

# ACM 模板

PocketCat

2021-11-12

## 目录

<b>1</b>	<b>高精</b>	<b>3</b>
1.1	两个非负整数相加	3
1.2	大的非负整数减小的非负整数	4
1.3	高精乘高精	4
1.4	高精度 a 乘低精度 b	4
1.5	高精除高精 + 高精除低精	5
1.6	高精度幂	6
1.7	阶层	8
1.8	求最大公约数	8
1.9	取模（高精）	10
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）	10
<b>2</b>	<b>计算几何</b>	<b>10</b>
2.1	基础模板	10
2.2	点积	11
2.3	叉积	11
2.4	模长	11
2.5	两个向量是否同象限	11
2.6	两向量是否共线	12
2.7	两向量是否垂直	12
2.8	两个向量是否同方向	12
2.9	点旋转	12
2.10	两点之间的距离	12
2.11	直线	12
2.11.1	直线和线段是否相交	12
2.11.2	直线和直线的交点	13
2.11.3	判断 b 和 c 的交点是否在直线右面	13
2.11.4	得到直线的角度	13
2.11.5	求点关于直线的对称点	13
2.11.6	点到直线的距离	13
2.11.7	点在直线的投影	14
2.12	求多边形的面积	14
2.13	线段	14
2.13.1	判断一个点是否在一个线段上	14
2.13.2	判断线和线段是否相交	14
2.14	圆	14
2.14.1	直线和圆的交点	14
2.15	凸包	15
2.16	三维凸包	15
2.17	半平面交	16
2.18	最小圆覆盖	17
2.19	圆的面积并	18

<b>3</b>	<b>数据结构</b>	<b>20</b>
3.1	单调栈	20
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	20
3.1.2	求一个柱状图的最大矩形面积	20
3.2	单调队列	21
3.2.1	区间 (长度固定) 最值问题	21
3.2.2	最大值减最小值小于等于 $k$ 的子区间数量	21
3.3	并查集	21
3.3.1	按秩合并	21
3.3.2	关押罪犯	22
3.3.3	食物链	22
3.4	AC 自动机	23
3.5	树状数组	24
3.5.1	单点修改, 区间查询	24
3.5.2	树状数组求逆序数	24
3.6	线段树	25
3.6.1	区间维护各种属性	25
3.6.2	区间维护最大连续子段和	26
3.6.3	区间维护最大公约数	28
3.6.4	区间同时维护乘法和加法	29
3.7	维护 $n$ 棵线段树	31
3.8	主席树 (可持久化权值线段树)	33
3.8.1	求区间第 $K$ 大、第 $K$ 小问题	33
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)	34
3.9	字典树	34
<b>4</b>	<b>图论</b>	<b>34</b>
4.1	tarjan	34
4.2	floyd 求最小环	36
<b>5</b>	<b>数论</b>	<b>37</b>
<b>6</b>	<b>杂类</b>	<b>37</b>
6.1	对拍	37
6.2	int128	37
6.3	$O(3)$ 优化	38
6.4	模拟退火	38
6.5	手写 Hash(int)	38
6.6	自适应辛普森积分	39
<b>7</b>	<b>字符串</b>	<b>39</b>
7.1	字符串哈希	39
7.2	KMP	40
7.3	扩展 KMP	41
7.4	马拉车算法	41

# 1 高精

## 1.1 两个非负整数相加

```
1
2 //加法
3 string add(string a,string b)//只限两个非负整数相加
4 {
5     const int L=1e5;
6     string ans;
7     int na[L]={0},nb[L]={0};
8     int la=a.size(),lb=b.size();
9     for(int i=0;i<la;i++) na[la-1-i]=a[i]-'0';
10    for(int i=0;i<lb;i++) nb[lb-1-i]=b[i]-'0';
11    int lmax=la>lb?la:lb;
12    for(int i=0;i<lmax;i++) na[i]+=nb[i],na[i+1]+=na[i]/10,na[i]%=10;
13    if(na[lmax]) lmax++;
14    for(int i=lmax-1;i>=0;i--) ans+=na[i]+'0';
15    return ans;
16 }
```

## 1.2 大的非负整数减小的非负整数

```
1
2 //减法
3 string sub(string a,string b)//只限大的非负整数减小的非负整数
4 {
5     const int L=1e5;
6     string ans;
7     int na[L]={0},nb[L]={0};
8     int la=a.size(),lb=b.size();
9     for(int i=0;i<la;i++) na[la-1-i]=a[i]-'0';
10    for(int i=0;i<lb;i++) nb[lb-1-i]=b[i]-'0';
11    int lmax=la>lb?la:lb;
12    for(int i=0;i<lmax;i++)
13    {
14        na[i]-=nb[i];
15        if(na[i]<0) na[i]+=10,na[i+1]--;
16    }
17    while(!na[--lmax]&& lmax>0) ;lmax++;
18    for(int i=lmax-1;i>=0;i--) ans+=na[i]+'0';
19    return ans;
20 }
```

## 1.3 高精乘高精

```
1
2 string mul(string a,string b)//高精度乘法a,b,均为非负整数
3 {
4     const int L=1e5;
5     string s;
6     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
7     fill(na,na+L,0);fill(nb,nb+L,0);fill(nc,nc+L,0);//将na,nb,nc都置为0
8     for(int i=La-1;i>=0;i--) na[La-1-i]=a[i]-'0';//将字符串表示的大整数数组转成i整数数组表示的大整数数
9     for(int i=Lb-1;i>=0;i--) nb[Lb-1-i]=b[i]-'0';
10    for(int i=1;i<=La;i++)
11    {
12        for(int j=1;j<=Lb;j++)
13            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)
14    }
15    for(int i=1;i<=La+Lb;i++)
16        nc[i+1]+=nc[i]/10,nc[i]%=10;//统一处理进位
17    if(nc[La+Lb]) s+=nc[La+Lb]+'0';//判断第i+j位上的数字是不是0
18    for(int i=La+Lb-1;i>=1;i--)
```

```

17     s+=nc[i]+'0';//将整形数组转成字符串
18     return s;
19 }

```

## 1.4 高精度 a 乘低精度 b

```

1
2 string mul(string a,int b)//高精度a乘单精度b
3 {
4     const int L=100005;
5     int na[L];
6     string ans;
7     int La=a.size();
8     fill(na,na+L,0);
9     for(int i=La-1;i>=0;i--) na[La-i-1]=a[i]-'0';
10    int w=0;
11    for(int i=0;i<La;i++) na[i]=na[i]*b+w,w=na[i]/10,na[i]=na[i]%10;
12    while(w) na[La++]=w%10,w/=10;
13    La--;
14    while(La>=0) ans+=na[La--]+'0';
15    return ans;
16 }

```

## 1.5 高精除高精 + 高精除低精

```

1
2 //除法
3 int sub(int *a,int *b,int La,int Lb)
4 {
5     if(La<Lb) return -1;//如果a小于b, 则返回-1
6     if(La==Lb)
7     {
8         for(int i=La-1;i>=0;i--)
9             if(a[i]>b[i]) break;
10            else if(a[i]<b[i]) return -1;//如果a小于b, 则返回-1
11    }
12    for(int i=0;i<La;i++)//高精度减法
13    {
14        a[i]-=b[i];
15        if(a[i]<0) a[i]+=10,a[i+1]--;
16    }
17    for(int i=La-1;i>=0;i--)
18        if(a[i]) return i+1;//返回差的位数
19    return 0;//返回差的位数
20 }
21
22 string div(string n1,string n2,int nn)
23 //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
24 {
25     const int L=1e5;
26     string s,v;//s存商,v存余数
27     int a[L],b[L],r[L],La=n1.size(),Lb=n2.size(),i,tp=La;
28     //a, b是整形数组表示被除数, 除数, tp保存被除数的长度
29     fill(a,a+L,0);fill(b,b+L,0);fill(r,r+L,0);//数组元素都置为0
30     for(i=La-1;i>=0;i--) a[La-1-i]=n1[i]-'0';
31     for(i=Lb-1;i>=0;i--) b[Lb-1-i]=n2[i]-'0';
32     if(La<Lb || (La==Lb && n1<n2)) {
33         //cout<<0<<endl;
34         return n1;}//如果a<b,则商为0, 余数为被除数
35     int t=La-Lb;//除被数和除数的位数之差
36     for(int i=La-1;i>=0;i--)//将除数扩大10^t倍
37         if(i>=t) b[i]=b[i-t];
38

```

```

39     else b[i]=0;
40     Lb=La;
41     for(int j=0;j<=t;j++)
42     {
43         int temp;
44         while((temp=sub(a,b+j,La,Lb-j))>=0)//如果被除数比除数大继续减
45         {
46             La=temp;
47             r[t-j]++;
48         }
49     }
50     for(i=0;i<L-10;i++) r[i+1]+=r[i]/10,r[i]%10;//统一处理进位
51     while(!r[i]) i--;//将整形数组表示的商转化成字符串表示的
52     while(i>=0) s+=r[i--]+'0';
53     //cout<<s<<endl;
54     i=tp;
55     while(!a[i]) i--;//将整形数组表示的余数转化成字符串表示的
56     while(i>=0) v+=a[i--]+'0';
57     if(v.empty()) v="0";
58     //cout<<v<<endl;
59     if(nn==1) return s;//返回商
60     if(nn==2) return v;//返回余数
61 }
62 string div(string a,int b)//高精度a除以单精度b
63 {
64     string r,ans;
65     int d=0;
66     if(a=="0") return a;//特判
67     for(int i=0;i<a.size();i++)
68     {
69         r+=(d*10+a[i]-'0')/b+'0';//求出商
70         d=(d*10+(a[i]-'0'))%b;//求出余数
71     }
72     int p=0;
73     for(int i=0;i<r.size();i++)
74     if(r[i]!='0') {p=i;break;}
75     return r.substr(p);
76 }

```

## 1.6 高精度幂

使用了 FFT, 时复  $(n \log(n) \log(n))$

```

1 //高精度幂(nlog(n)log(n))
2 #define L(x) (1 << (x))
3 const double PI = acos(-1.0);
4 const int Maxn = 133015;
5 double ax[Maxn], ay[Maxn], bx[Maxn], by[Maxn];
6 char sa[Maxn/2],sb[Maxn/2];
7 int sum[Maxn];
8 int x1[Maxn],x2[Maxn];
9 int revv(int x, int bits)
10 {
11     int ret = 0;
12     for (int i = 0; i < bits; i++)
13     {
14         ret <= 1;
15         ret |= x & 1;
16         x >>= 1;
17     }
18     return ret;
19 }
20 void fft(double * a, double * b, int n, bool rev)
21 {
22

```

```

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;

```

```

89     string ans;
90     memset(sum, 0, sizeof(sum));
91     l1 = sa.size();
92     l2 = sb.size();
93     for(i = 0; i < l1; i++)
94         x1[i] = sa[l1 - i - 1] - '0';
95     for(i = 0; i < l2; i++)
96         x2[i] = sb[l2 - i - 1] - '0';
97     l = solve(x1, l1, x2, l2, sum);
98     for(i = 0; i < l || sum[i] >= 10; i++) // 进位
99     {
100         sum[i + 1] += sum[i] / 10;
101         sum[i] %= 10;
102     }
103     l = i;
104     while(sum[l] <= 0 && l > 0) l--; // 检索最高位
105     for(i = l; i >= 0; i--) ans += sum[i] + '0'; // 倒序输出
106     return ans;
107 }
108 string Pow(string a, int n)
109 {
110     if(n == 0) return 1;
111     if(n == 1) return a;
112     if(n & 1) return mul(Pow(a, n - 1), a);
113     string ans = Pow(a, n / 2);
114     return mul(ans, ans);
115 }

```

## 1.7 阶层

```

1
2 //阶层
3 string fac(int n)
4 {
5     const int L = 100005;
6     int a[L];
7     string ans;
8     if(n == 0) return "1";
9     fill(a, a + L, 0);
10    int s = 0, m = n;
11    while(m) a[++s] = m % 10, m /= 10;
12    for(int i = n - 1; i >= 2; i--)
13    {
14        int w = 0;
15        for(int j = 1; j <= s; j++) a[j] = a[j] * i + w, w = a[j] / 10, a[j] = a[j] % 10;
16        while(w) a[++s] = w % 10, w /= 10;
17    }
18    while(!a[s]) s--;
19    while(s >= 1) ans += a[s--] + '0';
20    return ans;
21 }

```

## 1.8 求最大公约数

```

1
2 //gcd
3 string add(string a, string b)
4 {
5     const int L = 1e5;
6     string ans;
7     int na[L] = {0}, nb[L] = {0};
8     int la = a.size(), lb = b.size();
9     for(int i = 0; i < la; i++) na[la - 1 - i] = a[i] - '0';

```

```

10     for(int i=0;i<lb;i++) nb[lb-1-i]=b[i]-'0';
11     int lmax=la>lb?la:lb;
12     for(int i=0;i<lmax;i++) na[i]+=nb[i],na[i+1]+=na[i]/10,na[i]%=10;
13     if(na[lmax]) lmax++;
14     for(int i=lmax-1;i>=0;i--) ans+=na[i]+'0';
15     return ans;
16 }
17 string mul(string a,string b)
18 {
19     const int L=1e5;
20     string s;
21     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
22     fill(na,na+L,0);fill(nb,nb+L,0);fill(nc,nc+L,0);//将na,nb,nc都置为0
23     for(int i=La-1;i>=0;i--) na[La-i]=a[i]-'0';//将字符串表示的大整数数组转成i整数数组表示的大整数
24     for(int i=Lb-1;i>=0;i--) nb[Lb-i]=b[i]-'0';
25     for(int i=1;i<=La;i++)
26         for(int j=1;j<=Lb;j++)
27             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)
28     for(int i=1;i<=La+Lb;i++)
29         nc[i+1]+=nc[i]/10,nc[i]%=10;//统一处理进位
30     if(nc[La+Lb]) s+=nc[La+Lb]+'0';//判断第i+j位上的数字是不是0
31     for(int i=La+Lb-1;i>=1;i--)
32         s+=nc[i]+'0';//将整数数组转成字符串
33     return s;
34 }
35 int sub(int *a,int *b,int La,int Lb)
36 {
37     if(La<Lb) return -1;//如果a小于b,则返回-1
38     if(La==Lb)
39     {
40         for(int i=La-1;i>=0;i--)
41             if(a[i]>b[i]) break;
42             else if(a[i]<b[i]) return -1;//如果a小于b,则返回-1
43     }
44     for(int i=0;i<La;i++)//高精度减法
45     {
46         a[i]-=b[i];
47         if(a[i]<0) a[i]+=10,a[i+1]--;
48     }
49     for(int i=La-1;i>=0;i--)
50         if(a[i]) return i+1;//返回差的位数
51     return 0;//返回差的位数
52 }
53 }
54 }
55 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
56 {
57     const int L=1e5;
58     string s,v;//s存商,v存余数
59     int a[L],b[L],r[L],La=n1.size(),Lb=n2.size(),i,tp=La;//a,b是整数数组表示被除数,除数,tp保存被除数的长度
60     fill(a,a+L,0);fill(b,b+L,0);fill(r,r+L,0);//数组元素都置为0
61     for(i=La-1;i>=0;i--) a[La-1-i]=n1[i]-'0';
62     for(i=Lb-1;i>=0;i--) b[Lb-1-i]=n2[i]-'0';
63     if(La<Lb || (La==Lb && n1<n2)) {
64         //cout<<0<<endl;
65         return n1;}//如果a<b,则商为0,余数为被除数
66     int t=La-Lb;//除被数和除数的位数之差
67     for(int i=La-1;i>=0;i--)//将除数扩大10^t倍
68         if(i>=t) b[i]=b[i-t];
69         else b[i]=0;
70     Lb=La;
71     for(int j=0;j<=t;j++)
72     {
73         int temp;
74         while((temp=sub(a,b+j,La,Lb-j))>=0)//如果被除数比除数大继续减

```



```

75     {
76         La=temp;
77         r[t-j]++;
78     }
79 }
80 for(i=0;i<L-10;i++) r[i+1]+=r[i]/10,r[i]%=10;//统一处理进位
81 while(!r[i]) i--;//将整形数组表示的商转化成字符串表示的
82 while(i>=0) s+=r[i--]+'0';
83 //cout<<s<<endl;
84 i=tp;
85 while(!a[i]) i--;//将整形数组表示的余数转化成字符串表示的</span>
86 while(i>=0) v+=a[i--]+'0';
87 if(v.empty()) v="0";
88 //cout<<v<<endl;
89 if(nn==1) return s;
90 if(nn==2) return v;
91 }
92 bool judge(string s)//判断s是否为全0串
93 {
94     for(int i=0;i<s.size();i++)
95         if(s[i]!='0') return false;
96     return true;
97 }
98 string gcd(string a,string b)//求最大公约数
99 {
100     string t;
101     while(!judge(b))//如果余数不为0，继续除
102     {
103         t=a;//保存被除数的值
104         a=b;//用除数替换被除数
105         b=div(t,b,2);//用余数替换除数
106     }
107     return a;
108 }

```

## 1.9 取模（高精）

```

1 //取模
2 int mod(string a,int b)//高精度a除以单精度b
3 {
4     int d=0;
5     for(int i=0;i<a.size();i++) d=(d*10+(a[i]-'0'))%b;//求出余数
6     return d;
7 }
8 }

```

## 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）

```

1 //进制转换
2 //将字符串表示的10进制大整数转换为m进制的大整数
3 //并返回m进制大整数的字符串
4 bool judge(string s)//判断串是否为全零串
5 {
6     for(int i=0;i<s.size();i++)
7         if(s[i]!='0') return 1;
8     return 0;
9 }
10 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
11 {
12     string r,ans;
13     int d=0;
14     if(!judge(s)) return "0";//特判

```

```

16 while(judge(s))//被除数不为0则继续
17 {
18     for(int i=0;i<s.size();i++)
19     {
20         r+=(d*n+s[i]-'0')/m+'0';//求出商
21         d=(d*n+(s[i]-'0'))%m;//求出余数
22     }
23     s=r;//把商赋给下一次的被除数
24     r="";//把商清空
25     ans+=d+'0';//加上进制转换后数字
26     d=0;//清空余数
27 }
28 reverse(ans.begin(),ans.end());//倒置下
29 return ans;
30 }

```

## 2 计算几何

### 2.1 基础模板

```

1 //点和线的表示
2 typedef long double LD;
3 struct PII{
4     int x,y;
5     bool operator<(const Point &o)const{
6         if(x==o.x) return y<o.y;
7         return x<o.x;
8     }
9 };
10 struct Point{
11     double x,y;
12     bool operator<(const Point &o)const{
13         if(x==o.x) return y<o.y;
14         return x<o.x;
15     }
16 };
17 struct Line{
18     Point st,ed;
19 };
20 //判正负
21 int sign(double a){
22     if(fabs(a)<=eps) return 0;
23     return a>0?1:-1;
24 }
25 //比较大小
26 int dcmp(double a,double b){
27     if(fabs(a-b)<eps) return 0;
28     return a>b?1:-1;
29 }

```

### 2.2 点积

```

1 double Dot(Point a, Point b) {
2     return a.x*b.x+a.y*b.y;
3 }

```

### 2.3 叉积

```

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

```

```
3 }
```

## 2.4 模长

```
1 double ABS(Point a){
2     return sqrt(a.x*a.x+a.y*a.y);
3 }
4 double norm(Point a){
5     return a.x*a.x+a.y*a.y;
6 }
```

## 2.5 两个向量是否同象限

```
1 bool same_quadrant(Point v,Point p) {
2     LD a=v.x, b=v.y, c=p.x, d=p.y;
3     int aa=sign(a), bb=sign(b);
4     int cc=sign(c), dd=sign(d);
5     return aa*cc>=0 && bb*dd>=0;
6 }
```

## 2.6 两向量是否共线

```
1 int dcmp(double x) {
2     if (fabs(x)<eps) return 0;
3     else if (x<0) return -1;
4     else return 1;
5 }
6 double cross(Point a,Point b){
7     return a.x*b.y-a.y*b.x;
8 }
9 bool on_line(Point a, Point b) { //a和b是否共线
10     return dcmp(cross(a,b))==0;
11 }
```

## 2.7 两向量是否垂直

```
1 double Dot(Point a, Point b) {
2     return a.x*b.x+a.y*b.y;
3 }
4 bool is_vertical(Point a, Point b) {
5     return dcmp(Dot(a,b))==0;
6 }
```

## 2.8 两个向量是否同方向

```
1 int same_direction(Point v, Point p) { //判断向量v和向量p是否共线且同向
2     if (on_line(v, p) && same_quadrant(v,p)) return 1; //同向
3     else if(on_line(v, p) && !same_quadrant(v,p)) return -1; //反向
4     return 0; //不共线
5 }
```

## 2.9 点旋转

```
1 //点逆时针旋转a度后的坐标
2 Point rotate1(Point p,double a){
3     return {p.x*cos(a)-p.y*sin(a),p.x*sin(a)+p.y*cos(a)};
4 }
5 //点顺时针旋转a度后的坐标
6 Point rotate2(Point p,double a){
7     return {p.x*cos(a)+p.y*sin(a),-p.x*sin(a)+p.y*cos(a)};
8 }
```

## 2.10 两点之间的距离

```
1 double getdis(Point a,Point b) {
2     return hypot(a.x-b.x,a.y-b.y);
3 }
```

## 2.11 直线

### 2.11.1 直线和线段是否相交

```
1 //直线a是否经过线段b
2 bool on_segment(Line a,Line b){
3     point q1=a.st,q2=a.ed;
4     if(sign(area(q1,q2,b.st))*sign(area(q1,q2,b.ed))>0) return 0;
5     return 1;
6 }
```

### 2.11.2 直线和直线的交点

```
1 //求两直线的交点
2 Point get_line_intersection(Point p, Point v, Point q, Point w){
3     Point u = p - q;
4     LD t = cross(w, u) / cross(v, w);
5     return {p.x + t * v.x, p.y + t * v.y};
6 }
7 Point get_line_intersection(Line& a, Line& b){
8     return get_line_intersection(a.st, a.ed - a.st, b.st, b.ed - b.st);
9 }
```

### 2.11.3 判断 b 和 c 的交点是否在直线右面

```
1 bool on_right(Line& a, Line& b, Line& c){
2     auto o = get_line_intersection(b, c);
3     return sign(area(a.st, a.ed, o)) < 0;
4 }
```

### 2.11.4 得到直线的角度

```
1 double get_angle(const Line &a){
2     return atan2(a.ed.y-a.st.y,a.ed.x-a.st.x);
3 }
```

### 2.11.5 求点关于直线的对称点

```
1 Point point_line(Line l, Point p) { //点p关于直线l的对称点
2     Point p1 = l.st;
3     Point p2 = l.ed;
4     double _x, _y;
5     if(p1.x - p2.x == 0) { //l斜率不存在
6         _x = 2 * p1.x - p.x;
7         _y = p.y;
8         return Point{_x, _y};
9     } else if(p1.y - p2.y == 0) { //l斜率为0
10        _x = p.x;
11        _y = 2 * p1.y - p.y;
12        return Point{_x, _y};
13    } else {
14        double k1 = (p1.y - p2.y) / (p1.x - p2.x);
15        double b1 = p1.y - k1 * p1.x;
16        double k2 = -1 / k1;
17        double b2 = p.y - k2 * p.x;
18        _x = (b2 - b1) / (k1 - k2);
19        _y = k2 * _x + b2;
20        return Point{2 * _x - p.x, 2 * _y - p.y};
21    }
22 }
```

### 2.11.6 点到直线的距离

```
1 double PLDis(Point a, Line s) { //点到直线的距离
2     double A=s.st.y-s.ed.y;
3     double B=s.ed.x-s.st.x;
4     double C=(s.st.x-s.ed.x)*s.st.y-(s.st.y-s.ed.y)*s.st.x;
5     return fabs(A*a.x+B*a.y+C)/sqrt((A*A+B*B));
6 }
```

### 2.11.7 点在直线的投影

```
1 //点p在直线s上的投影
2 Point Projection(Point p, Line s) {
3     Point alp=p-s.st;
4     Point beta=s.ed-s.st;
5     double res=Dot(alp,beta)/norm(beta); //norm(): 模长的平方
6     return s.st+(res*beta);
7 }
```

## 2.12 求多边形的面积

```
1 vector<Point> polygon;
2 double get_Area(vector<Point> polygon) {
3     double ans=0;
4     int n=(int)polygon.size();
5     for(int i=0; i<n; i++) {
6         ans+=cross(polygon[i], polygon[(i+1)%n]);
7     }
8     return fabs(ans/2);
9 }
```

## 2.13 线段

### 2.13.1 判断一个点是否在一个线段上

```

1 bool on_segment(Point p, Point a, Point b){
2     return sign(cross(p - a, p - b)) == 0 && sign(dot(p - a, p - b)) <= 0;
3 }

```

## 2.14 判断两个线段是否相交

```

1 bool segment_intersection(Point a1, Point a2, Point b1, Point b2){
2     double c1 = cross(a2 - a1, b1 - a1), c2 = cross(a2 - a1, b2 - a1);
3     double c3 = cross(b2 - b1, a2 - b1), c4 = cross(b2 - b1, a1 - b1);
4     return sign(c1) * sign(c2) <= 0 && sign(c3) * sign(c4) <= 0;
5 }

```

### 2.14.1 判断线和线段是否相交

```

1 bool line_segment_intersection(Line a, Line b){
2     if(on_line(a.p2-a.p1, b.p2-b.p1)){
3         if(on_line(a.p2-a.p1, b.p1-a.p1)) return 1;
4         else return 0;
5     }
6     Point o=get_line_intersection(a,b);
7     if(on_segment(o,b)) return 1;
8     else return 0;
9 }

```

## 2.15 圆

### 2.15.1 直线和圆的交点

```

1 int CCL(Line s, Point o, double r, Point &o1, Point &o2) {
2     Point x=Projection(o,s);
3     double dis=PLDis(o,s);
4     if(dis>r) { //距离>r没有交点
5         return 0;
6     }
7     if(dis==r) { //只有一个交点
8         o1=x;
9         return 1;
10    }
11    double beta=sqrt(r*r-dis*dis); //勾股定理
12    Point pp=s.ed-s.st;
13    pp=pp/pp.ABS(); //单位向量
14    Point ans1=x-beta*pp;
15    Point ans2=x+beta*pp;
16    o1=ans1;
17    o2=ans2;
18    return 2;
19 }

```

## 2.16 凸包

求包含所有点的最小周长多边形

```

1 //极角排序比较函数
2 bool cmp(PDD a, PDD b){
3     a=a-bas; b=b-bas;
4     double ag1=atan2(a.y,a.x), ag2=atan2(b.y,b.x);
5     if(ag1==ag2) a.x<b.x;
6     else return ag1<ag2;
7 }

```

```

8 }
9 //二维凸包
10 void get_convex(){
11     sort(q+1,q+1+n,cmp);
12     stk[++top]=q[1];
13     stk[++top]=q[2];
14     for(int i=3;i<=n;i++){
15         while(top>=2 && area(stk[top-1],stk[top],q[i])<=0) --top;
16         stk[++top]=q[i];
17     }
18     return ;
19 }

```

## 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

```

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 using namespace std;
6 typedef long double LD;
7 const LD eps=1e-10;
8 const int N=110;
9 const double PI=acos(-1);
10 double rand_eps(){
11     return ((double)rand()/RAND_MAX-0.5)*eps;
12 }
13 struct Point{
14     double x,y,z;
15     void shake(){ //微小扰动
16         x+=rand_eps();
17         y+=rand_eps();
18         z+=rand_eps();
19     }
20     Point operator - (Point a){
21         return {x-a.x,y-a.y,z-a.z};
22     }
23     Point operator + (Point a){
24         return {x+a.x,y+a.y,z+a.z};
25     }
26     Point operator * (Point a){ //向量叉乘
27         return {y * a.z - z * a.y, z * a.x - x * a.z, x * a.y - y * a.x};
28     }
29     double operator & (Point a){ //向量点积
30         return x*a.x+y*a.y+z*a.z;
31     }
32     double len(){ //向量模长
33         return sqrt(x*x+y*y+z*z);
34     }
35 }q[N];
36 struct Plane{
37     int v[3];
38     Point norm(){
39         return (q[v[1]]-q[v[0]])*(q[v[2]]-q[v[0]]); //返回法向量
40     }
41     double area(){
42         return norm().len()/2; //面积
43     }
44     bool above(Point a){
45         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
46     }
47 }pl[N],bk[N];
48 bool g[N][N];

```

```

49 int n,tot;
50 void get_convex(){
51     pl[++tot]={1,2,3}; //放进去前3个点组成的两个平面（正反）
52     pl[++tot]={1,3,2};
53     for(int i=4;i<=n;i++){
54         int cnt=0; //把更新后的平面放进去备份数组
55         for(int j=1;j<=tot;j++){
56             bool flag=pl[j].above(q[i]); //表示q[i]照到了第j个平面
57             if(!flag) bk[++cnt]=pl[j]; //没有照到
58             for(int k=0;k<3;k++){
59                 g[pl[j].v[k]][pl[j].v[(k+1)%3]]=flag; //标记每一条边是否照到
60             }
61         }
62         for(int j=1;j<=tot;j++){
63             for(int k=0;k<3;k++){
64                 int a=pl[j].v[k],b=pl[j].v[(k+1)%3]; //两点的编号
65                 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
66             }
67         }
68         tot=cnt; //更新面的数量
69         for(int i=1;i<=tot;i++) pl[i]=bk[i]; //更新所有的面
70     }
71 }
72 int main()
73 {
74     ios;
75     cin>>n;
76     for(int i=1;i<=n;i++){
77         cin>>q[i].x>>q[i].y>>q[i].z;
78         q[i].shake();
79     }
80     get_convex();
81     double ans=0;
82     for(int i=1;i<=tot;i++){
83         ans+=pl[i].area();
84     }
85     cout<<fixed<<setprecision(6)<<ans<<endl;
86
87     return 0;
88 }

```

## 2.18 半平面交

求包含所有点的最小周长多边形

```

1
2 //极角排序比较函数
3 bool cmp(PDD a,PDD b){
4     a=a-bas; b=b-bas;
5     double ag1=atan2(a.y,a.x),ag2=atan2(b.y,b.x);
6     if(ag1==ag2) a.x<b.x;
7     else return ag1<ag2;
8 }
9 //二维凸包
10 void get_convex(){
11     sort(q+1,q+1+n,cmp);
12     stk[++top]=q[1];
13     stk[++top]=q[2];
14     for(int i=3;i<=n;i++){
15         while(top>=2 && area(stk[top-1],stk[top],q[i])<=0) --top;
16         stk[++top]=q[i];
17     }
18     return ;
19 }

```



## 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);
```

```

65     auto p=get_line_intersection(u.x,u.y,v.x,v.y);
66     return {p,getdis(p,a)};
67 }
68 PDD q[N];
69 int n;
70 Circle get_Circle() {
71     random_shuffle(q+1,q+1+n);
72     Circle c;
73     c.p=q[1];
74     c.r=0;
75     for(int i=2; i<=n; i++) {
76         if(fcmp(c.r,getdis(c.p,q[i]))<0) {
77             c= {q[i],0};
78             for(int j=1; j<i; j++) {
79                 if(fcmp(c.r,getdis(c.p,q[j]))<0) {
80                     c= {(q[i]+q[j])/2,getdis(q[i],q[j])/2};
81                     for(int k=1; k<j; k++) {
82                         if(fcmp(c.r,getdis(c.p,q[k]))<0) {
83                             c=get_circle(q[i],q[j],q[k]);
84                         }
85                     }
86                 }
87             }
88         }
89     }
90     return c;
91 }
92
93 int main() {
94     ios;
95     cin>>n;
96     for(int i=1; i<=n; i++) cin>>q[i].x>>q[i].y;
97     Circle c;
98     c=get_Circle();
99     cout<<fixed<<setprecision(10)<<c.r<<endl;
100    cout<<fixed<<setprecision(10)<<c.p.x<<" "<<c.p.y<<endl;
101    return 0;
102 }

```

## 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

```

1
2 int rmi[N],rmx[N];
3 void largest(int *arr,int n) {
4     stack<int> stmin,stmax;
5     for(int i=1;i<=n;i++){
6         while(!stmin.empty() && arr[i]<arr[stmin.top()]){
7             rmi[stmin.top()]=i-1;
8             stmin.pop();
9         }
10        stmin.push(i);
11        while(!stmax.empty() && arr[i]>arr[stmax.top()]) {
12            rmx[stmax.top()]=i-1;
13            stmax.pop();
14        }
15        stmax.push(i);
16    }
17    while(!stmin.empty()) {
18        rmi[stmin.top()]=n;
19        stmin.pop();

```

```

20     }
21     while(!stmax.empty()) {
22         rmx[stmax.top()]=n;
23         stmax.pop();
24     }
25     return ;
26 }

```

### 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

```

1
2 int largestRectangleArea(vector<int>& heights) {
3     heights.push_back(0);
4     stack<int> st;
5     int sz=heights.size();
6     int ans=0;
7     for(int i=0; i<sz; i++) {
8         while(!st.empty() && heights[st.top()]>heights[i]) {
9             int cur=heights[st.top()];
10            st.pop();
11            if(st.empty()) ans=max(ans,i*cur); //这里记得特判
12            else ans=max(ans,(i-st.top()-1)*cur);
13        }
14        st.push(i);
15    }
16    return ans;
17 }

```

## 3.2 单调队列

### 3.2.1 区间 (长度固定) 最值问题

```

1
2 int mi[N],mx[N];
3 void GET(int *arr,int n,int k){
4     deque<int> dqmin,dqmax;
5     for(int i=1;i<=n;i++){
6         while(!dqmax.empty() && arr[i]>=arr[dqmax.back()]) dqmax.pop_back();
7         dqmax.push_back(i);
8         while(!dqmin.empty() && arr[i]<=arr[dqmin.back()]) dqmin.pop_back();
9         dqmin.push_back(i);
10        if(i>=k){
11            mx[i-k+1]=arr[dqmax.front()];
12            mi[i-k+1]=arr[dqmin.front()];
13        }
14        if(i-dqmax.front()+1>=k) dqmax.pop_front();
15        if(i-dqmin.front()+1>=k) dqmin.pop_front();
16    }
17    return ;
18 }

```

### 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  $\leq k$  的所有子区间个数

```

1
2 ll GETNUM(vector<ll> nums,int k){
3     deque<ll> dqmin,dqmax;

```

```

4    11 sz=nums.size(),j=0,ans=0;
5    for(int i=0;i<sz;i++){
6        while(j<sz){
7            //维护从大到小的deque
8            while(!dqmax.empty() && nums[j]>=nums[dqmax.back()]) dqmax.pop_back();
9            dqmax.push_back(j);
10           //维护从小到大的deque
11           while(!dqmin.empty() && nums[j]<=nums[dqmin.back()]) dqmin.pop_back();
12           dqmin.push_back(j);
13           //不满足条件就break出去
14           if(nums[dqmax.front()]-nums[dqmin.front()]>k) break;
15           j++;
16       }
17       //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
18       if(dqmax.front()==i) dqmax.pop_front();
19       if(dqmin.front()==i) dqmin.pop_front();
20       //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
21       ans+=j-i;
22   }
23   return ans;
24 }

```

### 3.3 并查集

#### 3.3.1 按秩合并

```

1
2 void remerge(int a,int b,int c) {
3     int x=find(a), y=find(b);
4     if(x==y) return ;
5     if(dep[x]<dep[y]) fa[x]=y;
6     else {
7         fa[y]=x;
8         if(dep[x]==dep[y]) dep[x]++;
9     }
10 }

```

#### 3.3.2 关押罪犯

```

1
2 #include <bits/stdc++.h>
3 #define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
4 using namespace std;
5 const int N=1e6+100;
6 struct node{
7     int u,v,w;
8 }arr[100100];
9 int n,m;
10 int fa[N*2];
11 int find(int x){
12     if(x==fa[x]) return x;
13     else return fa[x]=find(fa[x]);
14 }
15 void remerge(int x,int y){
16     int fx=find(x),fy=find(y);
17     if(fx!=fy) fa[fx]=fy;
18 }
19 bool cmp(node a,node b){
20     return a.w>b.w;
21 }
22 int main(){
23     ios;
24     cin>>n>>m;

```

```

25     for(int i=1;i<=m;i++) cin>>arr[i].u>>arr[i].v>>arr[i].w;
26     sort(arr+1,arr+1+m,cmp);
27     for(int i=1;i<=2*n;i++) fa[i]=i;
28     for(int i=1;i<=m;i++){
29         int x=arr[i].u,y=arr[i].v;
30         if(find(x)==find(y) || find(x+n)==find(y+n)){
31             cout<<arr[i].w<<'\n';
32             return 0;
33         }
34         remerge(x+n,y);
35         remerge(x,y+n);
36     }
37     cout<<0<<'\n';
38     return 0;
39 }

```

### 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

```

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 using namespace std;
6 const int N=5e4+100;
7 int fa[N],dis[N];
8 int n,k;
9 int find(int x){
10     if(x==fa[x]) return x;
11     else{
12         int tmp=fa[x];
13         fa[x]=find(fa[x]);
14         dis[x]=(dis[x]+dis[tmp])%3;
15         return fa[x];
16     }
17 }
18
19 int main()
20 {
21     ios;
22     cin>>n>>k;
23     for(int i=1;i<=n;i++) fa[i]=i;
24     int ans=0;
25     while(k--){
26         int op,x,y;
27         cin>>op>>x>>y;
28         if(x<1 || x>n || y<1 || y>n) {
29             ans++;
30             continue;
31         }
32         if(op==1){
33             int fx=find(x), fy=find(y);
34             if(fx==fy){
35                 if((dis[y]-dis[x]+3)%3!=0) ans++;
36             }
37             else{
38                 dis[fx]=(dis[y]-dis[x]+3)%3;
39                 fa[fx]=fy;
40             }
41         }
42         else{
43             int fx=find(x), fy=find(y);
44             if(fx==fy){

```

```

45         if((dis[y]-dis[x]+3)%3!=1) ans++;
46     }
47     else{
48         dis[fx]=(dis[y]-dis[x]-1+3)%3;
49         fa[fx]=fy;
50     }
51 }
52 }
53 cout<<ans<<endl;
54 return 0;
55 }

```

### 3.4 AC 自动机

```

1
2 struct AC{
3     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
4     int tot; // 点编号
5     void insert(string s){ // 插入模式串, 构建字典树
6         int p=0,len=s.size();
7         for(int i=0;i<len;i++){
8             if(!tr[p][s[i]-'a']) tr[p][s[i]-'a']+=++tot;
9             p=tr[p][s[i]-'a'];
10        }
11        num[p]++; // 数量加一
12        return ;
13    }
14    queue<int> q;
15    void build(){
16        for(int i=0;i<26;i++){
17            if(tr[0][i]) q.push(tr[0][i]);
18        }
19        while(!q.empty()){
20            int u=q.front();
21            q.pop();
22            for(int i=0;i<26;i++){
23                if(tr[u][i]){
24                    fail[tr[u][i]]=tr[fail[u]][i];
25                    q.push(tr[u][i]);
26                }
27                else tr[u][i]=tr[fail[u]][i]; // 压缩路径
28            }
29        }
30        return ;
31    }
32    int query(string s){
33        int len=s.size(),u=0,ret=0;
34        for(int i=0;i<len;i++){
35            u=tr[u][s[i]-'a'];
36            for(int j=u;j && num[j]!=-1;j=fail[j]){
37                // 每一个字符都当成失配字符算一遍匹配模式串数量
38                ret+=num[j];
39                num[j]=-1; // 一个模式串被计算过了就要标记一下
40            }
41        }
42        return ret;
43    }
44 };
45 AC tree;

```

### 3.5 树状数组

#### 3.5.1 单点修改, 区间查询

```

1
2 int lowbit(int x){
3     return x&-x;
4 }
5 void add(int i) {
6     while(i<=n) {
7         c[i]++;
8         i+=lowbit(i);
9     }
10 }
11 ll getsum(int i) {
12     long long res=0;
13     while(i>0) {
14         res+=c[i];
15         i-=lowbit(i);
16     }
17     return res;
18 }

```

### 3.5.2 树状数组求逆序数

```

1
2 #include <bits/stdc++.h>
3 using namespace std;
4 const int N=500001;
5 typedef long long ll;
6 int c[N];
7 struct Node {
8     int v,id;
9     bool operator < (const Node &b) const {
10         return v<b.v; //从小到大排序
11     }
12 } arr[N];
13 int n;
14 int lowbit(int x){
15     return x&-x;
16 }
17 void add(int i) {
18     while(i<=n) {
19         c[i]++;
20         i+=lowbit(i);
21     }
22 }
23 ll getsum(int i) {
24     long long res=0;
25     while(i>0) {
26         res+=c[i];
27         i-=lowbit(i);
28     }
29     return res;
30 }
31
32 int main() {
33     while(1) {
34         cin>>n;
35         if(n==0) break;
36         int a;
37         memset(arr,0,sizeof arr);
38         memset(c,0,sizeof c);
39         for(int i=1; i<=n; i++) {
40             scanf("%d",&a);
41             arr[i].id=i;
42             arr[i].v=a;
43         }

```

```

44     sort(arr+1,arr+1+n);
45     ll ans=0;
46     for(int i=1; i<=n; i++) {
47         add(arr[i].id); //离散化结果—— 下标等效于数值
48         ans+=i-getsum(arr[i].id); //得到之前有多少个比你大的数（逆序对）
49     }
50     cout<<ans<<endl;
51 }
52 return 0;
53 }

```

## 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

```

1
2 #define ls u<<1
3 #define rs u<<1|1
4 struct node{
5     int l,r;
6     int sum;
7 }tr[N<<2];
8 int lazy[N<<2];
9 void pushup(int u){
10     tr[u].sum=tr[ls].sum+tr[rs].sum;
11 }
12 void pushdown(int u){
13     if(!lazy[u]) return ;
14     tr[ls].sum+=(tr[ls].r-tr[ls].l+1)*lazy[u];
15     tr[rs].sum+=(tr[rs].r-tr[rs].l+1)*lazy[u];
16     lazy[ls]+=lazy[u];
17     lazy[rs]+=lazy[u];
18     lazy[u]=0; //加法初始化为0
19 // lazy[u]=1; //乘法初始化为1
20 }
21 void build(int u,int l,int r){
22     lazy[u]=0; //初始化标记
23     if(l==r) tr[u]={l,r,0};
24     else{
25         tr[u]={l,r};
26         int mid=(l+r)>>1;
27         build(ls,l,mid);
28         build(rs,mid+1,r);
29         pushup(u);
30     }
31 }
32 void update(int u,int l,int r,int c){
33     if(tr[u].l>=l && tr[u].r<=r){
34         tr[u].sum+=(tr[u].r-tr[u].l+1)*c; //区间值加上c
35         lazy[u]+=c; //更新加法标记
36     }
37     else{
38         pushdown(u);
39         int mid=(tr[u].l+tr[u].r)>>1;
40         if(l<=mid) update(ls,l,r,c);
41         if(r>mid) update(rs,l,r,c);
42         pushup(u);
43     }
44 }
45 int query(int u,int l,int r){
46     if(tr[u].l>=l && tr[u].r<=r) return tr[u].sum;
47     else{
48         pushdown(u);

```



```

49     int mid=(tr[u].r+tr[u].l)>>1;
50     int res=0;
51     if(l<=mid) res=res+query(ls,l,r);
52     if(r>mid) res=res+query(rs,l,r);
53     return res;
54 }
55 }

```

### 3.6.2 区间维护最大连续子段和

```

1
2 #include <bits/stdc++.h>
3 #define endl '\n'
4 #define ls u<<1
5 #define rs u<<1|1
6 #define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
7 using namespace std;
8 const int N=500100;
9 struct node{
10     int l,r;
11     int lmx,rmx;
12     int sum,ms;
13 }tr[N<<2];
14 int arr[N];
15 void pushup(int u){
16     tr[u].sum=tr[ls].sum+tr[rs].sum;
17     tr[u].lmx=max(tr[ls].lmx,tr[ls].sum+tr[rs].lmx);
18     tr[u].rmx=max(tr[rs].rmx,tr[rs].sum+tr[ls].rmx);
19     tr[u].ms=max(tr[ls].ms,tr[rs].ms);
20     tr[u].ms=max(tr[ls].rmx+tr[rs].lmx,tr[u].ms);
21 }
22 void build(int u,int l,int r){
23     if(l==r){
24         tr[u]={l,r,arr[l],arr[l],arr[l],arr[l]};
25     }
26     else{
27         int mid=l+r>>1;
28         tr[u]={l,r};
29         build(u<<1,l,mid);
30         build(u<<1|1,mid+1,r);
31         pushup(u);
32     }
33 }
34 void update(int u,int pos,int c){
35     if(tr[u].l==tr[u].r && tr[u].l==pos){
36         tr[u].ms=c;
37         tr[u].lmx=c;
38         tr[u].rmx=c;
39         tr[u].sum=c;
40         return ;
41     }
42     else{
43         int mid=tr[u].l+tr[u].r>>1;
44         if(pos<=mid) update(ls,pos,c);
45         else update(rs,pos,c);
46         pushup(u);
47     }
48 }
49 node query(int u,int l,int r)
50 {
51     if(tr[u].l>=l&&tr[u].r<=r)return tr[u];
52     else {
53         int mid=(tr[u].l+tr[u].r)>>1;
54         if(r<=mid)return query(u<<1,l,r);

```

```

55     else if(l>mid)return query(u<<1|1,l,r);
56     else
57     {
58         node a=query(u<<1,l,r),b=query(u<<1|1,l,r);
59         node c;
60         c.sum=a.sum+b.sum;
61         c.lmx=max(a.lmx,a.sum+b.lmx);
62         c.rmx=max(b.rmx,a.rmx+b.sum);
63         c.ms=max(a.rmx+b.lmx,max(a.ms,b.ms));
64         return c;
65     }
66 }
67 }
68 int m,n;
69 signed main()
70 {
71     ios;
72     cin>>n>>m;
73     for(int i=1;i<=n;i++) cin>>arr[i];
74     build(1,1,n);
75     while(m--){
76         int k,x,y;
77         cin>>k>>x>>y;
78         if(k==1){
79             if(x>y) swap(x,y);
80             node tmp=query(1,x,y);
81             cout<<tmp.ms<<endl;
82         }
83         else{
84             update(1,x,y);
85         }
86     }
87
88     return 0;
89 }

```

### 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,

$$\gcd(a, b, c, d \dots) = \gcd(a, b - a, c - b, d - c \dots)$$

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  $r+1 \leq n$ , 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

```

1
2 #include<bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 const int maxn = 5e5 + 23;
6 struct Node {
7     int l, r;
8     ll v, d;
9 } tr[maxn * 4];
10 ll a[maxn], b[maxn];
11 void pushup(Node &u, Node &l, Node &r) {
12     u.v = l.v + r.v;
13     u.d = __gcd(l.d, r.d);
14 }
15 void pushup(int u) {
16     pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
17 }
18 void build(int u, int l, int r) {

```

```

19     if(l == r) tr[u] = {l, r, b[l], b[l]};
20     else {
21         tr[u] = {l, r};
22         int mid = l + r >> 1;
23         build(u << 1, l, mid);
24         build(u << 1 | 1, mid + 1, r);
25         pushup(u);
26     }
27 }
28 ll query(int u, int l, int r) { //区间gcd
29     if(tr[u].l >= l && tr[u].r <= r) return tr[u].d;
30     else {
31         int mid = tr[u].l + tr[u].r >> 1;
32         if(r <= mid) return query(u << 1, l, r);
33         else if(l > mid) return query(u << 1 | 1, l, r);
34         else return __gcd(query(u << 1, l, r), query(u << 1 | 1, l, r));
35     }
36 }
37 ll query2(int u, int l, int r) { //区间和
38     if(tr[u].l >= l && tr[u].r <= r) return tr[u].v;
39     else {
40         int mid = tr[u].l + tr[u].r >> 1;
41         if(r <= mid) return query2(u << 1, l, r);
42         else if(l > mid) return query2(u << 1 | 1, l, r);
43         else return query2(u << 1, l, r) + query2(u << 1 | 1, l, r);
44     }
45 }
46 void modify(int u, int p, ll v) {
47     if(tr[u].l == tr[u].r && tr[u].l == p) tr[u].d += v, tr[u].v += v;
48     else {
49         int mid = tr[u].l + tr[u].r >> 1;
50         if(p <= mid) modify(u << 1, p, v);
51         else modify(u << 1 | 1, p, v);
52         pushup(u);
53     }
54 }
55 int main() {
56     int n, m;
57     scanf("%d%d", &n, &m);
58     for(int i = 1; i <= n; i++) scanf("%lld", &a[i]), b[i] = a[i] - a[i - 1];
59     build(1, 1, n);
60     char op[4];
61     ll op1 = 0, op2 = 0, op3 = 0;
62     while(m--) {
63         scanf("%s%lld%lld", op, &op1, &op2);
64         if(*op == 'C') {
65             scanf("%lld", &op3);
66             modify(1, op1, op3);
67             if(op2 + 1 <= n) modify(1, op2 + 1, -op3);
68         } else {
69             ll t = query2(1, 1, op1);
70             printf("%lld\n", abs( __gcd(t, query(1, op1 + 1, op2))));
71         }
72     }
73 }

```

### 3.6.4 区间同时维护乘法和加法

注意 lazy 标记的初始化

```

1
2 #include <bits/stdc++.h>
3 #define ios ios::sync_with_stdio(0); cin.tie(0); cout.tie(0)
4 using namespace std;
5 typedef long long ll;

```

```

6  const ll MOD=1e9+7;
7  const ll SUP=0x800000;
8  const ll MAXN=1e5+10;
9  const ll INF=0x3f3f3f3f;
10 const double eps=1e-4;
11 struct node{
12     ll l,r,val;
13 }tr[MAXN<<2];
14 //lazy1加法标记 lazy2乘法标记
15 ll lazy1[MAXN<<2],lazy2[MAXN<<2],arr[MAXN];
16 ll n,m,p;
17 void pushup(ll u){
18     tr[u].val=(tr[u<<1].val+tr[u<<1|1].val)%p;
19 }
20 void pushdown(ll u){
21     ll l=u<<1,r=u<<1|1;
22     // 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
23     tr[l].val=(tr[l].val*lazy2[u]%p+(tr[l].r-tr[l].l+1)*lazy1[u]%p)%p;
24     tr[r].val=(tr[r].val*lazy2[u]%p+(tr[r].r-tr[r].l+1)*lazy1[u]%p)%p;
25     // 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
26     lazy1[l]=(lazy1[l]*lazy2[u]%p+lazy1[u])%p;
27     lazy1[r]=(lazy1[r]*lazy2[u]%p+lazy1[u])%p;
28     // 儿子乘法标记=儿子乘法标记*父亲乘法标记
29     lazy2[l]=lazy2[l]*lazy2[u]%p;
30     lazy2[r]=lazy2[r]*lazy2[u]%p;
31
32     lazy1[u]=0;
33     lazy2[u]=1;
34 }
35 void build(ll u,ll l,ll r){
36     lazy1[u]=0; //初始化标记
37     lazy2[u]=1;
38     if(l==r) tr[u]={l,r,arr[l]};
39     else{
40         tr[u]={l,r};
41         ll mid=(l+r)>>1;
42         build(u<<1,l,mid);
43         build(u<<1|1,mid+1,r);
44         pushup(u);
45     }
46 }
47 void add(ll u,ll l,ll r,ll c){
48     if(tr[u].l>=l && tr[u].r<=r){
49         tr[u].val=(tr[u].val+(tr[u].r-tr[u].l+1)*c)%p; //区间值加上c
50         lazy1[u]=(c+lazy1[u])%p; //更新加法标记
51     }
52     else{
53         pushdown(u);
54         ll mid=(tr[u].l+tr[u].r)>>1;
55         if(l<=mid) add(u<<1,l,r,c);
56         if(r>mid) add(u<<1|1,l,r,c);
57         pushup(u);
58     }
59 }
60 void mul(ll u,ll l,ll r,ll c){
61     if(tr[u].l>=l && tr[u].r<=r){
62         tr[u].val=tr[u].val*c%p; //区间的值乘上c
63         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
64         lazy2[u]=(lazy2[u]*c)%p; //更新乘法标记
65     }
66     else{
67         pushdown(u);
68         ll mid=(tr[u].l+tr[u].r)>>1;
69         if(l<=mid) mul(u<<1,l,r,c);
70         if(r>mid) mul(u<<1|1,l,r,c);
71         pushup(u);

```

```

72     }
73 }
74 ll query(ll u,ll l,ll r){
75     if(tr[u].l>=l && tr[u].r<=r) return tr[u].val;
76     else{
77         pushdown(u);
78         ll mid=(tr[u].r+tr[u].l)>>1,ret=0;
79         if(l<=mid) ret=(ret+query(u<<1,l,r))%p;
80         if(r>mid) ret=(ret+query(u<<1|1,l,r))%p;
81         return ret;
82     }
83 }
84 int main()
85 {
86     ios;
87     cin>>n>>m>>p;
88     for(ll i=1;i<=n;i++) cin>>arr[i];
89     build(1,1,n);
90     while(m--){
91         ll op,x,y,k;
92         cin>>op;
93         if(op==1){
94             cin>>x>>y>>k;
95             mul(1,x,y,k);
96         }
97         else if(op==2){
98             cin>>x>>y>>k;
99             add(1,x,y,k);
100        }
101        else{
102            cin>>x>>y;
103            cout<<query(1,x,y)<<'\n';
104        }
105    }
106    return 0;
107 }

```

### 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( $x \leq 10$ )：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  $x \leq 10$ ，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。

```

1  #include <bits/stdc++.h>
2  #define ls u<<1
3  #define rs u<<1|1
4  #define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
5  #define endl '\n'
6  using namespace std;
7  const int N=1e5+100;
8  int n,q;
9  struct node{
10     int l,r;
11     int sum,mx;
12 }tr[10][N<<2];
13 int lazy[10][N<<2];
14 void pushup(int id,int u){
15     tr[id][u].sum=tr[id][ls].sum+tr[id][rs].sum;
16     tr[id][u].mx=max(tr[id][ls].mx,tr[id][rs].mx);
17 }

```

```

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                 }

```

```

84         update(mp[i],1,l,r,cnt);
85     }
86 }
87 if(x>1) update(mp[x],1,l,r,1);
88 }
89 else{
90     int l,r;
91     cin>>l>>r;
92     int ans=0;
93     for(int i=1;i<=4;i++){
94         int tmp=query(i,1,l,r);
95         ans=max(ans,tmp);
96     }
97     cout<<"ANSWER "<<ans<<endl;
98 }
99 }
100 return 0;
101 }

```

### 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

```

1
2 #include <bits/stdc++.h>
3 #define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
4 using namespace std;
5 const int N=1e6+100;
6 struct node {
7     int ls,rs,sum;
8 } tr[N*40];
9 int rt[N*40];
10 int arr[N],brr[N];
11 int n,m,tot;
12 int build(int l,int r) { //新建一颗空树
13     int root=++tot;
14     if(l==r) return root;
15     int mid=l+r>>1;
16     tr[root].ls=build(l,mid);
17     tr[root].rs=build(mid+1,r);
18     return root;
19 }
20 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
21     int root=++tot;
22     tr[root]=tr[pre];
23     tr[root].sum++;
24     if(l==r) return root;
25     int mid=(l+r)>>1;
26     if(pos<=mid) tr[root].ls=insert(tr[pre].ls,l,mid,pos);
27     else tr[root].rs=insert(tr[pre].rs,mid+1,r,pos);
28     return root;
29 }
30 int query_k_min(int u,int v,int l,int r,int k) { //查询第k小
31     if(l==r) return l;
32     int cha=tr[tr[v].ls].sum-tr[tr[u].ls].sum;
33     int mid=l+r>>1;
34     if(cha>=k) return query_k_min(tr[u].ls,tr[v].ls,l,mid,k);
35     else return query_k_min(tr[u].rs,tr[v].rs,mid+1,r,k-cha);
36 }
37 int query_k_max(int u,int v,int l,int r,int k) { //查询第k大
38     if(l==r) return l;
39     int cha=tr[tr[v].rs].sum-tr[tr[u].rs].sum;
40     int mid=l+r>>1;
41     if(cha>=k) return query_k_max(tr[u].rs,tr[v].rs,l,mid,k);

```

```

42     else return query_k_max(tr[u].ls,tr[v].ls,mid+1,r,k-cha);
43 }
44 int main() {
45     ios;
46     cin>>n>>m;
47     for(int i=1; i<=n; i++) {
48         cin>>arr[i];
49         brr[i]=arr[i];
50     }
51     sort(arr+1,arr+1+n); //离散化, brr存离散值, arr原值
52     for(int i=1; i<=n; i++) brr[i]=lower_bound(arr+1,arr+1+n,brr[i])-arr;
53
54     rt[0]=build(1,n);
55     for(int i=1; i<=n; i++) rt[i]=insert(rt[i-1],1,n,brr[i]);
56
57     while(m--) {
58         int l,r,k;
59         cin>>l>>r>>k;
60         cout<<arr[query_k_min(rt[l-1],rt[r],1,n,k)]<<'\n';
61     }
62     return 0;
63 }

```

### 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）

```

1
2 int query_sum(int u,int v,int l,int r,int ql,int qr){ //查询区间和
3     if(l>=ql && r<=qr) return tr[v].sum-tr[u].sum;
4     else{
5         int mid=l+r>>1;
6         int res=0;
7         if(ql<=mid) res+=query_sum(tr[u].ls,tr[v].ls,l,mid,ql,qr);
8         if(qr>mid) res+=query_sum(tr[u].rs,tr[v].rs,mid+1,r,ql,qr);
9         return res;
10    }
11 }

```

## 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

```

1 #include<bits/stdc++.h>
2 #define endl '\n'
3 #define ios ios::sync_with_stdio(false),cin.tie(0),cout.tie(0)
4 using namespace std;
5 const int N=1e5+100;
6
7 //链式前向星存图
8 struct Edge{
9     int to,next;
10 }e[1000100];
11 int h[N],tot;
12 void add(int u,int v){
13     e[tot]={v,h[u]};
14     h[u]=tot++;
15 }

```



```

15 }
16
17 /*
18 dfn: 时间戳
19 siz: 联通块内部点数量
20 idx: 点所属联通块编号
21 instk: 是否在栈中
22 */
23 int dfn[N], low[N], siz[N], idx[N];
24 bool instk[N];
25 int tim, cnt; //时间戳, 联通块数量
26 stack<int> stk;
27 void tarjan(int x){ //缩点
28     dfn[x]=low[x]=++tim;
29     stk.push(x);
30     instk[x]=1;
31     for(int i=h[x]; ~i; i=e[i].next){
32         int v=e[i].to;
33         if(!dfn[v]){
34             tarjan(v);
35             low[x]=min(low[v], low[x]);
36         }
37         else if(instk[v]) low[x]=min(low[x], dfn[v]);
38     }
39     if(low[x]==dfn[x]){
40         cnt++;
41         while(stk.top()!=x){
42             siz[cnt]++;
43             idx[stk.top()]=cnt;
44             instk[stk.top()]=0;
45             stk.pop();
46         }
47         siz[cnt]++;
48         idx[stk.top()]=cnt;
49         instk[stk.top()]=0;
50         stk.pop();
51     }
52 }
53 /*
54 G是缩点后的图
55 in: 入度
56 mi: 联通块内部点权最小
57 mx: 联通块内部点权最大
58 */
59 vector<int> G[N];
60 int in[N], mi[N], mx[N];
61 void tupo(){
62     queue<int> q;
63     //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
64     for(int i=1; i<=cnt; i++){
65         if(!in[i] && i!=idx[1]) q.push(i);
66     }
67     while(!q.empty()){
68         int u=q.front();
69         q.pop();
70         for(auto v:G[u]){
71             if(v!=idx[1]) in[v]--;
72             if(!in[v]) q.push(v);
73         }
74     }
75     //到这里结束
76     q.push(idx[1]);
77     while(!q.empty()){
78         int u=q.front();
79         q.pop();
80         for(auto v:G[u]){

```

```

81         /*
82         mi[v]=min(mi[v],mi[u]);
83         ans[v]=max(ans[v],mx[v]-mi[v]);
84         ans[v]=max(ans[v],ans[u]);
85         */
86         in[v]--;
87         if(!in[v]) q.push(v);
88     }
89 }
90 }
91
92 int n,m;
93 int num[N];
94 int main() {
95     ios;
96     memset(h,-1,sizeof h); //记得初始化
97     cin>>n>>m;
98     for(int i=1;i<=n;i++) cin>>num[i];
99     while(m--){
100         int u,v,op;
101         cin>>u>>v>>op;
102         add(u,v);
103         if(op==2) add(v,u);
104     }
105     memset(mi,0x3f,sizeof mi);
106     //图可能不连通，所以需要遍历所有点
107     for(int i=1;i<=n;i++){
108         if(!dfn[i]) tarjan(i);
109     }
110     //每一个点去更新他所在联通块的最值
111     for(int i=1;i<=n;i++){
112         mi[idx[i]]=min(mi[idx[i]],num[i]);
113         mx[idx[i]]=max(mx[idx[i]],num[i]);
114     }
115     //建新图
116     for(int i=1;i<=n;i++){
117         for(int j=h[i];~j;j=e[j].next){
118             int v=e[j].to;
119             if(idx[v]!=idx[i]) G[idx[i]].push_back(idx[v]);
120         }
121     }
122     //新图入度
123     for(int i=1;i<=cnt;i++){
124         for(auto v:G[i]){
125             in[v]++;
126         }
127     }
128     //拓扑
129     tupo();
130     cout<<ans[idx[n]]<<endl;
131     return 0;
132 }
133 /*
134 6 6
135 2 6 2 1 8 9
136 1 4 1
137 4 5 1
138 5 1 1
139 1 2 1
140 2 3 1
141 2 6 1
142 */

```

## 4.2 floyed 求最小环

```

1  for(int k=1;k<=n;k++){
2      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
3          for(int j=i+1;j<k;j++){
4              ans=min(ans,dis[i][j]+G[j][k]+G[k][i]);
5          }
6      }
7      for(int i=1;i<=n;i++){
8          for(int j=1;j<=n;j++){
9              dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
10         }
11     }
12 }

```

## 5 数论

## 6 杂类

### 6.1 int128

```

1  void read(int128 &x) {
2      x = 0;
3      int f = 1;
4      char ch;
5      if((ch = getchar()) == '-') f = -f;
6      else x = x*10 + ch-'0';
7      while((ch = getchar()) >= '0' && ch <= '9')
8          x = x*10 + ch-'0';
9      x *= f;
10 }
11 void print(int128 x) { //输出
12     if(x < 0) {
13         x = -x;
14         putchar('-');
15     }
16     if(x > 9) print(x/10);
17     putchar(x%10 + '0');
18 }

```

### 6.2 O(3) 优化

```

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

```

### 6.3 模拟退火

```

1  //随机数
2  srand(time(NULL));
3  int rand_INT(int l,int r) { //产生[l,r]的一个随机数
4      return rand()%(r-l+1)+l;
5  }
6  double rand_DOUBLE(double l,double r){ //产生[l,r]的一个随机数
7      return (double)rand()/RAND_MAX*(r-l)+l;
8  }
9  //以Acwing进阶课题目为例
10 #include <bits/stdc++.h>
11 #define ios ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
12 #define endl '\n'
13 using namespace std;
14 const int N=110;
15 struct Point{

```

```

16     double x,y;
17 }q[N];
18 int n;
19 double ans=1e8;
20 double rand(double l,double r){ //产生[l,r]的一个随机数
21     return (double)rand()/RAND_MAX*(r-l)+l;
22 }
23 double getdis(Point x){
24     double res=0;
25     for(int i=1;i<=n;i++){
26         res+=sqrt(pow(q[i].x-x.x,2)+pow(q[i].y-x.y,2));
27     }
28     ans=min(res,ans);
29     return res;
30 }
31 void simulate_anneal(){ //退火函数
32     Point now={rand(0,10000),rand(0,10000)};
33     for(double t=1e4;t>1e-4;t*=0.999){ //t是温度,
34         Point np={rand(0,10000),rand(0,10000)}; //np是在[x-t,x+t]范围内产生的新点
35         double dt=getdis(np)-getdis(now); //dt是新点函数值和旧点函数值的差值
36         //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!!
37         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
38     }
39 }
40 int main()
41 {
42     ios;
43     cin>>n;
44     for(int i=1;i<=n;i++) cin>>q[i].x>>q[i].y;
45     for(int i=1;i<=100;i++) simulate_anneal();
46     cout<<fixed<<setprecision(0)<<ans<<endl;
47     return 0;
48 }

```

## 6.4 手写 Hash(int)

```

1 struct MAP{
2     int tot;
3     int nxt[M],las[P],to[M],w[M];
4     MAP(){
5         tot=0;
6         for(int i=0;i<=P-1;++i)
7             las[i]=0;
8     }
9     void insert(int y){
10         int x=y%P;
11         for(int e=las[x];e=e=nxt[e]){
12             if(to[e]==y){
13                 ++w[e];
14                 return;
15             }
16         }
17         nxt[++tot]=las[x];
18         las[x]=tot;
19         to[tot]=y;
20         w[tot]=1;
21     }
22     int find(int y){
23         int x=y%P;
24         for(int e=las[x];e=e=nxt[e]){
25             if(to[e]==y)
26                 return w[e];
27         }
28         return 0;

```

```

29     }
30 } mp1;

```

### section 字符串 subsection 字符串哈希

```

1 typedef unsigned long long ull;
2 ull ha[N];
3 ull po[N];
4 ull bas=1331;
5 void init_hash(string &s){
6     int len=s.size();
7     s=" "+s;
8     po[0]=1; ha[0]=0;
9     for(int i=1;i<=len;i++){
10         ha[i]=ha[i-1]*bas+s[i]-'0';
11         po[i]=po[i-1]*bas;
12     }
13 }
14 ull get_hash(int l,int r){
15     return ha[r]-ha[l-1]*po[r-l+1];
16 }

```

### subsectionKMP

```

1 int ne[N]; //next数组: i位置之前的最长公共前后缀
2 void get_next(string &s){
3     int len=s.size();
4     int i=0, j=-1;
5     ne[0]=-1;
6     while(i<len){
7         if(j== -1 || s[i]==s[j]){
8             ++i; ++j;
9             ne[i]=j;
10        }
11        else j=ne[j];
12    }
13    return ;
14 }
15 int kmp(string &s,string &t){
16     get_next(t);
17     int len1=s.size(),len2=t.size();
18     int i=0, j=0, cnt=0;
19     while(i<len1){
20         if(j== -1 || s[i]==t[j]){
21             ++i; ++j;
22         }
23         else j=ne[j];
24         if(j==len2) cnt++;
25     }
26     return cnt;
27 }
28 //nextval优化
29 void get_nextval(string s){
30     int i=0, j=-1;
31     nextval[0]=-1;
32     int len=s.size();
33     while(i<len){
34         if(j== -1 || s[i]==s[j]){
35             ++i; ++j;
36             // 和求解next数组唯一不一样的地方
37             if(s[i]!=s[j]) nextval[i]=j;
38             else nextval[i]=nextval[j]; //当前位置和next[i]的字符相同时, 则不需要回溯到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[nextval[i]], 当没有跳步回溯操作时next和nextval值是一样的
39         }
40         else j=nextval[j];
41     }

```

42 }

### subsection 扩展 KMP

```
1 void getnext(string T){
2     int len=T.size();
3     nex[0]=len;
4     int p=0;
5     while(p+1<len && T[p]==T[p+1]) p++; // 这里注意把边界写在前面
6     nex[1]=p;
7     int po=1;
8     for(int i=2;i<len;i++){
9         if(i+nex[i-po]<po+nex[po]) nex[i]=nex[i-po]; // 第一种情况, 直接得到答案
10        else{
11            int j=po+nex[po]-i;
12            if(j<0) j=0; // 超出已匹配的字符串长度, 需要重新匹配
13            while(i+j<len && T[i+j]==T[j]) j++;
14            nex[i]=j;
15            po=i; // 长度超出, 更新起始位置
16        }
17    }
18 }
19 void extmp(string S,string T){
20     int len1=S.size(), len2=T.size();
21     getnext(T);
22     int p=0;
23     while(p<len1 && p<len2 && S[p]==T[p]) p++; // 边界写到前面
24     ext[0]=p;
25     int po=0;
26     for(int i=1;i<len1;i++){
27         if(i+nex[i-po]<po+ext[po]) ext[i]=nex[i-po];
28         else{
29             int j=po+ext[po]-i;
30             if(j<0) j=0;
31             while(i+j<len1 && j<len2 && T[j]==S[i+j]) j++;
32             ext[i]=j;
33             po=i;
34         }
35     }
36 }
```

## 6.5 马拉车算法

```
1 int p[N];
2 string s;
3 int manacher(string s){
4     string t="";
5     t+='#';
6     for(int i=0;i<(int)s.size();i++){
7         t+=s[i];
8         t+='#';
9     }
10    int ans=0;
11    int pos=0;int maxxright=0;
12    for(int i=0;i<(int)t.length();i++){
13        p[i]=maxxright>i?min(p[2*pos-i],maxxright-i):1;//关键
14        while(i-p[i]>=0&&i+p[i]<(int)t.length()&&t[i-p[i]]==t[i+p[i]]) p[i]++;
15        if(i+p[i]-1>maxxright){
16            maxxright=i+p[i]-1;
17            pos=i;
18        }
19        ans=max(ans,p[i]);
20    }
21    return ans-1;
22 }
```