几种简单的字符串算法

by dxw

讲课顺序

- hash，double hash
- trie&（kmp）
- ac自动机
- +exkmp
- 今天以讲知识点为主，题目为辅，题目数量可能不多

懂的大佬请自行出门右拐


我就觉得知识分子爱吹牛

Hash&&double hash

- 用途：hash在字符串中主要用于记录下字符串（子串）的信息以判断这个字符串（子串）是否出现过集出现的次数。对于一个不大的数字，我们可以用桶来解决数字重复问题，但对于字符串，我们能用什么信息来解决重复问题呢？这个时候hash应运而生。

hash简略描述

- 对于一段字符串，我们考虑将它转换成一个数字代表，为了使每种字符数量完全相同而排列顺序不同的字符串也同样被区分，我们可以采用k进制的方法将每个字符的ASCII码乘起来。例如：aba的ASCII码分别为97、98、99，我们可以变成0,1,0，然后按26进制排列再转换成10进制，就是$0*26^2+1*26+0=26$，而bcq按上述转换则变成737，aab则是1。然后将这些数字作为这个字符串独特的标记存到一定大小的桶里。
- 但假如这是一个长度为$10^{2,3,4,5}$长度的字符串，即使是longlong也存不下这么大的标记，所以我们将这些标记对某些数取模，然后再打入桶中。但也正因为取模，导致标记不在独一无二，因此模数的选择非常关键，通常为一些大质数，如1e9+7，998244353……

hash在桶中操作

- 有时hash取的模数会远大于桶的大小，这时我们通常把hash出的值x放在x%桶大小的位置，假如那个位置已经有数了，我们就往右移动，直到找到一个空位为止。所以建议同学们桶一定要开够，能开大尽量开大，桶大不仅可以存更多字符串，而且x%桶大小的位置有数的可能性也更低，程序运行的也更快（前提是不会炸空间）。
- 正常情况下单个hash就足够了，比起双hash能用20%的代码量解决80%的hash问题。

double hash

- 如前面所说，由于取模的原因，导致字符串的标记不再唯一，是的不同的字符串取模后也有可能拥有相同的标记，准确性有所下降。（P.S.而且有些出题人就好这口，专门设数据卡取特定模数的）所以，为了提高准确性（不被出题人AC坑成WA），双哈希应运而生。
- 顾名思义，双哈希就是hash两次，将一个字符串的标记x分别对两个模数取模，然后分别作为标记，两个个字符串的两个标记只有同时相同才能被认为是相同的字符串。

- 单hash

```
int hash(ll x){
    ll t=x%maxn;
    while (h[t] && h[t]!=x) t=(t+1)%maxn;
    return t;
}
int main(){
    for (i=1;i<=n;i++)
        x=(x*26+s[i]-97)%mo;
    t=hash(x);
    h[t]=x;
```

- 双hash

```
int hash(ll x,ll y){
    ll t=x%maxn;
    while (h[t][0] && !(h[t][0]==x && h[t][1]==y)) t=(t+1)%maxn;
    return t;
}
int main(){
    for (i=1;i<=n;i++){
        x=(x*26+s[i]-97)%mo;
        y=(y*26+s[i]-97)%mo1;
    }
    t=hash(x,y);
    h[t][0]=x;h[t][1]=y;
```

hash字符串中的子串

- 对于字符串中的某一个子串[i..j]，我们不需要重新把它扫描一遍求出hash值，而是只用处理出字符串的前缀hash值s[i]，对于[i..j]的hash值则等于$(s[j]-s[i-1]*26^{j-i+1}\%mo+mo)\%mo$，简单画画就懂啦~

例题

- 很少有hash作为单独考察点出现，所以你要我找一道只有hash算法的字符串题目我也很为难的……
- 大家下去手玩几个样例对对自己程序跑出来的结果就好啦

trie&KMP

为什么要先讲trie&kmp?

因为后面的ac自动机是建立在trie的基础上的kmp，扩展kmp也需要懂kmp

- 听说你们都会kmp，那就不复赘述，先做几道kmp的题吧.....

jzoj4886. 字符串

- 某日mhy12345在教同学们写helloworld，要求同学们用程序输出一个给定长度的字符串，然而发现有些人输出了一些“危险”的东西，所以mhy12345想知道对于任意长度n的小写字母字符串，不包含危险串的字符串个数
- 对于100%的数据，0<=|str|<=100,0<=n<=10000

- 直接dp就好了，设f[i][j]表示当前字符串长度为i，匹配了危险串的前j个位置，然后枚举26个字母转移。
- 时间复杂度O(str*n*26)。

JZOJ2964. Memory

- 话说某一天，CD在爬树的时候发现了树干上有一大串奇怪的符文，于是好奇的CD就从头到尾看了一遍。看完一遍以后CD觉得，最后一段文字好像很眼熟，好像在前面见过。对于CD来说，一开始看到的符文会印象特别深刻，而且这段符文要出现多次，CD才会觉得眼熟。
- 其实现在就是，CD会告诉你他看到的符文具体是什么，你要告诉CD，这段符文里最长的既是前缀又是后缀还在中间某个地方出现过（非前缀非后缀的出现）的最长的子串是什么。
- 对于100%的数据保证字符串长度≤106

- 我们发现KMP中的P数组可以处理出当前位置i的后缀与字符串的前缀的最长公共前缀长度。这就很好办了。我们在计算P的过程中可以顺带统计出字符串每个前缀在字符串中出现次数。然后从j=P[n]开始检验，若j的前缀出现次数小于2则j=p[j]。值得注意的是：由于长度为p[j]的前缀包括长度为p[p[j]]前缀，所以检验的时候要把p[j]的次数累加到p[p[j]]上。
- 时间复杂度O(N)

trie

- trie是一棵字典树，顾名思义，就是将一堆字符串存进一个树形的结构中，树上的每一条从根到叶子节点的链就是一条字符串。
- 每遇到一条字符串，我们从头到尾枚举s[i]，同时标记点x从根节点开始走起，一路按照该字符串的字符排列走下去，遇到没有的就新建点。好难讲啊，还是上代码比较合适……

```
for (i=1;i<=n;i++){
        scanf("%s\n",&s);x=0;
        t=strlen(s);
        for (j=0;j<t;j++){
            if (!f[x][s[j]-97])f[x][s[j]-97]=++num;
            x=f[x][s[j]-97];
        }
        g[x]++;
    }
```

- 很简单对吧，那我们就直接练习一下吧。
- 由于本人能力有限，见识短浅，只找到一道只考trie的题目，大家将就一下……

jzoj3126大LCP

- LCP就是传说中的最长公共前缀，至于为什么要加上一个大字，那是因为…你会知道的。
- 首先，求LCP就要有字符串。既然那么需要它们，那就给出n个字符串好了。
- 于是你需要回答询问大LCP，询问给出一个k,你需要求出前k个字符串中两两的LCP最大值是多少，这就是传说中的大LCP。
- 对于100%的数据,字符串总长度不超过10^6，1<=N,Q<=10^5.

- 我们可以用trie来解决这个问题。每插入一个字符串i，这个字符串第一个需要新建点的位置-1即为这个字符串与前i-1个字符串的最长LCP，用这个与前i-1个字符串的darkLCP进行比较即得到前i个字符串的darkLCP。
- 时间复杂度O(sigma s[i])。

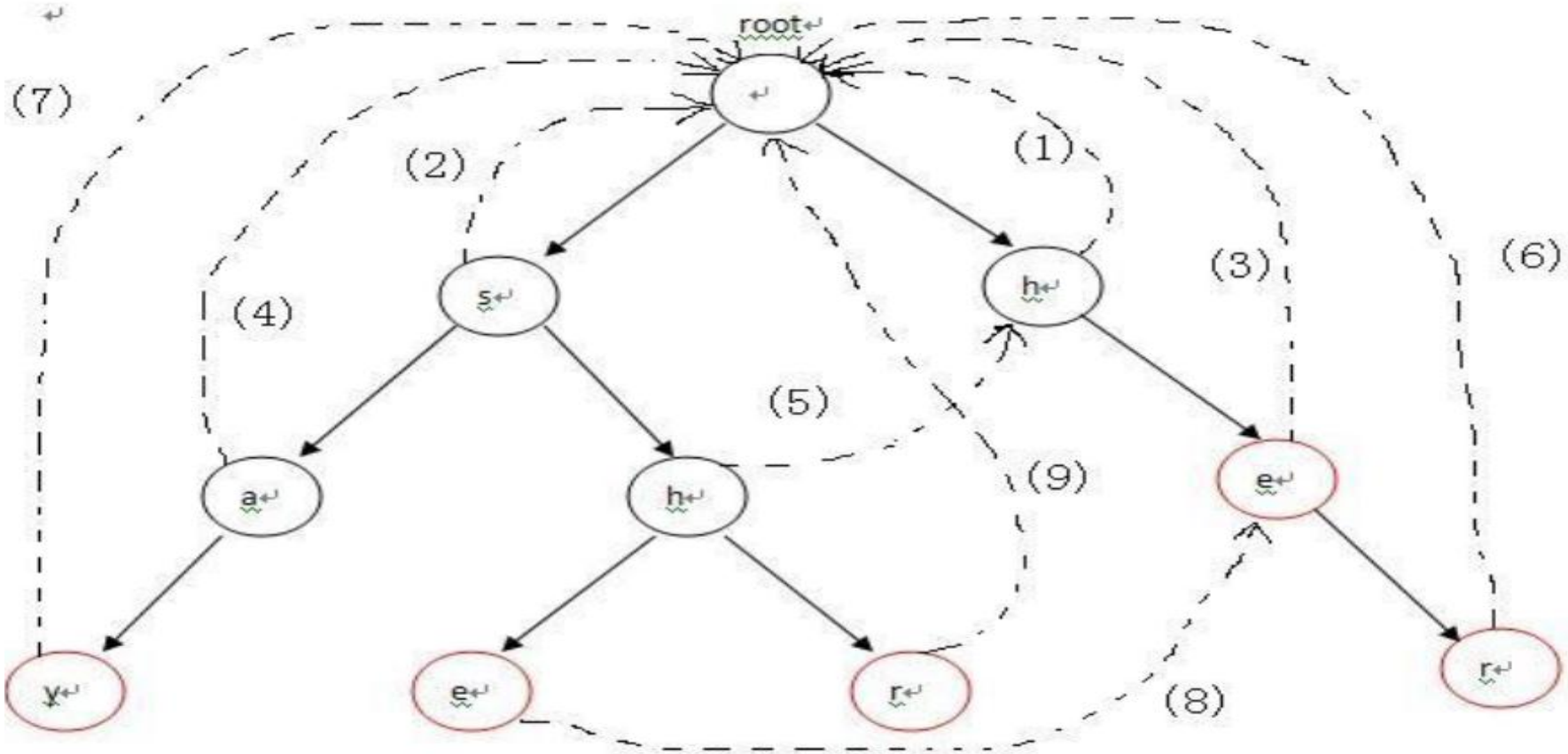
ac自动机

- kmp是一个匹配串在母串上寻求匹配，ac自动机本质上是在一颗trie上跑kmp，即一堆子串在母串上寻求匹配。
- 模仿kmp的思路，当母串的s[i+1]与某个匹配串sk的x+1位置才失配时，我们也同样希望不用进行从零开始的匹配，我们知道s[i-x+1..i]与sk[1..x]是相同的，若我们知道sk[1..x]的后缀与其它匹配串的前缀匹配，那此时匹配直接跳到另一个匹配串继续匹配即可。
- 由于ac自动机卓越的性能，它常用与在trie上dp转移相结合，经常还套上矩阵乘法。

构建

- 我们先在trie上跑bfs。对于一个点x，我们需要求出x的后缀与其它串的最长匹配前缀的点在trie上的位置p[x]。设x的父亲为y，显然y的后缀与根到p[y]的序列是匹配的。设指针为j=y，那么我们可以考虑根到p[y]的序列的是否存在下一位与x相匹配，若匹配，则p[x]=f[p[y]][x的字符]，否则我们就只能退而求其次，指针跳到j=p[y]。显然p[y]的后缀与根到p[p[y]]的序列是匹配的,那么我们可以考虑根到p[p[y]]的序列的是否存在下一位与x相匹配……直到匹配成功或指针跳回到根。
- 那么此时p[x]则为匹配成功的位置。由指针只会不断往深度浅的节点跳跃可知，p[x]的深度小于x。

构建出的p指针如图所示



构建代码

```
void bfs(){
    int i=0,j=1;
    while (i<j){
        x=v[++i];
        for (k=0;k<26;k++){
            if (!f[x][k]) continue;
            y=p[v[i]];z=f[x][k];
            while (!f[y][k] && y) y=p[y];
            p[f[x][k]]=f[y][k];c[f[x][k]]=max(c[f[x][k]],c[f[y][k]]);
            v[++j]=f[x][k];
        }
    }
}
```

匹配

- 构建好trie与P数组后，匹配时我们只需从头到尾枚举母串，同时一个指针在trie上移动，能匹配则下移，不能匹配则跳到该点的p[x]，直到匹配成功或跳到根节点。

```
for (i=1;i<=n;i++){
    while (!f[x][s[i]-97] && x) x=p[x];
    if (f[x][s[i]-97]){
        x=f[x][s[i]-97];
        if (c[x]) d[i-c[x]+1]++,d[i+1]--;
    }
}
```

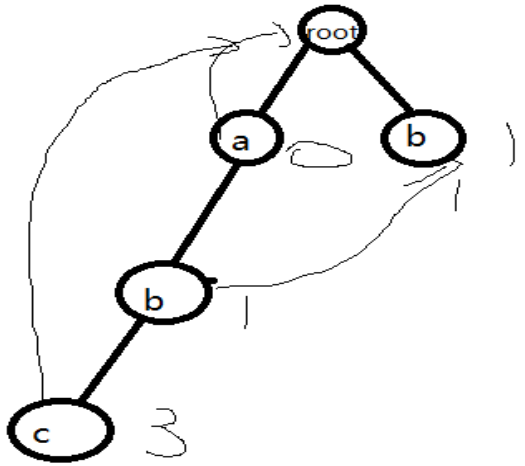
问题提问

- 对上述ac自动机构建及查询，大家有什么问题……
- 没有的话就做几道题练习一下。

JZOJ3172贴瓷砖

- A镇的主街是由N个小写字母构成，镇长准备在上面贴瓷砖，瓷砖一共有M种，第i种上面有Li个小写字母，瓷砖不能旋转也不能被分割开来，瓷砖只能贴在跟它身上的字母完全一样的地方，允许瓷砖重叠，并且同一种瓷砖的数量是无穷的。
- 问街道有多少字母(地方)不能被瓷砖覆盖。
- 街道长度N(1<=N<=300,000),瓷砖的种类M(1<=M<=5000)。
- 一种瓷砖长度为Li(1<=Li<=5000)，全部由小写字母构成。
- Time Limits: 4000 ms

- 先建出个ac自动机，在bfs过程中同时维护一个c[x]表示以从根到x的后缀为一种瓷砖的这种瓷砖的最长长度（假如这个x是一个字符串的结尾，那最长长度就是它的深度，否则c[x]=c[p[x]]）。（举个栗子：此时abc中的b
- 的c[]即为1,c中的为3。



- 然后我们拿母串在trie上跑自动机，当枚举到母串i指针x指到一个位置若它的c[x]不为0时我们在i-c[x]+1处打+1标记，在x+1处打-1标记，最后统计一下前缀和，为0的地方即为没被瓷砖覆盖到的地方。
- 时间复杂度O(N+sigma L[i])

jzoj4649项链

- 经过一番周折，Bob找到了Alice，为了安慰Alice惊魂未定的心，Bob决定给Alice买一条手链，这条手链由M个珍珠组成，每个珍珠上刻着不同的小写字母。当Alice看到一些字母按照一定的顺序排列成的字符串时，就会产生一定的愉悦值。Bob现在可以在这M个珍珠上刻上字母，现在他想知道，如何刻字母可以使得Alice的愉悦值最大。
- N<=200，所以字符串的总长度<=200，M<=10^14
- PS：字符串只含小写字母，可能有重复 Time Limits: 6000 ms

- 建出个自动机，设f[i][j]表示当前做到长度为i的珍珠，在自动机上的j节点的最大偷税值，显然可以枚举26个字母将f[i][j]转移到f[i+1][x]那这样复杂度是O(26*M*sigmas[i])，能过40%数据
- 我们发现i到i+1的转移并不随i的变化而变化，因此我们可以构建出一个转移矩阵，用矩阵乘法优化dp，时间复杂度$O((sigmas[i])^3*\log_2 M)$。

一道相似的题jzoj5078魔法咒语

Chandra 是一个魔法天才。

从一岁时接受火之教会洗礼之后，Chandra 就显示出对火元素无与伦比的亲和力，轻而易举地学会种种晦涩难解的法术。这也多亏 Chandra 有着常人难以企及的语言天赋，让她能轻松流利地说出咒语中那些极其拗口的魔法词汇。

直到十四岁，开始学习威力强大的禁咒法术时，Chandra 才遇到了障碍。

根据火之魔法规则，禁咒的构成单位是 N 个基本词汇。施法时只要凝聚精神力，说出一段用这些词语组成的长度恰好等于 L 的语言，就能释放威力超乎想象的火法术。过去的魔法师们总结了几种表达起来最连贯的组合方式，方便施法者以最快语速完成法术。

但具有魔法和语言双重天才的 Chandra 不满足于这几种流传下来的禁咒，因为她可以毫无困难地说出普通人几乎不可能表达的禁咒语句。然而，在实际施法时，Chandra 发现有些自创禁咒念出后不但没有预期效果，反而会使自己的精神力迅速枯竭，十分难受。

这个问题令 Chandra 万分不解。她大量阅读典籍，到处走访魔法学者，并且不顾精神折磨一次又一次尝试新咒语，希望找出问题的答案。

很多年过去了，在一次远古遗迹探险中，Chandra 意外闯进了火之神艾利克斯的不知名神殿。根据岩土特征分析，神殿应该有上万年的历史，这是极其罕见的。Chandra 小心翼翼地四处探索，沿着魔力流动来到一间密室。她看见密室中央悬浮着一本书籍。在魔法保护下书籍状况完好。精通上古语言的 Chandra 读过此书，终于解开了多年的困惑。

禁咒法术之所以威力强大，是因为咒语借用了火之神艾利克斯的神力。这本书里记载了艾利克斯生平忌讳的 M 个词语，比如情敌的名字、讨厌的植物等等。使用禁咒法术时，如果语言中含有任何忌讳词语，就会触怒神力而失效，施法者也一并遭受惩罚。

例如，若 "banana" 是唯一的忌讳词语，"an"、"ban"、"analysis" 是基本词汇，禁咒长度须是 11，则"bananalysis" 是无效法术，"analysisban"、"anbanbanban" 是两个有效法术。注意：一个基本词汇在禁咒法术中可以出现**零次、一次或多次；只要组成方式不同**就认为是不同的禁咒法术，即使书写形式相同。

谜题破解，Chandra 心情大好。她决定计算一共有多少种有效的禁咒法术。

由于答案可能很大，你只需要输出答案模 1,000,000,007 的结果。

本题一共有 10 个测试点。

下表是每个测试点的数据规模和约定：

#1	N = 5	M = 5	L ≤ 10
#2	N = 10	M = 1	L ≤ 50
#3	N = 20		
#4	N = 20	M = 20	L ≤ 100
#5	N = 40	M = 10	
#6	N = 50	M = 50	
#7	N = 10	M = 2	基本词汇长度不超过 1
#8	N = 26	M = 10	
#9	N = 20	M = 10	基本词汇长度不超过 2
#10	N = 50	M = 20	

对于 100%的数据，$1 \le N, M \le 50$，$1 \le L \le 10^8$，基本词汇的长度之和不超过 100，忌讳词语的长度之和不超过 100。保证基本词汇不重复，忌讳词语不重复。

- Time Limits： 2000ms

- 与上题相同，我们建出自动机并设出f[i][j]。但由于基本词汇的缘故，f[i][j]转移的目标为f[i+s[i]][x]，这个就搞不了矩阵乘法了。
- 但我们发现前60%的数据L<=100,这个可以直接用dp做，时间复杂度O(N*L*sigmas1[i])
- 后40%的数据基本词汇的长度<=2,这说明f[i][j]最多转移到f[i+2][x]，这个还是可以用矩阵搞搞事情的，我们的矩阵存下f[i],f[i-1]的信息，每次用f[i],f[i-1]的信息转移出f[i+1]，这样就可以在$O(logL*(2*sigmas1[i])^3)$内解决问题。
- 现在知道我为什么把数据另起一页了吧！敲黑板！！！（~~才，才不是页面太大放不下呢。~~）

jzoj2834喵星球上的点名

- a180285幸运地被选做了地球到喵星球的留学生。他发现喵星人在上课前的点名现象非常有趣。假设课堂上有N个喵星人，每个喵星人的名字由姓和名构成。喵星球上的老师会选择M个串来点名，每次读出一个串的时候，如果这个串是一个喵星人的姓或名的子串，那么这个喵星人就必须答到。然而，由于喵星人的字码过于古怪，以至于不能用ASCII码来表示。为了方便描述，a180285决定用数串来表示喵星人的名字。
- 现在你能帮助a180285统计每次点名的时候有多少喵星人答到，以及M次点名结束后每个喵星人答到多少次吗？
- 1≤N≤20000，1≤M≤50000，喵星人的名字总长和点名串的总长分别不超过100000 保证喵星人的字符串中作为字符存在的数不超过10000。
- Time Limits: 5000 ms Memory Limits: 131072 KB

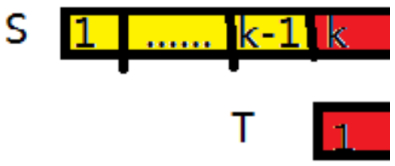
- 首先，考虑点名串建自动机。但现在问题来了，喵星人的作为字符存在的数为1e4，直接开数组估计(肯定)要炸，那怎么办？开map啊！！！然后正常跑一遍自动机构建出p数组，同时构建出c[x]表示以根到x后缀为命名串的数量。
- 然后拿姓名串s[i]去跑自动机，当跑到一个点x，设y=x，若此时这个点的c[y]不为0，且没被s[i]标记过，我们就将它头上的点名串的答案均+1，然后跳到p[y]再判断有没有被标记，重复以上操作直到c[y]=0或y被i标记过。同时统计出i的答案。
- 时间复杂度O(sigma s[i]+sigmas1[i]+ans)

扩展kmp

- 目标解决：给定串S，T，求出S的任意位置i的前缀[i..len]与T的最长公共前缀。
- 时间复杂度:线性。
- 既然叫扩展那他自然与kmp一脉相承，借鉴了kmp的思想，同时又有了新的突破与加深。

- 设extend[i]表示S[i..n]与T的最长公共前缀。
- 设现在extend[1..k-1]均算好，在之前i+extend[i]-1的最大值为p，即s之前最远匹配到的位置，同时设i+extend[i]-1为最大值p的i为a。
- 我们知道s[a..p]=t[1..p-a+1]
- 那么s[k..p]=t[k-a+1..p-a+1]
- 假如我知道t[k-a+1..m]与t的最长公共前缀就好了……
- 于是，设next[i]表示t[i..n]与t的最长公共前缀。

- 假如k+next[k-a+1]-1<p
- 由s[k..p]=t[k-a+1..p-a+1]可知，
- s[k..k+next[k-a+1]-1]=t[k-a+1..k-a+next[k-a+1]]=t[1..next[k-a+1]]
- s[k+next[k-a+1]]=t[k-a+1+next[k-a+1]]
- 由next为最长前缀可知,
- s[k+next[k-a+1]]=t[k-a+1+next[k-a+1]]≠t[next[k-a+1]+1]
- 故extend[k]=next[k-a+1]。如图，设L=next[k-a+1]。



- 假如k+next[k-a+1]-1>=p
- 由于s[k..p]=t[k-a+1..p-a+1]，我们只知道s[k..p]=t[1..p-k+1]并不知道s[p+1..k+next[k-a+1]-1]是否与t[p-a+2..k-a+next[k-a+1]]相等，这时候就要强行去匹配了。但我们不需要从k开始，而是从p+1开始去匹配，得到extend[k],同时我们更新p与a，p=k+extend[k]-1,a=k。
- 就不放图了，作图好累啊……
- 由于t和s的每个位置都只会被扫一遍，所以时间复杂度是线性的。
- next数组的处理与extend的处理相同（这不是和kmp一个道理吗！）

题目

- 找了半天找不到什么exkmp的题目，能力有限，搜肠刮肚也只找到一道,大家凑合着练练手。
- 有兴趣的同学可以课后自己找题做。

JZOJ3648 beyond

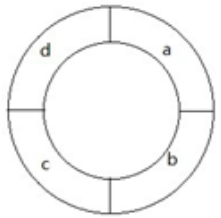
Jodie 慢慢地步入实验室，跟随在她身旁的灵体 Aiden 似乎有点不高兴，但还是形影不离地跟随着 Jodie。

今天 Jodie 要进行的实验在一个很大很大的圆环上面，圆环上有 L 个格子，每个格子上都显示着一个小写英文字母，Jodie 从**任意**格子开始当她离开一个格子的时候那个格子的字母就会改变，这个改变是随机的，没有人知道会变成什么。Jodie 在这个环上不回头顺时针地走，每进入一个格子就会在本子上写下这个格子当前显示的字母。由于 Jodie 不能回头而且不知道这个圆环上有多少个格子，她并不知道自己什么时候会走到重复的点，所以她让 Aiden 在她下一步走进重复格子的时候提醒一下。但可能他们闹了矛盾，Aiden 发了脾气，决定在 Jodie 走了 K（K >= 0）步重复的格子之后才告诉她。Jodie 进行了两次实验，记录了两次走的路径。第二次实验再进去之前，每个格子所显示的字母会被重设为第一次实验开始前的样子。Jodie 发现了 Aiden 的恶作剧，她只能把可能的最大的 L 告诉实验人员。

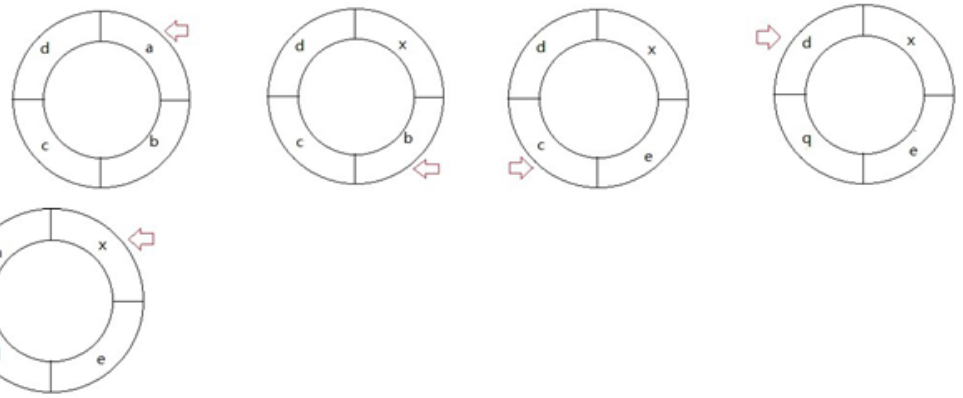
为了帮助你更好的理解题目，请仔细分析一下例子：

假设 L = 4，K = 1

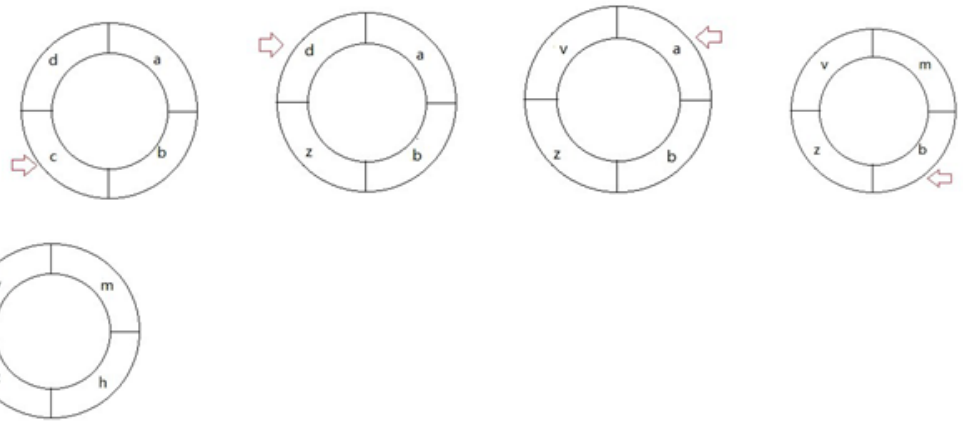
第一次实验开始前每个格子显示的字母如下图：



Jodie 从显示字母为'a'的格子开始走，Aiden 在她走了 K 步重复的格子之后告诉她停止，所以 Jodie 一共走了 5 步，每走一步，格子的变化如下（箭头指着 Jodie 所在的格子）：



Jodie 的第二次实验从显示字母为'c'的格子开始走，每走一步格子的变化如下（箭头指着 Jodie 所在的格子）：



Jodie 两次实验记录的路径分别为：

"abcdx"

"cdabz"

现在给出 Jodie 记录的两次路径的长度 N，以及 Jodie 所写的内容，但是并不知道 K 是多少，希望你能帮忙求出一个最大的可能的 L。

1 <= N <= 2,000,000

- 由题意得，s的某个位置i,若s[i..j]与t[1..j-i+1]是匹配的t[j-i+2..j]与s[1..i-1]是匹配的，那么就意味着L可能为j。这果断exkmp。
- 我们正着做一遍exkmp，然后交换s,t，再做一遍exkmp，得到ex[],ex1[],分别表示s[i..n]的前缀对t的extend和t[i..n]的前缀对s的extend。
- 然后我们枚举i，在[1..ex[i]]中找到一个j使得ex1[j]>=i-1,那么答案就是i+j-1。显然j是符合条件中的越大越好。我们知道，随着i的递增，若一个ex1[j]<i-1，那他永远不可能出现ex1[j]>=i-1的情况，所以我们可以维护一个并查集，每次从ex[i]往前跳，遇到ex1[j]<i-1就把它合并掉，直到跳到一个合法的j。
- 时间复杂度O(N)。

END

- 内容基础
- 题目很少
- 幻灯片简单
- 相信并希望今天大家过得都很愉快
- 谢谢大家

你以为讲完了吗

- 我也以为完了，结果还要我多讲一个马拉车
- 那就继续吧。

manachar

- manachar主要用于求以某个点为对称点的最长回文串。
- 它的主要思路与exkmp基本相同，都是保存一个最远点，用已知求未知。

构建

- 我们设rad[i]表示以i为对称点的最长回文串。
- 设现在rad[1..k-1]均已算好，以[1..k-1]为对称点的回文串最远匹配到的位置为p，对称点为a，即a+rad[a]-1=p
- 现在需要计算rad[k]，假如k>=p，那就暴力更新。
- 假如k<p，由回文的性质我们知道,s[a+1..p]=s[a-1..2*a-p]（倒序），那么对于k位置的rad，他可以参照2*a-k位置的rad。
- 若k+rad[2*a-k]-1<p,那由于s[k+rad[2*a-k]]=s[2*a-k-rad[2*a-k]],s[k-rad[2*a-k]]=s[2*a-k+rad[2*a-k]],s[2*a-k-rad[2*a-k]]≠s[2*a-k+rad[2*a-k]],所以s[k+rad[2*a-k]]≠s[k-rad[2*a-k]]。此时rad[k]=rad[2*a-k]。

- 若k+rad[2*a-k]-1>=p,那由前面的推倒可知，此时rad[k]>=rad[2*a-k]，但至于rad[k]最大能到多少，我们并不知道，所以我们需要暴力匹配，但与从零开始的暴力匹配不同，我们可以从rad[k]+1开始匹配，然后更新a和p。
- 由于每个点只会被访问一次，所以时间复杂度是O(N)的。

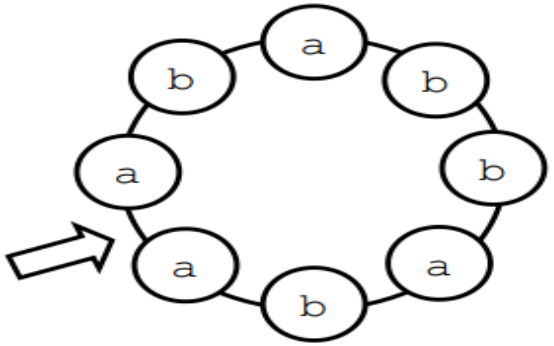
- 由rad的计算方式可知，这种方法只能计算有单一对称中心（即长度为奇数）的回文串，那偶数的回文串怎么办？
- 我们可以再相邻字符间插入一个字符串里没出现过的特殊符号，那长度为偶数的回文串相当于以这些特殊符号为对称中心的长度为奇数的回文串。

jzoj4002项链

强迫症非常麻烦，尤其是买东西的时候。冠璐女神的生日快到了，某强迫症要送一条项链送给她，重点是，这条项链要对称，这条项链要对称，这条项链要对称。（很重要所以说三遍）

项链都是环状的（废话...），每条项链上有颜色不同的珍珠，用小写字母 a-z 代表不同的颜色。一条项链是对称的，是指从项链的某个位置开始，两个方向看过去是一样的（珍珠的颜色一样）。

比方说，如图所示的项链（第一个输入样例），从箭头位置开始看，无论是顺时针还是逆时针，看到的都是 ababbaba，因此这条项链是对称的。



有的项链非常长，很难看出它们的对称性。你能快速地判断哪些项链是对称的吗？

- 对于 100%数据， 1≤项链≤5，项链长度≤10^5.

- 对每条项链复制一遍接到项链后，跑一遍马拉车，若出现回文串长度大于等于n则合法。
- 时间复杂度O(N)

JZOJ4793. 妮厨的愤怒

- 栋栋和标标都是厨力++的妮厨。俗话说“一机房不容二厨”，他们两个都加入了某OI( )交流♗( )群，在钦定老婆的时候出现了偏差，于是闹得不可开交。可是栋栋是群内的长者，斗权限标标斗不过他。
- 于是标标单方面找到了LL仲裁庭，还帮栋栋出了律师的钱，要求按基本法来判定。法官点点喝了口果汁，仔细审查了案子，说中央资瓷栋栋连任，这是最吼的；标标还naive，不要总想着搞一个大新闻，像那个南海某岛国一样。
- 标标不服，要到新日暮里和栋栋进行男人间的决斗♗。栋栋接住了标标丢去的蕾姆，并提出了一个问题：
- 给定一个长度为n的字符串s，给出 q个询问，每次询问子串S[l..r] 的最长回文子串长度。字符串下标从0开始。
- 标标被难住了，被禁言的他决定向你求助。（以下内容为无意义灌水，请要怒 本次比赛的神犇跳过。）
- 如果这是galgame，那么轮到你选选项的时候了！
- A.不帮并获得本题 分
- B.帮助并被栋栋禁言
- C.宣称自己也是妮厨与他们两个决斗
- 哪来的C选项啊QwQ
- 对于100% 的数据，满足1<=n ,q<=10^5，0<=l<=r<n ，s只包含小写拉丁字母。
- 其中最后一档部分分有一个数据点只存在前两个小写字母，有一个数据点只存在前五个小写字母。

- 跑一下马拉车。
- 对于每个询问，二分答案，然后判断[l+mid-1,r-mid+1]之间是否存在rad[x]>=mid。这个用ST表搞搞就好
- 时间复杂度O(NlogN)。

这回是真完了

- 谢谢大家！！！