一、头文件
二、简单函数
1.getline函数:读入整行
2.atoi函数: 将字符串转换成数字
3.reverse函数: 逆转字符串
4.查找函数: 返回查找字符的数组下标
5.查找函数2: 返回第一个不是输入字符的数组下标
6.整型转string函数
7.闰年判断
8.Unique函数示例
9.最大公约数+最小公倍数
10.next_permutation
三、数据结构
1.区间合并
2.线段树
3.矩阵快速幂
4.单调栈、单调队列
5.莫队算法
1）普通莫队
2）带修莫队
6.并查集
四、数论
1.扩展欧几里得
2.求逆元
1）扩展欧几里得求法
2）简洁写法
3）欧拉函数求法
3.中国剩余定理
4.埃式筛素数
5.线性筛素数
6.区间筛素数
7.Miller Robbin素数判断
8.快速幂和防爆乘
9.基本定理和公式
1）平方和公式
2）阶乘分解
3）算术基本定理拓展
4）费马小定理
5）多项式性质
10.二次剩余定理
1） $x^2 = n \pmod p$
2） $ax^2 + bx + c = 0 \pmod p$
五、字符串算法
1.kmp算法
1）计算 $s2$ 在 $s1$ 中出现的次数
2）将子串分为两个以上的相同子子串(循环节)
2.拓展kmp算法
3.Manacher算法
1）计算最长回文长度
2）处理多倍回文串
4.判断字符串是否是自己字典序最小的的子(循环)串
六、计算几何
1.凸包算法(计算凸包周长为例)
2.半平面交算法
3.有向直线平移(向左)

一、头文件

```
#include<bits/stdc++.h>
#include<tr1/unordered_map>
typedef long long ll;
using namespace std;
struct custom_hash{
    static uint64_t splitmix64(uint64_t x){
        x+=0x9e3779b97f4a7c15;
        x=(x^(x>>30))*0xbf58476d1ce4e5b9;
        x=(x^(x>>27))*0x94d049bb133111eb;
        return x^(x>>31);
    }
    size_t operator()(uint64_t x)const{
        static const uint64_t
FIXED_RANDOM=chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x+FIXED_RANDOM);
    }
};
```

二、简单函数

1.getline函数:读入整行

```
getline(cin,s);
```

2.atoi函数: 将字符串转换成数字

```
#include<stdlib.h>
int x=atoi(string.c_str());
int y=atoi(s.substr(0,4).c_str());
```

3.reverse函数: 逆转字符串

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

4.查找函数: 返回查找字符的数组下标

```
int x=s1.find(s2);//找不到时返回乱码,可用==string::npos判断
```

5.查找函数2: 返回第一个不是输入字符的数组下标

```
int ip=s.find_first_not_of('x');
```

6.整型转string函数

```
#include<sstream>
string tostr(char c){
  ostringstream s;
  s<<year<<c<<month<<c<<day;
  return s.str();
}
```

7.闰年判断

```
bool isLeapYear(int y){
  return (y%100!=0&&y%4==0)||(y%400==0);
}
```

8.Unique函数示例

```
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
int main(){
  int arr[5]={1,1,2,2,3};
  vector<int> a(arr,arr+5);
  for(int i=0;i<a.size();i++)
     cout<<a[i]<<" ";
  cout<<endl;
  vector<int>::iterator p=unique(a.begin(),a.end());
  a.erase(p,a.end());
  for(int i=0;i<a.size();i++)
     cout<<a[i]<<" ";
   cout<<endl;
}
```

9.最大公约数+最小公倍数

```
int gcd(int x, int y){
    if(x%y==0)return y;
    else return gcd(y,x%y);
}
int lcm(int x,int y){
    return x*y/gcd(x,y);
}
```

10.next_permutation

```
while(next_permutation(s.begin(),s.end())){}//字符串
while(next_permutation(a,a+n)){}//数组
while(next_permutation(a.begin(),a.end())){}//vector
```

三、数据结构

1.区间合并

```
#include<bits/stdc++.h>
using namespace std;
class Interval {//区间类
public:
    double start,end;
    Interval(double s,double e):start(s),end(e){}
};
bool cmp(Interval a,Interval b){//按左区间排序
    return a.start<b.start;
}
vector<Interval>merge(vector<Interval>s){//区间合并
    if (s.empty())
        return s;
    sort(s.begin(), s.end(),cmp);
    vector<Interval>res;
    res.push_back(s[0]);
    for (int i=1;i<s.size();++i){
        if(res.back().end<s[i].start)
            res.push_back(s[i]);
        else
            res.back().end=max(res.back().end,s[i].end);
    }
    return res;
}
int main(){
    vector<Interval>a;
    int n=4;//4个区间
    while(n--){
        double x,y;
        cin>>x>>y;
        a.push_back(Interval(x,y));
    }
    a=merge(a);
    for(int i=0;i<a.size();i++)//合并后按左区间排序
        cout<<a[i].start<<" "<<a[i].end<<endl;
}
```

2.线段树

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN=2e5+5;
struct node{//线段树节点
    int l,r;
    ll w,f;
}tree[4*MAXN];//四倍空间
void build(int k,int nl,int nr){//构造线段树
    tree[k].l=nl;
    tree[k].r=nr;
    if(tree[k].l==tree[k].r){
        scanf("%lld",&tree[k].w);
        return;
    }
    int m=(nl+nr)/2;
    build(2*k,nl,m);
    build(2*k+1,m+1,nr);
```

```
    tree[k].w=tree[2*k].w+tree[2*k+1].w;
}
void down(int k){//懒标记下传
    tree[2*k].f+=tree[k].f;
    tree[2*k+1].f+=tree[k].f;
    tree[2*k].w+=tree[k].f*(tree[2*k].r-tree[2*k].l+1);
    tree[2*k+1].w+=tree[k].f*(tree[2*k+1].r-tree[2*k+1].l+1);
    tree[k].f=0;
}
int ask_point(int k,int x){//单点查询
    if(tree[k].l==tree[k].r)
        return tree[k].w;
    if(tree[k].f)
        down(k);
    int m=(tree[k].l+tree[k].r)/2;
    if(x<=m)
        return ask_point(2*k,x);
    else
        return ask_point(2*k+1,x);
}
void point_add(int k,int x,int num){//单点修改
    if(tree[k].l==tree[k].r){
        tree[k].w+=num;
        return;
    }
    if(tree[k].f)
        down(k);
    int m=(tree[k].l+tree[k].r)/2;
    if(x<=m)
        point_add(2*k,x,num);
    else
        point_add(2*k+1,x,num);
    tree[k].w=tree[2*k].w+tree[2*k+1].w;
}
ll sum(int k,int x,int y){//区间求和
    if(x==tree[k].l&&y==tree[k].r)
        return tree[k].w;
    if(tree[k].f)
        down(k);
    int m=(tree[k].l+tree[k].r)>>1;
    if(y<=m)
        return sum(k<<1,x,y);
    else if(x>m)
        return sum(k<<1|1,x,y);
    else
        return sum(k<<1,x,m)+sum(k<<1|1,m+1,y);
}
void change_interval(int k,int a,int b,int x){//区间修改
    if(a<=tree[k].l&&b>=tree[k].r){
        tree[k].w+=(tree[k].r-tree[k].l+1)*x;
        tree[k].f+=x;
        return;
    }
    if(tree[k].f)
        down(k);
    int m=(tree[k].l+tree[k].r)/2;
    if(a<=m)
        change_interval(2*k,a,b,x);
```

```
    if(b>m)
        change_interval(2*k+1,a,b,x);
    tree[k].w=tree[2*k].w+tree[2*k+1].w;
}
int main(){
    int n,m;
    scanf("%d",&n);//n个底层节点
    scanf("%d",&m);//m个查询
    build(1,1,n);
    while(m--){
        int q;
        scanf("%d",&q);
        if(q==1){
            int a,b,x;
            scanf("%d%d%d",&a,&b,&x);
            change_interval(1,a,b,x);
        }
        else{
            int pos,p2;
            scanf("%d",&pos);
            printf("%d\n",ask_point(1,pos));
        }
    }
}
```

3.矩阵快速幂

```
#include<iostream>
#include<algorithm>
#include<cstring>
using namespace std;
typedef long long ll;
const int mod=1e9+7;
const int maxn=5;//最大矩阵边长
struct Matrix{//矩阵类
    int n,m;
    ll v[maxn][maxn];
    Matrix(int n,int m):n(n),m(m){init();}
    void init(){
        memset(v,0,sizeof(v));
    }
    Matrix(ll a[maxn][maxn],ll x,ll y):n(x),m(y){//用数组初始化矩阵
        for(int i=0;i<x;i++)
            for(int j=0;j<y;j++)
                v[i][j]=a[i][j];
    }
    Matrix operator*(const Matrix B)const{//重载矩阵乘法
        Matrix C(n,B.m);
        for(int i=0;i<n;i++)
            for(int j=0;j<B.m;j++)
                for(int k=0;k<m;k++)
                    C.v[i][j]=(v[i][k]*B.v[k][j]%mod+C.v[i][j]+mod)%mod;
        return C;
    }
};
Matrix unit(maxn,maxn);//单位矩阵
Matrix qpow(Matrix a,ll x){//矩阵快速幂
```

```
    Matrix ret=unit;
    while(x){
        if(x&1)ret=ret*a;
        a=a*a;
        x>>=1;
    }
    return ret;
}
int main(){
    for(int i=0;i<maxn;i++)//初始化单位矩阵
        unit.v[i][i]=1;
//数组一定要开maxn
    ll a[5][5]={1,ax*by%mod,ay*bx%mod,ax*bx%mod,ay*by%mod,
0,ax,0,0,ay,
                0,0,bx,0,by,
                0,ax*by%mod,ay*bx%mod,ax*bx%mod,ay*by%mod,
                0,0,0,0,1};
    Matrix left(a,5,5);
    ll b[5][5]={{a0*b0%mod},{a0},{b0},{a0*b0%mod},{1}};
    Matrix right(b,5,1);
    right=qpow(left,n-1)*right;//注意次数
}
```

4.单调栈、单调队列

```
#include<iostream>
#include<stdio.h>
using namespace std;
int main(){
    int n,h[80005];
    scanf("%d",&n);
    for(int i=0;i<n;i++)
        scanf("%d",&h[i]);
    int see[80005],top=1;
    long long sum=0;
    see[0]=h[0];
    for(int i=1;i<n;i++){
        while(see[top-1]<=h[i]&&top!=0)
            top--;
        see[top++]=h[i];
        sum+=top-1;
    }
    printf("%lld\n",sum);
}
```

5.莫队算法

1）普通莫队

```
#include<iostream>
#include<cmath>
#include<algorithm>
#include<stdio.h>
using namespace std;
typedef long long ll;
int block;//每个块的大小
```

```
int col[50005];
int bex[50005];//block_index
int cnum[50005];
struct query{
    ll l,r,ID;
    ll A,B;//A是分子,B是分母
}q[50005];
ll gcd(ll x, ll y){
    if(x%y==0)return y;
    else return gcd(y,x%y);
}
bool cmp(query a,query b){//玄学奇偶排序优化
    return (bex[a.l]^bex[b.l])?bex[a.l]<bex[b.l]:(bex[a.l]&1?a.r<b.r:a.r>b.r);
}
bool idcmp(query a,query b){
    return a.ID<b.ID;
}
ll ans=0;
void change(int x,int add){
    ans-=cnum[col[x]]*cnum[col[x]];
    cnum[col[x]]+=add;
    ans+=cnum[col[x]]*cnum[col[x]];
}
int main(){
    int n,m;
    scanf("%d%d",&n,&m);
    block=sqrt(n);
    for(int i=1;i<=n;i++){
        scanf("%d",&col[i]);
        bex[i]=i/block;
    }
    std::fill(cnum,cnum+50005,0);
    for(int i=1;i<=m;i++){
        scanf("%lld%lld",&q[i].l,&q[i].r);
        q[i].ID=i;
    }
    sort(q+1,q+m+1,cmp);
    int l=1,r=0;//注意l,r初值
    for(int i=1;i<=m;i++){
        while(l<q[i].l)
            change(l,-1),l++;
        while(l>q[i].l)
            change(l-1,1),l--;
        while(r<q[i].r)
            change(r+1,1),r++;
        while(r>q[i].r)
            change(r,-1),r--;
        if(q[i].l==q[i].r){
            q[i].A=0;
            q[i].B=1;
            continue;
        }
        q[i].A=ans-(q[i].r-q[i].l+1);
        q[i].B=(q[i].r-q[i].l+1)*(q[i].r-q[i].l);
        ll g=gcd(q[i].A,q[i].B);
        q[i].A/=g;
        q[i].B/=g;
    }
```

```
    sort(q+1,q+m+1,idcmp);
    for(int i=1;i<=m;i++)
        printf("%lld/%lld\n",q[i].A,q[i].B);
}
```

2）带修莫队

```
#include<bits/stdc++.h>
using namespace std;
struct query{//查询
    int l,r,id,pos,ans;
}q[200005];
struct replace{//单点修改
    int pos,col;
}r[200005];
int n,m,block;
int kind=0;//当前的颜色种类
int col[200005];
int cnt[1000005];
int bex[200005];//block_index
int ans[200005];
bool cmp(query a,query b){
    if(bex[a.l] != bex[b.l])//玄学优化
        return a.l < b.l;
    if(bex[a.r] != bex[b.r])
        return a.r < b.r;
    return a.id < b.id;
}
bool cmp2(query a,query b){
    return a.pos<b.pos;
}
int L=1,R=0;//注意初值
//inline优化时间
inline void update(int x){
    if(L<=r[x].pos&&r[x].pos<=R){
        kind-=!--cnt[col[r[x].pos]];
        //cnt[col[r[x].pos]]--;
        //if(cnt[col[r[x].pos]]==0)
        //  kind--;
        kind+=!cnt[r[x].col]++;
        //cnt[r[x].col]++;
        //if(cnt[r[x].col]==1)
        //  kind++;
    }
    swap(col[r[x].pos],r[x].col);//可来回修改
}
inline void erase(int x){
    kind-=!--cnt[col[x]];
    //cnt[col[x]]--;
    //if(cnt[col[x]]==0)
    //  kind--;
}
inline void insert(int x){
    kind+=!cnt[col[x]]++;
    //cnt[col[x]]++;
    //if(cnt[col[x]]==1)
    //  kind++;
```

```
}
int main(){
    scanf("%d%d",&n,&m);
    block=pow(n,2.0/3);
    for(int i=1;i<=n;i++){
        scanf("%d",col+i);
        bex[i]=i/block;
    }
    int qnum=0,rnum=0;
    scanf("\n");
    for(int i=1;i<=m;i++){
        char opt;
        //cin>>opt;
        //输入优化
        while(scanf("%c",&opt)==1)
            if(opt=='Q'||opt=='R')
                break;
        if(opt=='Q'){
            qnum++;
            scanf("%d%d",&q[qnum].l,&q[qnum].r);
            q[qnum].id=rnum;//时间戳
            q[qnum].pos=qnum;
        }
        else{
            rnum++;
            scanf("%d%d",&r[rnum].pos,&r[rnum].col);
        }
    }
    sort(q+1,q+qnum+1,cmp);
    int now=0;
    for(int i=1;i<=qnum;i++){
        while(L<q[i].l)
            erase(L++);
            //erase(L),L++;
        while(L>q[i].l)
            insert(--L);
            //insert(L-1),L--;
        while(R<q[i].r)
            insert(++R);
            //insert(R+1),R++;
        while(R>q[i].r)
            erase(R--);
            //erase(R),R--;
        while(now<q[i].id)
            update(++now);
            //update(now+1),now++;
        while(now>q[i].id)//相当于撤销now时的操作
            update(now--);
            //update(now),now--;
        q[i].ans=kind;
    }
    sort(q+1,q+qnum+1,cmp2);
    for(int i=1;i<=qnum;i++)
        printf("%d\n",q[i].ans);
}
```

6.并查集

```
#include<bits/stdc++.h>
using namespace std;
int pre[10005];
int unionsearch(int now){
    int son=now;
    while(now!=pre[now])
        now=pre[now];
    while(son!=now){//路径压缩,直连根节点
        int tmp=pre[son];
        pre[son]=now;
        son=tmp;
    }
    return now;//返回根
}
int main(){
    int T;
    scanf("%d",&T);
    while(T--){
        int n,m;
        scanf("%d%d",&n,&m);
        int total=n;
        for(int i=1;i<=n;i++)//一开始根就是自己
            pre[i]=i;
        while(m--){
            int s,e;
            scanf("%d%d",&s,&e);
            int r1=unionsearch(s);
            int r2=unionsearch(e);
            if(r1!=r2){
                pre[r1]=r2;
                total--;
            }
        }
        printf("%d\n",total);
    }
}
```

四、数论

1.扩展欧几里得

```
#include<bits/stdc++.h>
typedef long long ll;
using namespace std;
ll a,b,c,x,y;
int gcd(int x,int y){//最大公约数
    return x%y==0?y:gcd(y,x%y);
}
ll exgcd(ll a,ll b,ll &x,ll &y){
    if(a==0&&b==0)//无最大公约数
        return -1;
    if(b==0){
        x=1;y=0;
        return a;
```

```
    }
    ll d=exgcd(b,a%b,y,x);
    y-=a/b*x;
    return d;//返回d=gcd(a,b);和对应于等式ax+by=d中的x,y
}
int main(){
    while(~scanf("%lld%lld%lld",&a,&b,&c)//ax+by=c
    {
        if(c%gcd(a,b)!=0)
            printf("no answer\n");//当 c%gcd(a,b)!=0 时无整数解
        else {
            exgcd(a,b,x,y);
            ll d=gcd(a,b),mod1=b/d,mod2=a/d;
            ll x0=x*c/d;//x0
            ll y0=y*c/d;//y0
            printf("%lld %lld\n",x0,y0);
            //其他的解满足x=x0 +b/gcd*t, y=y0 -a/gcd*t ,t为任意整数
            x=(x0%mod1+mod1)%mod1;//x的最小正整数解
            y=(c-a*x)/b;//对应x的y
            printf("%lld%lld\n",x,y)
        }
    }
}
```

2.求逆元

1）扩展欧几里得求法

```
//ax=1(mod n)
ll mod_reverse(ll a,ll n){
 ll x,y;
 ll d=extend_gcd(a,n,x,y);
 if(d==1)
    return (x%n+n)%n;
 else
    return -1;
}
```

2）简洁写法

```
//注意:这个只能求a<m的情况,而且必须保证a和m互质
//求ax=1(mod m)的x值,就是逆元(0<a<m)
ll inv(ll a,llg m){
 if(a == 1)
    return 1;
 return inv(m%a,m)*(m-m/a)%m;
}
```

3）欧拉函数求法

```
//mod为素数,而且a和m互质
ll inv(ll a,ll mod){//mod为素数
return pow_m(a,mod-2,mod);
}
```

3.中国剩余定理

设正整数 $m_1, m_2, \cdots, m_k$ 两两互素，则同余方程组

$$\begin{aligned} x &\equiv a_1 (mod \quad m_1) \\ x &\equiv a_2 (mod \quad m_2) \\ x &\equiv a_3 (mod \quad m_3) \\ &\cdots \\ x &\equiv a_k (mod \quad m_k) \end{aligned}$$

有整数解。

并且在模 $M = m_1 * m_2 * \cdots * m_k$ 下的解是唯一的，

解为 $x \equiv (a_1 M_1 M_1^{-1} + a_2 M_2 M_2^{-1} + \cdots + a_k M_k M_k^{-1}) mod \quad M$ 。

其中 $M_i = M / m_i$ ，而 M_i^{-1} 为 M_i 模 m_i 的逆元。

```
#include<bits/stdc++.h>
#define ll long long
ll gcd(ll a,ll b){
    return b==0?a:gcd(b,a%b);
}
void exgcd(ll a,ll b,ll &d,ll &x,ll &y){//扩展欧几里得算法
    if(b==0){
        d=a;x=1;y=0;
    }
    else{
        exgcd(b,a%b,d,y,x);
        y-=(a/b)*x;
    }
}
ll China(int n,ll *m,ll *a){//中国剩余定理
    ll M=1,d,y,x=0;
    for(int i=0;i<n;i++)
        M*=m[i];
    for(int i=0;i<n;i++){
        ll w=M/m[i];
        exgcd(m[i],w,d,d,y);
        x=(x+y*w*a[i])%M;
    }
    return (x+M)%M;
}
ll m[15],a[15];
int main(){
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
        scanf("%lld%lld",&m[i],&a[i]);
    printf("%lld",China(n,m,a));
}
```

4.埃式筛素数

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN=1e9+5;//最大1e9
ll prime[MAXN+1];
//prime[0]存小于MAXN的素数的数量,prime[1]开始存素数
```

```
void getPrime(){
    memset(prime,0,sizeof(prime));
    for(int i=2;i<=MAXN;i++){
        if(!prime[i])
            prime[++prime[0]]=i;
        for(int j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++){
            prime[prime[j]*i]=1;
            if(i%prime[j]==0)
                break;
        }
    }
}
int main(){
    getPrime();
    for(int i=0;i<1005;i++)
    cout<<prime[i]<<endl;
}
```

5.线性筛素数

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN=1e9+5;//最大1e9
ll prime[MAXN+1];
//prime[0]存小于MAXN的素数的数量,prime[1]开始存素数
void getPrime(){
    memset(prime,0,sizeof(prime));
    for(int i=2;i<=MAXN;i++){
        if(!prime[i])
            prime[++prime[0]]=i;
        for(int j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++){
            prime[prime[j]*i]=1;
            if(i%prime[j]==0)
                break;
        }
    }
}
int main(){
    getPrime();
    for(int i=0;i<1005;i++)
    cout<<prime[i]<<endl;
}
```

6.区间筛素数

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll MAXN=1e6+5;
bool notp[MAXN];//下标是合数置1,素数置0
vector<ll>prime;//存小于MAXN的素数
void sieve(){//线性筛
    std::fill(notp,notp+MAXN,0);
    notp[0]=1;
```

```
    notp[1]=1;
    for(ll i=2;i<=MAXN;i++){
        if(notp[i]==0)
            prime.push_back(i);
        for(ll j=0;j<prime.size()&&prime[j]*i<=MAXN;j++){
            notp[i*prime[j]]=1;
            if(i%prime[j]==0)
                break;
        }
    }
}
bool pflag[MAXN];//下标[0,r-l],对应[l,r]
void interval_sieve(ll l,ll r){//区间筛
    std::fill(pflag,pflag+MAXN,1);
    if(l==1)//重要!!
        pflag[0]=0;//l>1时pflag[0]和pflag[1]为1
    for(ll i=0;prime[i]*prime[i]<=r;i++){
        for(ll j=(l-1)/prime[i]+1;j<=r/prime[i];j++)
            if(j>1)
                pflag[prime[i]*j-l]=0;
    }
}
int main(){
    sieve();
    int left,right;
    cin>>left>>right;
    interval_sieve(left,right);
    for(int i=0;i<=right-left;i++)
        cout<<i+left<<" "<<pflag[i]<<endl;
}
```

7.Miller Robbin素数判断

```
#include<cstdlib>
#include<ctime>
#include<cstdio>
using namespace std;
const int count=20;
int modular_exp(int a,int m,int n){
    if(m==0)
        return 1;
    if(m==1)
        return a%n;
    long long w=modular_exp(a,m/2,n);
    w=w*w%n;
    if(m&1)
        w=w*a%n;
    return w;
}
bool Miller_Rabin(int n){
    if(n==2)
        return true;
    for(int i=0;i<count;i++){
        int a=rand()%(n-2)+2;
        if(modular_exp(a,n,n)!=a)
            return false;
    }
```

```
    return true;
}
int main(){
    int n;
    scanf("%d",&n);
    if(Miller_Rabin(n))
        printf("Probably a prime.");
    else
        printf("A composite.");
}
```

8.快速幂和防爆乘

```
ll multi(ll a,ll b,ll mod){
    ll ans=0;
    while(b){
        if(b&1)
            ans=(ans+a)%mod;
        a=(a<<1)%mod;
        b>>=1;
    }
    return ans;
}
ll qpow(ll a,ll b,ll mod){
    ll ans=1;
    while(b){
        if(b&1)
            ans=multi(ans,a,mod);
        a=multi(a,a,mod);
        b>>=1;
    }
    return ans;
}
```

9.基本定理和公式

1）平方和公式

$$\sum_{k=1}^{n} k^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} = \frac{n(n+1)(2n+1)}{6}$$

$$= C_{n+2}^3 + C_{n+1}^3 = \frac{1}{4}C_{n+1}^2 = nC_{n+1}^2 - C_{n+1}^3$$

2）阶乘分解

阶乘 $N!$ 中包含质因子 x 的个数为 $\frac{N}{x} + \frac{N}{x^2} + \frac{N}{x^3} + \cdots$

3）算术基本定理拓展

N 的约数个数 $= (c_1 + 1) * (c_2 + 1) * \cdots * (c_n + 1)$

N 的约数之和

$$= (1 + p_1 + p_1^2 + \cdots p_1^{c_1}) * (1 + p_2 + p_2^2 + \cdots p_2^{c_2}) * \cdots * (1 + p_n + p_n^2 + \cdots + p_n^{c_n})$$

4）费马小定理

若 p 是质数，则对于任意正整数 a，有 $a^p \equiv a(mod \quad p)$

5）多项式性质

实数域不可拆分多项式只有两种：一次多项式和二次的 $b^2 < 4ac$

10.二次剩余定理

1) $x^2 = n \pmod p$

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
ll w;
struct num{
    ll x,y;
};
num mul(num a,num b,ll p){
    num ans={0,0};
    ans.x=((a.x*b.x%p+a.y*b.y%p*w%p)%p+p)%p;
    ans.y=((a.x*b.y%p+a.y*b.x%p)%p+p)%p;
    return ans;
}
ll powwR(ll a,ll b,ll p){
    ll ans=1;
    while(b){
        if(b&1)
            ans=1ll*ans%p*a%p;
        a=a%p*a%p;
        b>>=1;
    }
    return ans%p;
}
ll powwi(num a,ll b,ll p){
    num ans={1,0};
    while(b){
        if(b&1)
            ans=mul(ans,a,p);
        a=mul(a,a,p);
        b>>=1;
    }
    return ans.x%p;
}
ll solve(ll n,ll p){
    n%=p;
    if(p==2)
        return n;
    if(powwR(n,(p-1)/2,p)==p-1)
        return -1;//不存在
    ll a;
    while(1){
        a=rand()%p;
        w=((a*a%p-n)%p+p)%p;
        if(powwR(w,(p-1)/2,p)==p-1)
            break;
    }
    num x={a,1};
    return powwi(x,(p+1)/2,p);
}
int main(){
    int t;
```

```
    scanf("%d",&t);
    while(t--){
        ll n,p;//x^2=n (mod p)
        scanf("%lld%lld",&n,&p);
        if(!n){//n为0的情况,x1=x2=0
            printf("0\n");
            continue;
        }
        ll x1=solve(n,p),x2;
        if(x1==-1)//无解
            printf("Hola!\n");
        else{
            x2=p-x1;
            if(x1>x2)
                swap(x1,x2);
            if(x1==x2)
                printf("%lld\n",x1);
            else
                printf("%lld %lld\n",x1,x2);
        }
    }
}
```

2) $ax^2+bx+c=0 \quad (mod \quad p)$

转化为$y^2=b^2-4c \quad (mod \quad p), \quad y=2ax+b$

```
#include<iostream>
#include<cmath>
using namespace std;
typedef long long ll;
const ll p=1e9+7;
ll x1,x2,y,a,b,c;
ll qpow(ll a,ll b){
    ll ans=1,base=a;
    while(b!=0){
        if(b&1!=0)
            ans=ans*base%p;
        base=base*base%p;
        b>>=1;
    }
    return ans;
}
int main(){
    int t;
    scanf("%d",&t);
    while(t--){
        //ax^2+bx+c=0 (mod p)
        a=1;
        scanf("%lld%lld",&b,&c);
        //转化为 y^2=b^2-4c (mod p)
        //       y=2ax+b
        ll d=(b*b-4*c+p*4)%p;
        //       y^2=d (mod p)
        if(qpow(d,(p-1)/2)==0){//x1=x2
            y=qpow(d,(p+1)/4);
            x1=(((b-y)*(p+1)/2/a)%p+p)%p;
```

```
            printf("%lld %lld\n",x1,x1);
        }
        else if(qpow(d,(p-1)/2)==0||qpow(d,(p-1)/2)==1){//x1,x2不同
            y=qpow(d,(p+1)/4);
            x1=(((b-y)*(p+1)/2/a)%p+p)%p;
            x2=(b-x1+p)%p;
            if(x1<x2)//先输出小的,再输出大的
                printf("%lld %lld\n",x1,x2);
            else
                printf("%lld %lld\n",x2,x1);
        }
        else if(qpow(d,(p-1)/2)==p-1){//无解
            printf("-1 -1\n");
            continue;
        }
    }
}
```

五、字符串算法

1.kmp算法

1）计算 $s2$ 在 $s1$ 中出现的次数

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
const int maxn=1e6+5;
//构造next数组,存子串最大相同前缀和后缀的长度
//从a[1]开始存
void nex(string s,int a[]){
    a[0]=a[1]=0;
    for(int i=1;i<s.size();i++){
        int j=a[i];
        while(j&&s[i]!=s[j])
            j=a[j];
        a[i+1]=s[i]==s[j]?j+1:0;
    }
}
//计算s2在s1中出现的次数
int F[maxn];
void count(string s1,string s2){
    int cnt=0;
    nex(s2,F);
    for(int i=0,j=0;i<s1.size();i++){
        while(j&&s1[i]!=s2[j])
            j=F[j];
        if(s1[i]==s2[j])
            j++;
        if(j==s2.size())//匹配完成
            cnt++;
    }
    printf("%d\n",cnt);
}
```

```
char s1[maxn],s2[maxn];
int main(){
    int t;
    scanf("%d",&t);
    while(t--){
        scanf("%s%s",s2,s1);
        count(s1,s2);
    }
}
```

2）将子串分为两个以上的相同子子串(循环节)

```
int f[maxn];
nex(s,f);
    for(int i=2;i<=n;i++){
        if(f[i]>0&&i%(i-f[i])==0)
            printf("%d %d\n",i,i/(i-f[i]));
    }
```

2.拓展kmp算法

```
#include<bits/stdc++.h>
using namespace std;
#define N 1000010
//存s2中从i开始的后缀字串与s2的最长公共前缀
int nxt[N];//从nxt[0]到nxt[strlen(s2)-1]
void next(char*s2,int nxt[]){
    int n=strlen(s2);
    nxt[0]=n;
    nxt[1]=0;
    while (1+nxt[1]<n && s2[nxt[1]]==s2[1+nxt[1]])
        nxt[1]++;
    int pos=1,mr=1+nxt[1];
    for (int i=2;i<n;++i){
        if (i<mr)
            nxt[i]=min(i+nxt[i-pos],mr)-i;
        else
            nxt[i]=0;
        while (i+nxt[i]<n && s2[nxt[i]]==s2[i+nxt[i]])
            nxt[i]++;
        if (i+nxt[i]>mr){
            mr=i+nxt[i];
            pos=i;
        }
    }
}
//存s1从i开始的后缀字串和s2的最长公共前缀
int ex[N];//从ex[0]到ex[strlen(s1)-1]
void exkmp(char*s1,char*s2){
    next(s2,nxt);
    int n=strlen(s1),m=strlen(s2);
    int pos=-1,mr=0;
    for (int i=0;i<n;++i){
        if (i<mr)
            ex[i]=min(i+nxt[i-pos],mr)-i;
        else
```

```
            ex[i]=0;
        while (ex[i]<m && i+ex[i]<n && s2[ex[i]]==s1[i+ex[i]])
            ex[i]++;
        if (i+ex[i]>mr){
            mr=i+ex[i];
            pos=i;
        }
    }
}
char s1[N],s2[N];
int main(){
    long long ans=0;
    scanf("%s%s",s1,s2);
    exkmp(s1,s2);
    for(int i=0;i<strlen(s2);i++)
        cout<<nxt[i]<<" ";cout<<endl;
    for(int i=0;i<strlen(s1);i++)
        cout<<ex[i]<<" ";cout<<endl;
}
```

3.Manacher算法

1）计算最长回文长度

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e8+5;
void init(char *str){//改变原字符串
    for(int i=strlen(str);i>=0;i--)
    {
        str[2*i+2]=str[i];
        str[2*i+1]='#';
    }
    str[0]='$';
}
//计算最长回文长度
int rad[maxn];//存每个位置包括中点的回文半径
int Manacher(char*s){
    int len=strlen(s);
    init(s);
    int r=0,mid=0;
    for(int i=0;i<=2*len;i++){
        if(r>i)
            rad[i]=min(r-i,rad[2*mid-i]);
        else
            rad[i]=1;
        while(s[i-rad[i]]==s[i+rad[i]])
            rad[i]++;
        if(i+rad[i]>r){
            r=i+rad[i];
            mid=i;
        }
    }
    int ans=0;
    for(int i=2;i<len*2+1;i++)//注意范围
            ans=max(ans,rad[i]-1);
    return ans;
```

```
}
char s[maxn];
int main(){
    while(~scanf("%s",s))
    printf("%d\n",Manacher(s));
}
```

2）处理多倍回文串

```
for(int i=3;i/2<=len;i+=2)//处理双倍回文半径
        p[i/2]=rad[i]/2;
    int ans=0;
    set<int>t;
    for(int i=1;i<=len;i++) q[i]=i;
    sort(q+1,q+len+1,cmp);
    int now=1;
    for(int i=1;i<=len;i++){
        while(now<=len&&q[now]-p[q[now]]<=i){
            t.insert(q[now]);
            now++;
        }
        set<int>::iterator tmp=t.upper_bound(i+p[i]/2);
        if(tmp!=t.begin()){
            ans=max(ans,(*--tmp-i)*4);
        }
    }
```

4.判断字符串是否是自己字典序最小的的子(循环)串

```
bool judge(string s){
    for(int i=1;i<s.size();i++)//起点从i开始
        for(int j=0;j<s.size();j++){//依次比较
            if(s[j]>s[(i+j)%s.size()])
                return false;
            else if(s[j]<s[(i+j)%s.size()])
                break;
        }
    return true;
}
```

六、计算几何

1.凸包算法(计算凸包周长为例)

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<vector>
#include<algorithm>
#define eps 1e-7
using namespace std;
struct point{
    double x,y,v,l;
```

```
}p[15];
vector<point>d;//存放形成凸包的点集
double cross(point p0,point p1,point p2){//叉积
    return (p1.x-p0.x)*(p2.y-p0.y)-(p1.y-p0.y)*(p2.x-p0.x);
}
double dis(point p1,point p2){//计算距离
    return sqrt((double)(p2.x-p1.x)*(p2.x-p1.x)+(p2.y-p1.y)*(p2.y-p1.y));
}
bool cmp(point p1,point p2){//极角排序
    if(cross(d[0],p1,p2)>0)
        return true;
    else if(fabs(cross(d[0],p1,p2))<eps&&dis(d[0],p1)<dis(d[0],p2))
        return true;
    return false;
}
vector<point>s;//存放凸包的顶点
void get_convex_hull(){
    for(int i=1;i<d.size();i++)
        if(d[i].y<d[0].y||(d[i].y==d[0].y&&d[i].x<d[0].x))
            swap(d[0],d[i]);
    sort(d.begin()+1,d.end(),cmp);
    s.clear();
    s.push_back(d[0]);
    s.push_back(d[1]);
    s.push_back(d[2]);
    for(int i=3;i<d.size();i++){
        while(s.size()>=2&&cross(s[s.size()-2],s[s.size()-1],d[i])<eps)
            s.pop_back();
        s.push_back(d[i]);
    }
    s.push_back(s[0]);
}
double cal_peri(){//求凸包周长
    if(d.size()==1)//只有1个点
        return 0;
    else if(d.size()==2)//只有两个点
        return dis(d[0],d[1])*2;
    get_convex_hull();
    double ans=0;
    for(int i=0;i<s.size()-1;i++)
        ans+=dis(s[i],s[i+1]);
    return ans;
}
int n;
int main(){
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        point a;
        scanf("%lf%lf",&a.x,&a.y);
        d.push_back(a);
    }
    printf("%.2f\n",cal_peri());
}
```

2.半平面交算法

```
#include<iostream>
```

```cpp
#include<stdio.h>
#include<algorithm>
#include<cmath>
#define eps 1e-8
using namespace std;
const int maxn=1000;
typedef struct grid{
    double x,y;
    grid(double a=0,double b=0):x(a),y(b){}
}point,vec;
struct line{//有向直线
    point s,e;//s是起点,e是终点
    line(){}
    line(point a,point b):s(a),e(b){}
};
int n;//点的数量
point p[maxn];//p存所有点
line L[maxn],que[maxn];//L存所有边,que存半平面交的边
vec operator-(point a,point b){//两点相减得到一向量
    return vec(b.x-a.x,b.y-a.y);
}
double operator^(vec a,vec b){//向量叉积
    return a.x*b.y-a.y*b.x;
}
double Angle(vec a){//向量极角
    return atan2(a.y,a.x);
}
double Angle(line a){//有向直线极角
    return atan2(a.e.y-a.s.y,a.e.x-a.s.x);
}
bool Anglecmp(line a,line b){//极角排序从小到大
    vec va=a.e-a.s,vb=b.e-b.s;
    double Aa=Angle(a),Ab=Angle(b);
    if(fabs(Aa-Ab)<eps)//极角相同,把最左边的放最后面,以便去重
        return ((va)^(b.e-a.s))>=0;
    return Aa<Ab;
}
point intersect(line a,line b){//两直线交点
    double a1=a.s.y-a.e.y;
    double b1=a.e.x-a.s.x;
    double c1=a.s.x*a.e.y-a.e.x*a.s.y;
    double a2=b.s.y-b.e.y;
    double b2=b.e.x-b.s.x;
    double c2=b.s.x*b.e.y-b.e.x*b.s.y;
    point ans;
    ans.x=(c1*b2-c2*b1)/(a2*b1-a1*b2);
    ans.y=(c1*a2-c2*a1)/(a1*b2-a2*b1);
    return ans;
}
bool onRight(line a,line b,line c){//判断b,c的交点是否在a的右边
    point o=intersect(b,c);
    if(((a.e-a.s)^(o-a.s))<0)
        return true;
    return false;
}
bool halfplane(){//判断能否形成半平面交
    sort(L,L+n,Anglecmp);//对多边形的边极角排序
    int head=0,tail=0,cnt=0;//模拟双端队列
```

```
    for(int i=0;i<n-1;i++){//对边去重
        if(fabs(Angle(L[i])-Angle(L[i+1]))<eps)//极角相同只取最后一个,即最左边的
            continue;
        L[cnt++]=L[i];
    }
    L[cnt++]=L[n-1];
    for(int i=0;i<cnt;i++){//判断新加入直线的影响
        while(tail-head>1&&onRight(L[i],que[tail-1],que[tail-2]))
            tail--;
        while(tail-head>1&&onRight(L[i],que[head],que[head+1]))
            head++;
        que[tail++]=L[i];
    }
    //判段第一条直线的影响
    while(tail-head>1&&onRight(que[head],que[tail-1],que[tail-2]))
        tail--;
    //判断最后一条直线的影响
    while(tail-head>1&&onRight(que[tail-1],que[head],que[head+1]))
        head++;
    if(tail-head<3)//不能形成半平面交
        return false;
    return true;
}
bool judge(){//判断输入的点按顺时针顺序还是逆时针顺序
    double ans=0;
    for(int i=1;i<=n-2;i++)
        ans+=((p[i]-p[0])^(p[i+1]-p[0]));
    return ans>0;//面积>0则为逆时针
}
int main(){
    int t;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        for(int i=0;i<n;i++)
            scanf("%lf%lf",&p[i].x,&p[i].y);
        //判断点输入的顺序
        if(judge())//逆时针
            for(int i=0;i<n;i++)
                L[i]=line(p[i],p[(i+1)%n]);
        else//顺时针
            for(int i=n-1;i>=0;i--)
                L[n-1-i]=line(p[i],p[(i-1+n)%n]);
        if(halfplane())
            printf("YES\n");
        else
            printf("NO\n");
    }
}
```

3.有向直线平移(向左)

```
line tmp[maxn];//存平移后的直线
double r;
void change(){//将line向左边平移r的距离
    for(int i=0;i<n;i++){
        tmp[i]=L[i];
```

```
        double len=dis(L[i].s,L[i].e);
        double dx=(L[i].s.y-L[i].e.y)*r/len;
        double dy=(L[i].e.x-L[i].s.x)*r/len;
        tmp[i].s.x+=dx;
        tmp[i].e.x+=dx;
        tmp[i].s.y+=dy;
        tmp[i].e.y+=dy;
    }
}
```