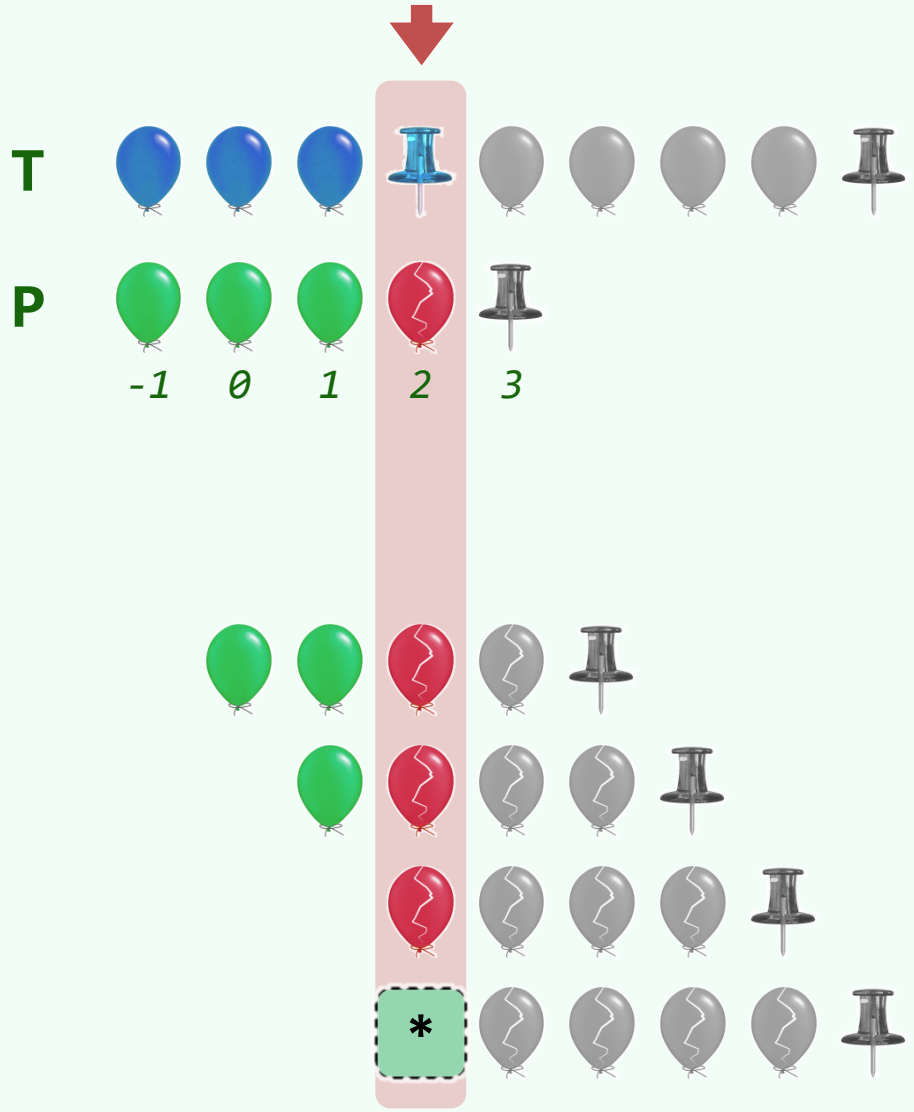
P[1] =

P[0] = 0

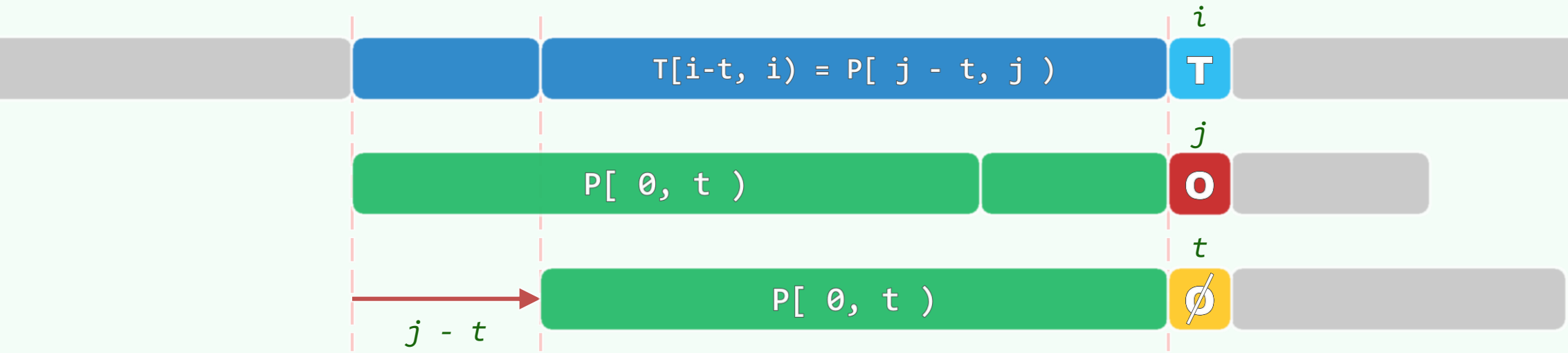
既然如此...

❖ 在发现 T[3] != P[3] 之后，为何还要一错再错?

❖ 事实上，后三次比对本来都是可以避免的!



改进：经验 + 教训



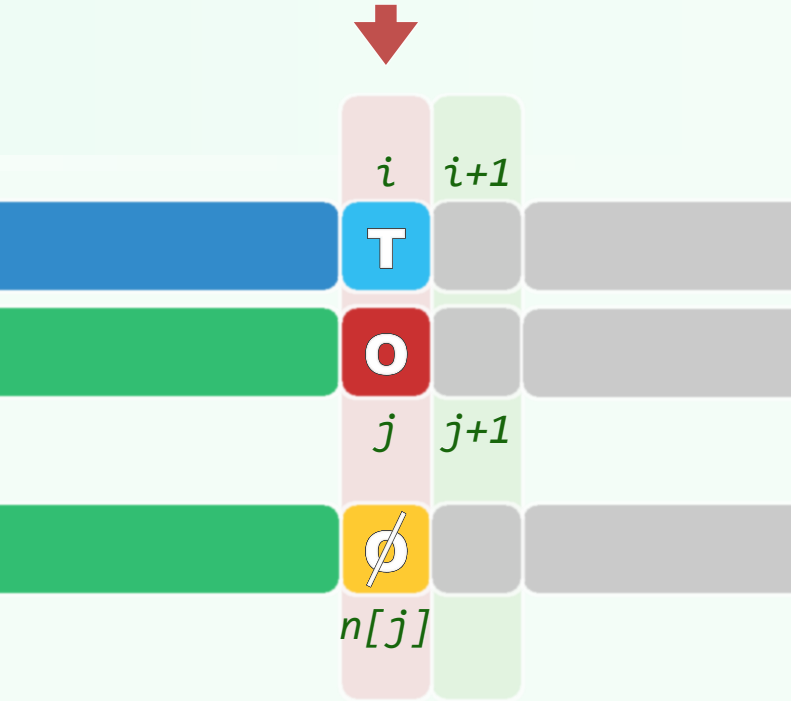
$$\forall j \geq 1,\ \mathbf{N}(P,j) = \{\ 0 \leq t < j \mid P[0,t) = P[j-t,j) \text{ and } P[t] \neq P[j]\ \} \cup \{\ -1\ \}$$

$$-1 \in \mathbf{N}(P,j) \neq \varnothing \qquad \text{//因总包含-1而非空，故可以}$$

$$\text{next}[j] = \max\{\ \mathbf{N}(P,j)\ \} \qquad \text{//取最大长度：位移最小，不致回溯}$$

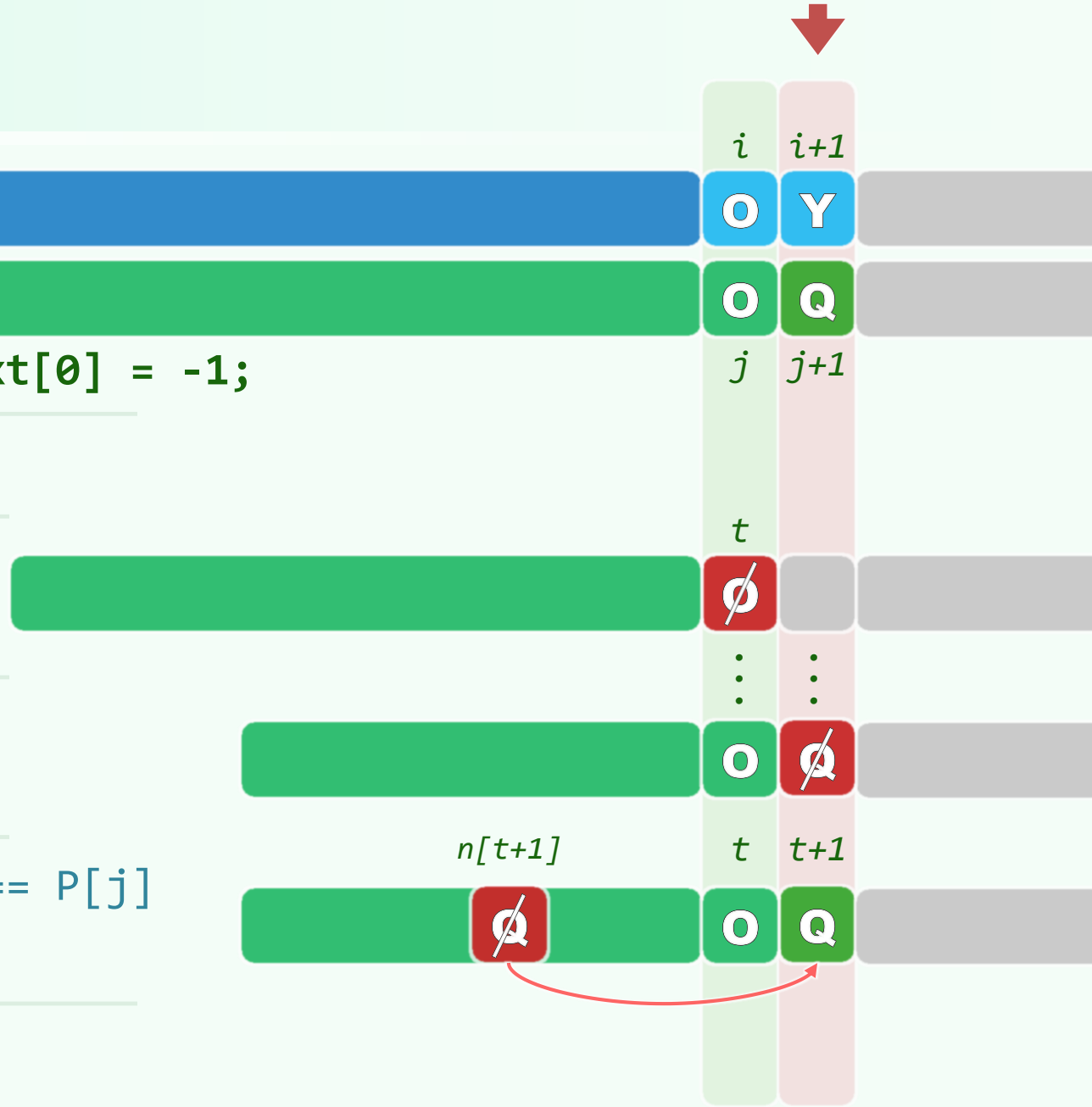
反观：next[j]在此前是如何确定的？

```
int* buildNext( char* P ) {
   int m = strlen(P), j = 0;
   int* next = new int[m]; int t = next[0] = -1;
   while ( j < m - 1 )
      if ( 0 <= t && P[t] != P[j] )
         t = next[t];
      else if ( P[++t] != P[++j] )
            next[j] = t;
         else //P[next[t]] != P[t] == P[j]
            next[j] = next[t];
   return next;
}
```

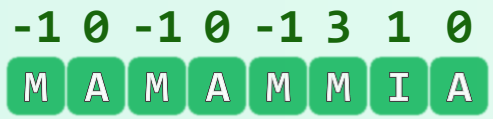


前进：接下来如何确定next[j+1]?

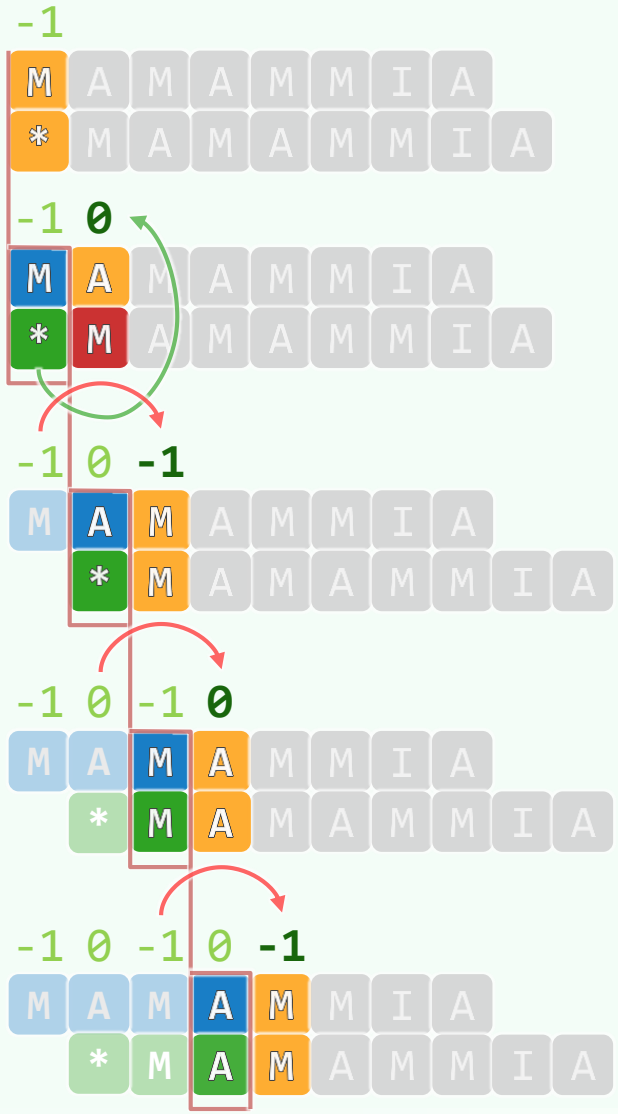
```
int* buildNext( char* P ) {
   int m = strlen(P), j = 0;
   int* next = new int[m]; int t = next[0] = -1;
   while ( j < m - 1 )
      if ( 0 <= t && P[t] != P[j] )
         t = next[t];
      else if ( P[++t] != P[++j] )
            next[j] = t;
         else //P[next[t]] != P[t] == P[j]
            next[j] = next[t];
   return next;
}
```

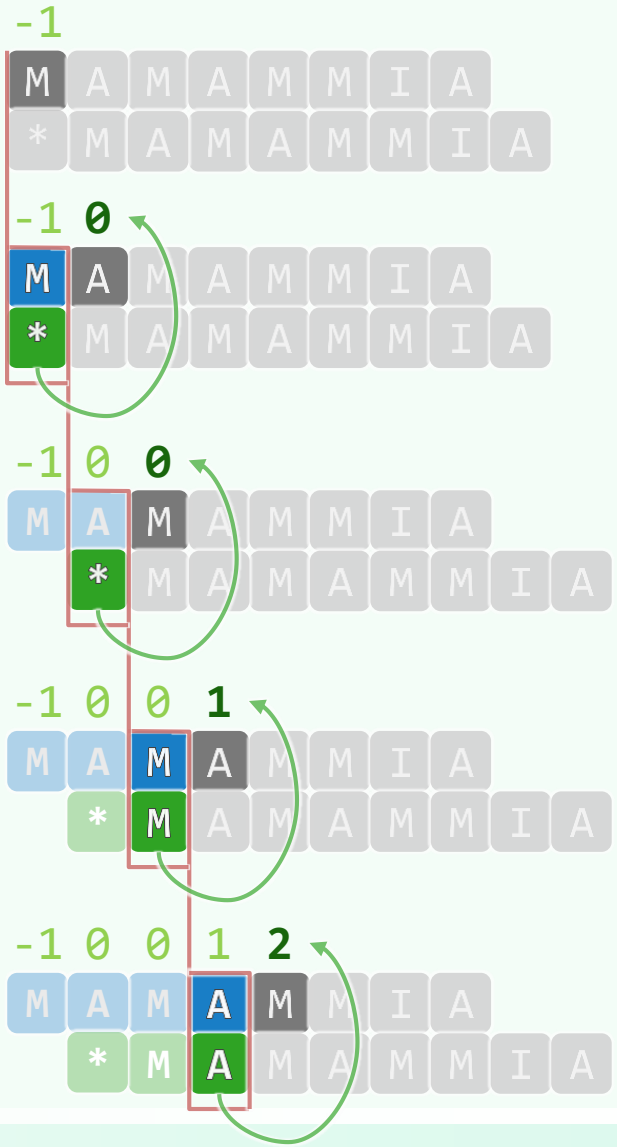


对比

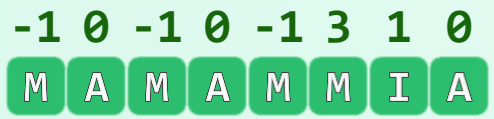

-1 0 -1 0 -1 3 1 0
M A M A M M I A

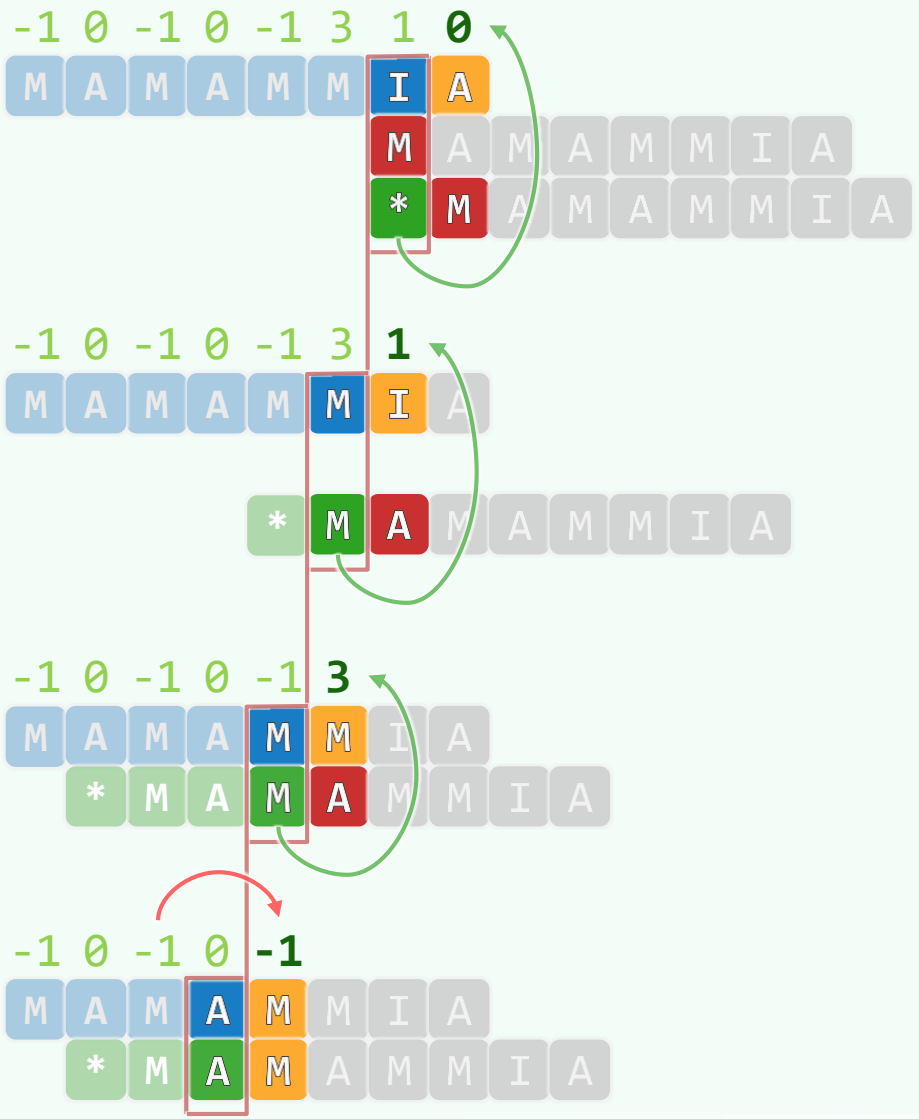

-1 0 0 1 2 3 1 0
M A M A M M I A

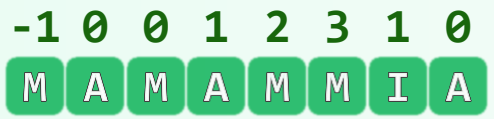

-1
M A M A M M I A
* M A M A M M I A
-1 0
M A M A M M I A
* M A M A M M I A
-1 0 -1
M A M A M M I A
* M A M A M M I A
-1 0 -1 0
M A M A M M I A
* M A M A M M I A
-1 0 -1 0 -1
M A M A M M I A
* M A M A M M I A

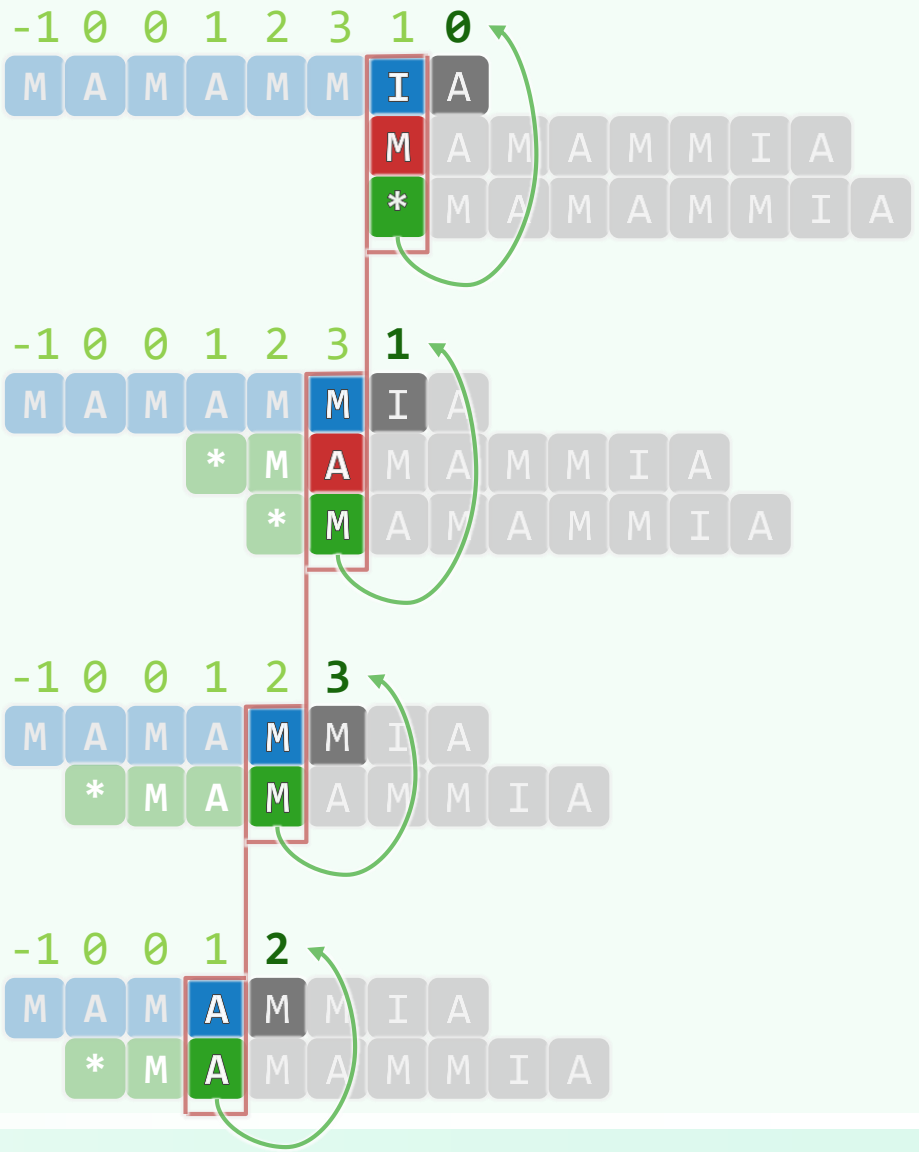

-1
M A M A M M I A
* M A M A M M I A
-1 0
M A M A M M I A
* M A M A M M I A
-1 0 0
M A M A M M I A
* M A M A M M I A
-1 0 0 1
M A M A M M I A
* M A M A M M I A
-1 0 0 1 2
M A M A M M I A
* M A M A M M I A

对比


-1 0 -1 0 -1 3 1 0
M A M A M M I A


-1 0 -1 0 -1 3 1 0
-1 0 -1 0 -1 3 1
-1 0 -1 0 -1 3
-1 0 -1 0 -1


-1 0 0 1 2 3 1 0
M A M A M M I A


-1 0 0 1 2 3 1 0
-1 0 0 1 2 3 1
-1 0 0 1 2 3
-1 0 0 1 2

小结

- 充分利用以往的比对所提供的信息

  模式串快速右移，文本串无需回退

- 经验 ~ 以往成功的比对：T[i-j, i)是什么

  教训 ~ 以往失败的比对：T[i]不是什么

- 特别适用于顺序存储介质

- 单次匹配概率越大（字符集越小），优势越明显 //比如二进制串

  否则，与蛮力算法的性能相差无几...