目录

**[STL](#_Toc23001_WPSOffice_Level1)** **[3](#_Toc23001_WPSOffice_Level1)**

[1.<Vector>](#_Toc17722_WPSOffice_Level1) [3](#_Toc17722_WPSOffice_Level1)

[2. <List>](#_Toc12633_WPSOffice_Level1) [3](#_Toc12633_WPSOffice_Level1)

[3. <deque>](#_Toc1415_WPSOffice_Level1) [4](#_Toc1415_WPSOffice_Level1)

[4.<map>](#_Toc13073_WPSOffice_Level1) [5](#_Toc13073_WPSOffice_Level1)

[5.<multimap>](#_Toc4389_WPSOffice_Level1) [5](#_Toc4389_WPSOffice_Level1)

[6.<unordered\_map>](#_Toc21505_WPSOffice_Level1) [5](#_Toc21505_WPSOffice_Level1)

[7.<set>](#_Toc149_WPSOffice_Level1) [6](#_Toc149_WPSOffice_Level1)

[8.<multiset>](#_Toc13842_WPSOffice_Level1) [6](#_Toc13842_WPSOffice_Level1)

[9.<String>](#_Toc21466_WPSOffice_Level1) [6](#_Toc21466_WPSOffice_Level1)

[10.<bitset>](#_Toc12281_WPSOffice_Level1) [7](#_Toc12281_WPSOffice_Level1)

[11.<queue>](#_Toc27308_WPSOffice_Level1) [8](#_Toc27308_WPSOffice_Level1)

[12.<stack>](#_Toc16146_WPSOffice_Level1) [8](#_Toc16146_WPSOffice_Level1)

[13.<priority\_queue>](#_Toc8022_WPSOffice_Level1) [8](#_Toc8022_WPSOffice_Level1)

[14. <algorithm>](#_Toc25943_WPSOffice_Level1) [9](#_Toc25943_WPSOffice_Level1)

**[字符串](#_Toc8538_WPSOffice_Level1)** **[10](#_Toc8538_WPSOffice_Level1)**

[1.Kmp](#_Toc15341_WPSOffice_Level1) [10](#_Toc15341_WPSOffice_Level1)

[2.马拉车](#_Toc22093_WPSOffice_Level1) [12](#_Toc22093_WPSOffice_Level1)

[3.拓展KMP](#_Toc1808_WPSOffice_Level1) [13](#_Toc1808_WPSOffice_Level1)

[4编辑距离](#_Toc31376_WPSOffice_Level1) [15](#_Toc31376_WPSOffice_Level1)

[5.Karp-Rabin算法](#_Toc15975_WPSOffice_Level1) [15](#_Toc15975_WPSOffice_Level1)

[6.AC自动加+trie 树](#_Toc18637_WPSOffice_Level1) [16](#_Toc18637_WPSOffice_Level1)

[7.查找母串中各单词出现次数](#_Toc21774_WPSOffice_Level1) [18](#_Toc21774_WPSOffice_Level1)

[8.最短公共祖先](#_Toc26209_WPSOffice_Level1) [21](#_Toc26209_WPSOffice_Level1)

[9.字符串Hash](#_Toc9941_WPSOffice_Level1) [23](#_Toc9941_WPSOffice_Level1)

[10.Sunday](#_Toc1055_WPSOffice_Level1) [24](#_Toc1055_WPSOffice_Level1)

[11.后缀数组](#_Toc23682_WPSOffice_Level1) [25](#_Toc23682_WPSOffice_Level1)

[12.后缀自动机](#_Toc24071_WPSOffice_Level1) [26](#_Toc24071_WPSOffice_Level1)

**[数论](#_Toc26394_WPSOffice_Level1)** **[29](#_Toc26394_WPSOffice_Level1)**

[1.Miller-Rabin素性测试--快速求一个数是否是素数-应用见例题](#_Toc23022_WPSOffice_Level1) [29](#_Toc23022_WPSOffice_Level1)

[2.杜教筛](#_Toc29308_WPSOffice_Level1) [30](#_Toc29308_WPSOffice_Level1)

[3.log(n)复杂度分解素因子](#_Toc17447_WPSOffice_Level1) [32](#_Toc17447_WPSOffice_Level1)

[4.基数排序MSD](#_Toc29141_WPSOffice_Level1) [32](#_Toc29141_WPSOffice_Level1)

[5.矩阵快速幂](#_Toc25258_WPSOffice_Level1) [34](#_Toc25258_WPSOffice_Level1)

[6.莫比乌斯表](#_Toc15545_WPSOffice_Level1) [35](#_Toc15545_WPSOffice_Level1)

[7.逆元模板](#_Toc6742_WPSOffice_Level1) [36](#_Toc6742_WPSOffice_Level1)

[8.欧拉函数](#_Toc10713_WPSOffice_Level1) [38](#_Toc10713_WPSOffice_Level1)

[9.素数筛](#_Toc32559_WPSOffice_Level1) [39](#_Toc32559_WPSOffice_Level1)

[10.拓展欧几里得及欧几里得-------求解](#_Toc15547_WPSOffice_Level1) [41](#_Toc15547_WPSOffice_Level1)

[11拓展欧几里得求整数解个数](#_Toc12251_WPSOffice_Level1) [41](#_Toc12251_WPSOffice_Level1)

[12.整除分块](#_Toc7586_WPSOffice_Level1) [43](#_Toc7586_WPSOffice_Level1)

[13组合数](#_Toc4383_WPSOffice_Level1) [43](#_Toc4383_WPSOffice_Level1)

[14.最长循环节](#_Toc11702_WPSOffice_Level1) [44](#_Toc11702_WPSOffice_Level1)

[15.高精度模拟乘法--实战](#_Toc23670_WPSOffice_Level1) [45](#_Toc23670_WPSOffice_Level1)

[16.N的大数次方](#_Toc17820_WPSOffice_Level1) [46](#_Toc17820_WPSOffice_Level1)

[17. 博弈论](#_Toc24876_WPSOffice_Level1) [48](#_Toc24876_WPSOffice_Level1)

[18. 线性基](#_Toc9186_WPSOffice_Level1) [51](#_Toc9186_WPSOffice_Level1)

**[数据结构](#_Toc3138_WPSOffice_Level1)** **[53](#_Toc3138_WPSOffice_Level1)**

[1.线段树](#_Toc12171_WPSOffice_Level1) [53](#_Toc12171_WPSOffice_Level1)

**[其他](#_Toc9896_WPSOffice_Level1)** **[56](#_Toc9896_WPSOffice_Level1)**

[1.对数器](#_Toc31704_WPSOffice_Level1) [56](#_Toc31704_WPSOffice_Level1)

[2.位运算加速](#_Toc29797_WPSOffice_Level1) [58](#_Toc29797_WPSOffice_Level1)

[3.输入外挂](#_Toc27477_WPSOffice_Level1) [58](#_Toc27477_WPSOffice_Level1)

[4.BM杜教-解决线性递推](#_Toc18927_WPSOffice_Level1) [60](#_Toc18927_WPSOffice_Level1)

[5.重载运算符](#_Toc29780_WPSOffice_Level1) [62](#_Toc29780_WPSOffice_Level1)

## STL

### 1.<Vector>

//将vec2拷贝进vec

vec = vec2;

//返回容器的最后一个元素

vec.back();

//返回容器的最后元素逆迭代器

vec.rbegin();

//返回容器的起始位置的逆迭代器

vec.rend();

//在插入位置插入若干元素 数量可选参数

vec.insert(插入位置迭代器, [数量], 元素);

//在vec的插入位置插入[vec2首迭代器, vec2尾迭代器)范围内元素

vec.insert(vec插入位置迭代器, vec2首迭代器, vec2尾迭代器);

//删除[首迭代器, 尾迭代器)范围内元素

vec.earse(首迭代器, [尾迭代器]);

//常数交换两vector元素

vec.swap(vec2);

//判断vec和vec2是否相等 不等!= 比较字典序< > <= >=

vec == vec2;

### <List>

//将ls2拷贝进ls

ls = ls2;

//返回容器第一个元素

ls.front();

//返回容器最后一个元素

ls.back();

//判断容器是否为空 常数复杂度

ls.empty();

//返回容器元素个数 线性复杂度

ls.size();

//将链表ls2合并进ls 要求两链表升序 合并后ls2清空

ls.merge(ls2);

//常数时间将ls2连接至ls指定位置

ls.splice(位置迭代器, ls2);

//移除所有指定的元素 remove\_if(判别函数)移除判别成功的元素

ls.remove(元素);

//将链表反转 线性复杂度

ls.reverse();

//将链表除重 要求有序链表

ls.unique();

//对链表快排

ls.sort();

//将迭代器向后移动若干次 可以为负 无随机访问容器线性复杂度

advance(迭代器, 次数);

//返回两个迭代器之间的距离 无随机访问容器线性复杂度

distance(迭代器, 迭代器2);

### <deque>

//访问指定位置的元素 常数复杂度

dq[位置];

//返回容器元素数量 常数复杂度

dq.size();

//将容器末尾的元素删除 常数复杂度

dq.pop\_back();

//在容器头部插入一个元素 常数复杂度

dq.push\_front();

//将容器头部的元素删除 常数复杂度

dq.pop\_front();

### 4.<map>

//返回对应键值的元素引用 无元素则创建

mp[键值];

//返回容器的元素数量 常数复杂度

mp.size();

//插入一个新元素 不使用C++11则make\_pair(键值, 元素);

mp.insert({ 键值, 元素 });

//删除[首迭代器, 尾迭代器)范围内元素

mp.earse(首迭代器, [尾迭代器]);

//查找对应键值是否出现 对数复杂度

mp.count(键值);

//返回对应键值元素的迭代器 没有精确匹配返回结束位置

mp.find(键值);

//返回第一个大于等于键值的元素迭代器

mp.lower\_bound(键值);

//返回第一个大于键值的元素迭代器

mp.upper\_bound(键值);

### **5.<multimap>**

//插入一个新元素

mp.insert({ 键值, 元素 });

//返回对应键值元素出现的个数 线性复杂度

mp.count(键值);

### **6.<unordered\_map>**

//需要引用头文件#include <unordered\_map> 无lower\_bound和upper\_bound其它同map

### **7.<set>**

//插入一个新键值

st.insert(键值);

//set其它操作同map 操作单键值

### **8.<multiset>**

//操作同multimap 操作单键值

<unordered\_set>

//需要引用头文件#include <unordered\_set> 无lower\_bound和 upper\_bound其它同set

### **9.<String>**

//返回s是否为空串 size复杂度不定

s.empty();

//在s尾部添加若干元素

s.append(数量, 元素);

//在s尾部增加s2的串

s += s2;

//返回一个从指定位置开始指定长度的字符串副本

s.substr(位置, 长度);

//从指定位置开始查找s2串的出现位置 未找到返回-1

s.find(s2, [位置]);

//反向查找字符串s到指定位置返回出现位置 位置为正向位置

s.rfind(s2, [位置]);

//从输入流读取一行字符到字符串

getline(输入流, s);

//字符串转换为整形 浮点型stod

stoi(s, [0], [进制]);

//整形/浮点型转字符串

s = to\_string([数值]

//字符串反转

reverse(s.begin(),s.end());

//截取string字符串i-j子串给s1

s1=s.substr(i,j)

//把整数变成字符串存到s里面

stringstream ss;

ss<<n;

ss>>s;

### **10.<bitset>**

//将容器内元素设置为指定整形的二进制

bt = 整形;

//判断两容器是否相等 不等!=

bt == bt2;

//访问容器指定位的元素

bt[位];

//判断是否所有位都为1

bt.all();

//判断是否有任意一位为1 都不为1bt.none();

bt.any();

//返回容器被设置为1的数量

bt.count();

//容器进行二进制运算&、|、^、~、<<、>>、&=、|=、^=、<<=、>>=

bt & bt2;

//将容器所有位设置为1

bt.set();

//将容器所有位设置为0

bt.reset();

//反转容器所有位的01

bt.flip();

//将容器内01转换为字符串并返回

bt.to\_string();

//将容器内的元素转换为十进制的unsigned long long并返回

bt.to\_ullong();

### **11.<queue>**

//返回两容器是否相等 不等!= 比较字典序< > <= >=

q == q2;

//返回容器最后一个元素

q.back();

//返回容器内元素个数 常数复杂度

q.size();

### **12.<stack>**

//返回容器内元素个数 常数复杂度

stk.size();

### **13.<priority\_queue>**

//返回队首元素

q.top();

//自定义排序

struct pqcmp

{

bool operator () (const int &元素, const int &元素2) const

{

return 比较方式;

}

};

### **<algorithm>**

//将[首迭代器, 尾迭代器)范围内元素初始化为对应值

fill(首迭代器, 尾迭代器, 值);

//将容器首迭代器位置的元素左旋至位置迭代器 线性复杂度

rotate(首迭代器, 位置迭代器, 尾迭代器);

//将范围内元素随机排列 srand初始化随机种子

random\_shuffle(首迭代器, 尾迭代器);

//对范围内元素进行稳定排序 NlogN^2

stable\_sort(首迭代器, 尾迭代器);

//将位置迭代器对应大小的值准确定位在对应位置上 保证左侧元素小于它 右侧元素大于等于它 线性复杂度

nth\_element(首迭代器, 位置迭代器, 尾迭代器);

//合并容器和容器2至容器3 要求两容器升序 线性复杂度

merge(容器首迭代器, 容器尾迭代器, 容器2首迭代器, 容器2尾迭代器, inserter(容器3, 容器尾迭代器);

//判断容器2是否是容器的子集 要求两容器升序

includes(容器首迭代器, 容器尾迭代器, 容器2首迭代器, 容器2尾迭代器);

//容器元素-容器2元素存入容器3 要求两容器升序

set\_difference(容器首迭代器, 容器尾迭代器, 容器2首迭代器, 容器2尾迭代器, inserter(容器3, 容器尾迭代器);

//将容器和容器2交集存入容器3 要求两容器升序 线性复杂度

set\_intersection(容器首迭代器, 容器尾迭代器, 容器2首迭代器, 容器2尾迭代器, inserter(容器3, 容器尾迭代器);

//容器元素^容器元素2存入容器3 要求两容器升序 线性复杂度

set\_symmetric\_difference(容器首迭代器, 容器尾迭代器, 容器2首迭代器, 容器2尾迭代器, inserter(容器3, 容器尾迭代器);

//合并容器和容器2至容器3 要求两容器升序 线性复杂度

set\_union(容器首迭代器, 容器尾迭代器, 容器2首迭代器, 容器2尾迭代器, inserter(容器3, 容器尾迭代器);

//返回容器中最大的元素 最小min\_element 线性复杂度

max\_element(首迭代器, 尾迭代器);

//将容器内元素调整为全排列的前一个状态

prev\_permutation(首迭代器, 尾迭代器);

//从data中查找第一个不小于K的的位置，返回的是一个迭代器，cmp为比较优先级不填就默认比大小

lower\_bound(data.begin(),data.end(),k,cmp)

//从data中查找第一个大于K的位置，返回的是一个迭代器

upper\_bound(data.begin(),data.end(),k.cmp)

//从data中查找是否存在k，返回一个bool型

binary\_search(data.begin(),data.end(),k)

## 字符串

### 1.Kmp

/\*

定义一个变量 j = i - next[i] 就是next数组下标和下标对应值的差，

如果这个差能整除下标 i，即 i%j==0 ,则说明下标i之前的字符串（周期性字符串长度为 i）

一定可以由一个前缀周期性的表示出来，这个前缀的长度为刚才求得的那个差，

即 j，则这个前缀出现的次数为 i/j 。所以最后输出i和i/j即可。

\*/

/\*

利用KMP的next数组性质；如果s[next[n-1]]=s[n],则此时前后缀相同，

然后再开始回滚，若s[next[n-1]] == s[n-1]，则子串s[0,1,2,...,next[n-1]]是满足条件的子串。

然后判断s[next[next[n-1]]] == s[n-1]是否成立，

这样一直回滚，直到next[next[.....next[n-1]]] == -1为止

\*/

#include<iostream>

#include<stdio.h>

#include<string.h>

using namespace std;

char s[1000005],t[200000];

int slen,tlen;

int nex[200000];//nex数组大小和短串一致

int ans,a,b,c,d,n,m;

inline void get\_nex()

{

int j=-1;//j初始化为-1

for (int i=0;i<tlen;i++){

while (t[i]!=t[j+1] && j!=-1)//如果下一个不同，那么j就变成next[j]，注意next[j]是小于j的，无论j取任何值

j=nex[j];//往前回溯

if (t[i]==t[j+1] && i!=0) j++;//如果相同，j++

nex[i]=j;//这个是把算的j的值（就是相同的最大前缀和最大后缀长）赋给next[i]

}

}

inline void kmp()

{

int j=-1;

for (int i=0;i<slen;i++){

while (s[i]!=t[j+1] && j!=-1)

j=nex[j];

if (s[i]==t[j+1])

j++;

if (j==tlen-1)

ans++,j=nex[j];

}

}

int main()

{

scanf("%d",&n);

for (int i=1;i<=n;i++){

ans=0;

scanf("%s %s",t,s);

slen=strlen(s);

tlen=strlen(t);//这两个长度应该设为全局变量最开始时求出，不能用一次求一次

get\_nex();

kmp();

printf("%d\n",ans);

}

}

### 2.马拉车

#include<stdio.h>

#include<iostream>

#include<string.h>

#include<algorithm>

using namespace std;

char s[111111];

char s\_new[111111\*2];

int p[111111\*2];

int Init()

{

int len = strlen(s);

s\_new[0] = '$';

s\_new[1] = '#';

int j = 2;

for (int i = 0; i < len; i++)

{

s\_new[j++] = s[i];

s\_new[j++] = '#';

}

s\_new[j] = '\0'; //别忘了哦

//printf("%s\n",s\_new);

return j; //返回s\_new的长度

}

int Manacher()

{

int len = Init(); //取得新字符串长度并完成向s\_new的转换

int maxLen = -1; //最长回文长度

int id;

int mx = 0;

for (int i = 1; i < len; i++)

{

if (i < mx)

p[i] = min(p[2 \* id - i], mx - i); //需搞清楚上面那张图含义, mx和2\*id-i的含义

else

p[i] = 1;

while (s\_new[i - p[i]] == s\_new[i + p[i]]) //不需边界判断，因为左有'$',右有'\0'

p[i]++;

//我们每走一步i，都要和mx比较，我们希望mx尽可能的远，这样才能更有机会执行if (i < mx)这句代码，从而提高效率

if (mx < i + p[i])

{

id = i;

mx = i + p[i];

}

maxLen = max(maxLen, p[i] - 1);

// printf("%d %d %d\n",mx,id,maxLen);

}

/\*for(int i=1;i<=len;i++)printf("%d ",p[i]);

printf("\n");\*/

return maxLen;

}

int main()

{

while (~scanf("%s", s))

{

printf("%d\n", Manacher());

}

return 0;

}

### 3.拓展KMP

#include<bits/stdc++.h>

using namespace std;

const int maxn=100010;//字符串长度最大值

int next[maxn],ex[maxn];//ex数组即为extend数组

char s[maxn],s2[maxn];

int n;

//预处理计算next数组

void getnext(){

int i=0,j,po,len=strlen(s);

next[0]=len;//初始化next[0]

while(s[i]==s[i+1]&&i+1<len)//计算next[1]

i++;

next[1]=i;

po=1;//初始化po的位置

for(i=2;i<len;i++){

if(next[i-po]+i<next[po]+po)//第一种情况，可以直接得到next[i]的值

next[i]=next[i-po];

else //第二种情况，要继续匹配才能得到next[i]的值

{

j=next[po]+po-i;

if(j<0)j=0;//如果i>po+next[po],则要从头开始匹配

while(i+j<len&&s[j]==s[j+i])//计算next[i]

j++;

next[i]=j;

po=i;//更新po的位置

}

}

}

//计算extend数组

void extend(){

int i=0,j,po,len=strlen(s),l2=strlen(s2);

getnext();//计算子串的next数组

while(s[i]==s2[i]&&i<len)i++;

ex[0]=i;

po=0;//初始化po的位置

for(i=1;i<len;i++){

if(next[i-po]+i<ex[po]+po)

ex[i]=next[i-po];//第一种情况

else{

j=ex[po]+po-i;

if(j<0)j=0;//如果j>ex[po]+po则从头开始匹配

while(i+j<len&&j<<l2&&s[j+i]==s2[j])//计算ex[i]

j++;

ex[i]=j;

po=i;

}

}

}

int main(){

}

### 4编辑距离

#include<bits/stdc++.h>

using namespace std;

const int N = 1e3 + 5;

int T, cas = 0;

int n, m;

int dp[N][N];

char s[N], t[N];

int main(){

while(scanf("%s%s",s,t)!=EOF){

int n=(int)strlen(s),m=(int)strlen(t);

for(int i=0;i<=n;i++){

dp[i][0]=i;

}

for(int i=0;i<=m;i++){

dp[0][i]=i;

}

for(int i=1;i<=n;i++){

for(int j=1;j<=m;j++){

dp[i][j]=min(dp[i-1][j],dp[i][j-1])+1;

dp[i][j]=min(dp[i][j],dp[i-1][j-1]+(s[i-1]!=t[j-1]));

}

}

printf("%d\n",dp[n][m]);

}

}

### 5.Karp-Rabin算法

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<string>

#include<cstring>

#define d 256 //字符表中字符数目

using namespace std;

string s,p;

void RK(int q){

//assert( s&& p && q > 0 ); //如果传递的有错，则打印提示

int m=p.size();

int n=s.size();

int p\_h=0; //模式串hash

int s\_h=0; //s串hash

int h=1;

for(int i=0;i<m-1;i++)h=(h\*d)%q; //h表示ts+1 = 10(31415 - 10000\*3) +2 = 14152中的10000

for(int i=0;i<m;i++){

p\_h= ( d \* p\_h + p[i] ) % q;

s\_h= ( d \* s\_h + s[i] ) % q;

} //求出开始p\_h 和 s\_h

for(int i=0;i<n-m;i++){

if(p\_h==s\_h){

int j;

for(j=0;j<m;j++)

if(s[i+j]!=p[j])break;

if(j==m)printf("P occurs with shifts: %d\n",i);

}

if(i<n-m){

s\_h=(d\*(s\_h-s[i]\*h)+s[i+m])%q;

if(s\_h<0)

s\_h+=q;

}

}

}

int main(){

s="GEEKlmnaS FOR GEEKlmnaS njknaskjdaskjbdkjasbdjas njabijbaslbckjsbfGEEKlmnaS FOR GEEKlmnaS";

p="GEEKlmna";

int mod=127; ////需要比s长度大

RK(mod);

}

### 6.AC自动加+trie 树

#include<stdio.h>

#include<bits/stdc++.h>

#define ll long long int

#define max\_n 1000050

#define max\_tot 500050

#define met(a) memset(a,0,sizeof(a))

#define fup(i,a,n,b) for(int i=a;i<n;i+=b)

#define fow(j,a,n,b) for(int j=a;j>0;j-=b)

#define MOD(x) (x)%mod

using namespace std;

const int maxn = 1e5 + 7;

struct Ac {

struct state { //节点状态

int next[26];

int fail, cnt;//指针fail 到这个节点有cnt个串结束

}stable[max\_tot];

int size; //当前AC自动机节点个数

queue<int>q;

void init() { //初始化

met(stable);

size = 1;

while (!q.empty())q.pop();

}

void insert(char \*s) { //将模式串插入trie树

int now = 0; //代表走到那个节点

for (int i = 0; s[i]; i++) {

char ch = s[i]-'a';

if (!stable[now].next[ch]) //节点不存在该字母边，则新建一个

stable[now].next[ch] = size++;

now = stable[now].next[ch];

}

stable[now].cnt++;//结束位置++；

}

void build() { //构造失配fail指针，要构造当前节点fail指针需先构造爸爸节点

for (int i = 0; i < 26; i++)if (stable[0].next[i])q.push(stable[0].next[i]);

while (!q.empty()) {

int u = q.front();

q.pop();

for (int i = 0; i < 26; i++) {

if (stable[u].next[i]) { //如果有i这条边 枚举下他儿子

int v = stable[u].fail;

int a = stable[u].next[i];

while (v) { //一直按箭头的fail

if (stable[v].next[i]) { //如果他某个祖先有i这条边

stable[a].fail = stable[v].next[i];

break;

}

v = stable[v].fail;

}

if (!stable[a].fail)stable[a].fail = stable[0].next[i];

q.push(stable[u].next[i]); //节点加进去

}

}

}

}

int get(int u) { //算所有祖先的和

int res = 0;

while (u) {

res = res + stable[u].cnt;

stable[u].cnt = 0; //计算后不再计算，如果要计算不清零

u = stable[u].fail;

}

return res;

}

int match(char \*s) { //匹配

int res = 0, now = 0;

for (int i = 0; s[i]; i++) {

char ch = s[i]-'a';

if (stable[now].next[ch]) //如果当前状态太能找到后继节点，则走他

now = stable[now].next[ch];

else { //否则只能跳爸爸

int p = stable[now].fail;

while (p && stable[p].next[ch] == 0)p = stable[p].fail; //始终没找到，只能指根节点 //找到就跳对应地方

now = stable[p].next[ch];

}

if (stable[now].cnt)res = res + get(now);//

}

return res;

}

}Ac;

char s[max\_n];

int main(int argc, char \*argv[]) {

#ifdef LOCAL

freopen("C:/input.txt", "r", stdin);

#endif

}

### 7.查找母串中各单词出现次数

#include<stdio.h>

#include<bits/stdc++.h>

#define ll long long int

#define max\_n 1000050

#define max\_tot 500050

#define met(a) memset(a,0,sizeof(a))

#define fup(i,a,n,b) for(int i=a;i<n;i+=b)

#define fow(j,a,n,b) for(int j=a;j>0;j-=b)

#define MOD(x) (x)%mod

using namespace std;

const int maxn = 1e5 + 7;

struct aa {

int i;

int cnt;

}ans[180];

int cmp(aa a, aa b) {

if (a.cnt == b.cnt)return a.i < b.i;

else return a.cnt > b.cnt;

}

struct Ac {

struct state { //节点状态

int next[26];

int fail, cnt;//指针fail 到这个节点有cnt个串结束

}stable[max\_tot];

int size; //当前AC自动机节点个数

queue<int>q;

void init() { //初始化

met(stable);

size = 1;

while (!q.empty())q.pop();

for (int i = 0; i <= 150; i++) {

ans[i].i = i;

ans[i].cnt = 0;

}

}

void insert(char \*s,int n) { //将模式串插入trie树

int now = 0; //代表走到那个节点

for (int i = 0; s[i]; i++) {

char ch = s[i]-'a';

if (!stable[now].next[ch]) //节点不存在该字母边，则新建一个

stable[now].next[ch] = size++;

now = stable[now].next[ch];

}

stable[now].cnt=n;//结束位置++；

}

void build() { //构造失配fail指针，要构造当前节点fail指针需先构造爸爸节点

for (int i = 0; i < 26; i++)if (stable[0].next[i])q.push(stable[0].next[i]);

while (!q.empty()) {

int u = q.front();

q.pop();

for (int i = 0; i < 26; i++) {

if (stable[u].next[i]) { //如果有i这条边 枚举下他儿子

int v = stable[u].fail;

int a = stable[u].next[i];

while (v) { //一直按箭头的fail

if (stable[v].next[i]) { //如果他某个祖先有i这条边

stable[a].fail = stable[v].next[i];

break;

}

v = stable[v].fail;

}

if (!stable[a].fail)stable[a].fail = stable[0].next[i];

q.push(stable[u].next[i]); //节点加进去

}

}

}

}

void get(int u) { //算所有祖先的和

int res = 0;

u = stable[u].fail;//从父节点开始算 不然会多算一个

while (u) {

if(stable[u].cnt)ans[stable[u].cnt].cnt++; //找个数

u = stable[u].fail;

}

return;

}

int match(char \*s) { //匹配

int res = 0, now = 0;

for (int i = 0; s[i]; i++) {

char ch = s[i]-'a';

if (stable[now].next[ch]) //如果当前状态太能找到后继节点，则走他

now = stable[now].next[ch];

else { //否则只能跳爸爸

int p = stable[now].fail;

while (p && stable[p].next[ch] == 0)p = stable[p].fail; //始终没找到，只能指根节点 //找到就跳对应地方

now = stable[p].next[ch];

}

if (stable[now].cnt) //如果是危险节点（单词结尾）

ans[stable[now].cnt].cnt++; //次单词数++

get(now); //查找后查重复 如在aaaaa中找单词 a aaa

}

return res;

}

}Ac;

char s[max\_n];

char s1[200][80];

int main(int argc, char \*argv[]) {

#ifdef LOCAL

freopen("C:/input.txt", "r", stdin);

#endif

return 0;

}

### 8.最短公共祖先

/\*

题意是给出两个串，用这两个串组成一个新串，使新串包含这两个串，

将KMP进⾏行行略略微的改动，依然是查找匹配段，要求要么⼀一个串串包含另⼀一个串串，要么⼀一个串串的 前缀等于另⼀一个串串的后缀。

问这个新串的长度最小是多少，显然，对于两个串A，B，A如果是B的字串或者B如果是A的字串的话，

直接输出那个母串的长度即可，如果没有这种关系，那么看一个串的后缀是否是另一个串的前缀，

如果某个串的后缀与另一个串的前缀的公共部分最长，则答案=A.length+B.length-公共长度。

\*/

#include<stdio.h>

#include<algorithm>

#include<iostream>

#include<cstring>

using namespace std;

const int N = 1000100;

char a[3][2\*N];

int fail[2\*N];

inline int max(int a, int b)

{

return (a > b) ? a : b;

}

int kmp(int &i, int &j, char\* str, char\* pat)

{

int k;

memset(fail, -1, sizeof(fail));

for (i = 1; pat[i]; ++i)

{

for (k = fail[i - 1]; k >= 0 && pat[i] != pat[k + 1]; k = fail[k]);

if (pat[k + 1] == pat[i])

{

fail[i] = k + 1;

}

}

i = j = 0;

while (str[i] && pat[j])

{

if (pat[j] == str[i])

{

i++;

j++;

}

else if (j == 0)

{

i++;

}

else

{

j = fail[j - 1] + 1;

}

}

if (pat[j])

{

return -1;

}

else

{

return i - j;

}

}

int main(int argc, const char \* argv[])

{

int T;

scanf("%d", &T);

while (T--)

{

int i, j, l1 = 0, l2 = 0;

cin >> a[0] >> a[1];

int len1 = (int)strlen(a[0]), len2 = (int)strlen(a[1]), val;

val = kmp(i, j, a[1], a[0]);

if (val != -1)

{

l1 = len1;

}

else

{

if (i == len2 && j - 1 >= 0 && a[1][len2 - 1] == a[0][j - 1])

{

l1 = j;

}

}

val = kmp(i, j, a[0], a[1]);

if (val != -1)

{

l2 = len2;

}

else

{

if (i == len1 && j - 1 >= 0 && a[0][len1 - 1] == a[1][j - 1])

{

l2 = j;

}

}

printf("%d\n", len1 + len2 - max(l1, l2));

}

return 0;

}

### 9.字符串Hash

#include<stdio.h>

#include<bits/stdc++.h>

#define ll long long int

using namespace std;

ll gethash(char \*s,int m){

ll h=0;

for(int i=0;s[i];i++)

h=((h<<8)+s[i])%m;

return h;

}

int main(){

#ifdef LOCAL

//freopen("C:/Users/Administrator/Desktop/input.txt","r",stdin);

#endif

char a[100]="22222222222222222222222222222";

char b[100]="22222222222222222222222223222";

printf("%lld\n",gethash(a,1e9+7));

printf("%lld\n",gethash(b,1e9+7));

}

### 10.Sunday

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<string>

#include<cstring>

using namespace std;

string s,p;

int next\_[257];

void getnext(){

int len=p.size();

for(int i=0;i<256;i++)next\_[i]=-1;

for(int i=0;i<len;i++)next\_[p[i]]=i; //打next\_表，记录模式串相应字符的位置

}

int sunday(){

int slen=s.size();

int plen=p.size();

if(slen==0)return -1; //如果s的长度为0，不需要匹配，直接返回-1

for(int i=0;i<slen-plen;){

int j=i; //s[j]

int k=0; //p[k]

for(;k<plen&&j<slen&&s[j]==p[k];j++,k++);//一直匹配，找到失配 j 和 k

if(k==plen) //说明已经找到一段匹配串

return i; //如果要查找出现次数，改成cnt++

else{

if(i+plen<slen)i+=(plen-next\_[s[i+plen]]);

else return -1;// //如果要查找出现次数，改成return cnt

}

}

return -1;

}

int main(){

s="I love DNF and Code";

p="love";

getnext();

if(sunday())printf("you find it at %d\n",sunday());

else printf("sorry,you do not find it!\n");

}

### 11.后缀数组

/\*

Problem: JZOJ1598(询问一个字符串中有多少至少出现两次的子串)

Content: SA's Code and Explanation

Author : YxuanwKeith

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

#include<iostream>

using namespace std;

const int MAXN = 100005;

char ch[MAXN], All[MAXN];

int SA[MAXN], rank[MAXN], Height[MAXN], tax[MAXN], tp[MAXN], a[MAXN], n, m;

char str[MAXN];

//rank[i] 第i个后缀的排名; SA[i] 排名为i的后缀位置; Height[i] 排名为i的后缀与排名为(i-1)的后缀的LCP

//tax[i] 计数排序辅助数组; tp[i] rank的辅助数组(计数排序中的第二关键字),与SA意义一样。

//a为原串

void RSort() {

//rank第一关键字,tp第二关键字。

for (int i = 0; i <= m; i ++) tax[i] = 0;

for (int i = 1; i <= n; i ++) tax[rank[tp[i]]] ++;

for (int i = 1; i <= m; i ++) tax[i] += tax[i-1];

for (int i = n; i >= 1; i --) SA[tax[rank[tp[i]]] --] = tp[i]; //确保满足第一关键字的同时，再满足第二关键字的要求

} //计数排序,把新的二元组排序。

int cmp(int \*f, int x, int y, int w) { return f[x] == f[y] && f[x + w] == f[y + w]; }

//通过二元组两个下标的比较，确定两个子串是否相同

void Suffix() {

//SA

for (int i = 1; i <= n; i ++) rank[i] = a[i], tp[i] = i;

m = 127 ,RSort(); //一开始是以单个字符为单位，所以(m = 127)

for (int w = 1, p = 1, i; p < n; w += w, m = p) { //把子串长度翻倍,更新rank

//w 当前一个子串的长度; m 当前离散后的排名种类数

//当前的tp(第二关键字)可直接由上一次的SA的得到

for (p = 0, i = n - w + 1; i <= n; i ++) tp[++ p] = i; //长度越界,第二关键字为0

for (i = 1; i <= n; i ++) if (SA[i] > w) tp[++ p] = SA[i] - w;

//更新SA值,并用tp暂时存下上一轮的rank(用于cmp比较)

RSort(), swap(rank, tp), rank[SA[1]] = p = 1;

//用已经完成的SA来更新与它互逆的rank,并离散rank

for (i = 2; i <= n; i ++) rank[SA[i]] = cmp(tp, SA[i], SA[i - 1], w) ? p : ++ p;

}

//离散：把相等的字符串的rank设为相同。

//LCP

int j, k = 0;

for(int i = 1; i <= n; Height[rank[i ++]] = k)

for( k = k ? k - 1 : k, j = SA[rank[i] - 1]; a[i + k] == a[j + k]; ++ k);

//这个知道原理后就比较好理解程序

}

void Init() {

scanf("%s", str+1);

n = strlen(str+1);

for (int i = 1; i <=n; i ++) a[i ] = str[i];

}

int main() {

Init();

Suffix();

for(int i=1;i<=n;i++)cout<<SA[i]<<endl;

for(int i=1;i<=n;i++){

printf("%s\n",str+SA[i]);

}

/\*int ans = Height[2];

for (int i = 3; i <= n; i ++) ans += max(Height[i] - Height[i - 1], 0);

printf("%d\n", ans); \*/

}

### 12.后缀自动机

#include<stdio.h>

#include<bits/stdc++.h>

#define maxc 28

using namespace std;

const int maxn = 1e6 + 5;

const int mod = 1e9 + 7;

typedef long long ll;

int len[maxn \* 2], //最长子串的长度(该节点字串数量=len[x]-len[link[x]])

link[maxn \* 2], //后缀链接(最短串前部减少一个字符所到达的状态)

cnt[maxn \* 2], //被后缀连接的数

nex[maxn \* 2][maxc], //状态转移(尾部加一个字符的下一个状态)(图)

idx, //节点编号

last; //最后节点

ll epos[maxn \* 2]; // enpos数（该状态子串出现数量）

char str[maxn];

ll a[maxn]; //长度为i的子串出现最大次数

void Iint() { //初始化

last = idx = 1; //1表示root起始点 空集

link[1] = len[1] = 0;

}

//SAM建图

void Extend(int c) { //插入字符，为字符ascll码值

int x = ++idx; //创建一个新节点x;

len[x] = len[last] + 1; // 长度等于最后一个节点+1

epos[x] = 1; //接受节点子串除后缀连接还需加一

int p; //第一个有C转移的节点;

for (p = last; p && !nex[p][c]; p = link[p])nex[p][c] = x;//沿着后缀连接 将所有没有字符c转移的节点直接指向新节点

if (!p)link[x] = 1, cnt[1]++; //全部都没有c的转移 直接将新节点后缀连接到起点

else {

int q = nex[p][c]; //p通过c转移到的节点

if (len[p] + 1 == len[q]) //pq是连续的

link[x] = q, cnt[q]++; //将新节点后缀连接指向q即可,q节点的被后缀连接数+1

else {

int nq = ++idx; //不连续 需要复制一份q节点

len[nq] = len[p] + 1; //令nq与p连续

link[nq] = link[q]; //因后面link[q]改变此处不加cnt

memcpy(nex[nq], nex[q], sizeof(nex[q])); //复制q的信息给nq

for (; p&&nex[p][c] == q; p = link[p])

nex[p][c] = nq; //沿着后缀连接 将所有通过c转移为q的改为nq

link[q] = link[x] = nq; //将x和q后缀连接改为nq

cnt[nq] += 2; // nq增加两个后缀连接

}

}

last = x; //更新最后处理的节点

}

void GetNpos() { //求npos数，即该节点字串出现次数

queue<int>q;

for (int i = 1; i <= idx; i++)

if (!cnt[i])q.push(i); //将所有没被后缀连接指向的节点入队

while (!q.empty()) {

int x = q.front();

q.pop();

epos[link[x]] += epos[x]; //子串数量等于所有后缀连接指向该节点的子串数量和+是否为接受节点

if (--cnt[link[x]] == 0)q.push(link[x]); //当所有后缀连接指向该节点的处理完毕后再入队

}

}

void GetSubNum() { //求不相同字串数量

ll ans = 0;

for (int i = 2; i <= idx; i++)ans += len[i] - len[link[i]]; //一状态子串数量等于len[i]-len[link[i]]

printf("%lld\n",ans);

}

void GetSubMax() { //求出所有长度为k的子串中出现次数最多的子串出现次数

GetNpos();

int n = strlen(str);

for (int i = 1; i <= idx; i++)a[len[i]] = max(a[len[i]], epos[i]); //长度≤k的子串中出现次数最多的子串出现次数的最小值

for (int i = n - 1; i >= 1; i--)a[i] = max(a[i], a[i + 1]); //求一遍后缀最大值就是答案

for (int i = 1; i <= n; i++)printf("%lld\n", a[i]);

}

int main() {

//freopen("c:/input.txt","r",stdin);

return 0;

}

## 数论

### 1.Miller-Rabin素性测试--快速求一个数是否是素数-应用见例题

#include<stdio.h>

#include<bits/stdc++.h>

#define ll long long int

#define met(a) memset(a,0,sizeof(a))

#define fup(i,a,n,b) for(int i=a;i<n;i+=b)

#define fow(j,a,n,b) for(int j=a;j>0;j-=b)

#define MOD(x) (x)%mod

using namespace std;

const int maxn = 2\*1e5 + 5;

long long quicks(long long a, long long b, long long c){

long long ans = 1;

a = a % c;

while (b != 0){

if (b & 1) ans = (ans\*a) % c;

b >>= 1;

a = (a\*a) % c;

}

return ans;

}

bool Miller\_Rabin\_1(long long n) { //标准代码

long long t = 0;

long long b = n - 1;

while ((b & 1) == 0){

t++;

b >>= 1;

}

//现在的a^(b\*2^t)=1(mod n)

long long a = rand() % (n - 1) + 1; //测试

long long x = quicks(a, b, n);

//个人认为这里如果加上优先判定是不是1，n-1的话，会更快一点？是不是呢？？？？？

for (long long i = 1; i <= t; i++){

long long y = quicks(x, 2, n);

if (y == 1 && x != 1 && x != n - 1) return false; //这里的意思是如果a^(d\*2^r)是1，但是a^(d\*2^(r-1))不是1也不是n-1的情况，这时候我们认为是合数

x = y;

}

if (x != 1) return false;

else return true;

}

int main() {

ll n;

cin >> n;

if (Miller\_Rabin\_1(n))cout << "Yes" << endl;

else cout << "No" << endl;

return 0;

}

### 2.杜教筛

#include<stdio.h>

#include<bits/stdc++.h>

#define ll long long int;

#define N 2001000

#define ni 500000004 //2的逆元

using namespace std;

const long long int mod=0x3f3f3f3f3f3f3f;

inline void read(long long &x) //输入外挂

{

x=0;

static int p;p=1;

static char c;c=getchar();

while(!isdigit(c)){if(c=='-')p=-1;c=getchar();}

while(isdigit(c)) {x=(x<<1)+(x<<3)+(c-48);c=getchar();}

x\*=p;

}

bool vis[N];

int mu[N],sum1[N];

long long phi[N],sum2[N];

int cnt,prim[N];

int e,e1;

map<long long ,int>w,w1; //哈希 w用来求phi前缀和 w1用来求miu前缀和

void get(int maxn)

{

phi[1]=mu[1]=1;

for(int i=2;i<=maxn;i++)

{

if(!vis[i])

{

prim[++cnt]=i;

mu[i]=-1;phi[i]=i-1;

}

for(int j=1;j<=cnt&&prim[j]\*i<=maxn;j++)

{

vis[i\*prim[j]]=1;

if(i%prim[j]==0)

{

phi[i\*prim[j]]=phi[i]\*prim[j];

break;

}

else mu[i\*prim[j]]=-mu[i],phi[i\*prim[j]]=phi[i]\*(prim[j]-1);

}

}

for(int i=1;i<=maxn;i++)sum1[i]=sum1[i-1]+mu[i],sum2[i]=(sum2[i-1]+phi[i])%mod; //打一个maxn的phi前缀和表 和miu前缀和表

}

int djsmu(long long x) // 求miu前缀和

{

if(x<=2000000)return sum1[x];

if(w.count(x))return w[x];

int ans=1;

for(long long l=2,r;l<=x;l=r+1)

{

r=x/(x/l);

ans-=(r-l+1ll)\*djsmu(x/l);

}

return w[x]=ans;

}

long long djsphi(long long x) //求phi 前缀和

{

if(x<=2000000)return sum2[x];

if(w1.count(x))return w1[x];

long long ans=(x %mod\* ((x + 1) % mod)) % mod\*ni%mod;

for(long long l=2,r;l<=x;l=r+1)

{

r=x/(x/l); //最后一个分块中的数

ans = (ans - ((r - l + 1) % mod\*djsphi(x / l)) % mod + mod) % mod;

}

return w1[x]=ans;

}

int main()

{

long long t;

read(t);

get(2000000);

while(t--){

static long long n;

read(n);

printf("%lld %d\n",djsphi(n),djsmu(n));

}

return 0;

}

### 3.log(n)复杂度分解素因子

//m为要分解的数

//a数组存所有质因子

//tot为质因子个数

void primeFactor(int n){

while(n%2==0){

Fac[cnt++]=2;

n/=2;

}

// 经过第二步, 此时 n 一定为奇数

// 并且不存在偶数的素因子

// 所以我们可以跳过所有偶数 (i += 2)

for(int i=3;i<=sqrt(n);i+=2){

while(n%i==0){

Fac[cnt++]=i;

n/=i;

}

}

//此处为了防止是一个大于 2 的素数

if(n>2)Fac[cnt++]=n;

}

### 4.基数排序MSD

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<string>

#include<malloc.h>

using namespace std;

const int maxn=1e6+7;

int arr[maxn]={12,14,54,5,6,3,9,8,47,89};

int getdigit(int x,int d){

int a[]={1,1,10,100,1e3,1e4,1e5,1e6,1e7,1e8,1e9}; ////因为待排数据最大数据位数

return ((x/a[d])%10);

}

void msdradix\_sort(int begin,int end,int d){

const int radix=10; //进制数

int count[radix],i,j; //count表示每个桶中元素个数

//置零

for(i=0;i<10;++i)count[i]=0;

//给桶分配空间

int \*bucket=(int \*)malloc((end-begin+1)\*sizeof(int));

//统计各桶需要装的元素的个数

for(i=begin;i<=end;++i){

count[getdigit(arr[i], d)]++;

}

//求出桶的边界索引，count[i]值为第i个桶的右边界索引+1

for(i=1;i<radix;++i){

count[i]+=count[i-1]; //每个桶的边界，便于下步将arr重新放入bucket里

}

//这里要从右向左扫描，保证排序稳定性

for(i=end;i>=begin;--i){

j=getdigit(arr[i],d); //求出关键码的第d位的数字， 例如：576的第3位是5

bucket[count[j]-1]=arr[i]; //放入对应的桶中，count[j]-1是第j个桶的右边界索引

--count[j]; //第j个桶放下一个元素的位置(右边界索引+1)

}

//注意：此时count[i]为第i个桶左边界

//从各个桶中收集数据

for(i = begin, j = 0;i <= end; ++i, ++j){

arr[i] = bucket[j];

}

//释放存储空间

free(bucket);

//对每个桶再次排序

for(i=0;i<radix;i++){

int p1=begin+count[i]; //第I个桶的左边界

int p2=begin+count[i+1]-1 ; //第i个桶的右边界

if(p1<p2&&d>1){

msdradix\_sort(p1, p2, d-1); //对第i个桶递归调用，进行基数排序，数位降 1

}

}

}

int main(){

int len=10;

for(int i=0;i<10;i++)cout<<arr[i]<<" ";

cout<<endl;

msdradix\_sort(0,10-1,2); //2表示最高位数

for(int i=0;i<10;i++)cout<<arr[i]<<" ";

cout<<endl;

}

### 5.矩阵快速幂

#include<bits/stdc++.h>

using namespace std;

int N=7;

void Matrix(int (&a)[2][2],int b[2][2]){

int tmp[2][2]={0};

for(int i=0;i<2;++i)

for(int j=0;j<2;++j)

for(int k=0;k<2;++k)

tmp[i][j]=(tmp[i][j]+a[i][k]\*b[k][j])%N;

for(int i=0;i<2;++i){

for(int j=0;j<2;++j){

a[i][j]=tmp[i][j];

}

}

}

int main(){

int a,b,n;

while(scanf("%d%d%d",&a,&b,&n)){

if(a==0&&b==0&&n==0)break;

if(n==1){

cout<<1<<endl;

continue;

}

int temp[2][2]={a,b,0,0},cot[2][2]={1,0,0,1}, x[2] = {1, 1};

n-=2;

while(n){

if(n&1)Matrix(cot,temp);

Matrix(temp,temp);

n/=2;

}

int ans=0;

for(int i=0;i<2;i++)

ans=(ans+x[i]\*cot[0][i])%N;

cout<<ans<<endl;

}

}

ll quick\_mod(ll a,ll b,ll c) //快速幂计算(a^b)%c

{

ll ans = 1;

while(b)

{

if(b&1) //相当于b%2==1

ans = (ans\*a)%c;

a = (a\*a)%c;

b>>=1; //相当于b/=2

}

return ans;

}

### 6.莫比乌斯表

int phi[maxn],miu[maxn],fac[maxn];//phi--欧拉函数表 miu--莫比乌斯函数表 fac--i最大的素因子辅助打phi表

ll f[maxn], F[maxn];

void init()

{

for (int i = 1; i < maxn; ++i) fac[i] = i;

phi[1] = miu[1] = 1;

for (int i = 2; i < maxn; ++i)

{

if (fac[i] == i)

for (int j = i << 1; j < maxn; j += i)

fac[j] = i;

if (i / fac[i] % fac[i]) phi[i] = (fac[i] - 1)\*phi[i / fac[i]], miu[i] = -miu[i / fac[i]]; //如果b质数 a%b！=0 phi(a\*b) = phi(a)\*b - phi(a)

else phi[i] = fac[i] \* phi[i / fac[i]], miu[i] = 0; //当b是质数，a%b==0，phi(a\*b)=phi(a)\*b

}

}

//求单个莫比乌斯

int miu(ll n){

int prime = 1;

int flag = 0;

for (int i = 2; i\*i <= n; i++) {

if (n%i == 0) {

prime++;

n /= i;

if (n%i == 0) {

flag = 1;

break;

}

}

}

if (flag)

return 0;

if (prime % 2)return -1;

else return 1;

}

### 7.逆元模板

求

a\*c(mod m)

//费马小定理求逆元

ll quick\_mod(ll a,ll b,ll c) //快速幂计算(a^b)%c

{

ll ans = 1;

while(b)

{

if(b&1) //相当于b%2==1

ans = (ans\*a)%c;

a = (a\*a)%c;

b>>=1; //相当于b/=2

}

return ans;

}

ll inv(ll b,ll c) //计算b的逆元

{

return quick\_mod(b,c-2,c);

}

ll div(ll a,ll b,ll c) //计算(a/b)%c

{

return ((a%c)\*(inv(b,c)%c))%c;

}

/---------------------------------------------------------------------------------/

/-------------------------------------------------------------------------------/

//扩展GCD求逆元

void ex\_gcd(ll a, ll b, ll &d, ll &x, ll &y){

if (!b) {d = a, x = 1, y = 0;}

else{

ex\_gcd(b, a % b, d, y, x);

y -= x \* (a / b);

}

}

ll inv(ll a, ll p){//如果不存在，返回-1 a,p互质

ll d, x, y;

ex\_gcd(a, p, d, x, y);

return d == 1 ? (x % p + p) % p : -1;

}

/---------------------------------------------------------------------------------/

/-------------------------------------------------------------------------------/

//递归求逆元

//当p是个质数的时候有inv(a) = (p - p / a) \* inv(p % a) % p,且1的逆元就是1

ll inv(ll t, ll p) {

//求t关于p的逆元，注意:t要小于p，最好传参前先把t%p一下 ,即inv(a%p, p)求a对p的逆元

return t == 1 ? 1 : (p - p / t) \* inv(p % t, p) % p;

}

/---------------------------------------------------------------------------------/

/-------------------------------------------------------------------------------/

//打表求逆元

ll inv[maxn];

void Prepare\_inv(ll n,ll M){

inv[1]=1;

for(ll i=2;i<=n;i++){

inv[i]=(ll)(M-M/i)\*inv[M%i]%M;

}

}

### 8.欧拉函数

ll p[maxn];

ll phi(ll x){ //求1~n与n互质的个数 // phi(1323)=phi(3^3\*7^2)=1323\*(1-1/3)\*(1-1/7)

ll i, ans = x;

for (i = 2; i\*i <= x; i++){

if (x%i == 0)

ans = ans - ans / i;

while(x%i == 0)

x /= i;

}

if (x > 1)

ans = ans - ans / x;

return ans;

}

/---------------------------------------------------------------------------------/

/-------------------------------------------------------------------------------/

ll \_phi(ll x) { //递推求欧拉函数 利用了欧拉函数的积性

//如果b质数 a%b！=0 phi(a\*b) = phi(a)\*b - phi(a)

//当b是质数，a%b==0，phi(a\*b)=phi(a)\*b

if (x == 0) return 0;

ll res = 1, t = x;

for (ll i = 2; i <= (ll)sqrt(1.\*x); i++) {

if (t%i == 0) {

res \*= (i - 1);

t /= i;

while (t%i == 0) {

res \*= i;

t /= i;

}

}

if (t == 1) break;

}

if (t > 1) { res \*= (t - 1); }

return res;

}

/---------------------------------------------------------------------------------/

/-------------------------------------------------------------------------------/

//递推打素数表

ll phi[maxn];

void init()

{

for(int i=1;i<=maxn;i++)

phi[i] = i;

for (int i = 2; i\*i < maxn; i++){ //最大素因子表

if (phi[i] == i){

for (int j = i \* i; j < maxn; j += i){

phi[j] = i;

}

}

}

phi[1] = 1;

for (int i = 2; i < maxn; i++){

if ((i / phi[i]) % phi[i] == 0){

phi[i] = phi[i / phi[i]] \* phi[i]; //当b是质数，a%b==0，phi(a\*b)=phi(a)\*b

}

else {

phi[i] = phi[i / phi[i]] \* (phi[i] - 1); //如果b质数 a%b！=0 phi(a\*b) = phi(a)\*b - phi(a)

}

}

}

### 9.素数筛

#include<stdio.h>

#include<cstring>

#include<bits/stdc++.h>

using namespace std;

const int MAX=1e7+7;//求MAX范围内的素数

long long su[MAX],cnt;

bool isprime[MAX];

void prime()

{

cnt=1;

memset(isprime,1,sizeof(isprime));//初始化认为所有数都为素数

isprime[0]=isprime[1]=0;//0和1不是素数

for(long long i=2;i<=MAX;i++)

{

if(isprime[i])

su[cnt++]=i;//保存素数i

for(long long j=1;j<cnt&&su[j]\*i<MAX;j++)

{

isprime[su[j]\*i]=0;//筛掉小于等于i的素数和i的积构成的合数

}

}

}

int main()

{

prime();

//for(long long i=1;i<cnt;i++)

// printf("%d ",su[i]);

return 0;

}

/---------------------------------------------------------------------------------/

/-------------------------------------------------------------------------------/

//欧拉筛 --线性

int p[maxn], check[maxn], tot = 0;

void prime()

{

check[1]=1;

check[0]=1;

for(int i=2;i<=n;i++)

{

if(!check[i])p[++tot]=i;

for(int j=1;j<=tot&&i\*p[j]<=n;j++)

{

check[i\*p[j]]=1;

if(!(i%p[j]))break;//\*\*\*\*\*关键

}

}

}

### 10.拓展欧几里得及欧几里得-------求解

int gcd(int a,int b){

return (b==0)?a:gcd(b,a%b); //一条语句搞定（三元运算符）装逼，跟上面略有不同，上面做到t=0,这里做到b=0

}

ll lcm(ll a, ll b) {

return a / gcd(a,b) \* b;

}

int exgcd(int a,int b,int &x,int &y){

if(b==0){

x=1;

y=0;

return a;

}

int r=exgcd(b,a%b,x,y);

int t=x;

x=y;

y=t-a/b\*y;

return r;

}

### 11拓展欧几里得求整数解个数

ll gcd(ll a, ll b) {

return b ? gcd(b, a%b) : a;

}

ll lcm(ll a, ll b) {

return a / gcd(a, b) \* b;

}

ll extend\_gcd(ll a, ll b, ll&x, ll&y) {

if (!b) {

x = 1;

y = 0;

return a;

}

ll xt = 0, yt = 0;

ll d = extend\_gcd(b, a % b, xt, yt);

x = yt;

y = xt - yt \* (a / b);

return d;

}

ll cal(ll a, ll b, ll n) { //计算ax+by == n的非负整数解组数

ll x = 0, y = 0, d;

d = extend\_gcd(a, b, x, y);

if (n % d != 0) {

return 0;

}

x \*= n / d, y \*= n / d;

ll LCM = lcm(a, b);

ll t1 = LCM / a, t2 = LCM / b;

if (x < 1) {

ll num = (1 - x) / t1;

x += num \* t1;

y -= num \* t2;

if (x < 1) {

y -= t2;

x += t1;

}

}

if (y < 1) {

ll num = (1 - y) / t2;

y += num \* t2;

x -= num \* t1;

if (y < 1) {

y += t2;

x -= t1;

}

}

ll ans = x > 0 && y > 0;

if (ans) {

ans += min((x - 1) / t1, ((n - 1) / b - y) / t2);

ans += min((y - 1) / t2, ((n - 1) / a - x) / t1);

}

return ans;

}

### 12.整除分块

for(int i=1,last;i<=n;i=last+1){

last=n/(n/i); //该区间的最后一个数

ans+=(last-i+1)\*(n/i);

}

### 13组合数

//组合数打表O(n\*n)

#include <bits/stdc++.h>

using namespace std;

int C[1005][1005];

int main()

{

C[0][0] = 1;

for (int i = 1; i <= 1000; i++)

{

C[i][0] = 1;

for (int j = 1; j <= 1000; j++)

C[i][j] = C[i - 1][j - 1] + C[i - 1][j];

}

cout << C[4][3] << endl;

return 0;

}

//Log(n)求C(n,m)

//快速幂求组合数

#include<stdio.h>

#include<bits/stdc++.h>

using namespace std;

typedef long long int ll;

const int mod=1e9+7;

ll qpow(ll a,ll b){

ll ans=1;

while(b){

if(b&1)ans=ans\*a%mod;

b>>=1;

a=a\*a%mod;

}

return ans;

}

ll jc(ll n){

ll ans=1;

for(ll i=1;i<=n;i++)ans=ans\*i%mod;

return ans;

}

ll C(ll n,ll m){

ll ans=1;

ans=ans\*jc(n)%mod;

ans=ans\*qpow(jc(m),mod-2)%mod;

ans=ans\*qpow(jc(n-m),mod-2)%mod;

return ans;

}

int main(){

int n,m;

scanf("%d%d",&n,&m);

printf("%lld\n",C(n,m));

}

### 14.最长循环节

/\*如果1<=b<a,a没有2或5的质因子，并且a与b互质，那么b/a 的循环节位数恰好等于e 满足min(10^e≡1(moda))),e是正整数。

如果1<=b<a,a没有2或5的质因子，并且a与b互质，那么b/a 的循环节位数必整除ϕ(a)。\*/

#include<stdio.h>

#include<bits/stdc++.h>

#define ll long long int

#define met(a) memset(a,0,sizeof(a))

#define fup(i,a,n,b) for(int i=a;i<n;i+=b)

#define fow(j,a,n,b) for(int j=a;j>0;j-=b)

using namespace std;

const int maxn = 1e6 + 7;

int main(int argc, char \*argv[]) {

int n,ans=1,maxx=0;

scanf("%d", &n);

for (int i = 2; i <= n; i++) {

int tmp = i, tmp1 = 1, tmp2 = 0;

while (tmp % 2 == 0)tmp /= 2;

while (tmp % 5 == 0)tmp /= 5;

if (tmp == 1)tmp2 = 0;

else {

do {

tmp1 = tmp1 \* 10 % tmp;

++tmp2;

} while (tmp1 != 1);

}

if (tmp2 > maxx) {

maxx = tmp2;

ans = i;

}

}

printf("%d\n", ans);

return 0;

}

### 15.高精度模拟乘法--实战

/\*有2堆石子。A B两个人轮流拿，A先拿。每次可以从一堆中取任意个或从2堆中取相同数量的石子，但不可不取。拿到最后1颗石子的人获胜。假设A B都非常聪明，拿石子的过程中不会出现失误。给出2堆石子的数量，问最后谁能赢得比赛。

例如：2堆石子分别为3颗和5颗。那么不论A怎样拿，B都有对应的方法拿到最后1颗。

Input

第1行：一个数T，表示后面用作输入测试的数的数量。（1 <= T <= 10000)

第2 - T + 1行：每行2个数分别是2堆石子的数量，中间用空格分隔。(1 <= N <= 10^18)

Output

共T行，如果A获胜输出A，如果B获胜输出B。

Input示例

3

3 5

3 4

1 9

Output示例

B

A

A

\*/

//注意：sqrt产生精度问题 double最多存小数点后14位，模拟乘法

#include<stdio.h>

#include<bits/stdc++.h>

using namespace std;

typedef long long int ll;

const int maxn = 1e4+5;

//0.618033988749894848204586834... 拆成整数放进数组里，

//拆成三部分即可

ll tmp[3] = { 618033988,749894848,204586834 };

ll MOD = 1e9;

int main(){

int t;

scanf("%d", &t);

while (t--){

ll a, b;

scanf("%lld%lld", &a, &b);

if (a > b){

ll t = a;

a = b;

b = t;

}

ll diff = b - a;

//把10^18分成两部分10^9

ll ta = diff / MOD;

ll tb = diff % MOD;

ll tp = tb \* tmp[2];

tp = ta \* tmp[2] + tb \* tmp[1] + tp / MOD;

tp = ta \* tmp[1] + tb \* tmp[0] + tp / MOD;

tp = ta \* tmp[0] + tp / MOD + diff;

if (tp == a)

printf("B\n");

else

printf("A\n");

}

return 0;

}

### 16.N的大数次方

#include <stdio.h>

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int INF = 0x3f3f3f3f;

const int MAXN = 1e5 + 10;

const int MOD = 1e9 + 7;

char s[MAXN];

long long phi(long long x)

{

long long ret = x;

for (int i = 2; i \* i <= x; ++i)

if (x % i == 0)

{

ret -= ret / i;

while (x % i == 0)

x /= i;

}

if (x > 1)

ret -= ret / x;

return ret;

}

ll mpow(ll a, ll n, ll m) //a ^ n % m

{

ll t = 1;

while (n)

{

if (n & 1)

t = (t \* a) % m;

a = (a \* a) % m, n >>= 1;

}

return t;

}

int main()

{

#ifdef LOCAL

//freopen("C:/input.txt", "r", stdin);

#endif

int T;

cin >> T;

while (T--)

{

ll n = 0, p = phi(MOD);

scanf("%s", s);

for (int i = 0; s[i]; i++)

n = (n \* 10 + s[i] - '0') % p;

n += p - 1;

ll ans = mpow(2, n, MOD);

cout << ans << endl;

}

return 0;

}

### 博弈论

#### 1**.Bash**

//两人从一堆a个石子里面轮流取石子，每次最多去b个，取到最后一个石子获胜

int main() {

int t;

scanf("%d", &t);

while (t--) {

int a, b,flag;

scanf("%d%d", &a, &b);

if (a % (b + 1) == 0)flag = 2;

else flag = 1;

if (flag == 1)printf("A\n");

else printf("B\n");

}

}

#### **2.Nim**

//两人从n堆石子中任选一堆取石子，不得不取，可以把一堆直接去玩，拿到最后一颗石子得人获胜

1.面对由 n 个游戏组合成的一个游戏时，只需对于每个游戏找出求它的每个局面的 SG 值的方法，

就可以把这些 SG 值全部看成 Nim 的石子堆，然后依照找 Nim 的必胜策略的方法来找这个游戏的必胜策略了！

int main() {

int n;

int stone, tag=0;

scanf("%d", &n);

while (n--) {

scanf("%d", &stone);

tag ^= stone;

}

// tag为0则为后⼿手赢，否则为先⼿手赢

if (tag == 0)printf("B\n");

else printf("A\n");

}

#### 3.威佐夫博弈

//有2堆石子。A B两个人轮流拿，A先拿。每次可以从一堆中取任意个或从2堆中取相同数量的石子，

但不可不取。拿到最后1颗石子的人获胜

int main() {

int n;

int stone, tag=0;

scanf("%d", &n);

while (n--) {

int a, b;

scanf("%d%d", &a,&b);

if (a < b)swap(a, b);

int flag = (a - b)\*(sqrt(5) + 1) / 2; //黄金分割线

// 如果flag == b，则为后⼿手赢，否则先⼿手赢（奇异局）

if (flag == b)printf("B\n");

else printf("A\n");

}

}

#### **SG打表**

解题模型：

1.把原游戏分解成多个独立的子游戏，则原游戏的SG函数值是它的所有子游戏的SG函数值的异或。

即sg(G)=sg(G1)^sg(G2)^...^sg(Gn)。

2.分别考虑没一个子游戏，计算其SG值。

SG值的计算方法：（重点）

1.可选步数为1~m的连续整数，直接取模即可，SG(x) = x % (m+1);

2.可选步数为任意步，SG(x) = x;

3.可选步数为一系列不连续的数，用模板计算。

//-----------模板：打表------------------------------------------------

//f[]：可以取走的石子个数

//sg[]:0~n的SG函数值

//hash[]:mex{}

int f[N],sg[N],hash[N];

void getSG(int n)

{

int i,j;

memset(sg,0,sizeof(sg));

for(i=1;i<=n;i++)

{

memset(hash,0,sizeof(hash));

for(j=1;f[j]<=i;j++)

hash[sg[i-f[j]]]=1;

for(j=0;j<=n;j++) //求mes{}中未出现的最小的非负整数

{

if(hash[j]==0)

{

sg[i]=j;

break;

}

}

}

}

// --------模板二：DFS----------------------------------------------

//注意 S数组要按从小到大排序 SG函数要初始化为-1 对于每个集合只需初始化1遍

//n是集合s的大小 S[i]是定义的特殊取法规则的数组

int s[110],sg[10010],n;

int SG\_dfs(int x)

{

int i;

if(sg[x]!=-1)

return sg[x];

bool vis[110];

memset(vis,0,sizeof(vis));

for(i=0;i<n;i++)

{

if(x>=s[i])

{

SG\_dfs(x-s[i]);

vis[sg[x-s[i]]]=1;

}

}

int e;

for(i=0;;i++)

if(!vis[i])

{

e=i;

break;

}

return sg[x]=e;

}

一般DFS只在打表解决不了的情况下用，首选打表预处理。

### 线性基

/\*

1.设线性基的异或集合中不存在00。

2.线性基的异或集合中每个元素的异或方案唯一，其实这个跟性质1是等价的。

3.线性基二进制最高位互不相同。

4.如果线性基是满的，它的异或集合为[1,2n−1][1,2n−1]。

5.线性基中元素互相异或，异或集合不变。

\*/

#include<stdio.h>

#include<bits/stdc++.h>

using namespace std;

typedef long long int ll;

const int maxn=1e5+7;

const int mod=1e9+7;

struct Linear\_Basis{

ll b[63],nb[63],tot; //b为线性基 nb用来求第K小异或值 tot为nb元素个数

bool flag=false;

void Init(){ //初始化

tot=0;

flag=false;

memset(b,0,sizeof(b));

memset(nb,0,sizeof(nb));

}

void Ins(ll x){ //插入

for(int i=62;i>=0;i--){

if(x&(1ll<<i)){

if(!b[i]){

b[i]=x;

return;

}

x^=b[i];

}

}

flag=true;

return;

}

bool Fin(ll x){ //验证存在性

if(x==0&&b[0])return 1;

for(int i=62;i>=1;i--){

int j=i-1;

if(x&(1<<j)){

x^=b[i];

if(!x)return 1;

}

}

return 0;

}

ll Max(ll x){ //求最大值

ll res=x;

for(int i=62;i>=0;i--){

res=max(res,res^b[i]);

}

return res;

}

ll Min(ll x){ //求最小值

ll res=x;

for(int i=0;i<=62;i++){

if(b[i])res^=b[i];

}

return res;

}

ll Rebuild(){ //第K大

for(int i=62;i>=0;i--){

if(b[i]==0)continue;

for(int j=i-1;j>=0;j--){

if(b[j]==0)continue;

if(b[i]&(1ll<<j))b[i]^=b[j];

}

}

for(int i=0;i<=62;i++){

if(b[i])nb[tot++]=b[i];

}

}

ll Kth\_Max(ll k){

if(flag)k--; //???

ll res=0;

if(k==0)return 0;

if(k>=(1ll<<tot))return -1;

for(int i=62;i>=0;i--){

if(k&(1ll<<i))res^=nb[i];

}

return res;

}

}LB;

void merge(Linear\_Basis &a,Linear\_Basis &b){//a和b都变成a+b

for(int i=31;i>=1;i--){

if(b.b[i]==0)continue;

a.Ins(b.b[i]);

}

b=a;

}

int main(){

int n;

scanf("%d",&n);

return 0;

}

## 数据结构

### 1.线段树

#include<stdio.h>

#include<bits/stdc++.h>

using namespace std;

int n,p,a,b,m,x,y,ans;

struct node

{

int l,r,w,f;

}tree[400001];

inline void build(int k,int ll,int rr) {//建树

tree[k].l=ll,tree[k].r=rr;

if(tree[k].l==tree[k].r){

scanf("%d",&tree[k].w);

return;

}

int m=(ll+rr)/2;

build(k\*2,ll,m);

build(k\*2+1,m+1,rr);

tree[k].w=tree[k\*2].w+tree[k\*2+1].w;

}

inline void down(int k) {//标记下传

tree[k\*2].f+=tree[k].f;

tree[k\*2+1].f+=tree[k].f;

tree[k\*2].w+=tree[k].f\*(tree[k\*2].r-tree[k\*2].l+1);

tree[k\*2+1].w+=tree[k].f\*(tree[k\*2+1].r-tree[k\*2+1].l+1);

tree[k].f=0;

}

inline void ask\_point(int k){//单点查询

if(tree[k].l==tree[k].r){

ans=tree[k].w;

return ;

}

if(tree[k].f) down(k);

int m=(tree[k].l+tree[k].r)/2;

if(x<=m) ask\_point(k\*2);

else ask\_point(k\*2+1);

}

inline void change\_point(int k) {//单点修改

if(tree[k].l==tree[k].r){

tree[k].w+=y;

return;

}

if(tree[k].f) down(k);

int m=(tree[k].l+tree[k].r)/2;

if(x<=m) change\_point(k\*2);

else change\_point(k\*2+1);

tree[k].w=tree[k\*2].w+tree[k\*2+1].w;

}

inline void ask\_interval(int k) {//区间查询

if(tree[k].l>=a&&tree[k].r<=b) {

ans+=tree[k].w;

return;

}

if(tree[k].f) down(k);

int m=(tree[k].l+tree[k].r)/2;

if(a<=m) ask\_interval(k\*2);

if(b>m) ask\_interval(k\*2+1);

}

inline void change\_interval(int k) {//区间修改

if(tree[k].l>=a&&tree[k].r<=b){

tree[k].w+=(tree[k].r-tree[k].l+1)\*y;

tree[k].f+=y;

return;

}

if(tree[k].f) down(k);

int m=(tree[k].l+tree[k].r)/2;

if(a<=m) change\_interval(k\*2);

if(b>m) change\_interval(k\*2+1);

tree[k].w=tree[k\*2].w+tree[k\*2+1].w;

}

int main(){

scanf("%d",&n);//n个节点

build(1,1,n);//建树

scanf("%d",&m);//m种操作

for(int i=1;i<=m;i++){

scanf("%d",&p);

ans=0;

if(p==1){

scanf("%d",&x);

ask\_point(1);//单点查询,输出第x个数

printf("%d",ans);

}

else if(p==2){

scanf("%d%d",&x,&y);

change\_point(1);//单点修改

}

else if(p==3){

scanf("%d%d",&a,&b);//区间查询

ask\_interval(1);

printf("%d\n",ans);

}

else{

scanf("%d%d%d",&a,&b,&y);//区间修改

change\_interval(1);

}

}

}

## 其他

### 1.对数器

//需要建立主路径 和主路径下sample/ 保证程序文件存在

#include <bits/stdc++.h>

using namespace std;

string mpat("D:/dsq/"); //主路径 注意末尾跟/!!!! linuxpwd查看当前路径

string an("A"), bn("B"); //程序名字

int brek = 5; //几个错误停止 -1不停止

int cmpt = 1; //编译模式

void exec(const string &cmd, string &res)

{

char buf[1024];

FILE \*fp = \_popen(cmd.c\_str(), "r"); //linux下改为popen

while (fgets(buf, 1024, fp))

res.append(buf);

\_pclose(fp); //linux下改为pclose

}

void wrfile(const string &path, stringstream &ss)

{

string str;

FILE \*fp = fopen(path.c\_str(), "wb");

while (getline(ss, str))

fprintf(fp, "%s\r\n", str.c\_str());

fclose(fp);

}

long long d\_rand(long long l, long long r)

{

int k = rand() % 4;

long long t = rand();

for (int i = 0; i < k; i++)

t = t << 16 | rand();

return l + t % (r - l + 1);

}

double f\_rand(double l, double r)

{

double t = rand() / 32767.0;

return l + t \* (r - l);

}

char c\_rand(const string &ch)

{

return ch[rand() % ch.size()];

}

void sample(stringstream &sout) //样例生成

{

}

int main()

{

srand(time(NULL));

rand();

if (cmpt) //编译功能

{

system(("g++ " + mpat + an + ".cpp -o " + mpat + an + ".exe -std=c++11").c\_str()); //C++11支持

system(("g++ " + mpat + bn + ".cpp -o " + mpat + bn + ".exe -std=c++11").c\_str());

}

string num, aout, bout, spat = mpat + "sample/"; //样例生成子路径

stringstream ss;

int cnt = 0, wa = 0;

for (int i = 1; i <= brek; i++)

{

ss.clear(), ss << i, ss >> num;

remove((spat + num + "samp.txt").c\_str());

remove((spat + num + "ans" + an + ".txt").c\_str());

remove((spat + num + "ans" + bn + ".txt").c\_str());

}

while (wa != brek) //几个错误停止

{

ss.str(""), ss.clear(), ss << wa + 1, ss >> num;

aout.clear(), bout.clear(), ss.clear();

sample(ss);

wrfile(spat + num + "samp.txt", ss);

cout << ++cnt;

exec(mpat + an + ".exe < " + spat + num + "samp.txt", aout);

exec(mpat + bn + ".exe < " + spat + num + "samp.txt", bout);

if (aout != bout)

{

ss.clear(), ss.str(aout);

wrfile(spat + num + "ans" + an + ".txt", ss);

ss.clear(), ss.str(bout);

wrfile(spat + num + "ans" + bn + ".txt", ss);

++wa;

}

cout << "\t" << wa << endl;

}

return 0;

}

### 2.位运算加速

还有这个 位运算加速至少一倍

#pragma GCC target("popcnt")

这个是开O2优化 不过只有部分oj能用

#pragma GCC optimize(2)

ios::sync\_with\_stdio(0);

cin.tie(0); cout.tie(0);

### 3.输入外挂

接受非负整数

void read(int &x){  
char ch = getchar(); x = 0;  
for (; ch < '0' || ch > '9'; ch = getchar());  
for (; ch >= '0' && ch <= '9'; ch = getchar()) x = x \* 10 + ch - '0';  
}

/------------------------------------/

/------------------------------------/

const int BufferSize=1<<16;

char buffer[BufferSize],\*head,\*tail;

inline char Getchar() {

if(head==tail) {

int l=fread(buffer,1,BufferSize,stdin);

tail=(head=buffer)+l;

}

return \*head++;

}

inline int read() {

int x=0,f=1;char c=Getchar();

for(;!isdigit(c);c=Getchar()) if(c=='-') f=-1;

for(;isdigit(c);c=Getchar()) x=x\*10+c-'0';

return x\*f;

}

/------------------------------------/

/------------------------------------/

namespace fastIO {

#define BUF\_SIZE 100000

//fread -> read

bool IOerror = 0;

inline char nc() {

static char buf[BUF\_SIZE], \*p1 = buf + BUF\_SIZE, \*pend = buf + BUF\_SIZE;

if (p1 == pend) {

p1 = buf;

pend = buf + fread(buf, 1, BUF\_SIZE, stdin);

if (pend == p1) {

IOerror = 1;

return -1;

}

}

return \*p1++;

}

inline bool blank(char ch) {

return ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t';

}

inline void read(int &x) {

char ch;

while (blank(ch = nc()));

if (IOerror) return;

for (x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x \* 10 + ch - '0');

}

#undef BUF\_SIZE

};

using namespace fastIO;

### 4.BM杜教-解决线性递推

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

#include <vector>

#include <string>

#include <map>

#include <set>

#include <cassert>

#include<bits/stdc++.h>

using namespace std;

#define rep(i,a,n) for (int i=a;i<n;i++)

#define per(i,a,n) for (int i=n-1;i>=a;i--)

#define pb push\_back

#define mp make\_pair

#define all(x) (x).begin(),(x).end()

#define fi first

#define se second

#define SZ(x) ((int)(x).size())

typedef vector<int> VI;

typedef long long ll;

typedef pair<int,int> PII;

const ll mod=1000000007;

ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1){if(b&1)res=res\*a%mod;a=a\*a%mod;}return res;}

// head

int \_,n;

namespace linear\_seq {

const int N=10010;

ll res[N],base[N],\_c[N],\_md[N];

vector<int> Md;

void mul(ll \*a,ll \*b,int k) {

rep(i,0,k+k) \_c[i]=0;

rep(i,0,k) if (a[i]) rep(j,0,k) \_c[i+j]=(\_c[i+j]+a[i]\*b[j])%mod;

for (int i=k+k-1;i>=k;i--) if (\_c[i])

rep(j,0,SZ(Md)) \_c[i-k+Md[j]]=(\_c[i-k+Md[j]]-\_c[i]\*\_md[Md[j]])%mod;

rep(i,0,k) a[i]=\_c[i];

}

int solve(ll n,VI a,VI b) { // a 系数 b 初值 b[n+1]=a[0]\*b[n]+...

// printf("%d\n",SZ(b));

ll ans=0,pnt=0;

int k=SZ(a);

assert(SZ(a)==SZ(b));

rep(i,0,k) \_md[k-1-i]=-a[i];\_md[k]=1;

Md.clear();

rep(i,0,k) if (\_md[i]!=0) Md.push\_back(i);

rep(i,0,k) res[i]=base[i]=0;

res[0]=1;

while ((1ll<<pnt)<=n) pnt++;

for (int p=pnt;p>=0;p--) {

mul(res,res,k);

if ((n>>p)&1) {

for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;

rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]\*\_md[Md[j]])%mod;

}

}

rep(i,0,k) ans=(ans+res[i]\*b[i])%mod;

if (ans<0) ans+=mod;

return ans;

}

VI BM(VI s) {

VI C(1,1),B(1,1);

int L=0,m=1,b=1;

rep(n,0,SZ(s)) {

ll d=0;

rep(i,0,L+1) d=(d+(ll)C[i]\*s[n-i])%mod;

if (d==0) ++m;

else if (2\*L<=n) {

VI T=C;

ll c=mod-d\*powmod(b,mod-2)%mod;

while (SZ(C)<SZ(B)+m) C.pb(0);

rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c\*B[i])%mod;

L=n+1-L; B=T; b=d; m=1;

} else {

ll c=mod-d\*powmod(b,mod-2)%mod;

while (SZ(C)<SZ(B)+m) C.pb(0);

rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c\*B[i])%mod;

++m;

}

}

return C;

}

int gao(VI a,ll n) {

VI c=BM(a);

c.erase(c.begin());

rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;

return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));

}

};

int main() {

while (~scanf("%d",&n)) {

vector<int>v;

v.push\_back(2);//前几项

v.push\_back(4);

v.push\_back(6);

v.push\_back(8);

v.push\_back(10);

v.push\_back(12);

//输入n ,输出第n项的值

printf("%d\n",linear\_seq::gao(v,n-1));

}

}

### 5.重载运算符

Triple& operator=(Triple& value){ // Triple为结构体名称

j = value.i;

i = value.j;

date=value.date;

return \*this;

}