



Star_Feel

彼岸花末、此岸花开

[首页](#)
[管理](#)
[联系](#)

算法导论——EXKMP

【例题传送门：[caioj1461](#)】

随笔 - 308 文章 - 0 评论 - 2

公告

机房小友

【AKCqhzdy】
 【Rose_max】
 【Mocha】
 【Hanks_o】
 【靖爷】
 【xgc_woker】
 【Cherish_OI】
 【MT_LI】



指教方式

QQ:1173544097
 VC:H85575285

昵称: Star_Feel

园龄: 1年2个月

粉丝: 7

关注: 3

+加关注

【EXKMP】最长共同前缀长度

【题意】

给出模板串A和子串B，长度分别为lenA和lenB，要求在线性时间内，对于每个A[i] ($1 \leq i \leq \text{lenA}$)，求出A[i..lenA]与B的最长公共前缀长度

【输入文件】

输入A, B两个串，($\text{lenB} \leq \text{lenA} \leq 1000000$)

【输出文件】

输出lenA个数，表示A[i..lenA]与B的最长公共前缀长度，每个数之前有空格

【样例输入】

aabbabaaab

aabb

【样例输出】

4 1 0 0 1 0 2 3 1 0

算法分析：

学EXKMP前，必须将KMP学透，如果仍未学KMP，请出门左转【[传送门](#)】

我们在KMP算法中可以理解，p数组是KMP的核心，p[i]代表着以i为结尾和以开头为首的最长公共子串长度，也就是说对于st字符串数组的p[i]代表的就是st字符串数组从1开始到p[i]和从i-p[i]+1到i是完全相同的 ($\text{st}[1 \dots p[i]] = \text{st}[i-p[i]+1 \dots i]$)

那么扩展KMP就高级了。一样还是p数组（还是原来的配方，还是熟悉的味道！），但是既然是扩展KMP就不要用p，我就改成extend数组了，表示的意义与普通KMP就大不相同。extend[i]表示的是以i为首和以开头为首的最长前缀，也就是说对于st字符串数组中的extend[i]表示的就是st字符串数组从1开始到extend[i]和从i到i+extend[i]-1是完全相同的 ($\text{st}[1 \dots \text{extend}[i]] = \text{st}[i \dots i+\text{extend}[i]-1]$)

首先extend[1]就不用说了，直接等于len这个没有问题吧（假如有问题就撞墙死算了），那么因为extend[1]这个是具有确定性，所以我们基本把这东西废掉..那么我们就直接从extend[2]开始。然后我们就定义一个k，这个k表示的就是在当前搜索过的范围以内（因为这是线性算法，所以是从1到len）能到达最远（也就是说k+extend[k]-1最大的）的编号，为什么要定义一个这样子的东西，等下你们就知道了。

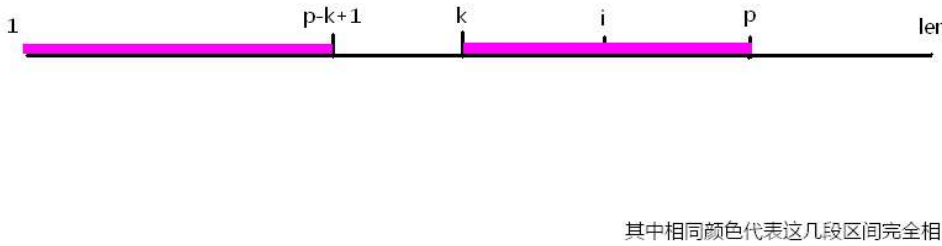
我们先定义一个p，让它等于k+extend[k]-1，那么由extend这个数组的定义我们就可以得到一个等式： $\text{st}[1 \dots \text{extend}[k]] = \text{st}[k \dots p]$ 。因为p=k+extend[k]-1，所以我们可以得到一条等式： $\text{extend}[k] = p - k + 1$ ，把这个代换到上一条等式上，就会——瞬间爆炸！（好吧开个玩笑..）就会变成： $\text{st}[1 \dots p - k + 1] = \text{st}[k \dots p]$ ，如下图：

2018年10月						
日	一	二	三	四	五	六
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

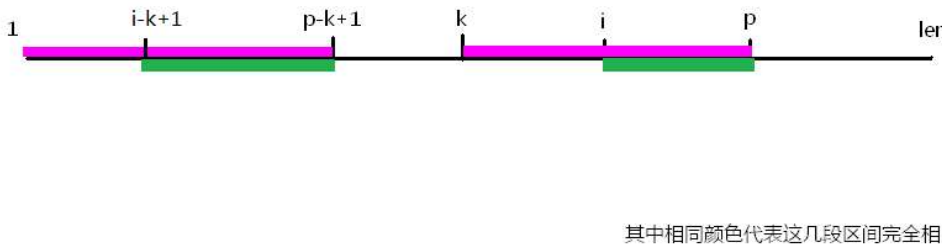
搜索

随笔分类

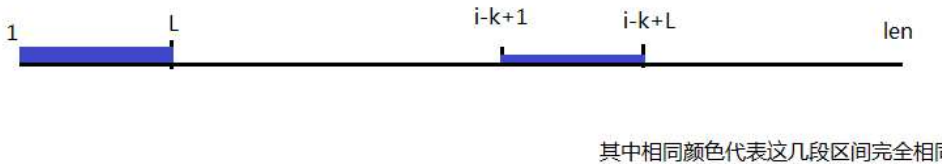
OJ&Competition——51nod(16)
 OJ&Competition——BZOJ(271)
 OJ&Competition——caioj(5)
 OJ&Competition——Codeforces(2)
 OJ&Competition——HDU(1)
 OJ&Competition——SPOJ(3)
 OJ&Competition——酱油记(2)
 The way——01分数规划(1)
 The way——CDQ分治
 The way——DFS(15)
 The way——ST表(RMQ)(3)



因为我们现在要求从 $\text{extend}[i]$ ，那么由上面这条等式又可以得到另一条等式：
 $\text{st}[i-k+1 \dots p-k+1] = \text{st}[i \dots p]$ ，如下图：

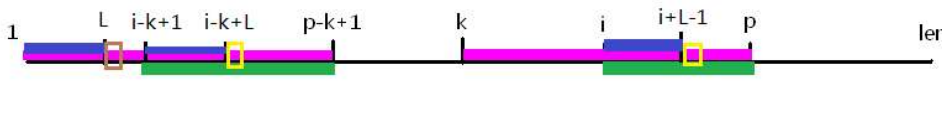


在看此证明过程中请各位一直记住 extend 数组所表达的含义，不然会有很多地方不懂的。那么我们再定义一个 $L = \text{extend}[i-k+1]$ ，我们又可以得到一条等式： $\text{st}[i-k+1 \dots i-k+L]$ （注意：本来得到的应该是 $i-k+1+L-1$ ，我直接把 $+1-1$ 省略了）
 $= \text{st}[1 \dots L]$ ，如下图：



这个时候有人就会问了：为什么你上面的图不和前几个合在一起呢？这个就是本算法的一个难点了！因为 L 的不定性，这个时候我们需要考虑 $i-k+L$ 和 $p-k+1$ 的大小！那我们就来分开来考虑。

(1) $i-k+L < p-k+1$ ，如下图：



从上图我们可以看到因为 $\text{st}[1 \dots L] = \text{st}[i-k+1 \dots i-k+L]$ ， $\text{st}[i-k+1 \dots p-k+1] = \text{st}[i \dots p]$ ，所以在 $\text{st}[i \dots p]$ 中肯定含有一段（从 i 开始）和 $\text{st}[1 \dots L]$ 是完全相同的，也就是上图标出来的蓝色部分 $\text{st}[i \dots i+L-1]$ 。因为 $\text{st}[i-k+1 \dots p-k+1] = \text{st}[i \dots p]$ 又因为 $i-k+L < p-k+1$ ，所以我们又可以得到： $\text{st}[i-k+L+1] = \text{st}[i+L]$ （也就是上图所标的黄色位置），而因为 $\text{extend}[i-k+1]$ 的定义，所以 $\text{st}[L+1] \neq \text{st}[i-k+L+1]$ ，所以我们可以得到： $\text{st}[i+L] \neq \text{st}[L+1]$ ，那么 $\text{extend}[i]$ 就直接等于 L 了。

The way——并查集&启发式合并

The way——博弈(2)

The way——差分约束(3)

The way——单调栈&单调队列(3)

The way——二分(11)

The way——分块(2)

The way——矩阵乘法(4)

The way——链表(1)

The way——乱搞(27)

The way——莫队算法(13)

The way——三分(1)

The way——随机(1)

The way——贪心(15)

动态规划——单调队列&斜率优化(11)

动态规划——概率期望DP(3)

动态规划——计数DP(1)

动态规划——区间DP(2)

动态规划——树形DP(5)

动态规划——数位DP(3)

动态规划——线性DP(18)

动态规划——状压DP(2)

树论——K-Dtree

树论——LCA(6)

树论——Link Cut Tree(4)

树论——prufer数列(1)

树论——SPLAY(6)

树论——点分治&动态点分治(3)

树论——树链剖分(7)

树论——树上差分(2)

树论——树状数组(12)

树论——线段树(14)

树论——主席树&线段树合并(12)

树论——最小生成树kruscal&prim(8)

树论——左偏树(1)

数论——BSGS

数论——FFT(5)

数论——基本数论(8)

数论——莫比乌斯反演(7)

算法导论(4)

图论——A*(1)

图论——SPFA&BFS&Floyd&Dijkstra(10)

图论——Tarjan强联通(7)

图论——并查集(8)

图论——二分图匹配(9)

图论——基环树(2)

图论——网络流&费用流&最小割&最大权闭合子图(34)

字符串处理——AC自动机(5)

字符串处理——EXKMP(2)

字符串处理——KMP(4)

字符串处理——Manacher(3)

字符串处理——后缀数组(2)

字符串处理——后缀自动机(SAM)(5)

字符串处理——回文树(1)

字符串处理——字典树(7)

随笔档案

其中相同颜色代表这几段区间完全相同

2018年10月 (27)

2018年9月 (10)

2018年8月 (1)

2018年5月 (8)

2018年4月 (76)

2018年3月 (83)

2018年2月 (11)

2018年1月 (13)

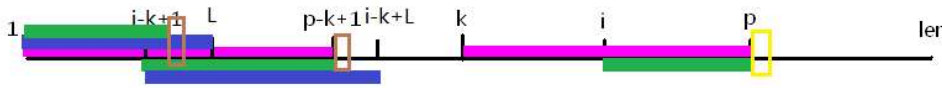
2017年12月 (21)

2017年11月 (24)

2017年10月 (18)

2017年9月 (17)

(2) $i-k+L > p-k+1$, 如下图:



其中相同颜色代表这几段区间完全相同

积分与排名

积分 - 7311

排名 - 52472



最新评论

1. Re:算法导论——EXKMP
@huyufeifei帮到你就好...

--Star_Feel

2. Re:算法导论——EXKMP
%%%讲的很详细，感谢！

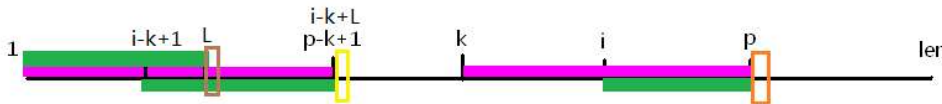
--huyufeifei

阅读排行榜

1. 算法导论——EXKMP(356)
2. 算法导论——KMP(238)
3. BZOJ5301: [Cqoi2018]异或序列(223)
4. 【为小白菜打call】(206)
5. caioj1442:第k小的数II(142)
6. 算法导论——斜率优化(138)
7. Self-Introduction(103)
8. 酱油记: GDKOI2018(92)
9. BZOJ1001: [BeiJing2006]狼抓兔子(83)
10. USACO2002 Open:雄伟的山峦(83)

从上图中我们看到因为 $st[1..L] = st[i-k+1..i-k+L]$, 又因为 $i-k+L > p-k+1$, 所以在 $st[1..L]$ 中肯定含有一段和 $st[i-k+1..p-k+1]$ 完全相同的 (图中绿色部分)。因为 $extend[k]$ 的意义, 所以 $st[p+1] \neq st[p-k+2]$ (图中第二个棕色和黄色位置, 为什么不相同不用解释吧), 又因为 $st[1..L] = st[i-k+1..i-k+L]$, 所以 $st[p-i+2]$ (图中并未标出, 第一个棕色位置) $= st[p-k+2]$, 那么就会得到: $st[p-i+2] \neq st[p+1]$, 所以 $extend[i]$ 就等于 $p-i+1$ 了。

(3) $i-k+L = p-k+1$, 如下图:



其中相同颜色代表这几段区间完全相同

评论排行榜

1. 算法导论——EXKMP(2)

推荐排行榜

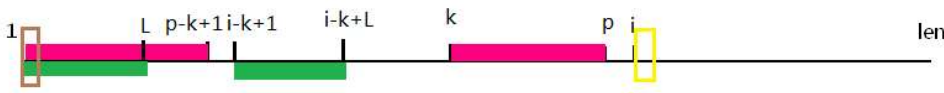
1. 算法导论——斜率优化(1)

从上图我们可以发现因为 $st[i-k+1..i-k+L] = st[1..L]$, $st[i-k+1..p-k+1] = st[i..p]$, 又因为 $i-k+L = p-k+1$, 所以 $st[1..L] = st[i-k+1..p-k+1]$ (可换成 $i-k+L$) $= st[i..p]$ (也就是指上图中三段绿色部分)。那么由于 $extend[i-k+1]$ 和 $extend[k]$ 的意义表达, 我们可以得到: $st[L+1] \neq st[i-k+L+1]$, $st[i-k+L+1] \neq st[p+1]$, 但是我们不能确定 $st[L+1]$ 和 $st[p+1]$ 是否相同, 我们只能确定从 i 开始和从 1 开始有 $p-i+1$ 这么长的公共前缀但并不一定是最长的 (这句话要好好理解, 这很重要)。

那么我们就设一个变量 $j = p-i+1$, 表示当前从 i 开始和从 1 开始的公共前缀长度, 由于上面加粗的那句话, 我们可以直接从 $st[j+1]$ 和 $st[i+j]$ 来累加 j 的值来得出最长的公共前缀。

注意事项:

因为 p 是一个不定的数 (由 k 和 $extend[k]$ 来定), 所以说有可能 $p-i+1$ 是有可能为负数, 那么第二种情况显然不对, 公共前缀怎么样也不能等于负数吧, 最小也会是 0 吧! 这个时候也许你就会冒出一个想法, 在第二种情况下取 $p-i+1$ 和 0 的最大值。很显然这是不对的。请看下图:



其中相同颜色代表这几段区间完全相同

从上图我们可以发现，因为 $p-i+1$ 是负数，所以上面所有条件都用不了，因为对 i 后面的字符没有作任何的计算，但这个时候是绝对不可以肯定 $st[1]$ 和 $st[i]$ 是不同的（也就是最长公共前缀为0），那么我们需要从 $st[1]$ 和 $st[i]$ 开始继续判断后面的字符是否相同。于是我们把第二种情况和第三种情况归纳为一种情况，因为两者都是要暴力处理未知的点，只是起始点不同

这仅仅是求`extend`数组的证明，也仅仅是在同一个字符串里的基本操作，接下来我就来讲下扩展KMP（简称EXKMP）的实际用途。EXKMP主要是利用于解决处理两个字符串的最长公共前缀长度，假如A是主串，B是副串，那么这时我们定义一个`ex`数组，`ex[i]`就表示`A[i...Alen]`和`B[1...Blen]`的最长公共前缀长度（这个概念需要好好注意）

其实在处理两者的匹配时，只需要注意将A串中的子串转移到B串中进行处理，那么这样我们实际上在求`ex`数组时，操作仍与上面的步骤相似

参考代码：

```
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<algorithm>
#include<cmath>
using namespace std;
char sa[1100000],sb[1100000];
int lena,lenb;
int p[1100000],ex[1100000];
//p数组是用来让B串自己匹配自己的
void exkmp()
{
    p[1]=lenb;
    int x=1;
    while (sb[x]==sb[x+1]&&x+1<=lenb) x++;//因为我们p[1]是具有确定性，所以我们不能直接用，所以要先暴力
    求出p[2]
    p[2]=x-1;
    int k=2;
    for(int i=3;i<=lenb;i++)
    {
        int pp=k+p[k]-1,L=p[i-k+1];//pp实际上是p
        if(i+L<pp+1) p[i]=L;//i-k+L<pp-k+1化简后i+L<pp
        else
        {
            int j=pp-i+1;
            if(j<0) j=0;
            while (sb[j+1]==sb[i+j]&&i+j<=lenb) j++;
            p[i]=j;
            k=i;
        }
    }
    x=1;
    while (sa[x]==sb[x]&&x<=lenb) x++;//ex[1]并不具有确定性，所以我们暴力求出ex[1]
    ex[1]=x-1;
    k=1;
    for(int i=2;i<=lena;i++)
    {
```

```
int pp=k+ex[k]-1,L=p[i-k+1];
if(i+L<pp+1) ex[i]=L;
else
{
    int j=pp-i+1;
    if(j<0) j=0;
    while(sb[j+1]==sa[i+j]&&i+j<=lena&&j<=lenb) j++;
    ex[i]=j;
    k=i;
}
}
}
int main()
{
    scanf("%s%s",sa+1,sb+1);
    lena=strlen(sa+1);lenb=strlen(sb+1);
    exkmp();
    for(int i=1;i<lena;i++) printf("%d ",ex[i]);
    printf("%d\n",ex[lena]);
    return 0;
}
```

渺渺时空，茫茫人海，与君相遇，幸甚幸甚

分类： 字符串处理——EXKMP ， 算法导论

好文要顶 关注我 收藏该文





[Star_Feel](#)
关注 - 3
粉丝 - 7

00

[+加关注](#)

« 上一篇: [BZOJ3376: \[Usaco2004 Open\]Cube Stacking 方块游戏](#)
» 下一篇: [BZOJ3477: \[Usaco2014 Mar\]Sabotage](#)
posted @ 2017-11-03 08:52 Star_Feel 阅读(356) 评论(2) 编辑 收藏

评论列表

- #1楼

2018-07-16 11:01

huyufeifei

%%%讲的很详细，感谢！

支持(0) 反对(0)
- #2楼

[楼主]

2018-09-30 11:21

Star_Feel

@ huyufeifei
帮到你就好

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

- 注册用户登录后才能发表评论，请[登录](#)或[注册](#)，[访问网站首页](#)。
- 【推荐】超50万VC++源码: 大型组态工控、电力仿真CAD与GIS源码库！
- 【推荐】华为云11.11普惠季 血拼风暴 一促即发
- 【拼团】腾讯云服务器拼团活动又双叒叕来了！
- 【推荐】腾讯云新注册用户域名抢购1元起