# 36KMP算法1

## KMP算法

相信很多鱼油（包括小甲鱼自己）在刚开始接触KMP算法的时候始终是丈二和尚摸不着头脑，要么完全不知所云，要么看不懂书上的解释，要么自己觉得好像心里了解KMP算法的意思，却说不出个究竟，所谓知其然不知其所以然是也。

KMP算法对大多数初学者来言是一项比较巨大的考验，特别是自学算法的朋友，很多资料对数学功底不牢固的学者来说看起来比较模糊。

所以，小甲鱼在讲解这一算法的实现前，特地添加了这一篇章：最高作战方针之算法思路养成篇！

KMP算法是三位老前辈（D.E.Knuth、J.H.Morris 和 V.R.Pratt）的研究结果，大大的避免重复遍历的情况，全称叫做克努特-莫里斯-普拉特算法，简称KMP算法。

上节课我们谈了BF算法，也说了BF算法虽然很黄很暴力，但是效率却不搞。

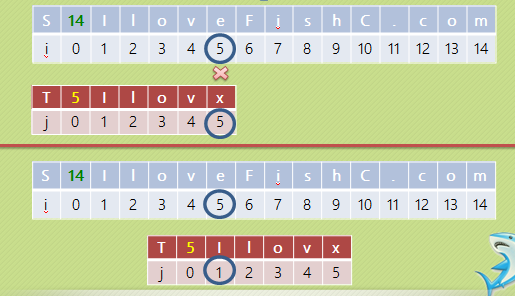
我们也给了一个例子。但似乎那个例子不足以反映出导致这种算法效率低下的致命缺陷。

因此，我们用下边一个例子来继续探讨：



* 回溯就是坚持条条大路通罗马的决心，然后遇到挫折就回到跌倒的地方重新爬起来，继续往前，这种思想是好的，但效率是低的。
* 因为牛在耕田的时候也是这么想的，但人家袁隆平懂得找捷径，才有了超级杂交水稻，这也告诉我们鱼油，学习除了勤奋，除了必要坚持，还更需要思考！
* KMP算法的核心就是避免不必要的回溯，那么什么是不必要的呢(me:记住每次匹配的过程，减少不必要的回溯)？问题由模式串决定，不是由目标决定！

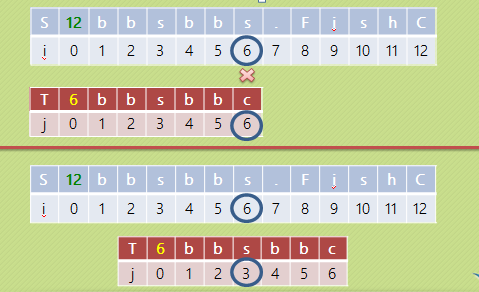
思路启发一



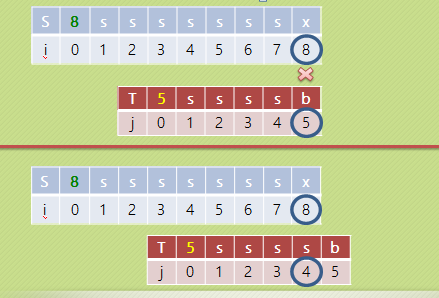
思路启发二



思路启发三



思路启发四



## 注重思考不要注重结果

如果平时有阅读英文计算机书籍的朋友会发现很多教学书籍都有课后习题，但大部分不会附带答案，市面也不会有所谓的“答案全解”。

其实在老外的教学中，他们更加注重思考而非正确的结果。回想我们之所以会这么在意答案完全是中国式教育的产物，在我们的学校，分数就是一切！

在小甲鱼的所有教学中，我希望大家可以培养独立思考的精神，因为为这是创新的根源所在！

希望你们可以通过以上引导，自己推导出KMP算法的原理！

# 37KMP算法2

## KMP算法之养成篇

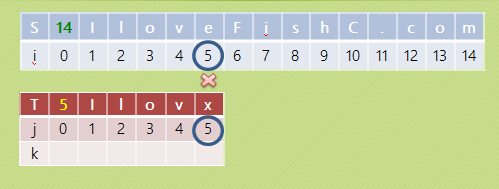
为了照顾广大喜欢细嚼慢咽的朋友，小甲鱼再添加一讲继续深入探讨思路。

这次我们给模式匹配串添加一个k数组（也就是KMP算法中非著名的next数组）。

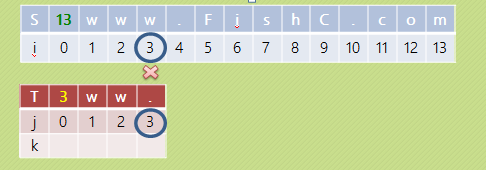
这是一个“智能”的数组，因为他指导着模式匹配串下一步改用第几号元素去进行匹配。

No pic you say a J8!

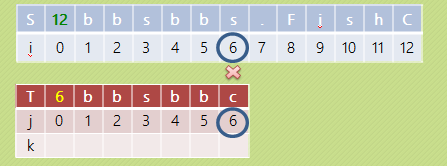
## 思路启发一



## 思路启发二



## 思路启发三



## 思路启发四



## 注重思考不要注重结果

* 如果平时有阅读鸟文计算机书籍的朋友会发现很多教学书籍都有课后习题，但大部分不会附带答案，市面也不会有所谓的“答案全解”。
* 其实在老外的教学中，他们更加注重思考而非正确的结果。回想我们之所以会这么在意答案完全是中国式教育的产物，在我们的学校，分数就是一切！
* 在小甲鱼的所有教学中，我希望大家可以培养独立思考的精神，因为为这是创新的根源所在！
* 希望你们可以通过以上引导，自己推导出KMP算法的原理！

# 38KMP算法之NEXT数组代码原理分析

## KMP算法之NEXT数组代码原理分析

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **T** | **5** | **a** | **b** | **a** | **b** | **a** | **a** | **a** | **b** | **a** |
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| next |  | 0 | 1 | 1 | 2 | 3 | 4 | 1 | 2 | 1 |

i（后缀）=

j（前缀）=

NEXT数组：当模式匹配串T失配的时候，NEXT数组对应的元素指导应该用T串的哪个元素进行下一轮的匹配。

Next.c

void get\_next( String T, int \*next )

{

j = 0;

i = 1;

next[1] = 0;

while( i < T[0] )

{

if( 0==j || T[i] == T[j] )

{

i++;

j++;

next[i] = j;

}

else

{

j = next[j];

}

}

// 因为前缀是固定的，后缀是相对的。

}

39KMP算法之实现及优化

KMP算法之实现及优化

* 搞定了NEXT数组，KMP算法就易如反掌了。
* 一起来完成：kmp.c
* KMP模式匹配算法改进
  + 后来有人发现，KMP算法是有缺陷的。比如我们的主串 S =“aaaabcde”，子串 T =“aaaaax”，其中很容易得到next数组为012345。

那么我们看以下比较图：

KMP模式匹配算法改进

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S** | **8** | **a** | **a** | **a** | **a** | **b** | **c** | **d** | **e** |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **T** | **5** | **a** | **a** | **a** | **a** | **a** | **x** |
| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| k |  | 0 | 1 | 2 | 3 | 4 | 5 |

Kmp.c

#include <stdio.h>

typedef char\* String;

void get\_next( String T, int \*next )

{

int j = 0;

int i = 1;

next[1] = 0;

while( i < T[0] )

{

if( 0 == j || T[i] == T[j] )

{

i++;

j++;

next[i] = j;

}

else

{

j = next[j];

}

}

}

// 返回子串T在主串S第pos个字符之后的位置

// 若不存在，则返回0

int Index\_KMP( String S, String T, int pos )

{

int i = pos;

int j = 1;

int next[255];

get\_next( T, next );

while( i <= S[0] && j <= T[0] )

{

if( 0 == j || S[i] == T[j] )

{

i++;

j++;

}

else

{

j = next[j];

}

}

if( j > T[0] )

{

return i - T[0];

}

else

{

return 0;

}

}

Kmp2.c

#include <stdio.h>

typedef char\* String;

void get\_next( String T, int \*next )

{

int j = 0;

int i = 1;

next[1] = 0;

while( i < T[0] )

{

if( 0 == j || T[i] == T[j] )

{

i++;

j++;

if( T[i] != T[j] )

{

next[i] = j;

}

else

{

next[i] = next[j];

}

}

else

{

j = next[j];

}

}

}

// 返回子串T在主串S第pos个字符之后的位置

// 若不存在，则返回0

int Index\_KMP( String S, String T, int pos )

{

int i = pos;

int j = 1;

int next[255];

get\_next( T, next );

while( i <= S[0] && j <= T[0] )

{

if( 0 == j || S[i] == T[j] )

{

i++;

j++;

}

else

{

j = next[j];

}

}

if( j > T[0] )

{

return i - T[0];

}

else

{

return 0;

}

}

# 40树

## 树









之前我们一直在讨论的是一对一的线性结构，无论是线性表也好，栈和队列也罢，都是2P模式。

可现实生活中，3P、4P等现象比比皆是，例如一个年轻的妈妈生了4个孩子，而每个孩子都不像他们的爸爸，那么这类情况我们用线性结构的形式就不足以描述了！

所以我们需要研究这种一对多的数据结构：树

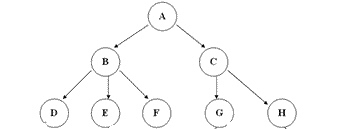
解下来的一些列教程，小甲鱼会充分考虑它的各种特性，结合现实中的实例给大家讲解。

## 树的定义

树(Tree)是n(n>=0)个结点的有限集。当n=0时成为空树，在任意一棵非空树中：

有且仅有一个特定的称为根(Root)的结点；

当n>1时，其余结点可分为m(m>0)个互不相交的有限集T1、T2、...、Tm，其中每一个集合本身又是一棵树，并且称为根的子树(SubTree)。

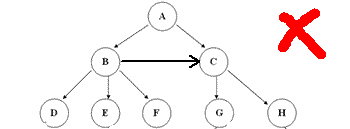


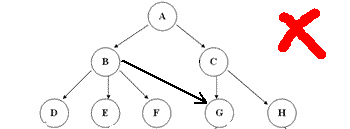
回忆下我们在讲解斐波那契数列的时候的那个“No pic you say a J8”环节画的那个图，是不是跟树长得一模一样？

虽然从概念上很容易理解树，但是有两点还是需要大家注意下：

n>0时，根结点是唯一的，坚决不可能存在多个根结点。

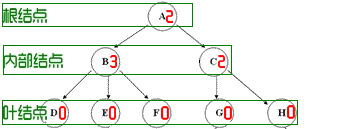
m>0时，子树的个数是没有限制的，但它们互相是一定不会相交的。





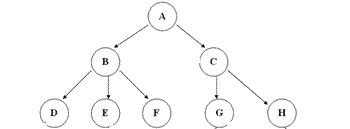
结点分类

* 刚才所有图片中，每一个圈圈我们就称为树的一个结点。结点拥有的子树数称为结点的度-(Degree)，树的度取树内各结点的度的最大值。
  + 度为0的结点称为叶结点(Leaf)或终端结点；
  + 度不为0的结点称为分支结点或非终端结点，除根结点外，分支结点也称为内部结点。



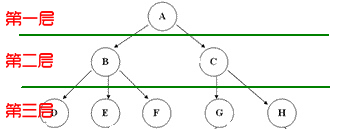
结点间的关系

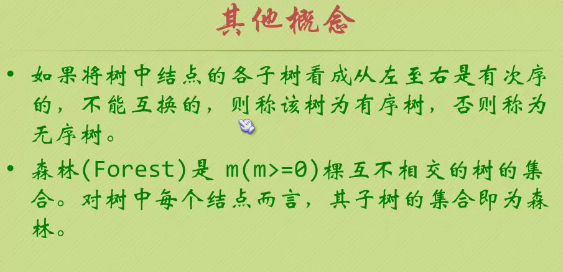
* 结点的子树的根称为结点的孩子(Child)，相应的，该结点称为孩子的双亲(Parent)，同一双亲的孩子之间互称为兄弟(Sibling)。
* 结点的祖先是从根到该结点所经分支上的所有结点。



结点的层次

* 结点的层次(Level)从根开始定一起，根为第一层，根的孩子为第二层。
* 其双亲在同一层的结点互为堂兄弟。
* 树中结点的最大层次称为树的深度(Depth)或高度。





有序树 无序树 森林