数学:

数论:

素数筛选:

```
//素数筛选法打表
void Prim() {
    int i,j;
    len = 0;
    memset(f, 0, sizeof(f));
    f[0] = f[1] = 1;
    for (i = 2; i * i < maxn; ++i) {
       if (!f[i]) {
           for (j = 2 * i; j < maxn; j = j + i) {
               f[j] = 1;
           }
       }
    }
    for (i = 2; i < maxn; ++i) {
        if (!f[i]) p[len++] = i;
    }
}
```

反素数求解+反素数打表

问题描述: 对于任何正整数`x`,起约数的个数记做`g(x)`.例如`g(1)=1`,`g(6)=4`. 如果某个正整数`x`满足:对于任意`i(0<i<x)`,都有`g(i)<g(x)`,则称`x`为反素数. 现在给一个**N**,求出不超过**N**的最大的反素数. 比如:输入**1000** 输出**840** 思维过程: 求`[1..N]`中约数在大的反素数**-->**求约数最多的数 如果求约数的个数 **756=2^{2*3}3*7^1** `(2+1)*(3+1)*(1+1)=24` 基于上述结论,给出算法:按照质因数大小递增顺序搜索每一个质因子,枚举每一个质因子 为了剪枝: 性质一:一个反素数的质因子必然是从`2`开始连续的质数. 因为最多只需要`10`个素数构造:2,3,5,7,11,13,17,19,23,29 性质二:**p=2^{t1*3}t25^*t37*t4.....必然t1>=t2>=t3>=....**

反素数求解函数:

```
#include <cstdio>
#include <cstring>
#define ll long long
using namespace std;

const int inf = 999999999;
const int prime[16]= {1,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47};
int n;
ll bnum,bsum;
//当前枚举到的数; 枚举到的第K大的质因子; 该数的约数个数; 质因子个数上限。
void getantiprime(ll num,ll k,ll sum,ll limit)
```

```
{
    //如果约数个数更多，将最优解更新为当前数。
    if (bsum < sum)
    {
        bsum = sum;
        bnum = num;
    }
    //如果约数个数相同，将最优解更新为较小的数。
    if (bsum == sum && bnum > num) bnum = num;
    if (k > 10) return ;
    ll tmp = num;
     //开始枚举每个质因子的个数。
    for (int i = 1; i <= limit; ++i)
    {
        if (tmp*prime[k] > n) break;
        tmp = tmp*prime[k];
        getantiprime(tmp,k + 1,sum*(i + 1),i);
    }
}
int main()
{
    int t;
    scanf("%d",&t);
    while (t--)
    {
        bsum = -inf; bnum = inf;
        scanf("%d",&n);
        getantiprime(1,1,1,50);
        printf("%lld\n",bnum);
    }
    return 0;
}
```

反素数打表函数：

```
#include<iostream>
#include<cstring>
#include <cstdio>
#define maxn 600000
using namespace std;

int dp[600001];
int main()
{
    int i,j;
    freopen("out.txt","w",stdout);
    memset(dp,0,sizeof(dp));
    for(i=1;i<=maxn;i++)
    {
        for(j=1;i*j<=maxn;j++)
        {
                dp[i*j]++;
```

```
        }
    }
    int max=0;
    for(i=2;i<=maxn;i++)
    {
        if(dp[i]>max)
        {
            max=dp[i];
            cout<<i<<",";
        }
    }
    cout<<endl<<endl;
    max=0;
    for(i=2;i<=maxn;i++)
    {
        if(dp[i]>max)
        {
            max=dp[i];
            cout<<dp[i]<<",";
        }
    }
    return 0;
}
int antip[] = {1,2,4,6,12,24,36,48,60,120,180,240,360,720,
840,1260,1680,2520,5040,7560,10080,15120,20160,25200,27720,
45360,50400,55440,83160,110880,166320,221760,277200,
332640,498960,554400};
int pnum[] = {1,2,3,4,6,8,9,10,12,16,18,20,24,30,32,36,40,48,
60,64,72,80,84,90,96,100,108,120,128,144,160,168,180,192,200,
216};
```

欧拉函数：

欧拉函数:

定义：用于计算 p(n)，比n小的所有与n互质的数。

计算公式：p(n)=n*(1-1/p1)*(1-1/p2)....*(1-1/pk)【p1,p2,pk都是n的素因子】

另：若n=p1^{q1*p2}q2.....pk^qk

则，p(n)=(p1-1)*p1^(q1-1)*(p1-1)*p2^(q2-1)*.....(pk-1)*pk^(qk-1)

性质：若m,n互质，φ(mn)=φ(m)φ(n)。当n为奇数时，φ(2n)=φ(n)

欧拉定理：

a,m互质，a^φ(m)≡1(mod m)

例：2,3互质，那么，2^2%3=1

推论：对于互质的数a、n，满足a^(φ(n)+1) ≡ a (mod n)

欧拉公式的延伸：一个数的所有质因子之和是euler(n)*n/2。

费马小定理：

假如p是质数，且(a,p)=1，那么 a^(p-1) ≡1（mod p） 假如p是质数，且a,p互质，那么 a的(p-1)次方除以p的余数恒等于1

费马大定理：

当整数n > 2时，关于x, y, z的不定方程 x^n + y^n = z^n. 无正整数解

威尔逊定理：

当仅当p是素数：(p -1)! ≡ -1 (mod p)

```
欧拉函数：
定义：用于计算 p(n)，小于等于n的所有与n互质的数。
计算公式：p(n)=n*(1-1/p1)*(1-1/p2)....*(1-1/pk)【p1,p2,pk都是n的素因子】
另：若n=p1^q1*p2^q2*.....*pk^qk
则，p(n)=(p1-1)*p1^(q1-1)*(p1-1)*p2^(q2-1)......*(pk-1)*pk^(qk-1)
性质：若m,n互质，φ(mn)=φ(m)φ(n)。当n为奇数时，φ(2n)=φ(n)
```

欧拉函数模板：

```
int euler(int n)//返回euler(n)
{
     int i;
     int res = n,a = n;
     for(i = 2;i*i <= a; ++i)
     {
         if(a%i == 0)
         {
             res -= res/i; //p(n) = (p - p/p1)(1 - 1/p2)......
             while(a%i == 0) a/=i;
         }
     }
     if(a > 1) res -= res/a;//存在大于sqrt(a)的质因子
     return res;
}
```

欧拉函数打表：

```
void SE()//select euler//类似于素数筛选法
{
    int i,j;
    euler[1] = 1;
    for(i = 2;i < Max; ++i)  euler[i]=i;
    for(i = 2;i < Max; ++i)
    {
         if(euler[i] == i)//这里出现的肯定是素数
         {
```

```
            for(j = i; j < Max; j += i)//然后更新含有它的数
            {
               euler[j] = euler[j]/i*(i - 1);
            }
        }
    }
     //for (int i = 1; i <= 20; ++i) printf("%d ",euler[i]);
}
```

扩展欧几里得：

```
同余：这里主要运用了（2）
同余有三种说法都是等价的,分别为:
   (1) a和b是模d同余的. a=b(mod d)这里的=是三道杠的。
   (2) 存在某个整数n,使得a=b+nd .
   (3) d整除a-b.
```

```
LL Pow(int a,int k)
{
    LL sum = 1;
    for (int i = 1; i <= k; ++i)
    sum *= a;
    return sum;
}
LL exp_gcd(LL a,LL b,LL &x,LL &y)
{
    if (b == 0)
    {
        x = 1; y  =0;
        return a;
    }
    else
    {
        LL d = exp_gcd(b,a%b,x,y);
        LL tmp = x;
        x = y; y = (tmp - (a/b)*y);
        return d;
    }
}
if (p%d != 0)
        {
            printf("FOREVER\n");
        }
        else
        {
            /*
            这里自己开始没理解好，我们求出来的x,y是a*x +  b*y = gcd(a,b)的解,
            而我们要的则是a*x + b*y = p的解，所以要转化一下
            */
            p /= d;
            b /= d;
```

```
            a /= d;
            x *= p;
            x = (x%b + b)%b;
            printf("%I64d\n",x);
        }

```

必须注意的是：在利用扩展欧几里得求解时，要求的a,b必须是非负数，这里a是负数了，所以要处理一下，在利用扩展欧几里德求解时将a = -a;置为正数，把负数加大x上面，于是我们求解出的x事最大的负数，只要将其取反就是最小的正整数了。

```
if (p%d == 0) {
     p /= d;
     a /= d;
     b /= d;
     x*=p;
     x = (x%b + b)%b;
     while (x >= 0) x -= b;//求出最大分数
     x = -x;
     if (ans > x) ans = x;
 }
```

确定三角形内点的个数：

```
struct node
{
    double x,y;
}Nve[N],Mve[M];
int Right[N][N],s[N][N];
double save[N][N],angle[M];
int n,m;
//寻找第一个大于等于val的坐标
int bsearch(double val)
{
    int l = 0,r = m - 1;
    int mid = 0,ans = -1;
    while (l <= r)
    {
        mid = (l + r)/2;
        if (angle[mid] > val) r = mid - 1;
        else
        {
            l = mid + 1;
            ans = mid;
        }
    }
    return ans;
}
void init()
{
    int i,j;
```

```
    int pos1,pos2;
    //求出任意两点的角度(-pi,pi]
    for (i = 0; i < n; ++i)
    {
        for (j = 0; j < n; ++j)
        {
            if (i == j) continue;
            save[i][j] = atan2((Nve[j].y - Nve[i].y), (Nve[j].x - Nve[i].x));
        }
    }

    for (i = 0; i < n; ++i)
    {
        for (j = 0; j < m; ++j)//枚举i点以i点为中心按极角对m个点排序
        {
            angle[j] = atan2((Mve[j].y - Nve[i].y), (Mve[j].x - Nve[i].x));
        }
        sort(angle,angle + m);
        for (j = 0; j < n; ++j)
        {
            if (i == j) continue;
            double ang = save[i][j];
            pos1 = bsearch(ang);//寻找i->j右边的第一个点也表示1-pos1的个数
            s[i][j] = pos1;
            if (ang >= 0)
            {
                ang -= pi;
                pos2 = bsearch(ang);
                Right[i][j] = pos1 - pos2;
            }
            else
            {
                ang += pi;
                pos2 = bsearch(ang);
                Right[i][j] = m - (pos2 - pos1);
            }
        }
    }
}
int getS(int i,int k,int j)//求i,k,j这个角里面的点的个数
{
    int ang1 = save[k][i];
    int ang2 = save[k][j];
    if (ang1 > ang2)
    {
        if (ang1 - ang2 < pi) return (s[k][i] - s[k][j]);
        else  return (m - (s[k][i] - s[k][j]));
    }
    else
    {
        if (ang2 - ang1 < pi) return s[k][j] - s[k][i];
        else return (m - (s[k][j] - s[k][i]));
    }
```

```
}
int main()
{
    //freopen("din.txt","r",stdin);
    int i,j,h;
    int cas = 1;
   while (~scanf("%d%d",&n,&m))
   {
       for (i = 0; i < n; ++i) scanf("%lf%lf",&Nve[i].x,&Nve[i].y);

       for (i = 0; i < m; ++i) scanf("%lf%lf",&Mve[i].x,&Mve[i].y);

       init();

       int ans = 0;
       for (i = 0; i < n; ++i)
       {
           for (j = i + 1; j < n; ++j)
           {
               for (h = j + 1; h < n; ++h)
               {
                   int tmp = getS(i,j,h) + getS(j,h,i) + getS(h,i,j) + Right[i][j]
+ Right[j][h] + Right[h][i] - 2*m;
                   //printf(">>%d\n",tmp);
                   if (tmp&1) ans++;
               }
           }
       }
       printf("Case %d: %d\n",cas++,ans);
   }
    return 0;
}
```

计算几何：

快速幂取模：

```
ll modmul(ll a,ll b, ll mod)
{
    ll res = 0;
    ll tmp = a;
    while (b)
    {
        if (b&1) res = (res + tmp)%mod;
        tmp = (tmp+tmp)%mod;
        b>>=1;
    }
    return res;
}
ll modexp(ll a,ll b,ll mod)
```

```
{
    ll res = 1;
    ll tmp = a;
    while (b)
    {
        if (b&1) res = modmul(res,tmp,mod);
        tmp = modmul(tmp,tmp,mod);
        b>>=1;
    }
    return res;
}
```

矩阵快速幂：

```
/*数据结构*/
struct Mat{
    ll mat[3][3];
    void init(){
        for (int i = 0; i < 2; ++i){
            for (int j = 0; j < 2; ++j){
                mat[i][j] = ;
            }
        }
        mat[1][1] = 0;
    }
};
Mat operator * (Mat a,Mat b){
    Mat c;
    int i,j,k;
    CL(c.mat,0);
    for (i = 0; i < 2; ++i){
        for (j = 0; j < 2; ++j){
            for (k = 0; k < 2; ++k){
                if (!a.mat[i][k] || !b.mat[k][j]) continue;
                c.mat[i][j] += a.mat[i][k]*b.mat[k][j];
                if (c.mat[i][j] > MOD) c.mat[i][j] %= MOD;
            }
        }
    }
    return c;
}
Mat operator ^ (Mat a,ll k){
    Mat c;
    int i,j;
    //单位矩阵
    for (i = 0; i < 2; ++i){
        for (j = 0; j < 2; ++j){
            c.mat[i][j] = (i == j);
        }
    }
    //求k次幂
    while (k){
```

```
        if (k&1) c = c*a;
        k >>= 1;
        a = a*a;
    }
    return c;
}
```

高斯消元：

```
int a[N][N],X[N];//分别记录增广矩阵和解集
int free_x[N];//记录自由变量
int equ,var;//分别表示方程组的个数和变量的个数

int GCD(int x,int y){
    if (y == 0) return x;
    return GCD(y,x%y);
}
int LCM(int x,int y){
    return x/GCD(x,y)*y;
}
void Debug(void)
{
    int i, j;
    for (i = 0; i < equ; i++)
    {
        for (j = 0; j < var + 1; j++)
        {
            cout << a[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}
// 高斯消元法解方程组(Gauss-Jordan elimination).(-2表示有浮点数解，但无整数解，-1表示无解，0表示唯一解，大于0表示无穷解，并返回自由变元的个数)
int Guass(){
    int i,j,k,col;
    CL(X,0); CL(free_x,1);//把解集清空，所有变量都标为自由变量

    for (k = 0,col = 0; k < equ && col < var; ++k, ++col){//枚举行列
        int max_r = k;//找到该col列元素绝对值最大的那行与第k行交换.(为了在除法时减小误差)
        for (i = k + 1; i < equ; ++i){
            if (iabs(a[i][col]) > iabs(a[max_r][col])) max_r = i;
        }
        if (max_r != k){//交换
            for (i = k; i < var + 1; ++i) swap(a[k][i],a[max_r][i]);
        }
        if (a[k][col] == 0){//如果对应该列都为0，枚举该行的下一列
            k--; continue;
        }
```

```
        for (i = k + 1; i < equ; ++i){//将k后边的col进行初等变换成行阶梯矩阵
            if (a[i][col] != 0){
                int lcm = LCM(a[k][col],a[i][col]);
                int ta = lcm/iabs(a[i][col]); int tb = lcm/iabs(a[k][col]);
                if (a[i][col]*a[k][col] < 0) tb = -tb;
                for (j = col; j < var + 1; ++j){
                    a[i][j] = ta*a[i][j] - tb*a[k][j];
                }
            }
        }
    }
    Debug();
    // 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0). 即R(A) != R(A')无解
    for (i = k; i < equ; ++i){
        if (a[i][col] != 0) return -1;
    }
   // 2. 无穷解的情况: 在var * (var + 1)的增广阵中出现(0, 0, ..., 0)这样的行, 即说明没有形成严格的上三角阵.
   // 且出现的行数即为自由变元的个数.    即R(A) = R(A') < n
    if (k < var){
         // 首先, 自由变元有var - k个, 即不确定的变元至少有var - k个.
        int num = 0,freeidx;
        for (i = k - 1; i >= 0; --i){
            num = 0;// 用于判断该行中的不确定的变元的个数, 如果超过1个, 则无法求解, 它们仍然为不确定的变元.
            int tmp = a[i][var];
            // 第i行一定不会是(0, 0, ..., 0)的情况, 因为这样的行是在第k行到第equ行.
            // 同样, 第i行一定不会是(0, 0, ..., a), a != 0的情况, 这样的无解的.
            for (j = 0; j < var; ++j){
                if (a[i][j] != 0 && free_x[j]){
                    num++;
                    freeidx = j;
                }
            }
            if (num > 1) continue; // 无法求解出确定的变元.
          // 说明就只有一个不确定的变元free_index, 那么可以求解出该变元, 且该变元是确定的.
            tmp = a[i][var];
            for (j = 0; j < var; ++j){
                if (a[i][j] && j != freeidx) tmp -= a[i][j]*X[j];
            }
            X[freeidx] = tmp/a[i][freeidx];
            free_x[freeidx] = 0;
        }
        return var - k;
    }
    // 3. 唯一解的情况: 在var * (var + 1)的增广阵中形成严格的上三角阵.
    // 计算出Xn-1, Xn-2 ... X0.
    for (i = k - 1; i >= 0; --i){
        int tmp = a[i][var];
        for (j = i + 1; j < var; ++j){
            tmp -= a[i][j]*X[j];
        }
        X[i] = tmp/a[i][i];
```

```
    }
    return 0;
}
```

多边形模板:

点结构:

```
struct point
{
    double x,y;
    point(double a = 0,double b = 0): x(a),y(b){}
};
```

浮点数处理:

```
int dblcmp(double x)
{
    if(fabs(x) < eps)
        return 0;
    return x > 0 ? 1:-1;
}
```

或者

```
或者
int dblcmp(double x)
{
    if (x > eps) return 1;
    else if (x < -eps) return -1;
    else return 0;
}
判断线段是否相交并求交点（规范相交）
double det(double x1, double y1, double x2, double y2)
{    //求叉积
    return x1*y2 - x2*y1;
}

double cross(Point a, Point b, Point c)
{    //向量ab，和向量ac
    return det(b.x - a.x, b.y - a.y, c.x - a.x, c.y - a.y);
}

double dotdet(double x1, double y1, double x2, double y2)
{    //点积
    return x1*x2 + y1*y2;
}
```

```
23
24  double dot(Point a, Point b, Point c)
25  {
26      return dotdet(b.x - a.x, b.y - a.y, c.x - a.x, c.y - a.y);
27  }
28
29  int betweenCmp(Point a, Point b, Point c)
30  {    //判断a是不是在bc范围内
31      return dbcmp(dot(a, b, c));
32  }
33
34  bool segcross(Point a, Point b, Point c, Point d)
35  {
36      double s1, s2;
37      int d1, d2, d3, d4;
38      d1 = dbcmp(s1 = cross(a, b, c));
39      d2 = dbcmp(s2 = cross(a, b, d));
40      d3 = dbcmp(cross(c, d, a));
41      d4 = dbcmp(cross(c, d, b));
42      if((d1^d2) == -2 && (d3^d4) == -2)
43      {    //规范相交
44          return true;
45      }
46      //非规范相交
47      if((d1 == 0 && betweenCmp(c, a, b) <= 0) ||
48         (d2 == 0 && betweenCmp(d, a, b) <= 0) ||
49         (d3 == 0 && betweenCmp(a, c, d) <= 0) ||
50         (d4 == 0 && betweenCmp(b, c, d) <= 0))
51          return true;
52      return false;
53  }
```

叉积求多边形面积：

```
1  double area(point p[],int n)
2   { //这里是相对于原点(0, 0), 也可以在多边形上找一个点作为向量的起点
3      double s = 0;
4      int i;
5      p[n].x = p[0].x;
6      p[n].y = p[0].y;
7      for(i = 0; i < n; ++i)
8          s += det(p[i].x, p[i].y, p[i+1].x, p[i+1].y);
9      return fabs(s / 2.0);
10 }
```

三角形公式：

1.海伦公式：

```
p = (a+b+c)/2;
S = sqrt(p*(p-a)*(p-b)*(p-c));
```

2:叉积求解：一直三点：

```
A(x1,y1),B(x2,y2),C(x3,y3);
S = fabs(-x2 * y1 + x3*y1+x1*y2-x3*y2-x1*y3+x2*y3) / 2.0;
```

判断多边形是否为凸多边形

输入p[1],p[2] ...p[n]。 令p[0] = p[n] p[n + 1] = p[1];

```
bool isconvexpg()
{
    int dir = 0;
    for (int i = 0; i <= n - 1; ++i)
    {
        int temp = dblcmp(cross(p[i],p[i + 1],p[i + 2]));
        if (!dir) dir = temp;
        if (dir*temp < 0) return false;
    }
    return true;
}
```

判断一点是否在多边形内，环顾法：

```
double getdis(point a,point b)
{
    double x = a.x - b.x;
    double y = a.y - b.y;
    return sqrt(x*x + y*y);
}
//利用点积求角度
double getangle(point a,point b,point c)
{
    double dj = dotdet(b.x - a.x,b.y - a.y,c.x - a.x,c.y - a.y);
    double dis = getdis(a,b)*getdis(a,c);
    double tmp = dj/dis;
    return acos(tmp);
}
//判断是否在多边形内
bool IsIn()
{
    int i;
    double angle = 0.0;
    for (i = 1; i <= n; ++i)
    {
        if (dblcmp(cross(cir,p[i],p[i + 1])) >= 0)
```

```
        angle += getangle(cir,p[i],p[i + 1]);
        else
        angle -= getangle(cir,p[i],p[i + 1]);
    }
    //printf("angle == %lf\n",angle);
    if (dblcmp(angle) == 0) return false;//在外边
    else if (dblcmp(angle - pi) == 0 || dblcmp(angle + pi) == 0)//在边上
    {
        if (dblcmp(r) == 0) return true;
    }
    else if (dblcmp(angle - 2.0*pi) == 0 || dblcmp(angle + 2.0*pi) == 0)//在里面
    {
        return true;
    }
    else //在多边行顶点
    {
        if (dblcmp(r) == 0) return true;
    }
    return false;
}
```

求凸包的graham_scan算法模板：

```
//这里起点与终点肯定在凸包上，不知道怎么证明
int graham(point *p,int len)
{
    top = 0;  int i;
    //先排序，lrj黑书上的排序方法
    sort(p,p + len,cmp);
    stack[top++] = p[0];
    stack[top++] = p[1];
    //求右链
    for (i = 2; i < len; ++i)
    {
        while (top > 1 && dblcmp(cross(stack[top - 2],stack[top - 1],p[i])) <= 0)
top--;
        stack[top++] = p[i];
    }
    //求左链
    int tmp = top;
    for (i =len - 2; i >= 0; --i)
    {
        while (top > tmp && dblcmp(cross(stack[top - 2],stack[top - 1],p[i]))<= 0)
top--;
        stack[top++] = p[i];
    }
    return top - 1;//起点两次进栈 - 1
}
```

利用凸包求最远点距离——旋转卡壳法：

```
int rotaing(point *p,int len)
{
    p[len] = p[0];
    int ans = 0,q = 1;
    for (int i = 0; i < len; ++i)
    {
        while (cross(p[i],p[i + 1],p[q + 1]) > cross(p[i],p[i + 1],p[q]))
        q = (q + 1)%len;
        ans = max(ans,max(dis2(p[i],p[q]),dis2(p[i + 1],p[q + 1])));
        //这里之所以计算i+1与q+1的距离是考虑到在凸多边形中存在平行边的问题
    }
    return ans;
}
```

数据结构：

哈希方法：

开散列：

挂链：

```
void Gua(int tmp){
    int tp = tmp%N;
    if (tp < 0) tp = -tp;
    node *t = &H[pos++];
    t->num = tmp;
    t->next = tagp[tp];
    tagp[tp] = t;
}
int Cha(int tmp){
    int tp = tmp%N;
    if (tp < 0) tp = -tp;
    node *q;
    int count = 0;
    for (q = tagp[tp]; q != NULL; q = q->next){
        if (q->num == tmp) count++;
    }
    return count;
}
```

Vector 实现：

```
```

```
vector<int>hash[N];
void Gua(int tmp){
    int tp = tmp%N;
    if (tp < 0) tp = -tp;
    hash[tp].push_back(tmp);
}
int Cha(int tmp){
    int tp = tmp%N;
    if (tp < 0) tp = -tp;
    int count = 0,k;
    int sz = hash[tp].size();
    for (k = 0; k < sz; ++k){
        if (hash[tp][k] == tmp) count++;
    }
    return count;
}
```

几种经典方法：

Unix的ELF哈希函数：

```
unsigned int ELFHash(char* str)
{
    unsigned int hash = 0;
    unsigned int x = 0;

    while (*str){
        hash = (hash << 4) + (*str++);
        if ((x = hash & 0xF0000000L) != 0){
            hash ^= (x >> 24);
            hash &= ~x;
        }
    }
    return (hash & 0x7FFFFFFF);
}
```

BKDRHash 哈希：

```
unsigned int BKDRHash(char * str)
{
    unsigned int seed = 31;//31 131 1313 13131
    unsigned int key = 0;
    while (*str)
        key = key*seed+ *str++;
    return key&(0x7fffffff);
}
```

KMP模板：

```
void GetNext(char *s)
{
```

```
    int len = strlen(s);
    int i,j;
    i = 0; j = -1;//j从0开始, i从1开始
    next[0] = -1;
    for (i = 1; i < len; ++i)
    {
        while (j > -1 && s[j + 1] != s[i])//不相同的话j就跳跃知道相同或者成为-1
        j = next[j];
        if (s[j + 1] == s[i]) ++j;//相同更新j
        next[i] = j;//每一次都要求出next[i]
    }
}

void kmp(char *s1,char *s2)//s1副串 s2主串
{
    int len1 = strlen(s1);
    int len2 = strlen(s2);
    int i,j = -1;
    int ans = 0;
    for (i = 0; i < len2; ++i)
    {
        while (j > -1 && s1[j + 1] != s2[i]) j = next[j];
        if (s1[j + 1] == s2[i]) ++j;
        if (j == len1 - 1)//找到子串, 后继续查找
        {
            ans++;
            j = next[j];
        }
    }
    printf("%d\n",ans);
}
```

并查集模板:

```
for (i = 0; i <= n; ++i)
 f[i] = i;

int find(int x)
{
    if (x != f[x])
    {
        f[x] = find(f[x]);
    }
    return f[x];
}

经常使用的:
void Union(int x,int y)
{
    x = find(x);
    y = find(y);
    if (x != y)
```

```
    {
        f[y] = x;
        num[x] += num[y];//根记录子系的个数
    }
}
```

还有一种写法:

```
void Union(int root1, int root2)
{
     int x = FindSet(root1), y = FindSet(root2);
     if( x == y ) return ;
     if( rank[x] > rank[y] ) parent[y] = x;
     else{
         parent[x] = y;
         if( rank[x] == rank[y] ) ++rank[y];
     }
}
```

种类并查集:

pku 1703 Find them, Catch them

题意:

总共存在两个帮派，有两种操作D [a][b] 说明a,b属于不同的帮派，A [a][b] 需要我们来判断他们的关系，属于同一帮派输出，属于不同的帮派，或者不确定。

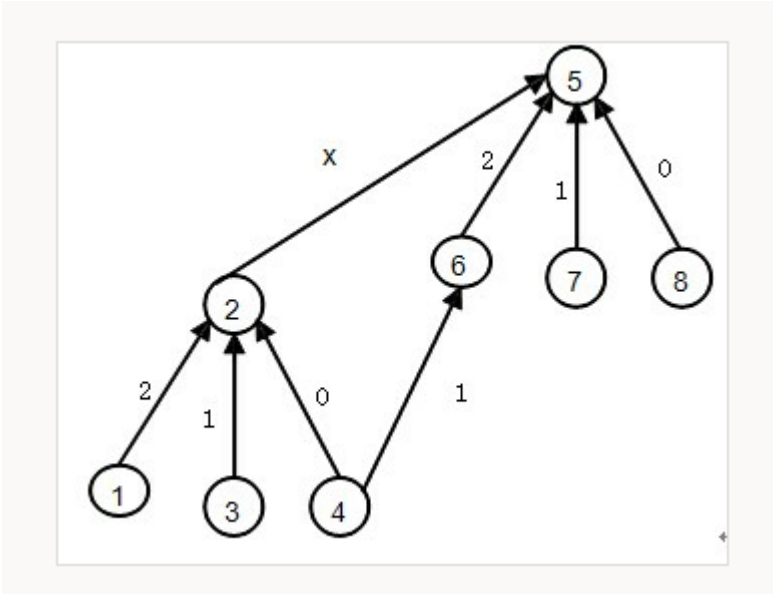
思路:

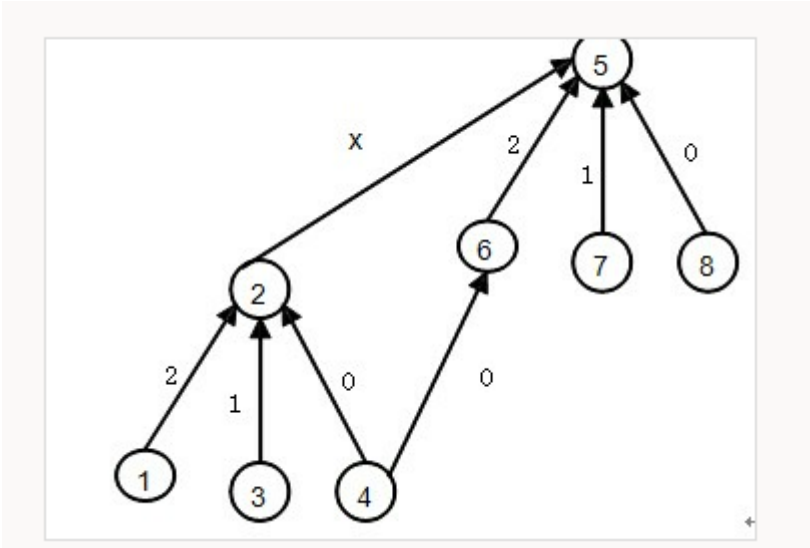
利用并查集构成的树的长度关系，我们把不同的利用并查集合并，然后维持一个他们到根节点的距离dep我们只要判断他们的距离的差值是否为2的倍数就可判断时代否在同以帮派了。

```
void init(){
    int i;
    for (i = 0; i <= n; ++i){
        f[i] = i;
        dep[i] = 0;
    }
}
//find函数里面维护dep
int find(int x){
    if (x != f[x]){
        int tmp = f[x];
        f[x] = find(f[x]);
        dep[x] = (dep[tmp] + dep[x])%2;//更新未压缩的点，保持dep的正确性
    }
    return f[x];
}
void Union(int x,int y){
    int tx = find(x);
    int ty = find(y);
```

```
    if (tx != ty){
        f[ty] = tx;
        dep[ty] = (dep[x] + dep[y] + 1)%2;//维护根节点的正确性
    }
}
int main(){
    //freopen("din.txt","r",stdin);
    int i;
    int T,m;
    char op[3];
    int x,y;
    scanf("%d",&T);
    while (T--){
        scanf("%d%d",&n,&m);
        init();
        for (i = 0; i < m; ++i){
            scanf("%s%d%d",op,&x,&y);
            if (op[0] == 'D'){
                Union(x,y);
            }
            else{
                int tx = find(x);
                int ty = find(y);
                if (tx != ty) printf("Not sure yet.\n");
                else{
                    if (iabs(dep[x] - dep[y])%2 == 0) printf("In the same gang.\n");
                    else printf("In different gangs.\n");
                }
            }
        }
    }
    return 0;
}
```

（pku 食物链 1182）






```
int dep[N],f[N];
int n,k;
int ct;

void init(){
    int i;
    for (i = 0; i <= n; ++i){
        f[i] = i;
        dep[i] = 0;
    }
}
int find(int x){
    if (x != f[x]){
        int tmp = f[x];
        f[x] = find(f[x]);
        dep[x] = (dep[x] + dep[tmp])%3;
    }
    return f[x];
}
void Union(int x,int y,int mk){
```

```
    int tx = find(x);
    int ty = find(y);
    if (tx != ty){
        f[tx] = ty;
        dep[tx] = (dep[y] + mk - dep[x])%3;//维护dep数组
    }
    else{
        if ((dep[x] - dep[y] + 3)%3 != mk) ct++;
    }
}
int main(){
  //  freopen("din.txt","r",stdin);
    ct = 0;
    int op,x,y;
    scanf("%d%d",&n,&k);
    init();
    while (k--){
        scanf("%d%d%d",&op,&x,&y);
        if (x > n || y > n || (op == 2 && x == y)){
            ct++;
            continue;
        }
        if (op == 1){
            Union(x,y,0);//同类合并时，间距为0
        }
        else{
            Union(x,y,1);//不是关系时，间距为1
        }
    }
    printf("%d\n",ct);
    return 0;
}
```

字典树模板:

```
struct node
{
    int flag;
    node *next[27];
}*head;

/*动态分配内存*/
node * newnode()
{
    int i;
    node * p = new node;    // c语言用(node * )malloc(sizeof(node)，这里是动态分配内存，时间上可能消耗的多一些
    p->flag = 0;
    for(i = 0; i < 26; i++)
        p->next[i] = NULL;
```

```
    return p;
}

/*静态分配内存*/
node T[1000000];
int t = 0;
node * newnode()
{
    int i;
    node * p = &T[t++];
    p->flag = 0;
    for(i = 0; i < 26; i++)
        p->next[i] = NULL;
    return p;
}
```

注意：

1：在此之前head一定要先分配，否则无法执行，这里自己老是出错。

2：还有如果处理多组数据的话一定要注意清空T[]数组，否则影响后边的处理。

```
void insert(char *s)
{
    int i,k;
    int len = strlen(s);
    node *p = head;
    for (i = 0; i < len; ++i)
    {
        k = s[i] - 'a';
        if (p->next[k] == NULL)
        p->next[k] = newnode();
        p = p->next[k];
    }
    p->flag = 1;
}

bool search(char *s)
{
    int i,k;
    int len = strlen(s);
    node *p = head;
    for (i = 0; i < len; ++i)
    {
        k = s[i] - 'a';
        if (p->next[k]) p = p->next[k];
        else
        return false;
    }
    if (p->flag) return true;
    else return false;
```

```
}

void del(node * p) {
    int i;
    if(p)                   //p不为空
    {
        for(i =0; i <26; i++)
            if(p->next[i])
                del(p->next[i]);        //递归删除每一个p->next[]
    }
    free(p);
    p = NULL;
}
```

拓扑排序：

```
int topor()
{
    int i,j,k;
    bool flag;
    memset(ind,0,sizeof(ind));
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= n; ++j)
        {
            if (map[i][j])
            ind[j]++;
        }
    }
    flag = false;//判断是否存在多个入度为0的点
    memset(as,0,sizeof(as));
    for (k = 0; k < n; ++k)
    {
        i = 0;
        for (j = 1; j <= n; ++j)
        {
            if (ind[j] == 0)
            {
                if (i == 0)//判断入度为0的点就一个
                 i = j;
                else//出现多个
                flag = true;
            }
        }
        if (i == 0) return 1;//没有入度为0的点
        as[k] = i + 'A' - 1;
        ind[i] = -1;
        for (j = 1; j <= n; ++j)
        {
            if (map[i][j])
```

```
            ind[j]--;
        }
    }
    if (!flag) return 0;
    else  return 2;
}
```

RMQ算法：

```
void init_rmqMIN()
{

    for(int i = 1;i <= n; ++i){
        dp1[i][0] = a[i];
    }
    for(int j = 1; pow2[j] <= n; ++j){
        for(int i = 1; (i + pow2[j] - 1) <= n; ++i){
                dp1[i][j] = min(dp1[i][j-1],dp1[i + (1 << (j-1))][j-1]);
        }

    }
}
int rmqMIN(int l,int r){
    int d = log((double)(r - l + 1)) / log(2.0);
    return min(dp1[l][d],dp1[r - pow2[d] + 1][d]);
}
void init_rmqMAX()
{
    for(int i = 1;i <= n; ++i){
        dp2[i][0] = a[i];
    }
    for(int j = 1; pow2[j] <= n; ++j){
        for(int i = 1; (i + pow2[j] - 1) <= n; ++i){
                dp2[i][j] = max(dp2[i][j-1],dp2[i + (1 << (j-1))][j-1]);
        }

    }
}
int rmqMAX(int l,int r){
    int d = log((double)(r - l + 1)) / log(2.0);
    return max(dp2[l][d],dp2[r - pow2[d] + 1][d]);
}
```

树状数组：

首先要分清a[], c[], sum[] 他们各自所代表的意思；

a[]就是输入的数组；

c[]就是建立的树状数组；

c[i] = a[i - 2^k +1] + ... + a[i];

a有多少个**c**就有多少个，而且**c[i]**肯定包含相应的**a[i]**;

lowbit(i) = 2^k 表示i的二进制数表示形式留下左右边的**1**其余为取**0**得到的数

sum[k] = c[N1] + c[N2] + c[N3].....+ c[Nm];

Ni-1 = Ni - lowbit(i);

求和的话，就是有**c[Nm] c[Nm-1] c[Nm-2] .... c[N1]**的过程 **Ni - lowbit(i)**的过程是将**Ni**的二进制的最右边的**1**去掉的过程，所以求和的时间复杂度为O(log(k));

而更新也是如果**a[i]**更新了 那么**c[N1] c[N2] ..... C[Nm]**也要更新**N1= i**; 就是从**c[N1] c[N2] ... c[Nm]**的过程 **Ni = Ni-1 + lowbit(i)**;就是在**Ni - 1**的二进制最右边**1**后边填**1**的过程，就是在时间复杂度也是O(log(n))级别。

适用于：单点更新，区间求和;

```
int lowbit(int x)
{
   //return x^(x&(x - 1));
    return x&(-x);
}
void modify(int pos,int sc)//位置pos更新sc
{
    while (pos <= n)//由C[N1]到C[Nm]的过程
    {
        c[pos] += sc;
        pos += lowbit(pos);
    }
}
int getsum(int pos)
{
    int sum = 0;
    while (pos > 0)//由C[Nm]到C[N1]的过程
    {
        sum += c[pos];
        pos -= lowbit(pos);
    }
    return sum;
}
二位树状数组模板题目:

给出n*n的矩阵，有两种操作,

1 x,y,z  将(x,y)方格的数更新z       2 x1,y1,x2,y2求方格(x1,y1)到(x2,y2)中的总数。

注意求和公式后面要加上多减去的部分。
int lowbit(int x)
{
    return x&(-x);
```

```
}

void modify(int x,int y,int sc)
{
    int i,j;
    for (i = x; i <= n; i += lowbit(i))
    {
        for (j = y; j <= n; j += lowbit(j))
        {
            c[i][j] += sc;
        }
    }
}
int getsum(int x,int y)
{
    int i,j;
    int sum = 0;
    for (i = x; i > 0; i -= lowbit(i))
    {
        for (j = y; j > 0; j -= lowbit(j))
        {
            sum += c[i][j];
        }
    }
    return sum;
}
int main()
{
    //freopen("d.txt","r",stdin);
    int op,x1,y1,z,x2,y2;
    memset(c,0,sizeof(c));
    while (scanf("%d",&op))
    {
        if (op == 3) break;
        else if (op == 0) scanf("%d",&n);
        else if (op == 1)
        {
            scanf("%d%d%d",&x1,&y1,&z);
            modify(x1 + 1,y1 + 1,z);
        }
        else
        {
            scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
            printf("%d\n",getsum(x2 + 1,y2 + 1) - getsum(x1,y2 + 1)
                    - getsum(x2 + 1,y1) + getsum(x1,y1));
        }
    }
    return 0;
}
```

线段树:

成端更新：注意有累加有覆盖

```
void pushup(int rt)
{
    val[rt] = val[rt<<1] + val[rt<<1|1];
}
void pushdown(int rt,int m)
{
    if (lz[rt])
    {
        lz[rt<<1] = lz[rt<<1|1] = lz[rt];
        val[rt<<1] = (m - (m>>1))*lz[rt];//左边是[1,m/2]有m-m/2个
        val[rt<<1|1] = (m>>1)*lz[rt];//右边[m/2+1,m]有m/2个
        lz[rt] = 0;
    }
}

void build(int l,int r,int rt)
{
    lz[rt] = 0;
    if (l == r)
    {
        val[rt] = 1;
        return ;
    }
    int m = (l + r)>>1;
    build(l,m,rt<<1);
    build(m + 1,r,rt<<1|1);
    pushup(rt);
}

void update(int L,int R,int sc,int l,int r,int rt)
{
    if (l >= L && r <= R)
    {
        lz[rt] = sc;
        val[rt] = lz[rt]*(r - l + 1);
        return ;
    }
    pushdown(rt,r - l + 1);
    int m = (l + r)>>1;
    if (L <= m) update(L,R,sc,l,m,rt<<1);
    if (R > m) update(L,R,sc,m + 1,r,rt<<1|1);
    pushup(rt);
}
```

区间合并：

```

```

```cpp
void pushup(int rt)
{
    //如果不存在区间合并的话正常的更新
    p[rt].lm = p[rt<<1].lm;
    p[rt].rm = p[rt<<1|1].rm;
    p[rt].sm = max(p[rt<<1].sm,p[rt<<1|1].sm);
    int m = (p[rt].l + p[rt].r)>>1;
    //存在可合并的区间
    if (val[m] < val[m + 1])
    {
        int L = m - p[rt].l + 1;
        int R = p[rt].r - m;
        if (p[rt<<1].lm == L) p[rt].lm += p[rt<<1|1].lm;
        if (p[rt<<1|1].rm == R) p[rt].rm += p[rt<<1].rm;
        p[rt].sm = max(p[rt].sm,p[rt<<1].rm + p[rt<<1|1].lm);
    }
}
void build(int l,int r,int rt)
{
   p[rt].l = l; p[rt].r = r;
   p[rt].lm = p[rt].rm = p[rt].sm = 1;
   if (l == r)
   {
       scanf("%d",&val[l]);
       return ;
   }
   int m = (l + r)>>1;
   build(l,m,rt<<1);
   build(m + 1,r,rt<<1|1);
   pushup(rt);
}
void update(int pos,int sc,int rt)
{
    if (p[rt].l == p[rt].r)
    {
        val[p[rt].l] = sc;
        return ;
    }
    int m = (p[rt].l + p[rt].r)>>1;
    if (pos <= m) update(pos,sc,rt<<1);
    else update(pos,sc,rt<<1|1);
    pushup(rt);

}

int query(int L,int R,int rt)
{
    if (p[rt].l >= L && p[rt].r <= R) return p[rt].sm;
    int m = (p[rt].l + p[rt].r)>>1;
    int res = 0;
    if (L <= m) res = max(res,query(L,R,rt<<1));
    if (R > m) res = max(res,query(L,R,rt<<1|1));
    //分出的两个小区间可以合并，记住这里有L，R左右临界的限制。
```

```
    if (val[m] < val[m + 1])
    res = max(res,min(m - L + 1,p[rt<<1].rm) + min(R - m,p[rt<<1|1].lm));
    return res;
}
lcs
同上，只不过在处理区间合并的判断条件时有所不同，上题因为每次都能更新到叶节点所以不必担心lz标记还没更新下来这一说。而这里lz大多数情况下不能更新到叶节点也就导致在判断区间合并的条件时产生了难点，才开始的时候我记录了每个树上节点总共加了多少add[rt]，样例过了可是老是wa无语了，后来想了想，如果我的子区间被更新后那么我只记录了子区间加了多少，而区间没有被更新，所以add记录的就是错误的信息了。

这里我们记录每个区间的左右点的值，每次更新的时候往上更新就好了，这样就保证了每个区间的左右端点值的正确性，因为我们在区间合并时需要的只是端点。
struct node{
    int lm,rm,sm;
    int l,r;
    int al,ar;//这里记录左右端点信息
    int mid(){
        return (l + r)>>1;
    }
}tre[N<<2];
int val[N],lz[N<<2],add[N<<2];

void pushup(int rt,int Ls,int Rs,int m){
    tre[rt].lm = tre[rt<<1].lm;
    tre[rt].rm = tre[rt<<1|1].rm;
    tre[rt].sm = max(tre[rt<<1].sm,tre[rt<<1|1].sm);
    tre[rt].al = tre[rt<<1].al; tre[rt].ar = tre[rt<<1|1].ar;

    if (tre[rt<<1].ar < tre[rt<<1|1].al){
        if (tre[rt].lm == Ls) tre[rt].lm += tre[rt<<1|1].lm;
        if (tre[rt].rm == Rs) tre[rt].rm += tre[rt<<1].rm;
        tre[rt].sm = max(tre[rt].sm,tre[rt<<1].rm + tre[rt<<1|1].lm);
    }
}
void pushdown(int rt){
    if (lz[rt] != 0){
        lz[rt<<1] += lz[rt];
        lz[rt<<1|1] += lz[rt];
        tre[rt<<1].al += lz[rt];
        tre[rt<<1].ar += lz[rt];
        tre[rt<<1|1].al += lz[rt];
        tre[rt<<1|1].ar += lz[rt];
        lz[rt] = 0;
    }
}
void build(int l,int r,int rt){
    lz[rt] = 0;
    tre[rt].l = l; tre[rt].r = r;
    tre[rt].ar = tre[rt].al = 0;
    if (l == r){
        tre[rt].lm = tre[rt].rm = tre[rt].sm = 1;
        scanf("%d",&tre[rt].al);
        tre[rt].ar = tre[rt].al;
```

```
        return ;
    }
    int m = tre[rt].mid();
    build(lc);
    build(rc);
    pushup(rt,m - l + 1,r - m,m);
}
void update(int L,int R,int sc,int rt){
    if (tre[rt].l >= L && tre[rt].r <= R){
        lz[rt] += sc;
        tre[rt].al += sc;
        tre[rt].ar += sc;
        return ;
    }
    int m = tre[rt].mid();
    pushdown(rt);
    if (L <= m) update(L,R,sc,rt<<1);
    if (R > m) update(L,R,sc,rt<<1|1);
    pushup(rt,m - tre[rt].l + 1,tre[rt].r - m,m);
}
int query(int L,int R,int rt){
    if (tre[rt].l >= L && tre[rt].r <= R){
        return tre[rt].sm;
    }
    pushdown(rt);
    int res = 0;
    int m = tre[rt].mid();
    if (L <= m) res = max(res,query(L,R,rt<<1));
    if (R > m) res = max(res,query(L,R,rt<<1|1));
    if (tre[rt<<1].ar < tre[rt<<1|1].al){
        res = max(res,min(m - L + 1,tre[rt<<1].rm) + min(R - m,tre[rt<<1|1].lm));
    }
    pushup(rt,m - tre[rt].l + 1,tre[rt].r - m,m);
    return res;
}

int main(){
    //freopen("data.in","r",stdin);
    int cas = 1;
    int T,n,q;
    char op[3];
    int x,y,z;
    scanf("%d",&T);
    while (T--){
        printf("Case #%d:\n",cas++);
        scanf("%d%d",&n,&q);
        build(1,n,1);
        while (q--){
            scanf("%s%d%d",op,&x,&y);
            if (op[0] == 'a'){
                scanf("%d",&z);
                update(x,y,z,1);
            }
```

```
            else{
                printf("%d\n",query(x,y,1));
            }
        }
    }
    return 0;
}
```

扫描线模板:

矩形面积并:

```
struct Seg{
    double l,r,h;
    int mk;
    Seg(){}
    Seg(double L,double R,double H,int MK){
        l = L; r = R; h = H; mk = MK;
    }
    bool operator < (const Seg &tmp) const{
        return h < tmp.h;
    }
}seg[N];

int cnt[N<<2];
double sum[N<<2],X[N];

void pushup(int rt,int l,int r){
    if (cnt[rt]) sum[rt] = X[r + 1] - X[l];//下边比上边多此区间仍满足条件
    else if (l == r) sum[rt] = 0;//单个的一条线段且x长度不满足条件肯定为0
    else sum[rt] = sum[rt<<1] + sum[rt<<1|1];//x不满足条件但其子树可能存在满足条件的
}
void update(int L,int R,int sc,int l,int r,int rt){
    if (L <= l && r <= R){
        cnt[rt] += sc;//更新
        pushup(rt,l,r);
        return ;
    }
    int m = (l + r)>>1;
    if (L <= m) update(L,R,sc,lc);
    if (m < R) update(L,R,sc,rc);
    pushup(rt,l,r);
}
//离散化后二分查找
int bSearch(double val,int n){
    int l = 0;
    int r = n;
    while (l <= r){
        int m = (l + r)>>1;
        if (val == X[m]) return m;
```

```
        else if (val < X[m]) r = m - 1;
        else l = m + 1;
    }
    return l;
}
for (i = 0; i < len - 1; ++i){
            int l = bSearch(seg[i].l,k - 1);
            int r = bSearch(seg[i].r,k - 1) - 1;//这里讲点转化成线段（难理解）
            if (l <= r) update(l,r,seg[i].mk,0,k - 2,1);
            res += sum[1]*(seg[i + 1].h - seg[i].h);//sum[1]记录整个区间满足条件的x的长度
        }
```

矩形周长并：

```
struct Seg{
    int l,r,h;
    int mk;
    Seg(){}
    Seg(int L,int R,int H,int MK){
        l = L; r = R; h = H; mk = MK;
    }
    bool operator < (const Seg &tmp) const{
        return h < tmp.h;
    }
}seg[M];

int len[N<<2],cnt[N<<2],numseg[N<<2];
int lbd[N<<2],rbd[N<<2];

void pushup(int rt,int l,int r){
    if (cnt[rt]){
        len[rt] = r - l + 1;
        lbd[rt] = rbd[rt] = 1;//标记边界
        numseg[rt] = 2;
    }
    else if (l == r) len[rt] = lbd[rt] = rbd[rt] = numseg[rt] = 0;
    else{
        len[rt] = len[rt<<1] + len[rt<<1|1];
        numseg[rt] = numseg[rt<<1] + numseg[rt<<1|1];
        lbd[rt] = lbd[rt<<1];
        rbd[rt] = rbd[rt<<1|1];
        if (lbd[rt<<1|1] && rbd[rt<<1]) numseg[rt] -= 2;//边界重合重合
    }
}
void update(int L,int R,int sc,int l,int r,int rt){
    if (L <= l && r <= R){
        cnt[rt] += sc;
        pushup(rt,l,r);
        return ;
    }
```

```
    int m = (l + r)>>1;
    if (L <= m) update(L,R,sc,lc);
    if (m < R) update(L,R,sc,rc);
    pushup(rt,l,r);
}
for (i = 0; i < m; ++i){
        if (seg[i].l < seg[i].r)
        update(seg[i].l,seg[i].r - 1,seg[i].mk,L,R - 1,1);
        res += numseg[1]*(seg[i + 1].h - seg[i].h);//竖直方向上有效长度
        res += iabs(len[1] - last);//水平方向上的长度
        last = len[1];//last记录上次的横向的有效长度
}
```

线段树区间部分更新

给出三种操作add x 向集合里添加x（这里保证集合中不存在该元素）, del x删除集合里的x（这里保证集合里面不存在x），

$$\sum_{1\le i\le k} a_i \quad \text{where } i \bmod 5 = 3$$

然后给出N个操作，输出每次的sum；

思路：

才开始看到这题就想到了上次那个维护区间（i-a）%k == 0的题目区间部分更新的题目，每个线段树上的节点添加记录该区间内模k余(0到k-1)的值，可是这里是求整个区间，而且一旦删除一个值它们原来满足模k的余数就会发生改变。直接就晕了。。。看了下解题报告原来和上次的题目是类似的，我们还是记录当前区间模5余（0到4）的值，只是稍微进行一下区间合并即可，解题的关键就在于这里的区间合并。当向上更新的时候当前节点的左子树模5余数不变，只要加上就好，而对于右子树由于在该区间的前边又添加len[rt<<1]这些长度的值，所以原来他们模5的余数对于整体的父亲节点来说已经改变了我们只要右子树模5等于i的现在对应于父亲节点为模5等与（len[rt<<1] + i ）%5.然后合并完毕了。。

离散化+离线处理 +区间合并

```
ll Mod[N<<2][5];
int len[N<<2];
int val[N],a[N];
char op[N][5];

void pushup(int rt){
    int i;
    len[rt] = len[rt<<1] + len[rt<<1|1];
    //关键理解合并
    for (i = 0; i < 5; ++i) Mod[rt][i] = Mod[rt<<1][i];
    int Lx = len[rt<<1];
```

```
    for (i = 0; i < 5; ++i) Mod[rt][(Lx + i)%5] += Mod[rt<<1|1][i];
}
void build(int l,int r,int rt){
    for (int i = 0; i < 5; ++i) Mod[rt][i] = 0;
    len[rt] = 0;
    if (l == r) return ;
    int m = (l + r)>>1;
    build(lc);
    build(rc);
}
void update(int pos,int sc,int l,int r,int rt){
    if (l == r){
        len[rt] += sc;
        Mod[rt][1] += sc*a[l];//对于一个叶子节点来说区间是[1,1]所以模5等于1
        return ;
    }
    int m = (l + r)>>1;
    if (pos <= m) update(pos,sc,lc);
    else update(pos,sc,rc);
    pushup(rt);
}
int bSearch(int L,int R,int v){
    int l = L;
    int r = R;
    while (l <= r){
        int m = (l + r)>>1;
        if (a[m] == v) return m;
        else if (v < a[m]) r = m - 1;
        else l = m + 1;
    }
    return -1;
}
int main(){
    //freopen("din.txt","r",stdin);
    int i,n;
    while (~scanf("%d",&n)){
        int ct = 0;
        for (i = 0; i < n; ++i){
            scanf("%s",op[i]);
            if (op[i][0] != 's'){
                scanf("%d",&val[i]);
                a[++ct] = val[i];
            }
        }
        //离散化
        sort(a + 1,a + 1 + ct);
        int nn = 1;
        for (i = 2; i <= ct; ++i){
            if (a[i] != a[i - 1]) a[++nn] = a[i];
        }
        //离线处理
        build(1,nn,1);
        for (i = 0; i < n; ++i){
```

```
            if (op[i][0] == 'a') update(bSearch(1,nn,val[i]),1,1,nn,1);
            else if (op[i][0] == 'd') update(bSearch(1,nn,val[i]),-1,1,nn,1);
            else printf("%I64d\n",Mod[1][3]);
        }
    }
    return 0;
}
```

给你N个数，有两种操作：

2 x询问a[x]的值；

1 a b k c a <= i <= b 满足(i - a)%k == 0的a[i] += c;

(1 <= N <= 50000) (1 <= a <= b <= N, 1 <= k <= 10, -1,000 <= c <= 1,000)

思路：

一看就知道这肯定是线段树的题目，可是区间内满足(i - a)%k == 0的点才进行处理怎么办，如果i i + k,i + 2k的循环处理的话，就都变成的了单点更新了，时间复杂度O(n^2)就不行可。

这里肯定要优化到O(nlogn)，才开始想记录1操作，区间[a,b]按平常加，每次询问点时再枚举所有的1减去多加了，发现只能优化到O(n^2/2)写了一下TLE.后来看了解题报告才恍然大悟。

线段树上的每个点表示一个区间，我们记录该区间所有模i于j的点的值,由于1 <= k <= 10 所以总共有55中可能，把每个区间的所有可能记录，然后区间更新时，只要将该区间所有满足的情况累加一边就好了，单点询问的话只需要将所有情况累加即可。

```
i%k = a%k;
struct node{
    int l,r;
    int mod[55];//记录模k的各种情况
    int lz;
    int mid(){
       return (l + r)>>1;
    }
}a[4*N];
int mk[11][11],b[N];

void init(){
    int i,j;
    int ct = 0;
    for (i = 1; i <= 10; ++i){
        for (j = 0; j < i; ++j)
        mk[i][j] = ct++;//将其转变为1维
    }
}
void pushdown(int rt){
    if (a[rt].lz){
        a[rt<<1].lz += a[rt].lz;
        a[rt<<1|1].lz += a[rt].lz;
        a[rt].lz = 0;
```

```
        for (int i = 0; i < 55; ++i){
            a[rt<<1].mod[i] += a[rt].mod[i];
            a[rt<<1|1].mod[i] += a[rt].mod[i];
            a[rt].mod[i] = 0;
        }
    }
}
void build(int l,int r,int rt){
    a[rt].l = l; a[rt].r = r;
    a[rt].lz = 0;
    CL(a[rt].mod,0);
    if (l == r) return;
    int m = a[rt].mid();
    build(lc);
    build(rc);
}
void update(int L,int R,int k,int MOD,int sc,int rt){
    if (a[rt].l >= L && a[rt].r <= R){
        a[rt].lz += sc;
        a[rt].mod[mk[k][MOD]] += sc;//模k等于MOD的累加
        return;
    }
    pushdown(rt);
    int m = a[rt].mid();
    if (L <= m) update(L,R,k,MOD,sc,rt<<1);
    if (R > m) update(L,R,k,MOD,sc,rt<<1|1);
}
int query(int pos,int rt){
    if (a[rt].l == a[rt].r){
        int tmp = 0;
        for (int i = 1; i <= 10; ++i){
            tmp += a[rt].mod[mk[i][pos%i]];//枚举当前点模1-10的所有可能
        }
        return b[a[rt].l] + tmp;
    }
    pushdown(rt);
    int res = 0;
    int m = a[rt].mid();
    if (pos <= m) res = query(pos,rt<<1);
    else res = query(pos,rt<<1|1);
    return res;
}
int main(){
    //freopen("din.txt","r",stdin);
    int i,n,m;
    int op,x,y,c,k;
    init();
    while (~scanf("%d",&n)){
        for (i = 1; i <= n; ++i) scanf("%d",&b[i]);
        build(1,n,1);
        scanf("%d",&m);
        while (m--){
            scanf("%d",&op);
```

```
            if (op == 2){
                scanf("%d",&x);
                printf("%d\n",query(x,1));
            }
            else{
                scanf("%d%d%d%d",&x,&y,&k,&c);
                update(x,y,k,x%k,c,1);
            }
        }
    }
    return 0;
}
```

划分树模板:

求区间第**K**大的数

```
struct node{
    int l,r;
    int mid(){
        return (l + r)>>1;
    }
}tt[N<<2];

int toLeft[20][N];
int val[20][N],sorted[N];
int n,q;

void build(int l,int r,int rt,int d){
    int i;
    tt[rt].l = l;
    tt[rt].r = r;
    if (l == r) return ;
    int m = tt[rt].mid();
    int lsame = m - l + 1;//先假设做区间与m相等的放满
    //一一排除, 保证左区间放的是<=sorted[m]的值
    for (i = l; i <= r; ++i){
        if (val[d][i] < sorted[m]) lsame--;
    }

    int lpos = l;
    int rpos = m + 1;
    int same = 0;

    for (i = l; i <= r; ++i){
        if (i == l) toLeft[d][i] = 0;//toLeft[i]表示[ tt[rt].l , i ]区域里有多少个数分到左边
        else toLeft[d][i] = toLeft[d][i - 1];

        if (val[d][i] < sorted[m]){
```

```
            toLeft[d][i]++;
            val[d + 1][lpos++] = val[d][i];
        }
        else if (val[d][i] > sorted[m]){
            val[d + 1][rpos++] = val[d][i];
        }
        else{
            if (same < lsame){//有lsame的数是分到左边的
                toLeft[d][i]++;
                val[d + 1][lpos++] = val[d][i];
                same++;
            }
            else{
                val[d + 1][rpos++] = val[d][i];
            }
        }
    }
    build(lc,d + 1);
    build(rc,d + 1);
}

int query(int L,int R,int k,int d,int rt){
   // printf("%d %d\n",L,R);
    if (L == R){
        return val[d][L];
    }
    int s = 0;;//s表示[ L , R ]有多少个分到左边
    int ss = 0;//ss表示 [tt[rt].l , L-1 ]有多少个分到左边
    if (L == tt[rt].l){
        ss = 0;
        s = toLeft[d][R];
    }
    else{
        ss = toLeft[d][L - 1];
        s = toLeft[d][R] - toLeft[d][L - 1];
    }
    if (k <= s){//有多于k个分到左边,显然去左儿子区间找第k个
        int newl = tt[rt].l + ss;
        int newr = newl + s - 1;
        return query(newl,newr,k,d + 1,rt<<1);
    }
    else{
        int m = tt[rt].mid();
        int bb = L - tt[rt].l - ss;//bb表示 [tt[rt].l , L-1 ]有多少个分到右边
        int b = R - L + 1 - s;//b表示 [L , R]有多少个分到右边
        int newl = m + bb + 1;
        int newr = newl + b - 1;
        return query(newl,newr,k - s,d + 1,rt<<1|1);
    }
}
```

题意：

给定一个长度为n的序列，求区间[L,R]中小于h的个数；

思路：

分三种情况：

1：如果该区间最小值都大于h输出0；

2：如果该区间最大值小于等于h输出区间长度：

3：否则，二分枚举该区间的第m大，直到找到第m大为最后一个小于等于h的；

```
struct node{
    int l,r;
    int mid(){
        return (l + r)>>1;
    }
}tt[N<<2];

int toLeft[20][N],val[20][N];
int sorted[N];
int n,m;
int MIN,MAX;

void build(int l,int r,int rt,int d){
    int i;
    tt[rt].l = l;
    tt[rt].r = r;
    if (l == r) return ;
    int m = tt[rt].mid();
    int lsame = m - l + 1;
    for (i = l; i <= r; ++i){
        if (val[d][i] < sorted[m]) lsame--;
    }

    int lpos = l,rpos = m + 1;
    int same = 0;
    for (i = l; i <= r; ++i){
        if (i == l) toLeft[d][i] = 0;
        else toLeft[d][i] = toLeft[d][i - 1];

        if (val[d][i] < sorted[m]){
            toLeft[d][i]++;
            val[d + 1][lpos++] = val[d][i];
        }
        else if (val[d][i] > sorted[m]){
            val[d + 1][rpos++] = val[d][i];
        }
        else{
            if (same < lsame){
                same++;
                val[d + 1][lpos++] = val[d][i];
                toLeft[d][i]++;
```

```
            }
            else{
                val[d + 1][rpos++] = val[d][i];
            }
        }
    }
    build(lc,d + 1);
    build(rc,d + 1);
}
int query(int L,int R,int k,int d,int rt){
    if (L == R) return val[d][L];
    int s,ss;
    if (L == tt[rt].l){
        ss = 0;
        s = toLeft[d][R];
    }
    else{
        ss = toLeft[d][L - 1];
        s = toLeft[d][R] - toLeft[d][L - 1];
    }

    if (s >= k){
        int newl = tt[rt].l + ss;
        int newr = newl + s - 1;
        return query(newl,newr,k,d + 1,rt<<1);
    }
    else{
        int m = tt[rt].mid();
        int bb = L - tt[rt].l - ss;
        int b = R - L + 1 - s;
        int newl = m + bb + 1;
        int newr = newl + b - 1;
        return query(newl,newr,k - s,d + 1,rt<<1|1);
    }
}
int bSearch(int l,int r,int h,int x,int y){
    int ans = 0;
    while (l <= r){
        int m = (l + r)>>1;
        if (query(x,y,m,0,1) > h)
        r = m - 1;
        else{
            l = m + 1;
            ans = m;
        }
    }
    return ans;
}
int Input(){
    char ch = getchar();
    while (ch < '0' || ch > '9') ch = getchar();
    int num = 0;
    while (ch >= '0' && ch <= '9'){
```

```
        num = num*10 + ch - '0';
        ch = getchar();
    }
    return num;
}
int main(){
    //freopen("din.txt","r",stdin);
    int t,i;
    int x,y,h;
    int cas = 1;
    t = Input();
    while (t--){
        printf("Case %d:\n",cas++);
        n = Input();
        m = Input();
        for (i = 1; i <= n; ++i){
            val[0][i] = Input();
            sorted[i] = val[0][i];
        }
        sort(sorted + 1,sorted + 1 + n);
        build(1,n,1,0);
        while (m--){
            x = Input();
            y = Input();
            h = Input();
            x++; y++;
            MIN = 1;
            MAX = y - x + 1;
            if (query(x,y,MIN,0,1) > h) puts("0");
            else if (query(x,y,MAX,0,1) <= h) printf("%d\n",MAX);
            else{
                int pos = bSearch(MIN,MAX,h,x,y);
                printf("%d\n",pos);
            }
        }
    }
    return 0;
}
```

给定一个长度为n的序列，求区间[l,r]内的一个点值为x，使得最小。

思路：

很多解题报告都说找出区间**[l,r]**的中位数即可，可是我百度了一下中位数，他的定义是这样的：

当变量值的项数**N**为奇数时，处于中间位置的变量值即为中位数；当**N**为偶数时，中位数则为处于中间位置的**2**个变量值的平均数，也即当**N**为偶数时，中位数就不一定属于该序列了。所以这里不是求得中位数。

而是当**N**为奇数时得到中位数**k**，偶数时**N/2(k)**位置的那个数，也即这里保证x来自该序列。然后利用划分树求出区间**[l,r]**中小于**k**的个数以及他们的和，然后利用公式计算即可
