ACM 模板

PocketCat

2021-11-12

目录

1 高精

1.1 两个非负整数相加

```

//加法
string add(string a,string b)//只限两个非负整数相加
{
    const int L=1e5;
    string ans;
    int na[L]={0},nb[L]={0};
    int la=a.size(),lb=b.size();
    for(int i=0;i<la;i++) na[la-1-i]=a[i]-'0';
    for(int i=0;i<lb;i++) nb[lb-1-i]=b[i]-'0';
    int lmax=la>lb?la:lb;
    for(int i=0;i<lmax;i++) na[i]+=nb[i],na[i+1]+=na[i]/10,na[i]%=10;
    if(na[lmax]) lmax++;
    for(int i=lmax-1;i>=0;i--) ans+=na[i]+'0';
    return ans;
}
```

1.2 大的非负整数减小的非负整数

```

//减法
string sub(string a,string b)//只限大的非负整数减小的非负整数
{
    const int L=1e5;
    string ans;
    int na[L]={0},nb[L]={0};
    int la=a.size(),lb=b.size();
    for(int i=0;i<la;i++) na[la-1-i]=a[i]-'0';
    for(int i=0;i<lb;i++) nb[lb-1-i]=b[i]-'0';
    int lmax=la>lb?la:lb;
    for(int i=0;i<lmax;i++)
    {
        na[i]-=nb[i];
        if(na[i]<0) na[i]+=10,na[i+1]--;
    }
    while(!na[--lmax]&&lmax>0) ;lmax++;
    for(int i=lmax-1;i>=0;i--) ans+=na[i]+'0';
    return ans;
}
```

1.3 高精乘高精

```

string mul(string a,string b)//高精度乘法a,b,均为非负整数
{
    const int L=1e5;
    string s;
    int na[L],nb[L],nc[L],La=a.size(),Lb=b.size();//na存储被乘数，nb存储乘数，nc存储积
    fill(na,na+L,0);fill(nb,nb+L,0);fill(nc,nc+L,0);//将na,nb,nc都置为0
    for(int i=La-1;i>=0;i--) na[La-i]=a[i]-'0';//将字符串表示的大整形数转成i整形数组表示的大整形数
    for(int i=Lb-1;i>=0;i--) nb[Lb-i]=b[i]-'0';
    for(int i=1;i<=La;i++)
        for(int j=1;j<=Lb;j++)
        nc[i+j-1]+=na[i]*nb[j];//a的第i位乘以b的第j位为积的第i+j-1位（先不考虑进位）
    for(int i=1;i<=La+Lb;i++)
        nc[i+1]+=nc[i]/10,nc[i]%=10;//统一处理进位
    if(nc[La+Lb]) s+=nc[La+Lb]+'0';//判断第i+j位上的数字是不是0
    for(int i=La+Lb-1;i>=1;i--)
```

```
        s+=nc[i]+'0';//将整形数组转成字符串
    return s;
}
```

1.4 高精度 a 乘低精度 b

```

string mul(string a,int b)//高精度a乘单精度b
{
    const int L=100005;
    int na[L];
    string ans;
    int La=a.size();
    fill(na,na+L,0);
    for(int i=La-1;i>=0;i--) na[La-i-1]=a[i]-'0';
    int w=0;
    for(int i=0;i<La;i++) na[i]=na[i]*b+w,w=na[i]/10,na[i]=na[i]%10;
    while(w) na[La++]=w%10,w/=10;
    La--;
    while(La>=0) ans+=na[La--]+'0';
    return ans;
}
```

1.5 高精除高精 + 高精除低精

```

//除法
int sub(int *a,int *b,int La,int Lb)
{
    if(La<Lb) return -1;//如果a小于b，则返回-1
    if(La==Lb)
    {
        for(int i=La-1;i>=0;i--)
            if(a[i]>b[i]) break;
            else if(a[i]<b[i]) return -1;//如果a小于b，则返回-1

    }
    for(int i=0;i<La;i++)//高精度减法
    {
        a[i]-=b[i];
        if(a[i]<0) a[i]+=10,a[i+1]--;
    }
    for(int i=La-1;i>=0;i--)
        if(a[i]) return i+1;//返回差的位数
    return 0;//返回差的位数

}
string div(string n1,string n2,int nn)
//n1,n2是字符串表示的被除数，除数,nn是选择返回商还是余数
{
    const int L=1e5;
    string s,v;//s存商,v存余数
     int a[L],b[L],r[L],La=n1.size(),Lb=n2.size(),i,tp=La;
     //a，b是整形数组表示被除数，除数，tp保存被除数的长度
     fill(a,a+L,0);fill(b,b+L,0);fill(r,r+L,0);//数组元素都置为0
     for(i=La-1;i>=0;i--) a[La-1-i]=n1[i]-'0';
     for(i=Lb-1;i>=0;i--) b[Lb-1-i]=n2[i]-'0';
     if(La<Lb || (La==Lb && n1<n2)) {
           //cout<<0<<endl;
     return n1;}//如果a<b,则商为0，余数为被除数
     int t=La-Lb;//除被数和除数的位数之差
     for(int i=La-1;i>=0;i--)//将除数扩大10^t倍
        if(i>=t) b[i]=b[i-t];
```

```
        else b[i]=0;
    Lb=La;
    for(int j=0;j<=t;j++)
    {
        int temp;
        while((temp=sub(a,b+j,La,Lb-j))>=0)//如果被除数比除数大继续减
        {
            La=temp;
            r[t-j]++;
        }
    }
    for(i=0;i<L-10;i++) r[i+1]+=r[i]/10,r[i]%=10;//统一处理进位
    while(!r[i]) i--;//将整形数组表示的商转化成字符串表示的
    while(i>=0) s+=r[i--]+'0';
    //cout<<s<<endl;
    i=tp;
    while(!a[i]) i--;//将整形数组表示的余数转化成字符串表示的</span>
    while(i>=0) v+=a[i--]+'0';
    if(v.empty()) v="0";
    //cout<<v<<endl;
    if(nn==1) return s;//返回商
    if(nn==2) return v;//返回余数
}
string div(string a,int b)//高精度a除以单精度b
{
    string r,ans;
    int d=0;
    if(a=="0") return a;//特判
    for(int i=0;i<a.size();i++)
    {
           r+=(d*10+a[i]-'0')/b+'0';//求出商
           d=(d*10+(a[i]-'0'))%b;//求出余数
    }
    int p=0;
    for(int i=0;i<r.size();i++)
    if(r[i]!='0') {p=i;break;}
    return r.substr(p);
}
```

1.6 高精度幂

使用了 FFT，时复 (nlog(n)log(n))

```

//高精度幂(nlog(n)log(n))
#define L(x) (1 << (x))
const double PI = acos(-1.0);
const int Maxn = 133015;
double ax[Maxn], ay[Maxn], bx[Maxn], by[Maxn];
char sa[Maxn/2],sb[Maxn/2];
int sum[Maxn];
int x1[Maxn],x2[Maxn];
int revv(int x, int bits)
{
    int ret = 0;
    for (int i = 0; i < bits; i++)
    {
        ret <<= 1;
        ret |= x & 1;
        x >>= 1;
    }
    return ret;
}
void fft(double * a, double * b, int n, bool rev)
{
```

```
23      int bits = 0;
24      while (1 << bits < n) ++bits;
25      for (int i = 0; i < n; i++)
26      {
27          int j = revv(i, bits);
28          if (i < j)
29              swap(a[i], a[j]), swap(b[i], b[j]);
30      }
31      for (int len = 2; len <= n; len <<= 1)
32      {
33          int half = len >> 1;
34          double wmx = cos(2 * PI / len), wmy = sin(2 * PI / len);
35          if (rev) wmy = -wmy;
36          for (int i = 0; i < n; i += len)
37          {
38              double wx = 1, wy = 0;
39              for (int j = 0; j < half; j++)
40              {
41                  double cx = a[i + j], cy = b[i + j];
42                  double dx = a[i + j + half], dy = b[i + j + half];
43                  double ex = dx * wx - dy * wy, ey = dx * wy + dy * wx;
44                  a[i + j] = cx + ex, b[i + j] = cy + ey;
45                  a[i + j + half] = cx - ex, b[i + j + half] = cy - ey;
46                  double wnx = wx * wmx - wy * wmy, wny = wx * wmy + wy * wmx;
47                  wx = wnx, wy = wny;
48              }
49          }
50      }
51      if (rev)
52      {
53          for (int i = 0; i < n; i++)
54              a[i] /= n, b[i] /= n;
55      }
56  }
57  int solve(int a[],int na,int b[],int nb,int ans[])
58  {
59      int len = max(na, nb), ln;
60      for(ln=0; L(ln)<len; ++ln);
61      len=L(++ln);
62      for (int i = 0; i < len ; ++i)
63      {
64          if (i >= na) ax[i] = 0, ay[i] =0;
65          else ax[i] = a[i], ay[i] = 0;
66      }
67      fft(ax, ay, len, 0);
68      for (int i = 0; i < len; ++i)
69      {
70          if (i >= nb) bx[i] = 0, by[i] = 0;
71          else bx[i] = b[i], by[i] = 0;
72      }
73      fft(bx, by, len, 0);
74      for (int i = 0; i < len; ++i)
75      {
76          double cx = ax[i] * bx[i] - ay[i] * by[i];
77          double cy = ax[i] * by[i] + ay[i] * bx[i];
78          ax[i] = cx, ay[i] = cy;
79      }
80      fft(ax, ay, len, 1);
81      for (int i = 0; i < len; ++i)
82          ans[i] = (int)(ax[i] + 0.5);
83      return len;
84  }
85  string mul(string sa,string sb)
86  {
87      int l1,l2,l;
88      int i;
```

```
    string ans;
    memset(sum, 0, sizeof(sum));
    l1 = sa.size();
    l2 = sb.size();
    for(i = 0; i < l1; i++)
        x1[i] = sa[l1 - i - 1]-'0';
    for(i = 0; i < l2; i++)
        x2[i] = sb[l2-i-1]-'0';
    l = solve(x1, l1, x2, l2, sum);
    for(i = 0; i<l || sum[i] >= 10; i++) // 进位
    {
        sum[i + 1] += sum[i] / 10;
        sum[i] %= 10;
    }
    l = i;
    while(sum[l] <= 0 && l>0) l--; // 检索最高位
    for(i = l; i >= 0; i--) ans+=sum[i] + '0'; // 倒序输出
    return ans;
}
string Pow(string a,int n)
{
    if(n==0) return 1;
    if(n==1) return a;
    if(n&1) return mul(Pow(a,n-1),a);
    string ans=Pow(a,n/2);
    return mul(ans,ans);
}
```

1.7 阶层

```

//阶层
string fac(int n)
{
    const int L=100005;
    int a[L];
    string ans;
    if(n==0) return "1";
    fill(a,a+L,0);
    int s=0,m=n;
    while(m) a[++s]=m%10,m/=10;
    for(int i=n-1;i>=2;i--)
    {
        int w=0;
        for(int j=1;j<=s;j++) a[j]=a[j]*i+w,w=a[j]/10,a[j]=a[j]%10;
        while(w) a[++s]=w%10,w/=10;
    }
    while(!a[s]) s--;
    while(s>=1) ans+=a[s--]+'0';
    return ans;
}
```

1.8 求最大公约数

```

//gcd
string add(string a,string b)
{
    const int L=1e5;
    string ans;
    int na[L]={0},nb[L]={0};
    int la=a.size(),lb=b.size();
    for(int i=0;i<la;i++) na[la-1-i]=a[i]-'0';
```

```
    for(int i=0;i<lb;i++) nb[lb-1-i]=b[i]-'0';
    int lmax=la>lb?la:lb;
    for(int i=0;i<lmax;i++) na[i]+=nb[i],na[i+1]+=na[i]/10,na[i]%=10;
    if(na[lmax]) lmax++;
    for(int i=lmax-1;i>=0;i--) ans+=na[i]+'0';
    return ans;
}
string mul(string a,string b)
{
    const int L=1e5;
    string s;
    int na[L],nb[L],nc[L],La=a.size(),Lb=b.size();//na存储被乘数，nb存储乘数，nc存储积
    fill(na,na+L,0);fill(nb,nb+L,0);fill(nc,nc+L,0);//将na,nb,nc都置为0
    for(int i=La-1;i>=0;i--) na[La-i]=a[i]-'0';//将字符串表示的大整形数转成i整形数组表示的大整形数
    for(int i=Lb-1;i>=0;i--) nb[Lb-i]=b[i]-'0';
    for(int i=1;i<=La;i++)
        for(int j=1;j<=Lb;j++)
        nc[i+j-1]+=na[i]*nb[j];//a的第i位乘以b的第j位为积的第i+j-1位（先不考虑进位）
    for(int i=1;i<=La+Lb;i++)
        nc[i+1]+=nc[i]/10,nc[i]%=10;//统一处理进位
    if(nc[La+Lb]) s+=nc[La+Lb]+'0';//判断第i+j位上的数字是不是0
    for(int i=La+Lb-1;i>=1;i--)
        s+=nc[i]+'0';//将整形数组转成字符串
    return s;
}
int sub(int *a,int *b,int La,int Lb)
{
    if(La<Lb) return -1;//如果a小于b，则返回-1
    if(La==Lb)
    {
        for(int i=La-1;i>=0;i--)
            if(a[i]>b[i]) break;
            else if(a[i]<b[i]) return -1;//如果a小于b，则返回-1

    }
    for(int i=0;i<La;i++)//高精度减法
    {
        a[i]-=b[i];
        if(a[i]<0) a[i]+=10,a[i+1]--;
    }
    for(int i=La-1;i>=0;i--)
        if(a[i]) return i+1;//返回差的位数
    return 0;//返回差的位数

}
string div(string n1,string n2,int nn)//n1,n2是字符串表示的被除数，除数,nn是选择返回商还是余数
{
    const int L=1e5;
    string s,v;//s存商,v存余数
     int a[L],b[L],r[L],La=n1.size(),Lb=n2.size(),i,tp=La;//a，b是整形数组表示被除数，除数，tp保存被除数的长
         度
     fill(a,a+L,0);fill(b,b+L,0);fill(r,r+L,0);//数组元素都置为0
     for(i=La-1;i>=0;i--) a[La-1-i]=n1[i]-'0';
     for(i=Lb-1;i>=0;i--) b[Lb-1-i]=n2[i]-'0';
     if(La<Lb || (La==Lb && n1<n2)) {
           //cout<<0<<endl;
     return n1;}//如果a<b,则商为0，余数为被除数
     int t=La-Lb;//除被数和除数的位数之差
     for(int i=La-1;i>=0;i--)//将除数扩大10^t倍
        if(i>=t) b[i]=b[i-t];
        else b[i]=0;
     Lb=La;
     for(int j=0;j<=t;j++)
     {
         int temp;
         while((temp=sub(a,b+j,La,Lb-j))>=0)//如果被除数比除数大继续减
```

```
        {
            La=temp;
            r[t-j]++;
        }
    }
    for(i=0;i<L-10;i++) r[i+1]+=r[i]/10,r[i]%=10;//统一处理进位
    while(!r[i]) i--;//将整形数组表示的商转化成字符串表示的
    while(i>=0) s+=r[i--]+'0';
    //cout<<s<<endl;
    i=tp;
    while(!a[i]) i--;//将整形数组表示的余数转化成字符串表示的</span>
    while(i>=0) v+=a[i--]+'0';
    if(v.empty()) v="0";
    //cout<<v<<endl;
    if(nn==1) return s;
    if(nn==2) return v;
}
bool judge(string s)//判断s是否为全0串
{
    for(int i=0;i<s.size();i++)
        if(s[i]!='0') return false;
    return true;
}
string gcd(string a,string b)//求最大公约数
{
    string t;
    while(!judge(b))//如果余数不为0，继续除
    {
        t=a;//保存被除数的值
        a=b;//用除数替换被除数
        b=div(t,b,2);//用余数替换除数
    }
    return a;
}
```

1.9 取模（高精）

```

//取模
int mod(string a,int b)//高精度a除以单精度b
{
    int d=0;
    for(int i=0;i<a.size();i++) d=(d*10+(a[i]-'0'))%b;//求出余数
    return d;
}
```

1.10 进制转换（将字符串表示的 10 进制大整数转换为 m 进制的大整数）

```

//进制转换
//将字符串表示的10进制大整数转换为m进制的大整数
//并返回m进制大整数的字符串
bool judge(string s)//判断串是否为全零串
{
    for(int i=0;i<s.size();i++)
        if(s[i]!='0') return 1;
    return 0;
}
string solve(string s,int n,int m)//n进制转m进制只限0-9进制，若涉及带字母的进制，稍作修改即可
{
    string r,ans;
    int d=0;
    if(!judge(s)) return "0";//特判
```

```
    while(judge(s))//被除数不为0则继续
    {
        for(int i=0;i<s.size();i++)
        {
            r+=(d*n+s[i]-'0')/m+'0';//求出商
            d=(d*n+(s[i]-'0'))%m;//求出余数
        }
       s=r;//把商赋给下一次的被除数
       r="";//把商清空
        ans+=d+'0';//加上进制转换后数字
        d=0;//清空余数
    }
    reverse(ans.begin(),ans.end());//倒置下
    return ans;
}
```

2 计算几何

2.1 基础模板

```
//点和线的表示
typedef long double LD;
struct PII{
    int x,y;
    bool operator<(const Point &o)const{
        if(x==o.x) return y<o.y;
        return x<o.x;
    }
};
struct Point{
    double x,y;
    bool operator<(const Point &o)const{
        if(x==o.x) return y<o.y;
        return x<o.x;
    }
};
struct Line{
    Point st,ed;
};
//判正负
int sign(double a){
    if(fabs(a)<=eps) return 0;
    return a>0?1:-1;
}
//比较大小
int dcmp(double a,double b){
    if(fabs(a-b)<eps) return 0;
    return a>b?1:-1;
}
```

2.2 点积

```
double Dot(Point a, Point b) {
    return a.x*b.x+a.y*b.y;
}
```

2.3 叉积

```
double cross(Point a,Point b){
    return a.x*b.y-a.y*b.x;
```

```
}
```

2.4 模长

```
double ABS(Point a){
    return sqrt(a.x*a.x+a.y*a.y);
}
double norm(Point a){
    return a.x*a.x+a.y*a.y;
}
```

2.5 两个向量是否同象限

```
bool same_quadrant(Point v,Point p) {
    LD a=v.x, b=v.y, c=p.x, d=p.y;
    int aa=sign(a), bb=sign(b);
    int cc=sign(c), dd=sign(d);
    return aa*cc>=0 && bb*dd>=0;
}
```

2.6 两向量是否共线

```
int dcmp(double x) {
    if (fabs(x)<eps) return 0;
    else if (x<0) return -1;
    else return 1;
}
double cross(Point a,Point b){
    return a.x*b.y-a.y*b.x;
}
bool on_line(Point a, Point b) { //a和b是否共线
    return dcmp(cross(a,b))==0;
}
```

2.7 两向量是否垂直

```
double Dot(Point a, Point b) {
    return a.x*b.x+a.y*b.y;
}
bool is_vertical(Point a, Point b) {
    return dcmp(Dot(a,b))==0;
}
```

2.8 两个向量是否同方向

```
int same_direction(Point v, Point p) { //判断向量v和向量p是否共线且同向
    if (on_line(v, p) && same_quadrant(v,p)) return 1; //同向
    else if(on_line(v, p) && !same_quadrant(v,p)) return -1; //反向
    return 0; //不共线
}
```

2.9 点旋转

```
//点逆时针旋转a度后的坐标
Point rotate1(Point p,double a){
    return {p.x*cos(a)-p.y*sin(a),p.x*sin(a)+p.y*cos(a)};
}
//点顺时针旋转a度后的坐标
Point rotate2(Point p,double a){
    return {p.x*cos(a)+p.y*sin(a),-p.x*sin(a)+p.y*cos(a)};
}
```

2.10 两点之间的距离

```
double getdis(Point a,Point b) {
    return hypot(a.x-b.x,a.y-b.y);
}
```

2.11 直线

2.11.1 直线和线段是否相交

```
//直线a是否经过线段b
bool on_segment(Line a,Line b){
    point q1=a.st,q2=a.ed;
    if(sign(area(q1,q2,b.st))*sign(area(q1,q2,b.ed))>0) return 0;
    return 1;
}
```

2.11.2 直线和直线的交点

```
//求两直线的交点
Point get_line_intersection(Point p, Point v, Point q, Point w){
    Point u = p - q;
    LD t = cross(w, u) / cross(v, w);
    return {p.x + t * v.x, p.y + t * v.y};
}
Point get_line_intersection(Line& a, Line& b){
    return get_line_intersection(a.st, a.ed - a.st, b.st, b.ed - b.st);
}
```

2.11.3 判断 b 和 c 的交点是否在直线右面

```
bool on_right(Line& a, Line& b, Line& c){
    auto o = get_line_intersection(b, c);
    return sign(area(a.st, a.ed, o)) < 0;
}
```

2.11.4 得到直线的角度

```
double get_angle(const Line &a){
    return atan2(a.ed.y-a.st.y,a.ed.x-a.st.x);
}
```

2.11.5 求点关于直线的对称点

```
Point point_line(Line l, Point p) { //点p关于直线l的对称点
    Point p1 = l.st;
    Point p2 = l.ed;
    double _x, _y;
    if(p1.x - p2.x == 0) { //l斜率不存在
        _x = 2 * p1.x - p.x;
        _y = p.y;
        return Point{_x,_y};
    } else if(p1.y - p2.y == 0) { //l斜率为0
        _x = p.x;
        _y = 2 * p1.y - p.y;
        return Point{_x,_y};
    } else {
        double k1 = (p1.y - p2.y) / (p1.x - p2.x);
        double b1 = p1.y - k1 * p1.x;
        double k2 = -1 / k1;
        double b2 = p.y - k2 * p.x;
        _x = (b2 - b1) / (k1 - k2);
        _y = k2 * _x + b2;
        return Point{2 * _x - p.x, 2 * _y - p.y};
    }
}
```

2.11.6 点到直线的距离

```
double PLDis(Point a,Line s) { //点到直线的距离
    double A=s.st.y-s.ed.y;
    double B=s.ed.x-s.st.x;
    double C=(s.st.x-s.ed.x)*s.st.y-(s.st.y-s.ed.y)*s.st.x;
    return fabs(A*a.x+B*a.y+C)/sqrt((A*A+B*B));
}
```

2.11.7 点在直线的投影

```
//点p在直线s上的投影
Point Projection(Point p,Line s) {
    Point alp=p-s.st;
    Point beta=s.ed-s.st;
    double res=Dot(alp,beta)/norm(beta); //norm(): 模长的平方
    return s.st+(res*beta);
}
```

2.12 求多边形的面积

```
vector<Point> polygon;
double get_Area(vector<Point> polygon) {
    double ans=0;
    int n=(int)polygon.size();
    for(int i=0; i<n; i++) {
        ans+=cross(polygon[i],polygon[(i+1)%n]);
    }
    return fabs(ans/2);
}
```

2.13 线段

2.13.1 判断一个点是否在一个线段上

```
bool on_segment(Point p, Point a, Point b){
    return sign(cross(p - a, p - b)) == 0 && sign(dot(p - a, p - b)) <= 0;
}
```

2.14 判断两个线段是否相交

```
bool segment_intersection(Point a1, Point a2, Point b1, Point b2){
    double c1 = cross(a2 - a1, b1 - a1), c2 = cross(a2 - a1, b2 - a1);
    double c3 = cross(b2 - b1, a2 - b1), c4 = cross(b2 - b1, a1 - b1);
    return sign(c1) * sign(c2) <= 0 && sign(c3) * sign(c4) <= 0;
}
```

2.14.1 判断线和线段是否相交

```
bool line_segment_intersection(Line a,Line b){
    if(on_line(a.p2-a.p1,b.p2-b.p1)){
        if(on_line(a.p2-a.p1,b.p1-a.p1)) return 1;
        else return 0;
    }
    Point o=get_line_intersection(a,b);
    if(on_segment(o,b)) return 1;
    else return 0;
}
```

2.15 圆

2.15.1 直线和圆的交点

```
int CCL(Line s,Point o,double r,Point &o1,Point &o2) {
    Point x=Projection(o,s);
    double dis=PLDis(o,s);
    if(dis>r) { //距离>r没有交点
        return 0;
    }
    if(dis==r) { //只有一个交点
        o1=x;
        return 1;
    }
    double beta=sqrt(r*r-dis*dis);//勾股定理
    Point pp=s.ed-s.st;
    pp=pp/pp.ABS();//单位向量
    Point ans1=x-beta*pp;
    Point ans2=x+beta*pp;
    o1=ans1;
    o2=ans2;
    return 2;
}
```

2.16 凸包

求包含所有点的最小周长多边形

```

//极角排序比较函数
bool cmp(PDD a,PDD b){
    a=a-bas; b=b-bas;
    double ag1=atan2(a.y,a.x),ag2=atan2(b.y,b.x);
    if(ag1==ag2) a.x<b.x;
    else return ag1<ag2;
```

```
}
//二维凸包
void get_convex(){
    sort(q+1,q+1+n,cmp);
    stk[++top]=q[1];
    stk[++top]=q[2];
    for(int i=3;i<=n;i++){
        while(top>=2 && area(stk[top-1],stk[top],q[i])<=0) --top;
        stk[++top]=q[i];
    }
    return ;
}
```

2.17 三维凸包

求将三维空间中的 n 个点包含进去所需要的最小多边形面积

```

#include <bits/stdc++.h>
#define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define endl '\n'
using namespace std;
typedef long double LD;
const LD eps=1e-10;
const int N=110;
const double PI=acos(-1);
double rand_eps(){
    return ((double)rand()/RAND_MAX-0.5)*eps;
}
struct Point{
    double x,y,z;
    void shake(){ //微小扰动
        x+=rand_eps();
        y+=rand_eps();
        z+=rand_eps();
    }
    Point operator - (Point a){
        return {x-a.x,y-a.y,z-a.z};
    }
    Point operator + (Point a){
        return {x+a.x,y+a.y,z+a.z};
    }
    Point operator * (Point a){ //向量叉乘
        return {y * a.z - z * a.y, z * a.x - x * a.z, x * a.y - y * a.x};
    }
    double operator & (Point a){ //向量点积
        return x*a.x+y*a.y+z*a.z;
    }
    double len(){ //向量模长
        return sqrt(x*x+y*y+z*z);
    }
}q[N];
struct Plane{
    int v[3];
    Point norm(){
        return (q[v[1]]-q[v[0]])*(q[v[2]]-q[v[0]]); //返回法向量
    }
    double area(){
        return norm().len()/2; //面积
    }
    bool above(Point a){
        return ((a-q[v[0]]) & norm()) >= 0; //返回一个点是否在一个平面上方，也就是平面能不能被照到
    }
}pl[N],bk[N];
bool g[N][N];
```

```
int n,tot;
void get_convex(){
    pl[++tot]={1,2,3}; //放进去前3个点组成的两个平面（正反）
    pl[++tot]={1,3,2};
    for(int i=4;i<=n;i++){
        int cnt=0; //把更新后的平面放进去备份数组
        for(int j=1;j<=tot;j++){
            bool flag=pl[j].above(q[i]); //表示q[i]照到了第j个平面
            if(!flag) bk[++cnt]=pl[j]; //没有照到
            for(int k=0;k<3;k++){
                g[pl[j].v[k]][pl[j].v[(k+1)%3]]=flag; //标记每一条边是否照到
            }
        }
        for(int j=1;j<=tot;j++){
            for(int k=0;k<3;k++){
                int a=pl[j].v[k],b=pl[j].v[(k+1)%3]; //两点的编号
                if(g[a][b] && !g[b][a]) bk[++cnt]={a,b,i}; //正着可以照到，反着照不到，把照到的边扔掉
            }
        }
        tot=cnt; //更新面的数量
        for(int i=1;i<=tot;i++) pl[i]=bk[i]; //更新所有的面
    }
}
int main()
{
    ios;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>q[i].x>>q[i].y>>q[i].z;
        q[i].shake();
    }
    get_convex();
    double ans=0;
    for(int i=1;i<=tot;i++){
        ans+=pl[i].area();
    }
    cout<<fixed<<setprecision(6)<<ans<<endl;

    return 0;
 }
```

2.18 半平面交

求包含所有点的最小周长多边形

```

//极角排序比较函数
bool cmp(PDD a,PDD b){
    a=a-bas; b=b-bas;
    double ag1=atan2(a.y,a.x),ag2=atan2(b.y,b.x);
    if(ag1==ag2) a.x<b.x;
    else return ag1<ag2;
}
//二维凸包
void get_convex(){
    sort(q+1,q+1+n,cmp);
    stk[++top]=q[1];
    stk[++top]=q[2];
    for(int i=3;i<=n;i++){
        while(top>=2 && area(stk[top-1],stk[top],q[i])<=0) --top;
        stk[++top]=q[i];
    }
    return ;
}
```

2.19 最小圆覆盖

```
1
2  #include <bits/stdc++.h>
3  #define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
4  #define endl '\n'
5  #define x first
6  #define y second
7  using namespace std;
8  typedef long double LD;
9  const LD eps=1e-12;
10 const int N=101000;
11 const double PI=acos(-1);
12 //点和线的表示
13 typedef long double LD;
14 typedef pair<double,double> PDD;
15 struct Circle {
16     PDD p;
17
18     double r;
19 };
20 //重载运算符"-"
21 PDD operator-(const PDD &a,const PDD &b) {
22     return {a.x-b.x,a.y-b.y};
23 }
24 PDD operator+(const PDD &a,const PDD &b) {
25     return {a.x+b.x,a.y+b.y};
26 }
27 PDD operator/ (const PDD &a,double t) {
28     return {a.x/t,a.y/t};
29 }
30 //求向量a和b的叉积
31 double cross(PDD a,PDD b) {
32     return a.x*b.y-a.y*b.x;
33 }
34 //求向量ab和向量ac的叉积，也就是abc三角形的面积，顺时针为负，逆时针为正
35 double area(PDD a,PDD b,PDD c) {
36     return cross(b-a,c-a);
37 }
38 //判正负
39 int sign(double a) {
40     if(fabs(a)<=eps) return 0;
41     return a>0?1:-1;
42 }
43 //比较大小
44 int fcmp(double a,double b) {
45     if(fabs(a-b)<eps) return 0;
46     return a>b?1:-1;
47 }
48 PDD get_line_intersection(PDD p, PDD v, PDD q, PDD w) {
49     PDD u = p - q;
50     LD t = cross(w, u) / cross(v, w);
51     return {p.x + t * v.x, p.y + t * v.y};
52 }
53 //点逆时针旋转a度后的坐标
54 PDD rotate(PDD p,double a) {
55     return {p.x*cos(a)-p.y*sin(a),p.x*sin(a)+p.y*cos(a)};
56 }
57 pair<PDD,PDD> getline(PDD a,PDD b) {
58     return {(a+b)/2,rotate(b-a,PI/2)};
59 }
60 double getdis(PDD a,PDD b) {
61     return sqrt(pow(b.y-a.y,2)+pow(b.x-a.x,2));
62 }
63 Circle get_circle(PDD a,PDD b,PDD c) {
64     auto u=getline(a,b),v=getline(a,c);
```

```
    auto p=get_line_intersection(u.x,u.y,v.x,v.y);
    return {p,getdis(p,a)};
}
PDD q[N];
int n;
Circle get_Circle() {
    random_shuffle(q+1,q+1+n);
    Circle c;
    c.p=q[1];
    c.r=0;
    for(int i=2; i<=n; i++) {
        if(fcmp(c.r,getdis(c.p,q[i]))<0) {
            c= {q[i],0};
            for(int j=1; j<i; j++) {
                if(fcmp(c.r,getdis(c.p,q[j]))<0) {
                    c= {(q[i]+q[j])/2,getdis(q[i],q[j])/2};
                    for(int k=1; k<j; k++) {
                        if(fcmp(c.r,getdis(c.p,q[k]))<0) {
                            c=get_circle(q[i],q[j],q[k]);
                        }
                    }
                }
            }
        }
    }
    return c;
}

int main() {
    ios;
    cin>>n;
    for(int i=1; i<=n; i++) cin>>q[i].x>>q[i].y;
    Circle c;
    c=get_Circle();
    cout<<fixed<<setprecision(10)<<c.r<<endl;
    cout<<fixed<<setprecision(10)<<c.p.x<<" "<<c.p.y<<endl;
    return 0;
}
```

3 数据结构

3.1 单调栈

3.1.1 求某组数以其中某一个数字为最小值的最大延伸区间

```

int rmi[N],rmx[N];
void largest(int *arr,int n) {
    stack<int> stmin,stmax;
    for(int i=1;i<=n;i++){
        while(!stmin.empty() && arr[i]<arr[stmin.top()]){
            rmi[stmin.top()]=i-1;
            stmin.pop();
        }
        stmin.push(i);
        while(!stmax.empty() && arr[i]>arr[stmax.top()]) {
            rmx[stmax.top()]=i-1;
            stmax.pop();
        }
        stmax.push(i);
    }
    while(!stmin.empty()) {
        rmi[stmin.top()]=n;
        stmin.pop();
```

```
    }
    while(!stmax.empty()) {
        rmx[stmax.top()]=n;
        stmax.pop();
    }
    return ;
}
```

3.1.2 求一个柱状图的最大矩形面积

维护栈内元素单调递增，如果来一个元素比栈顶元素小，那么栈顶元素右边就没法再增大，所以栈顶位置的最大面积就可以直接算了，s=(i-st.top())(heights[st.top()])，需要注意的是初始数组最后添加一个能把前面都小的值 0

```

int largestRectangleArea(vector<int>& heights) {
    heights.push_back(0);
    stack<int> st;
    int sz=heights.size();
    int ans=0;
    for(int i=0; i<sz; i++) {
        while(!st.empty() && heights[st.top()]>heights[i]) {
            int cur=heights[st.top()];
            st.pop();
            if(st.empty()) ans=max(ans,i*cur); //这里记得特判
            else ans=max(ans,(i-st.top()-1)*cur);
        }
        st.push(i);
    }
    return ans;
}
```

3.2 单调队列

3.2.1 区间 (长度固定) 最值问题

```

int mi[N],mx[N];
void GET(int *arr,int n,int k){
    deque<int> dqmin,dqmax;
    for(int i=1;i<=n;i++){
        while(!dqmax.empty() && arr[i]>=arr[dqmax.back()]) dqmax.pop_back();
        dqmax.push_back(i);
        while(!dqmin.empty() && arr[i]<=arr[dqmin.back()]) dqmin.pop_back();
        dqmin.push_back(i);
        if(i>=k){
            mx[i-k+1]=arr[dqmax.front()];
            mi[i-k+1]=arr[dqmin.front()];
        }
        if(i-dqmax.front()+1>=k) dqmax.pop_front();
        if(i-dqmin.front()+1>=k) dqmin.pop_front();
    }
    return ;
}
```

3.2.2 最大值减最小值小于等于 k 的子区间数量

求一个数组中区间最大值减去最小值 <=k 的所有子区间个数

```

ll GETNUM(vector<ll> nums,int k){
    deque<ll> dqmin,dqmax;
```

```
    ll sz=nums.size(),j=0,ans=0;
    for(int i=0;i<sz;i++){
        while(j<sz){
            //维护从大到小的deque
            while(!dqmax.empty() && nums[j]>=nums[dqmax.back()]) dqmax.pop_back();
            dqmax.push_back(j);
            //维护从小到大的deque
            while(!dqmin.empty() && nums[j]<=nums[dqmin.back()]) dqmin.pop_back();
            dqmin.push_back(j);
            //不满足条件就break出去
            if(nums[dqmax.front()]-nums[dqmin.front()]>k) break;
            j++;
        }
        //L指针右走时要看看需不需要更新dq，如果dq队头是这个元素就需要pop出去
        if(dqmax.front()==i) dqmax.pop_front();
        if(dqmin.front()==i) dqmin.pop_front();
        //这个时候R-L就是以L开头的所有最大减最小<=k的区间个数
        ans+=j-i;
    }
    return ans;
}
```

3.3 并查集

3.3.1 按秩合并

```

void remerge(int a,int b,int c) {
    int x=find(a), y=find(b);
    if(x==y) return ;
    if(dep[x]<dep[y]) fa[x]=y;
    else {
        fa[y]=x;
        if(dep[x]==dep[y]) dep[x]++;
    }
}
```

3.3.2 关押罪犯

```

#include <bits/stdc++.h>
#define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
using namespace std;
const int N=1e6+100;
struct node{
    int u,v,w;
}arr[100100];
int n,m;
int fa[N*2];
int find(int x){
    if(x==fa[x]) return x;
    else return fa[x]=find(fa[x]);
}
void remerge(int x,int y){
    int fx=find(x),fy=find(y);
    if(fx!=fy) fa[fx]=fy;
}
bool cmp(node a,node b){
    return a.w>b.w;
}
int main(){
    ios;
    cin>>n>>m;
```

```
    for(int i=1;i<=m;i++) cin>>arr[i].u>>arr[i].v>>arr[i].w;
    sort(arr+1,arr+1+m,cmp);
    for(int i=1;i<=2*n;i++) fa[i]=i;
    for(int i=1;i<=m;i++){
        int x=arr[i].u,y=arr[i].v;
        if(find(x)==find(y) || find(x+n)==find(y+n)){
            cout<<arr[i].w<<'\n';
            return 0;
        }
        remerge(x+n,y);
        remerge(x,y+n);
    }
    cout<<0<<'\n';
    return 0;
}
```

3.3.3 食物链

dis[i] 是 i 指向其父亲的向量模长，向量的方向也是有意义的，模长决定了 i 属于哪个集合，而方向表示 i 是被父亲吃还是吃父亲

```

#include <bits/stdc++.h>
#define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define endl '\n'
using namespace std;
const int N=5e4+100;
int fa[N],dis[N];
int n,k;
int find(int x){
    if(x==fa[x]) return x;
    else{
        int tmp=fa[x];
        fa[x]=find(fa[x]);
        dis[x]=(dis[x]+dis[tmp])%3;
        return fa[x];
    }
}

int main()
{
    ios;
    cin>>n>>k;
    for(int i=1;i<=n;i++) fa[i]=i;
    int ans=0;
    while(k--){
        int op,x,y;
        cin>>op>>x>>y;
        if(x<1 || x>n || y<1 || y>n) {
            ans++;
            continue;
        }
        if(op==1){
            int fx=find(x), fy=find(y);
            if(fx==fy){
                if((dis[y]-dis[x]+3)%3!=0) ans++;
            }
            else{
                dis[fx]=(dis[y]-dis[x]+3)%3;
                fa[fx]=fy;
            }
        }
        else{
            int fx=find(x), fy=find(y);
            if(fx==fy){
```

```
            if((dis[y]-dis[x]+3)%3!=1) ans++;
        }
        else{
            dis[fx]=(dis[y]-dis[x]-1+3)%3;
            fa[fx]=fy;
        }
    }
}
cout<<ans<<endl;
return 0;
}
```

3.4 AC 自动机

```

struct AC{
    int tr[N][27],num[N],fail[N]; //tr: 字典树数组、num: 记录模式串数量、失配跳转指针
    int tot; // 点编号
    void insert(string s){ // 插入模式串，构建字典树
        int p=0,len=s.size();
        for(int i=0;i<len;i++){
            if(!tr[p][s[i]-'a']) tr[p][s[i]-'a']=++tot;
            p=tr[p][s[i]-'a'];
        }
        num[p]++; // 数量加一
        return ;
    }
    queue<int> q;
    void build(){
        for(int i=0;i<26;i++){
            if(tr[0][i]) q.push(tr[0][i]);
        }
        while(!q.empty()){
            int u=q.front();
            q.pop();
            for(int i=0;i<26;i++){
                if(tr[u][i]){
                    fail[tr[u][i]]=tr[fail[u]][i];
                    q.push(tr[u][i]);
                }
                else tr[u][i]=tr[fail[u]][i]; // 压缩路径
            }
        }
        return ;
    }
    int query(string s){
        int len=s.size(),u=0,ret=0;
        for(int i=0;i<len;i++){
            u=tr[u][s[i]-'a'];
            for(int j=u;j && num[j]!=-1;j=fail[j]){
            // 每一个字符都当成失配字符算一遍匹配模式串数量
                ret+=num[j];
                num[j]=-1; // 一个模式串被计算过了就要标记一下
            }
        }
        return ret;
    }
};
AC tree;
```

3.5 树状数组

3.5.1 单点修改，区间查询

```

int lowbit(int x){
    return x&-x;
}
void add(int i) {
    while(i<=n) {
        c[i]++;
        i+=lowbit(i);
    }
}
ll getsum(int i) {
    long long res=0;
    while(i>0) {
        res+=c[i];
        i-=lowbit(i);
    }
    return res;
}
```

3.5.2 树状数组求逆序数

```
#include <bits/stdc++.h>
using namespace std;
const int N=500001;
typedef long long ll;
int c[N];
struct Node {
    int v,id;
    bool operator < (const Node &b) const {
        return v<b.v; //从小到大排序
    }
} arr[N];
int n;
int lowbit(int x){
    return x&-x;
}
void add(int i) {
    while(i<=n) {
        c[i]++;
        i+=lowbit(i);
    }
}
ll getsum(int i) {
    long long res=0;
    while(i>0) {
        res+=c[i];
        i-=lowbit(i);
    }
    return res;
}

int main() {
    while(1) {
        cin>>n;
        if(n==0) break;
        int a;
        memset(arr,0,sizeof arr);
        memset(c,0,sizeof c);
        for(int i=1; i<=n; i++) {
            scanf("%d",&a);
            arr[i].id=i;
            arr[i].v=a;
        }
```

```
        sort(arr+1,arr+1+n);
        ll ans=0;
        for(int i=1; i<=n; i++) {
            add(arr[i].id); //离散化结果—— 下标等效于数值
            ans+=i-getsum(arr[i].id); //得到之前有多少个比你大的数（逆序对）
        }
        cout<<ans<<endl;
    }
    return 0;
}
```

3.6 线段树

3.6.1 区间维护各种属性

注意 lazy 标记的初始化

```

#define ls u<<1
#define rs u<<1|1
struct node{
    int l,r;
    int sum;
}tr[N<<2];
int lazy[N<<2];
void pushup(int u){
    tr[u].sum=tr[ls].sum+tr[rs].sum;
}
void pushdown(int u){
    if(!lazy[u]) return ;
    tr[ls].sum+=(tr[ls].r-tr[ls].l+1)*lazy[u];
    tr[rs].sum+=(tr[rs].r-tr[rs].l+1)*lazy[u];
    lazy[ls]+=lazy[u];
    lazy[rs]+=lazy[u];
    lazy[u]=0; //加法初始化为0
// lazy[u]=1; //乘法法初始化为1
}
void build(int u,int l,int r){
    lazy[u]=0; //初始化标记
    if(l==r) tr[u]={l,r,0};
    else{
        tr[u]={l,r};
        int mid=(l+r)>>1;
        build(ls,l,mid);
        build(rs,mid+1,r);
        pushup(u);
    }
}
void update(int u,int l,int r,int c){
    if(tr[u].l>=l && tr[u].r<=r){
        tr[u].sum+=(tr[u].r-tr[u].l+1)*c; //区间值加上c
        lazy[u]+=c; //更新加法标记
    }
    else{
        pushdown(u);
        int mid=(tr[u].l+tr[u].r)>>1;
        if(l<=mid) update(ls,l,r,c);
        if(r>mid) update(rs,l,r,c);
        pushup(u);
    }
}
int query(int u,int l,int r){
    if(tr[u].l>=l && tr[u].r<=r) return tr[u].sum;
    else{
        pushdown(u);
```

```
        int mid=(tr[u].r+tr[u].l)>>1;
        int res=0;
        if(l<=mid) res=res+query(ls,l,r);
        if(r>mid) res=res+query(rs,l,r);
        return res;
    }
}
```

3.6.2 区间维护最大连续子段和

```

#include <bits/stdc++.h>
#define endl '\n'
#define ls u<<1
#define rs u<<1|1
#define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
using namespace std;
const int N=500100;
struct node{
    int l,r;
    int lmx,rmx;
    int sum,ms;
}tr[N<<2];
int arr[N];
void pushup(int u){
    tr[u].sum=tr[ls].sum+tr[rs].sum;
    tr[u].lmx=max(tr[ls].lmx,tr[ls].sum+tr[rs].lmx);
    tr[u].rmx=max(tr[rs].rmx,tr[rs].sum+tr[ls].rmx);
    tr[u].ms=max(tr[ls].ms,tr[rs].ms);
    tr[u].ms=max(tr[ls].rmx+tr[rs].lmx,tr[u].ms);
}
void build(int u,int l,int r){
    if(l==r){
        tr[u]={l,r,arr[l],arr[l],arr[l],arr[l]};
    }
    else{
        int mid=l+r>>1;
        tr[u]={l,r};
        build(u<<1,l,mid);
        build(u<<1|1,mid+1,r);
        pushup(u);
    }
}
void update(int u,int pos,int c){
    if(tr[u].l==tr[u].r && tr[u].l==pos){
        tr[u].ms=c;
        tr[u].lmx=c;
        tr[u].rmx=c;
        tr[u].sum=c;
        return ;
    }
    else{
        int mid=tr[u].l+tr[u].r>>1;
        if(pos<=mid) update(ls,pos,c);
        else update(rs,pos,c);
        pushup(u);
    }
}
node query(int u,int l,int r)
{
    if(tr[u].l>=l&&tr[u].r<=r)return tr[u];
    else {
        int mid=(tr[u].l+tr[u].r)>>1;
        if(r<=mid)return query(u<<1,l,r);
```

```
        else if(l>mid)return query(u<<1|1,l,r);
        else
        {
            node a=query(u<<1,l,r),b=query(u<<1|1,l,r);
            node c;
            c.sum=a.sum+b.sum;
            c.lmx=max(a.lmx,a.sum+b.lmx);
            c.rmx=max(b.rmx,a.rmx+b.sum);
            c.ms=max(a.rmx+b.lmx,max(a.ms,b.ms));
            return c;
        }
    }
}
int m,n;
signed main()
{
    ios;
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>arr[i];
    build(1,1,n);
    while(m--){
        int k,x,y;
        cin>>k>>x>>y;
        if(k==1){
            if(x>y) swap(x,y);
            node tmp=query(1,x,y);
            cout<<tmp.ms<<endl;
        }
        else{
            update(1,x,y);
        }
    }

    return 0;
}
```

3.6.3 区间维护最大公约数

$gcd(a,b) = gcd(a,b-a)$

$(a,b,c) = ((a,b),(b,c)) = ((a,b-a),(b,c-b)) = ((a,b),(b-a,c-b)) = (a,b-a,c-b)$

通过这个式子可以发现，

$gcd(a,b,c,d...) = gcd(a,b-a,c-b,d-c...)$

这样可以维护一个 $b[i] = a[i] - a[i-1]$

区间加 d 时，只有 $b[l]$ $b[r+1]$ 的位置需要修改，也就变成了单点修改，区间查询问题，需要注意 r+1<=n，还有查询 $gcd(l\ r)$ 时，$a[l]$ 是原值，不是 $b[i]$，而 $sum(a[1]\ a[l])$ 正好是原值，就省去了懒标记操作了

```

#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 5e5 + 23;
struct Node {
    int l, r;
    ll v, d;
} tr[maxn * 4];
ll a[maxn], b[maxn];
void pushup(Node &u, Node &l, Node &r) {
    u.v = l.v + r.v;
    u.d = __gcd(l.d, r.d);
}
void pushup(int u) {
    pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
}
void build(int u, int l, int r) {
```

```
    if(l == r) tr[u] = {l, r, b[l], b[l]};
    else {
        tr[u] = {l, r};
        int mid = l + r >> 1;
        build(u << 1, l, mid);
        build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}
ll query(int u, int l, int r) {  //区间gcd
    if(tr[u].l >= l && tr[u].r <= r) return tr[u].d;
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if(r <= mid) return query(u << 1, l, r);
        else if(l > mid) return query(u << 1 | 1, l, r);
        else return __gcd(query(u << 1, l, r), query(u << 1 | 1, l, r));
    }
}
ll query2(int u, int l, int r) { //区间和
    if(tr[u].l >= l && tr[u].r <= r) return tr[u].v;
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if(r <= mid) return query2(u << 1, l, r);
        else if(l > mid) return query2(u << 1 | 1, l, r);
        else return query2(u << 1, l, r) + query2(u << 1 | 1, l, r);
    }
}
void modify(int u, int p, ll v) {
    if(tr[u].l == tr[u].r && tr[u].l == p) tr[u].d += v, tr[u].v += v;
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if(p <= mid) modify(u << 1, p, v);
        else modify(u << 1 | 1, p, v);
        pushup(u);
    }
}
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i++) scanf("%lld", &a[i]), b[i] = a[i] - a[i - 1];
    build(1, 1, n);
    char op[4];
    ll op1 = 0, op2 = 0, op3 = 0;
    while(m--) {
        scanf("%s%lld%lld", op, &op1, &op2);
        if(*op == 'C') {
            scanf("%lld", &op3);
            modify(1, op1, op3);
            if(op2 + 1 <= n) modify(1, op2 + 1, -op3);
        } else {
            ll t = query2(1, 1, op1);
            printf("%lld\n", abs( __gcd(t, query(1, op1 + 1, op2))));
        }
    }
}
```

3.6.4 区间同时维护乘法和加法

注意 lazy 标记的初始化

```

#include <bits/stdc++.h>
#define ios ios::sync_with_stdio(0); cin.tie(0); cout.tie(0)
using namespace std;
typedef long long ll;
```

```
const ll MOD=1e9+7;
const ll SUP=0x800000;
const ll MAXN=1e5+10;
const ll INF=0x3f3f3f3f;
const double eps=1e-4;
struct node{
    ll l,r,val;
}tr[MAXN<<2];
//lazy1加法标记 lazy2乘法标记
ll lazy1[MAXN<<2],lazy2[MAXN<<2],arr[MAXN];
ll n,m,p;
void pushup(ll u){
    tr[u].val=(tr[u<<1].val+tr[u<<1|1].val)%p;
}
void pushdown(ll u){
    ll l=u<<1,r=u<<1|1;
    // 儿子的值=儿子的值*父亲的乘法标记+儿子区间*父亲加法标记
    tr[l].val=(tr[l].val*lazy2[u]%p+(tr[l].r-tr[l].l+1)*lazy1[u]%p)%p;
    tr[r].val=(tr[r].val*lazy2[u]%p+(tr[r].r-tr[r].l+1)*lazy1[u]%p)%p;
    // 儿子加法标记更新，儿子加法标记=儿子加法标记*父亲乘法标记+父亲加法标记
    lazy1[l]=(lazy1[l]*lazy2[u]%p+lazy1[u])%p;
    lazy1[r]=(lazy1[r]*lazy2[u]%p+lazy1[u])%p;
    // 儿子乘法标记=儿子乘法标记*父亲乘法标记
    lazy2[l]=lazy2[l]*lazy2[u]%p;
    lazy2[r]=lazy2[r]*lazy2[u]%p;

    lazy1[u]=0;
    lazy2[u]=1;
}
void build(ll u,ll l,ll r){
    lazy1[u]=0; //初始化标记
    lazy2[u]=1;
    if(l==r) tr[u]={l,r,arr[l]};
    else{
        tr[u]={l,r};
        ll mid=(l+r)>>1;
        build(u<<1,l,mid);
        build(u<<1|1,mid+1,r);
        pushup(u);
    }
}
void add(ll u,ll l,ll r,ll c){
    if(tr[u].l>=l && tr[u].r<=r){
        tr[u].val=(tr[u].val+(tr[u].r-tr[u].l+1)*c%p)%p; //区间值加上c
        lazy1[u]=(c+lazy1[u])%p; //更新加法标记
    }
    else{
        pushdown(u);
        ll mid=(tr[u].l+tr[u].r)>>1;
        if(l<=mid) add(u<<1,l,r,c);
        if(r>mid) add(u<<1|1,l,r,c);
        pushup(u);
    }
}
void mul(ll u,ll l,ll r,ll c){
    if(tr[u].l>=l && tr[u].r<=r){
        tr[u].val=tr[u].val*c%p; //区间的值乘上c
        lazy1[u]=(lazy1[u]*c)%p; //每次更新乘法标记时要顺带着把加法标记也更新了，目的是确定优先级
        lazy2[u]=(lazy2[u]*c)%p; //更新乘法标记
    }
    else{
        pushdown(u);
        ll mid=(tr[u].l+tr[u].r)>>1;
        if(l<=mid) mul(u<<1,l,r,c);
        if(r>mid) mul(u<<1|1,l,r,c);
        pushup(u);
```

```
    }
}
ll query(ll u,ll l,ll r){
    if(tr[u].l>=l && tr[u].r<=r) return tr[u].val;
    else{
        pushdown(u);
        ll mid=(tr[u].r+tr[u].l)>>1,ret=0;
        if(l<=mid) ret=(ret+query(u<<1,l,r))%p;
        if(r>mid) ret=(ret+query(u<<1|1,l,r))%p;
        return ret;
    }
}
int main()
{
    ios;
    cin>>n>>m>>p;
    for(ll i=1;i<=n;i++) cin>>arr[i];
    build(1,1,n);
    while(m--){
        ll op,x,y,k;
        cin>>op;
        if(op==1){
            cin>>x>>y>>k;
            mul(1,x,y,k);
        }
        else if(op==2){
            cin>>x>>y>>k;
            add(1,x,y,k);
        }
        else{
            cin>>x>>y;
            cout<<query(1,x,y)<<'\n';
        }
    }
    return 0;
 }
```

3.7 维护 n 棵线段树

例题：

初始有 n 个 1，接下来 q 次操作

1. MULTIPLY l r x(x<=10)：给 [l,r] 区间上的数字乘上 x

2. MAX l r：求出区间 [l,r] 内哪一个质因子数量最多，输出数量

因为 x<=10，所以最后 [1,n] 区间内的数字最多就 4 种质因子，2,3,5,7，给每一个质因子建议一颗线段树，区间乘上 x，就对 x 进行质因子分解，去每一颗质因子的线段树上进行区间加的修改，询问的时候输出四种质因子区间内最大数是多少即可。

```
#include <bits/stdc++.h>
#define ls u<<1
#define rs u<<1|1
#define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define endl '\n'
using namespace std;
const int N=1e5+100;
int n,q;
struct node{
    int l,r;
    int sum,mx;
}tr[10][N<<2];
int lazy[10][N<<2];
void pushup(int id,int u){
    tr[id][u].sum=tr[id][ls].sum+tr[id][rs].sum;
    tr[id][u].mx=max(tr[id][ls].mx,tr[id][rs].mx);
}
```

```
18 void pushdown(int id,int u){
19     if(lazy[id][u]){
20         tr[id][ls].sum+=(tr[id][ls].r-tr[id][ls].l+1)*lazy[id][u];
21         tr[id][rs].sum+=(tr[id][rs].r-tr[id][rs].l+1)*lazy[id][u];
22         tr[id][ls].mx+=lazy[id][u];
23         tr[id][rs].mx+=lazy[id][u];
24         lazy[id][ls]+=lazy[id][u];
25         lazy[id][rs]+=lazy[id][u];
26         lazy[id][u]=0;
27     }
28 }
29 void build(int id,int u,int l,int r){
30     if(l==r) tr[id][u]={l,r,0,0};
31     else{
32         int mid=l+r>>1;
33         tr[id][u]={l,r};
34         build(id,ls,l,mid);
35         build(id,rs,mid+1,r);
36     }
37 }
38 void update(int id,int u,int l,int r,int c){
39     if(tr[id][u].l>=l && tr[id][u].r<=r){
40 //  cout<<tr[id][u].l<<" "<<tr[id][u].r<<" "<<l<<" "<<r<<endl;
41         tr[id][u].sum+=c*(tr[id][u].r-tr[id][u].l+1);
42         tr[id][u].mx+=c;
43         lazy[id][u]+=c;
44     }
45     else{
46         pushdown(id,u);
47         int mid=tr[id][u].l+tr[id][u].r>>1;
48         if(l<=mid) update(id,ls,l,r,c);
49         if(r>mid) update(id,rs,l,r,c);
50         pushup(id,u);
51     }
52 }
53 int query(int id,int u,int l,int r){
54     if(tr[id][u].l>=l && tr[id][u].r<=r) return tr[id][u].mx;
55     else{
56         pushdown(id,u);
57         int mid=tr[id][u].l+tr[id][u].r>>1;
58         int res=0;
59         if(l<=mid) res=max(res,query(id,ls,l,r));
60         if(r>mid) res=max(res,query(id,rs,l,r));
61         return res;
62     }
63 }
64 int mp[10];
65 int main()
66 {
67     ios;
68     cin>>n>>q;
69     mp[2]=1; mp[3]=2; mp[5]=3; mp[7]=4;
70     for(int i=1;i<=4;i++) build(i,1,1,n);
71     while(q--){
72         string s;
73         cin>>s;
74         if(s[1]=='U'){
75             int l,r,x;
76             cin>>l>>r>>x;
77             for(int i=2;i<=sqrt(x);i++){
78                 if(x%i==0){
79                     int cnt=0;
80                     while(x%i==0){
81                         cnt++;
82                         x/=i;
83                     }
```

```
                update(mp[i],1,l,r,cnt);
            }
        }
        if(x>1) update(mp[x],1,l,r,1);
    }
    else{
        int l,r;
        cin>>l>>r;
        int ans=0;
        for(int i=1;i<=4;i++){
            int tmp=query(i,1,l,r);
            ans=max(ans,tmp);
        }
        cout<<"ANSWER "<<ans<<endl;
    }
  }
  return 0;
}
```

3.8 主席树 (可持久化权值线段树)

3.8.1 求区间第 K 大、第 K 小问题

```

#include <bits/stdc++.h>
#define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
using namespace std;
const int N=1e6+100;
struct node {
    int ls,rs,sum;
} tr[N*40];
int rt[N*40];
int arr[N],brr[N];
int n,m,tot;
int build(int l,int r) { //新建一颗空树
    int root=++tot;
    if(l==r) return root;
    int mid=l+r>>1;
    tr[root].ls=build(l,mid);
    tr[root].rs=build(mid+1,r);
    return root;
}
int insert(int pre,int l,int r,int pos) { //在上一个版本的基础上新建一个版本并插入pos值
    int root=++tot;
    tr[root]=tr[pre];
    tr[root].sum++;
    if(l==r) return root;
    int mid=(l+r)>>1;
    if(pos<=mid) tr[root].ls=insert(tr[pre].ls,l,mid,pos);
    else tr[root].rs=insert(tr[pre].rs,mid+1,r,pos);
    return root;
}
int query_k_min(int u,int v,int l,int r,int k) { //查询第k小
    if(l==r) return l;
    int cha=tr[tr[v].ls].sum-tr[tr[u].ls].sum;
    int mid=l+r>>1;
    if(cha>=k) return query_k_min(tr[u].ls,tr[v].ls,l,mid,k);
    else return query_k_min(tr[u].rs,tr[v].rs,mid+1,r,k-cha);
}
int query_k_max(int u,int v,int l,int r,int k) { //查询第k大
    if(l==r) return l;
    int cha=tr[tr[v].rs].sum-tr[tr[u].rs].sum;
    int mid=l+r>>1;
    if(cha>=k) return query_k_max(tr[u].rs,tr[v].rs,l,mid,k);
```

```
    else return query_k_max(tr[u].ls,tr[v].ls,mid+1,r,k-cha);
}
int main() {
    ios;
    cin>>n>>m;
    for(int i=1; i<=n; i++) {
        cin>>arr[i];
        brr[i]=arr[i];
    }
    sort(arr+1,arr+1+n); //离散化，brr存离散值，arr原值
    for(int i=1; i<=n; i++) brr[i]=lower_bound(arr+1,arr+1+n,brr[i])-arr;

    rt[0]=build(1,n);
    for(int i=1; i<=n; i++) rt[i]=insert(rt[i-1],1,n,brr[i]);

    while(m--) {
        int l,r,k;
        cin>>l>>r>>k;
        cout<<arr[query_k_min(rt[l-1],rt[r],1,n,k)]<<'\n';
    }
    return 0;
}
```

3.8.2 维护区间内存在多少个不同的数字（单纯的权值线段树做不到）

```

int query_sum(int u,int v,int l,int r,int ql,int qr){ //查询区间和
    if(l>=ql && r<=qr) return tr[v].sum-tr[u].sum;
    else{
        int mid=l+r>>1;
        int res=0;
        if(ql<=mid) res+=query_sum(tr[u].ls,tr[v].ls,l,mid,ql,qr);
        if(qr>mid) res+=query_sum(tr[u].rs,tr[v].rs,mid+1,r,ql,qr);
        return res;
    }
}
```

3.9 字典树

4 图论

4.1 tarjan

缩点经常和拓扑 dp 一起使用，缩点的目的就是把图中所有可以互相到达的点看成一个点 (强连通分量)，然后开一个数组维护强连通分量内部的信息，缩点后图就变成了 DAG 有向无环图，跑拓扑的原因是可以降低时复，bfs 会跑重复的点，例如一条链，最开始的地方有一个分叉，之后又合上了，但是后面的部分非常长，这样小分叉多点，时复就炸了

```
#include<bits/stdc++.h>
#define endl '\n'
#define ios ios::sync_with_stdio(false),cin.tie(0),cout.tie(0)
using namespace std;
const int N=1e5+100;

//链式前向星存图
struct Edge{
    int to,next;
}e[1000100];
int h[N],tot;
void add(int u,int v){
    e[tot]={v,h[u]};
    h[u]=tot++;
```

```
}

/*
dfn: 时间戳
siz: 联通块内部点数量
idx: 点所属联通块编号
instk: 是否在栈中
*/
int dfn[N],low[N],siz[N],idx[N];
bool instk[N];
int tim,cnt;  //时间戳，联通块数量
stack<int> stk;
void tarjan(int x){  //缩点
    dfn[x]=low[x]=++tim;
    stk.push(x);
    instk[x]=1;
    for(int i=h[x];~i;i=e[i].next){
        int v=e[i].to;
        if(!dfn[v]){
            tarjan(v);
            low[x]=min(low[v],low[x]);
        }
        else if(instk[v]) low[x]=min(low[x],dfn[v]);
    }
    if(low[x]==dfn[x]){
        cnt++;
        while(stk.top()!=x){
            siz[cnt]++;
            idx[stk.top()]=cnt;
            instk[stk.top()]=0;
            stk.pop();
        }
        siz[cnt]++;
        idx[stk.top()]=cnt;
        instk[stk.top()]=0;
        stk.pop();
    }
}
/*
G是缩点后的图
in: 入度
mi: 联通块内部点权最小
mx: 联通块内部点权最大
*/
vector<int> G[N];
int in[N],mi[N],mx[N];
void tupo(){
    queue<int> q;
    //从这里开始是把1前面的点跑拓扑给释放掉，因为起点是1，不释放的话会把起点之前的点也算进去
    for(int i=1;i<=cnt;i++){
        if(!in[i] && i!=idx[1]) q.push(i);
    }
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(auto v:G[u]){
            if(v!=idx[1]) in[v]--;
            if(!in[v]) q.push(v);
        }
    }
    //到这里结束
    q.push(idx[1]);
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(auto v:G[u]){
```

```
            /*
            mi[v]=min(mi[v],mi[u]);
            ans[v]=max(ans[v],mx[v]-mi[v]);
            ans[v]=max(ans[v],ans[u]);
            */
            in[v]--;
            if(!in[v]) q.push(v);
        }
    }
}

int n,m;
int num[N];
int main() {
    ios;
    memset(h,-1,sizeof h); //记得初始化
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>num[i];
    while(m--){
        int u,v,op;
        cin>>u>>v>>op;
        add(u,v);
        if(op==2) add(v,u);
    }
    memset(mi,0x3f,sizeof mi);
    //图可能不连通，所以需要遍历所有点
    for(int i=1;i<=n;i++){
        if(!dfn[i]) tarjan(i);
    }
    //每一个点去更新他所在联通块的最值
    for(int i=1;i<=n;i++){
        mi[idx[i]]=min(mi[idx[i]],num[i]);
        mx[idx[i]]=max(mx[idx[i]],num[i]);
    }
    //建新图
    for(int i=1;i<=n;i++){
        for(int j=h[i];~j;j=e[j].next){
            int v=e[j].to;
            if(idx[v]!=idx[i]) G[idx[i]].push_back(idx[v]);
        }
    }
    //新图入度
    for(int i=1;i<=cnt;i++){
        for(auto v:G[i]){
            in[v]++;
        }
    }
    //拓扑
    tupo();
    cout<<ans[idx[n]]<<endl;
    return 0;
}
/*
6 6
2 6 2 1 8 9
1 4 1
4 5 1
5 1 1
1 2 1
2 3 1
2 6 1
*/
```

4.2 floyed 求最小环

```
for(int k=1;k<=n;k++){
    for(int i=1;i<k;i++){ //这段代码执行时k还没有更新其他点的最短距离
        for(int j=i+1;j<k;j++){
                ans=min(ans,dis[i][j]+G[j][k]+G[k][i]);
        }
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
        }
    }
}
```

5 数论

6 杂类

6.1 int128

```
void read(int128 &x) {
    x = 0;
    int f = 1;
    char ch;
    if((ch = getchar()) == '-') f = -f;
    else x = x*10 + ch-'0';
    while((ch = getchar()) >= '0' && ch <= '9')
        x = x*10 + ch-'0';
    x *= f;
}
void print(int128 x) { //输出
    if(x < 0) {
        x = -x;
        putchar('-');
    }
    if(x > 9) print(x/10);
    putchar(x%10 + '0');
}
```

6.2 O(3) 优化

```
#pragma GCC optimize(3,"Ofast","inline")
```

6.3 模拟退火

```
//随机数
srand(time(NULL));
int rand_INT(int l,int r) { //产生[l,r]的的一个随机数
    return rand()%(r-l+1)+l;
}
double rand_DOUBLE(double l,double r){ //产生[l,r]的的一个随机数
    return (double)rand()/RAND_MAX*(r-l)+l;
}
//以Acwing进阶课题目为例
#include <bits/stdc++.h>
#define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define endl '\n'
using namespace std;
const int N=110;
struct Point{
```

```
    double x,y;
}q[N];
int n;
double ans=1e8;
double rand(double l,double r){ //产生[l,r]的的一个随机数
    return (double)rand()/RAND_MAX*(r-l)+l;
}
double getdis(Point x){
    double res=0;
    for(int i=1;i<=n;i++){
        res+=sqrt(pow(q[i].x-x.x,2)+pow(q[i].y-x.y,2));
    }
    ans=min(res,ans);
    return res;
}
void simulate_anneal(){ //退火函数
    Point now={rand(0,10000),rand(0,10000)};
    for(double t=1e4;t>1e-4;t*=0.999){ //t是温度,
        Point np={rand(0,10000),rand(0,10000)}; //np是在[x-t,x+t]范围内产生的新点
        double dt=getdis(np)-getdis(now); //dt是新点函数值和旧点函数值的差值
        //这里需要注意这道题取得是最小值，所以加了负号，如果求最大值就要把负号去掉！！
        if(exp(-dt/t)>rand(0,1)) now=np; //如果新点好就一定跳过去，否则以一定概率跳过去
    }
}
int main()
{
    ios;
    cin>>n;
    for(int i=1;i<=n;i++) cin>>q[i].x>>q[i].y;
    for(int i=1;i<=100;i++) simulate_anneal();
    cout<<fixed<<setprecision(0)<<ans<<endl;
    return 0;
}
```

6.4 手写 Hash(int)

```
struct MAP{
    int tot;
    int nxt[M],las[P],to[M],w[M];
    MAP(){
        tot=0;
        for(int i=0;i<=P-1;++i)
            las[i]=0;
    }
    void insert(int y){
        int x=y%P;
        for(int e=las[x];e;e=nxt[e]){
            if(to[e]==y){
                ++w[e];
                return;
            }
        }
        nxt[++tot]=las[x];
        las[x]=tot;
        to[tot]=y;
        w[tot]=1;
    }
    int find(int y){
        int x=y%P;
        for(int e=las[x];e;e=nxt[e]){
            if(to[e]==y)
                return w[e];
        }
        return 0;
```

```
    }
} mp1;
```

section 字符串 subsection 字符串哈希

```
typedef unsigned long long ull;
ull ha[N];
ull po[N];
ull bas=1331;
void init_hash(string &s){
    int len=s.size();
    s=" "+s;
    po[0]=1; ha[0]=0;
    for(int i=1;i<=len;i++){
        ha[i]=ha[i-1]*bas+s[i]-'0';
        po[i]=po[i-1]*bas;
    }
}
ull get_hash(int l,int r){
    return ha[r]-ha[l-1]*po[r-l+1];
}
```

subsectionKMP

```
int ne[N]; //next数组: i位置之前的最长公共前后缀
void get_next(string &s){
    int len=s.size();
    int i=0, j=-1;
    ne[0]=-1;
    while(i<len){
        if(j==-1 || s[i]==s[j]){
            ++i; ++j;
            ne[i]=j;
        }
        else j=ne[j];
    }
    return ;
}
int kmp(string &s,string &t){
    get_next(t);
    int len1=s.size(),len2=t.size();
    int i=0, j=0, cnt=0;
    while(i<len1){
        if(j==-1 || s[i]==t[j]){
            ++i; ++j;
        }
        else j=ne[j];
        if(j==len2) cnt++;
    }
    return cnt;
}
//nextval优化
void get_nextval(string s){
    int i=0, j=-1;
    nextval[0]=-1;
    int len=s.size();
    while(i<len){
        if(j==-1 || s[i]==s[j]){
            ++i; ++j;
            // 和求解next数组唯一不一样的地方
            if(s[i]!=s[j]) nextval[i]=j;
            else nextval[i]=nextval[j]; //当前位置和next[i]的字符相同时，则不需要回朔到next[i]位置，因为这个位置的字符一定会失配，所以让nextval[i]直接指向nextval[next[i]]，当没有跳步回朔操作时next和nextval值是一样的
        }
        else j=nextval[j];
    }
```

```
}
```

subsection 扩展 KMP

```
void getnext(string T){
    int len=T.size();
    nex[0]=len;
    int p=0;
    while(p+1<len && T[p]==T[p+1]) p++; // 这里注意把边界写在前面
    nex[1]=p;
    int po=1;
    for(int i=2;i<len;i++){
        if(i+nex[i-po]<po+nex[po]) nex[i]=nex[i-po]; // 第一种情况，直接得到答案
        else{
            int j=po+nex[po]-i;
            if(j<0) j=0; // 超出已匹配的字符串长度，需要重新匹配
            while(i+j<len && T[i+j]==T[j]) j++;
            nex[i]=j;
            po=i; // 长度超出，更新起始位置
        }
    }
}
void extmp(string S,string T){
    int len1=S.size(), len2=T.size();
    getnext(T);
    int p=0;
    while(p<len1 && p<len2 && S[p]==T[p]) p++; // 边界写到前面
    ext[0]=p;
    int po=0;
    for(int i=1;i<len1;i++){
        if(i+nex[i-po]<po+ext[po]) ext[i]=nex[i-po];
        else{
            int j=po+ext[po]-i;
            if(j<0) j=0;
            while(i+j<len1 && j<len2 && T[j]==S[i+j]) j++;
            ext[i]=j;
            po=i;
        }
    }
}
```

6.5 马拉车算法

```
int p[N];
string s;
int manacher(string s){
    string t="";
    t+='*';
    for(int i=0;i<(int)s.size();i++){
        t+=s[i];
        t+='*';
    }
    int ans=0;
    int pos=0;int maxxright=0;
    for(int i=0;i<(int)t.length();i++){
        p[i]=maxxright>i?min(p[2*pos-i],maxxright-i):1;//关键
        while(i-p[i]>=0&&i+p[i]<(int)t.length()&&t[i-p[i]]==t[i+p[i]]) p[i]++;
        if(i+p[i]-1>maxxright){
            maxxright=i+p[i]-1;
            pos=i;
        }
        ans=max(ans,p[i]);
    }
    return ans-1;
}
```