串

KMP算法：再改进

13-C6

有颜回者好学，不迁怒，不贰过

母亲心疼地看了我好久，然后叹口气：“好吧，你这个倔强的孩子，那条路很难走，一路小心！”

邓俊辉

deng@tsinghua.edu.cn

反例

❖ 在T[3]处

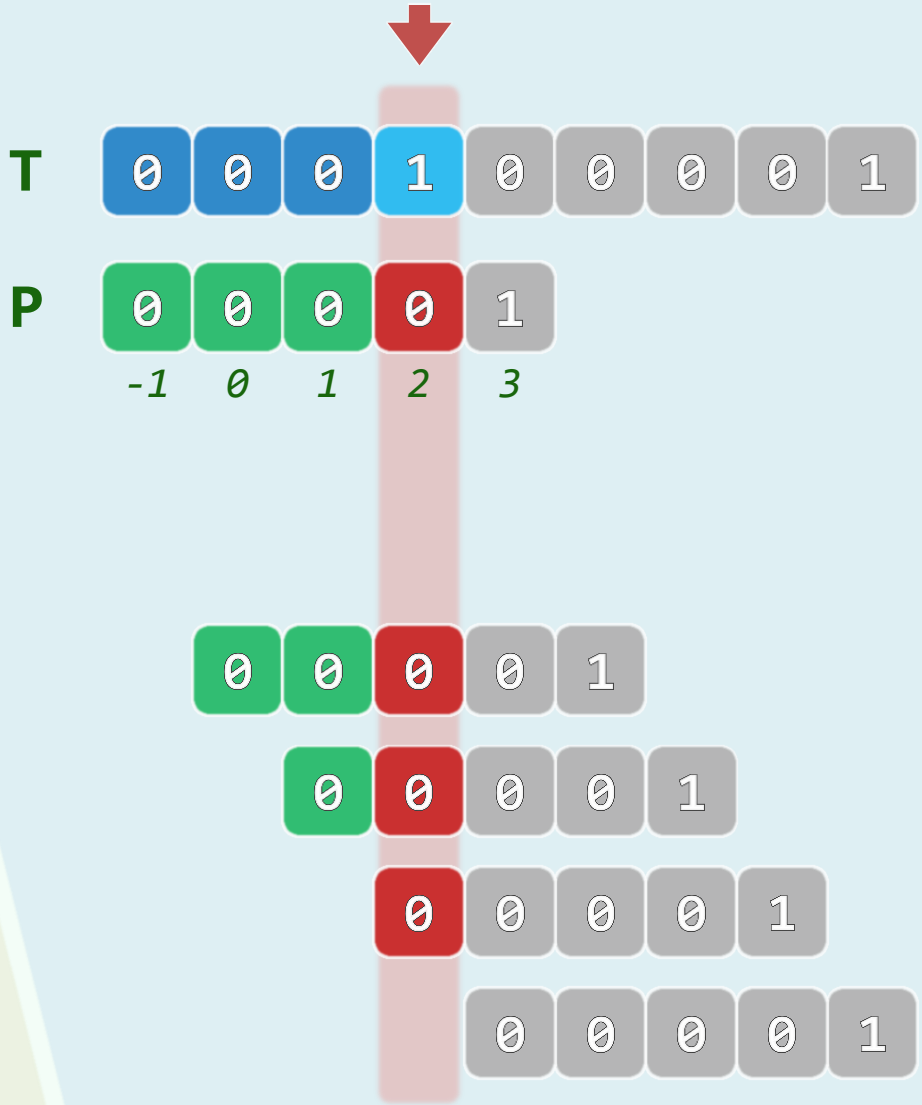
又：与 P[3] 比对，失败

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] =

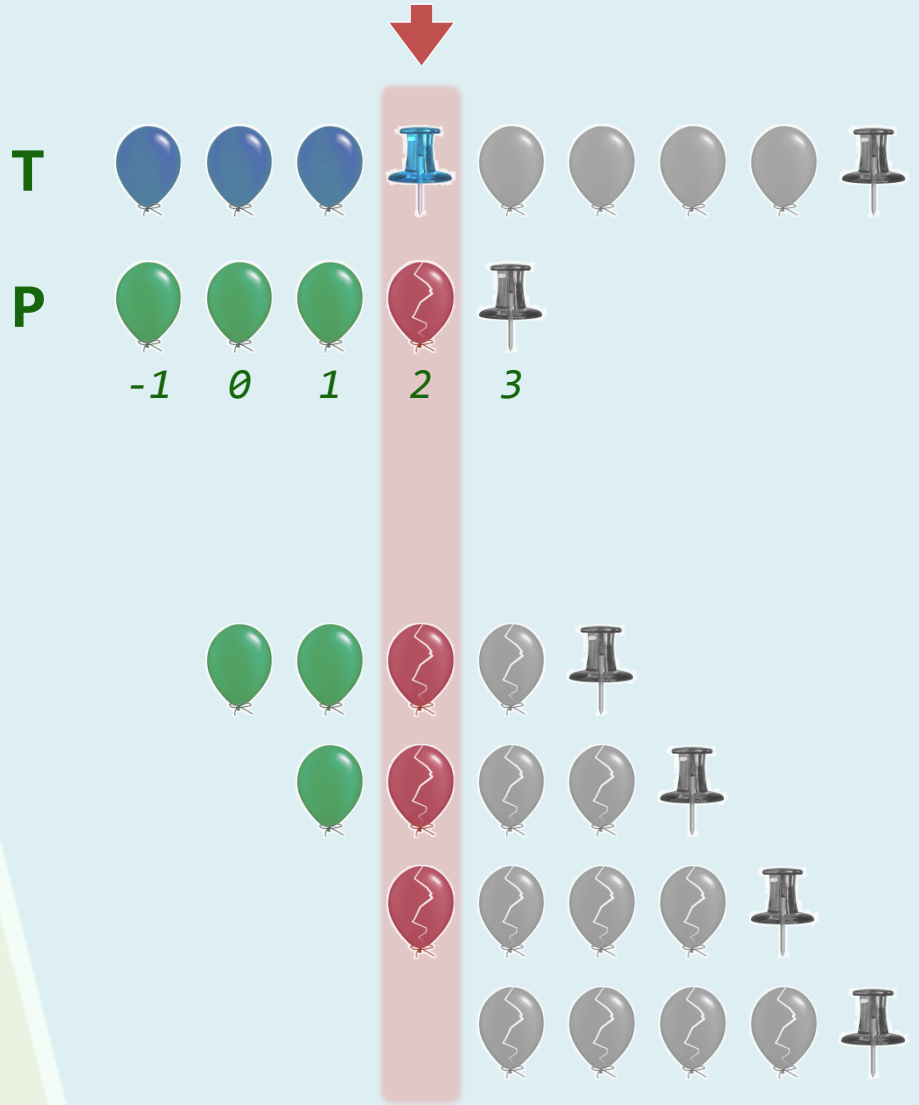
P[1] =

P[0] = 0

既然如此...

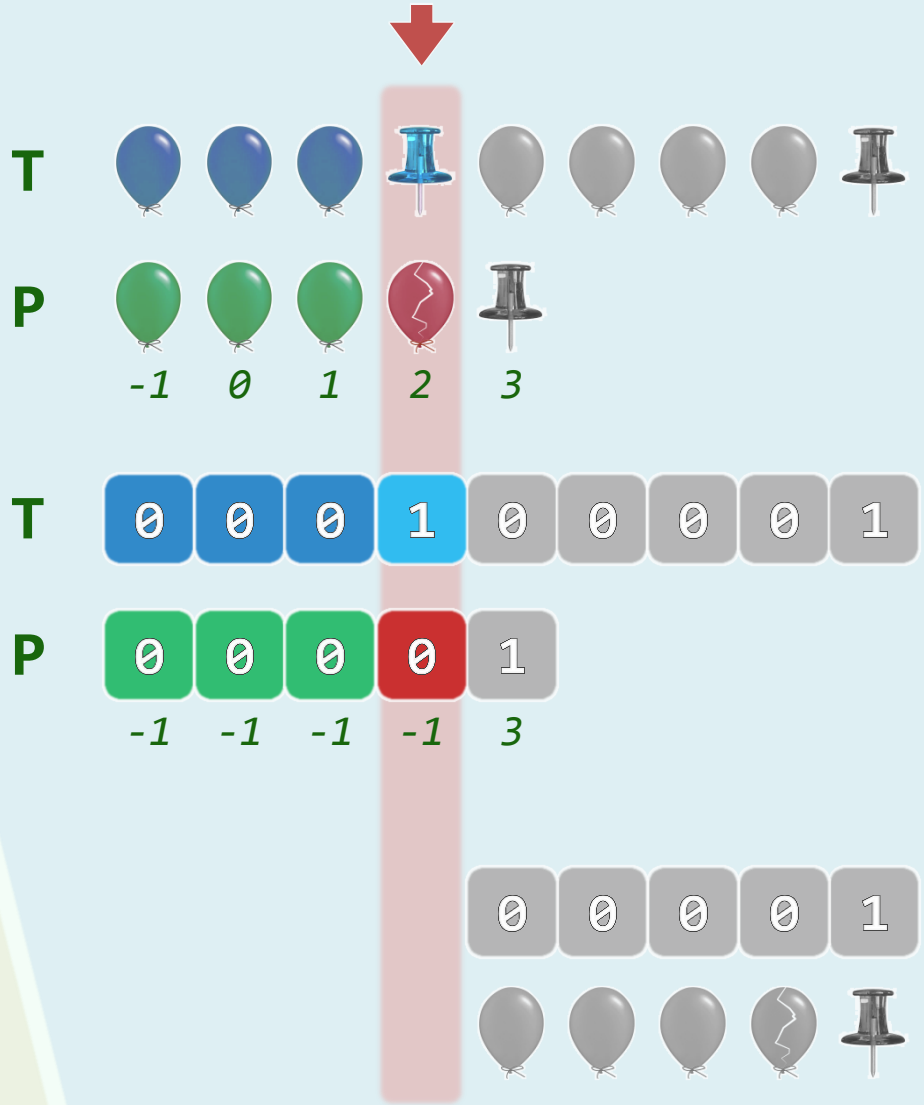
❖ 在发现 T[3] != P[3] 之后，为何还要一错再错？

❖ 事实上，后三次比对本来都是可以避免的！

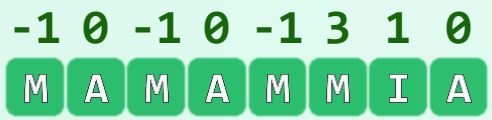


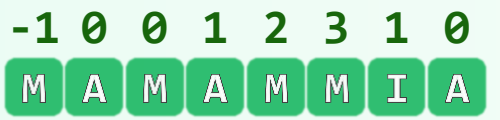
改进

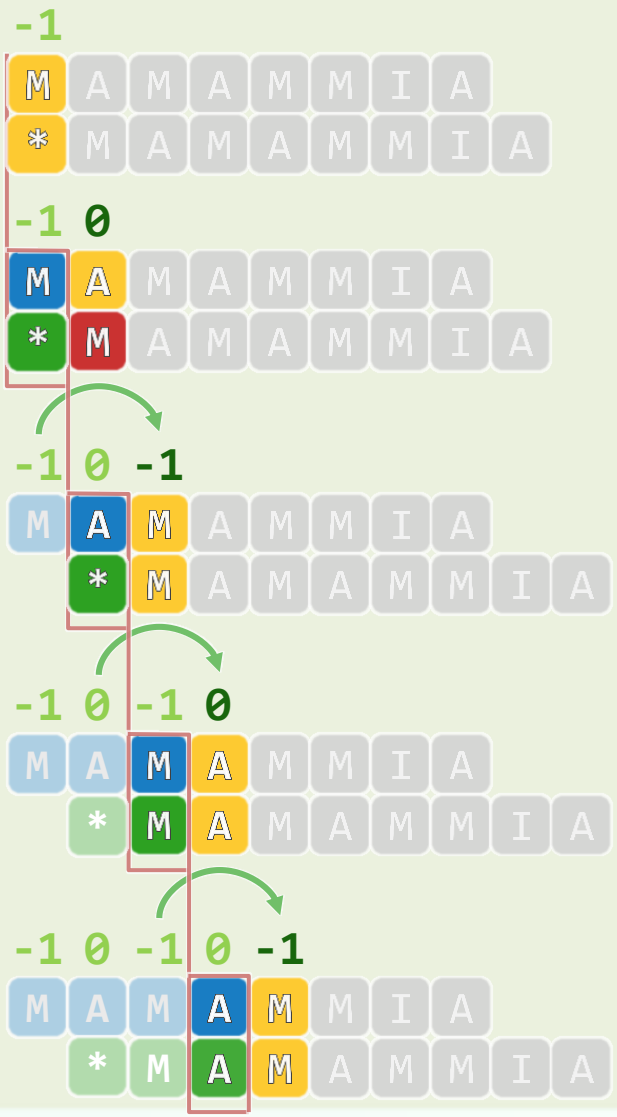
```
int * buildNext( char * P ) {
   size_t m = strlen(P), j = 0;
   int * N = new int[m];
   int t = N[0] = -1;
   while ( j < m - 1 )
      if ( 0 > t || P[j] == P[t] ) {
         j ++; t ++;
         N[j] = ( P[j] != P[t] ? t : N[t] );
      } else
         t = N[t];
   return N;
}
```

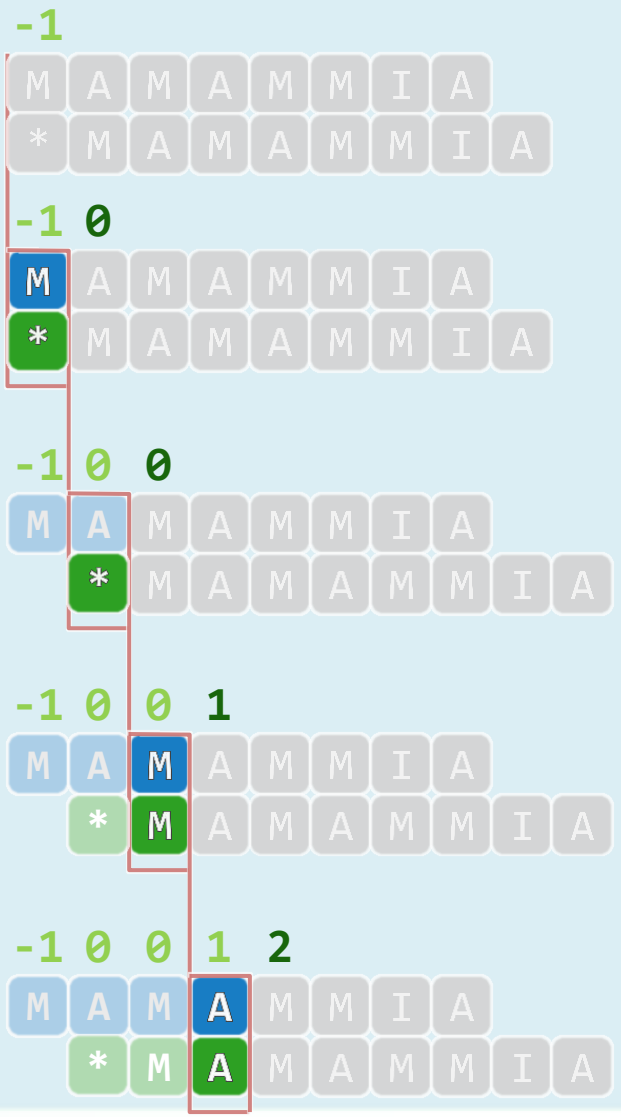


对比

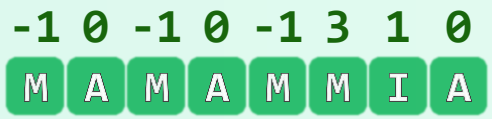

-1 0 -1 0 -1 3 1 0
M A M A M M I A

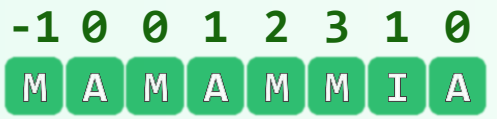

-1 0 0 1 2 3 1 0
M A M A M M I A

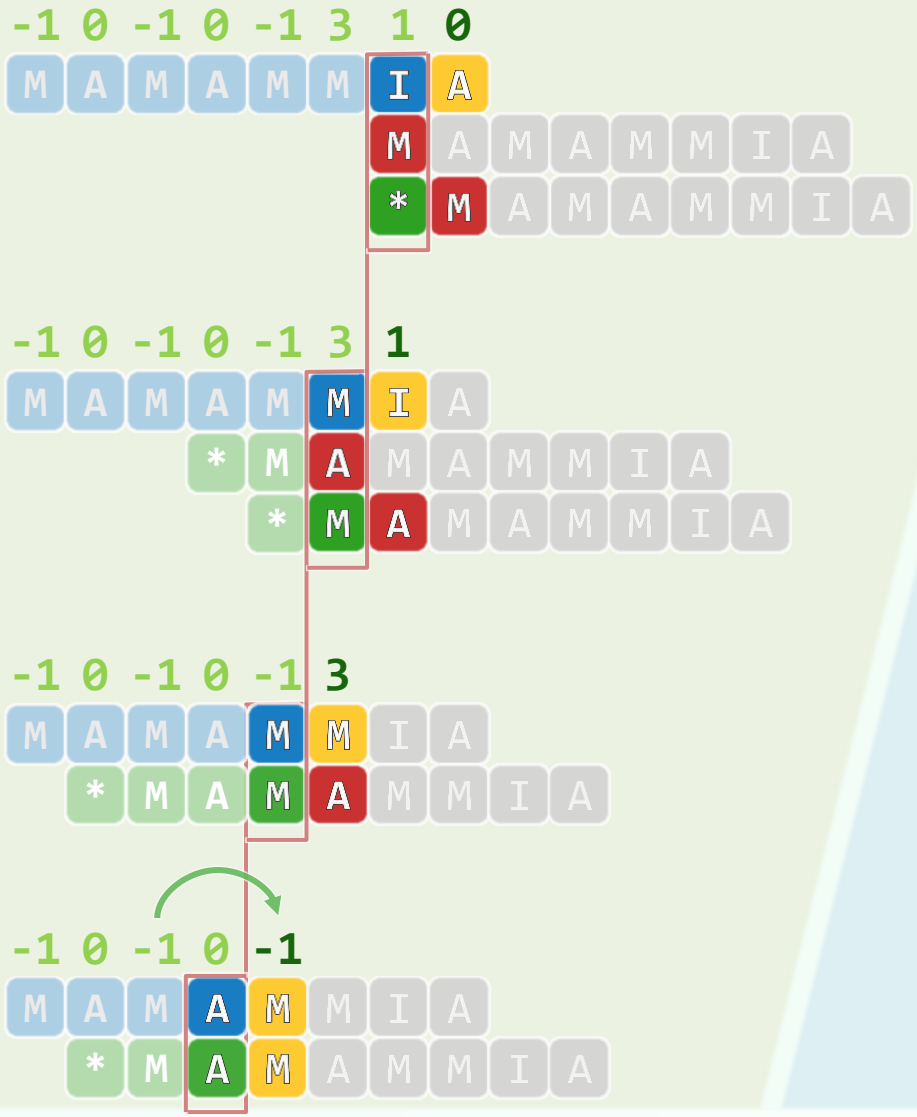

-1
M A M A M M I A
* M A M A M M I A
-1 0
M A M A M M I A
* M A M A M M I A
-1 0 -1
M A M A M M I A
* M A M A M M I A
-1 0 -1 0
M A M A M M I A
* M A M A M M I A
-1 0 -1 0 -1
M A M A M M I A
* M A M A M M I A

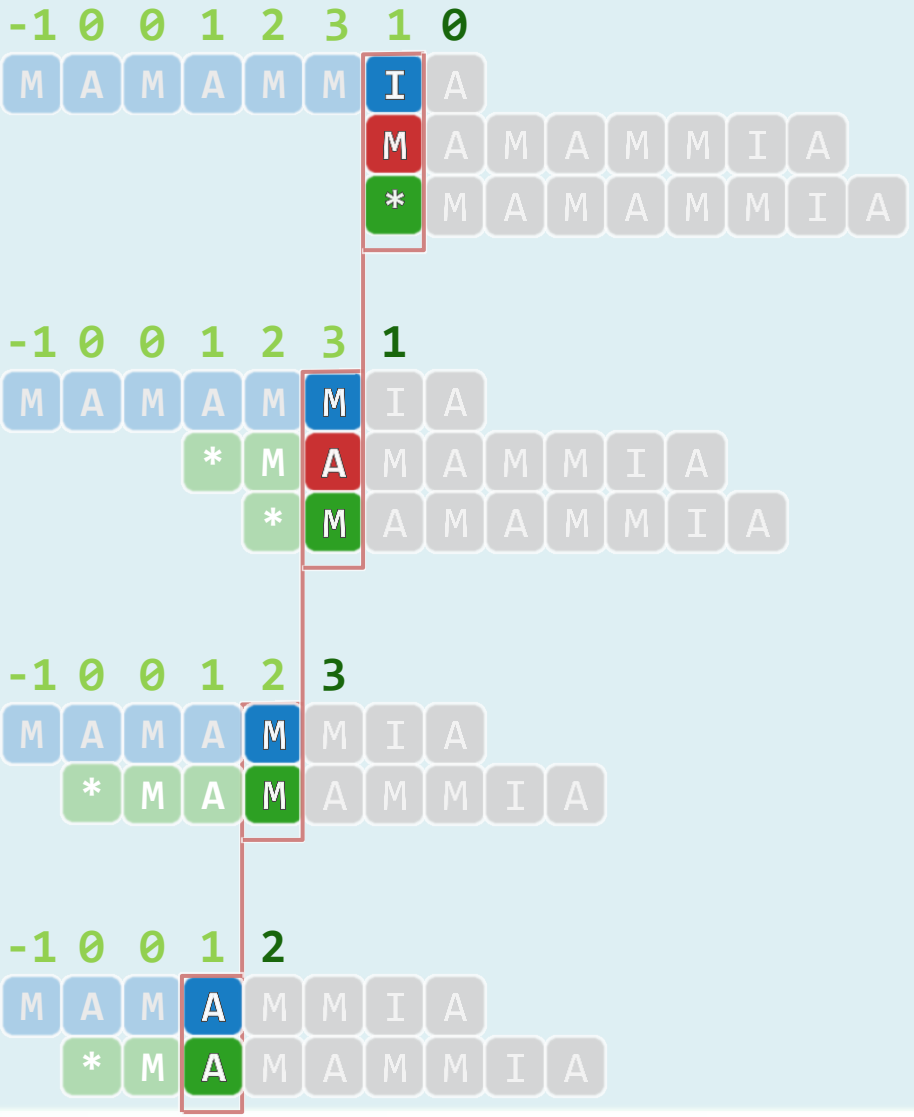

-1
M A M A M M I A
* M A M A M M I A
-1 0
M A M A M M I A
* M A M A M M I A
-1 0 0
M A M A M M I A
* M A M A M M I A
-1 0 0 1
M A M A M M I A
* M A M A M M I A
-1 0 0 1 2
M A M A M M I A
* M A M A M M I A

对比


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


-1 0 0 1 2 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
-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...