

# ACM 模板

无敌爆哥

2023 年 9 月 13 日



## 目录

1 数学	5
1.1 康托展开	5
1.2 逆康托展开	6
1.3 线性逆元	7
1.4 线性筛求因子数量	8
1.5 二进制 GCD	9
1.6 快速 gcd	10
1.7 求矩阵行列式值	12
1.8 线性基	13
1.9 拓展中国剩余定理 (EXCRT)	15
1.10 BSGS	17
1.11 卢卡斯求组合数	19
1.12 错排	20
1.13 高精度	21
1.14 高精度 GCD	26
1.15 拉格朗日插值	27
1.16 FFT(快速傅里叶变换)	28
1.16.1 单位根	28
1.16.2 代码实现	28
1.17 原根	32
1.18 NTT (快速数论变换)	33
1.19 任意模数 MTT	35
1.20 牛顿迭代法	40
1.21 分治 NTT	41
1.22 二次剩余	42
1.23 多项式通用模板 (NTT)	45
1.24 类欧几里得	54
1.24.1 拓展	54
1.25 区间维护等差数列	56
1.26 随机化	59
1.26.1 随机数据生成器	59
1.26.2 随机排列数组	59
1.27 二项式反演	60
1.28 Pollard-Rho	61
1.29 杜教筛	63

1.30 min25 筛	65
1.31 斯特林数	67
1.31.1 第一类斯特林数 (斯特林轮换数)	67
1.31.2 第二类斯特林数 (斯特林轮换数)	67
1.31.3 行	67
1.31.4 列	67
1.31.5 下降幂	67
1.31.6 变换	67
1.31.7 上升幂	67
1.31.8 变换	67
1.32 十二种球盒模型	68
1.33 贝尔数	73
1.34 欧拉数	73
1.35 小公式	73
1.36 佩尔 (pell) 方程	73
1.37 LGV 引理	73
1.38 伯特兰-切比雪夫定理	73
1.39 基础博弈论	74
2 几何	76
2.1 几何通用	76
2.2 极角序	83
2.3 凸包	84
2.3.1 二维凸包	84
2.4 三维凸包	85
2.4.1 暴力	85
2.5 增量法	87
2.6 线段到线段最小距离	89
2.7 旋转卡壳	90
2.7.1 凸包直径	90
2.7.2 两个不相交凸包最小距离	91
2.7.3 最小矩阵覆盖	93
2.8 闵可夫斯基和	95
2.9 半平面交	97
2.10 最小圆覆盖	98
2.11 三角形和圆交面积	99
2.12 二维向量绕 0 点旋转 $n$ 度	101

2.13	椭圆	101
2.14	皮克定理	101
2.15	浮点输出	101
2.16	扫描线	102
2.17	平面最近点对	105
2.18	自适应辛普森公式	106
3	其它	107
3.1	快读快写开 02	107
3.2	bitset 处理多维偏序	108
3.3	CDQ 分治	111
3.4	Tanjan	114
3.4.1	边双连通分量	114
3.4.2	缩点	117
3.5	可持久化 kmp	121
3.6	map 排序	122

# 1 数学

## 1.1 康托展开

可以  $O(n \log n)$  复杂度求一个排列对于排列中的排名

```
1  const int N=1e6+10,mod=998244353;
2  long long fac[N];
3  int tr[N];
4  int n;
5  int a[N];
6  int lowbit(int x)
7  {
8      return x&-x;
9  }
10 void add(int x)
11 {
12     for(int i=x;i<=n;i+=lowbit(i)) tr[i]+=1;
13 }
14 int qr(int x)
15 {
16     int res=0;
17     for(int i=x;i;i-=lowbit(i)) res+=tr[i];
18     return res;
19 }
20 long long cantor(){
21     fac[0]=1;
22     for(int i=1;i<=n;i++) {
23         fac[i]=fac[i-1]*i%mod;
24     }
25     long long ans=0;
26     for(int i=1;i<=n;i++){
27         ans=(ans+(a[i]-1-qr(a[i]))*fac[n-i]%mod)%mod;
28         add(a[i]);
29     }
30     return (ans+1)%mod;
31 }
```

## 1.2 逆康托展开

已经知道排名得到原排列，逆向推出原序列

```
1 vector<int> incantor(int x,int n) {
2     x--; // 得到以0开始的排名
3     vector<int> res(n); // 保存数列答案
4     int cnt;
5     bool st[10]; // 标记数组
6     memset(st,0,sizeof st);
7     for(int i = 0;i < n; ++i) {
8         cnt = x/fact[n - i - 1]; // 比a[i]小且没有出现过的
            数的个数
9         x %= fact[n - i - 1]; // 更新 x
10        for(int j = 1;j <= n; ++j) { // 找到a[i], 从1开始向
            后找
11            if(st[j]) continue; // 如果被标记过, 就跳过
12            if(!cnt) { // 如果cnt == 0说明当前数是a[i]
13                st[j] = 1; // 标记
14                res[i] = j; // 第i位是j
15                break;
16            }
17            cnt --; // 如果当前不是0, 就继续往后找
18        }
19    }
20    return res; // 返回答案
21 }
```

## 1.3 线性逆元

```
1 in[0]=in[1]=1;
2 for(int i=2;i<=n;i++){
3     in[i]=1ll*(mod-mod/i)*in[mod%i]%mod;
4 }
```

## 1.4 线性筛求因子数量

```
1 int fac[N],q[N],top;
2 bool vis[N];
3 int cnt[N];
4 void init(int n=1e6)
5 {
6     vis[1]=true;
7     fac[1]=1;
8     for(int i=2;i<=n;i++)
9     {
10         if(vis[i]==false)
11         {
12             q[top++]=i;
13             fac[i]=2;
14             cnt[i]=1;
15         }
16         for(int j=0;1ll*q[j]*i<=n;j++)
17         {
18             int nu=i*q[j];
19             vis[nu]=true;
20             if(i%q[j]==0)
21             {
22                 fac[nu]=fac[i]/(cnt[i]+1)*(cnt[i]+2);
23                 cnt[nu]=cnt[i]+1;
24                 break;
25             }
26             fac[nu]=fac[i]*fac[q[j]];
27             cnt[nu]=1;
28         }
29     }
30 }
```



## 1.5 二进制 GCD

快!

```
1 int gcd(int a, int b) {
2     int az = __builtin_ctz(a);
3     int bz = __builtin_ctz(b);
4     int z = min(az, bz);
5     b >>= bz;
6     while (a) {
7         a >>= az;
8         int diff = a - b;
9         az = __builtin_ctz(diff);
10        b = min(a, b), a = abs(diff);
11    }
12    return b << z;
13 }
```

## 1.6 快速 gcd

预处理值域，常数访问 gcd (T 取  $\sqrt{max}$ )

```
1  const int M=1e6+10,T=1e3+10;//值域
2  bool vis[M];
3  int q[M],cnt;
4  int fac[M][3];
5  int gc[T][T];
6  int gcd(int a,int b)
7  {
8      return b?gcd(b,a%b):a;
9  }
10 void init(int n=M-10)
11 {
12     vis[1]=true;
13     vis[0]=true;
14     fac[1][0]=fac[1][1]=fac[1][2]=1;
15     for(int i=2;i<=n;i++)
16     {
17         if(!vis[i])
18         {
19             q[cnt++]=i;
20             fac[i][0]=fac[i][1]=1;
21             fac[i][2]=i;
22         }
23         for(int j=0;1ll*q[j]*i<=n;j++)
24         {
25             int nu=q[j]*i;
26             vis[nu]=true;
27             fac[nu][0]=fac[i][0]*q[j];
28             fac[nu][1]=fac[i][1];
29             fac[nu][2]=fac[i][2];
30             if(fac[nu][0]>fac[nu][1]) swap(fac[nu][0],fac[nu][1]);
31             if(fac[nu][1]>fac[nu][2]) swap(fac[nu][1],fac[nu][2]);
32             if(i%q[j]==0) break;
```

```
33     }
34 }
35 for(int i=0;i<=T-10;i++)
36 {
37     for(int j=0;j<=T-10;j++)
38     {
39         if(!i&&!j) continue;
40         gc[i][j]=gcd(i,j);
41     }
42 }
43 }
44 int qgcd(int x,int y)
45 {
46     int res=1;
47     for(int i=0,t=1;i<3;i++)
48     {
49         if(fac[x][i]>T-10)
50         {
51             if(y%fac[x][i]) t=1;
52             else t=fac[x][i];
53         }
54         else t=gc[fac[x][i]][y%fac[x][i]];
55         res=res*t;y/=t;
56     }
57     return res;
58 }
```

## 1.7 求矩阵行列式值

原理是通过高斯消元将行列式下三角清 0，那么答案就是主对角线的累乘，这里由于模值不一定是质数，不能保障每一个元素都在 P 取模情况下有逆元，所以采用了特别妙的方法，通过主元行与其他行进来辗转相除来进行清 0 过程。原理是行列式中一行减去另外一行的乘积，行列式答案不变。

```
1 int mod=998244353;
2 const int N=610;
3 long long mp[N][N];
4 int n;
5 long long gauss()//求矩阵行列式的值
6 {
7     long long res=1;
8     for(int i=0;i<n;i++)
9     {
10         for(int j=i+1;j<n;j++)
11         {
12             while(mp[i][i])
13             {
14                 long long div=mp[j][i]/mp[i][i];
15                 for(int k=i;k<n&&div;k++) {
16                     mp[j][k]=(mp[j][k]-div*mp[i][k])%mod+
17                         mod)%mod;
18                 }
19                 for(int k=i;k<n;k++) swap(mp[i][k],mp[j][k]);res=-res;
20             }
21             for(int k=i;k<n;k++) swap(mp[i][k],mp[j][k]);
22             res=-res;
23         }
24         res=(res+mod)%mod;
25         for(int i=0;i<n;i++) res=res*mp[i][i]%mod;
26         return res;
27     }
```

## 1.8 线性基

```
1 struct Vec{
2     vector<long long>d,g;
3     int n;
4     int le;
5     Vec(){};
6     Vec(int _n){n=_n;d.resize(n+1);g.resize(n+1);}
7     bool insert(long long x){
8         for(int i=n;i>=0;i--){
9             if((x>>i)&1){
10                 if(d[i]==0){
11                     d[i]=x;
12                     return true;
13                 }else{
14                     x^=d[i];
15                 }
16             }
17         }
18         return false;
19     }
20     void re(){
21         for(int i=0;i<=n;i++){
22             for(int j=i-1;j>=0;j--){
23                 if((d[i]>>j)&1){
24                     d[i]^=d[j];
25                 }
26             }
27         }
28         le=0;
29         for(int i=0;i<=n;i++){
30             if(d[i]){
31                 g[le++]=d[i];
32             }
33         }
34     }
35     long long qrk(long long x){
```

```
36         if(x>(1ll<<le)) return -1;
37         x--;
38         long long res=0;
39         for(int i=0;i<le;i++){
40             if((x>>i)&1){
41                 res^=g[i];
42             }
43         }
44         return res;
45     }
46 };
```

## 1.9 拓展中国剩余定理 (EXCRT)

CRT 求的问题是有一个数余上不同的模值  $M$  之后, 求余数  $R$  值通过公式分析可以推出最终的模值为  $M=\text{LCM}(m_1,m_2,m_3\dots)$ ,  $R=\text{LCM}*k+q.m_1+r_1$  有时候如果可以用中国剩余定理搞的题目, 需要质数分别当模数, 如果 wa 了试一下不用 5 什么的

```
1 long long gcd(long long a,long long b)
2 {
3     return b?gcd(b,a%b):a;
4 }
5 long long exgcd(long long a,long long b,long long &x,long
    long &y)
6 {
7     if(b==0)
8     {
9         x=1;
10        y=0;
11        return a;
12    }
13    long long x1,y1,d;
14    d=exgcd(b,a%b,x1,y1);
15    x=y1;
16    y=x1-a/b*y1;
17    return d;
18 }
19 long long lcm(long long a,long long b)
20 {
21     return a/gcd(a,b)*b;
22 }
23 long long m[20],r[20];
24 int tp;
25 long long CRT()
26 {
27     for(int i=1;i<tp;i++)
28     {
29         long long d,x,y;
30         d=exgcd(m[i-1],m[i],x,y);
31         if((r[i]-r[i-1])%d) return -1;
```

```

32     x=x*(r[i]-r[i-1])/d;
33     x=(x%(m[i]/d)+m[i]/d)%(m[i]/d);
34     r[i]=m[i-1]*x+r[i-1];
35     m[i]=lcm(m[i-1],m[i]);
36     r[i]=(r[i]%m[i]+m[i])%m[i];
37 }
38 return r[tp-1]%m[tp-1];
39 }

```

解  $at[i]*x\%m[i]=r[i]$ //带有系数的 exCRT

```

1 long long CRT()
2 {
3     tp=n;
4     long long ans=0,M=1,A,B,C,x,y;
5     for(int i=1;i<=tp;i++)
6     {
7         A((__int128)at[i]*M%m[i];
8         B=m[i];
9         C=(r[i]-at[i]*ans%m[i]+m[i])%m[i];
10        long long d=exgcd(A,B,x,y);
11        if(C%d) return -1;
12        x=(x%B+B)%B;
13        ans+=((__int128)(C/d)*x%(B/d)*M%(M*=B/d);
14        ans%=M;
15    }
16    return ans;
17 }

```



## 1.10 BSGS

$$a^x = b \pmod{p}$$

```
1 long long gcd(long long a, long long b)
2 {
3     return b?gcd(b,a%b):a;
4 }
5 long long BSGS(long long a, long long b, long long p)
6 {
7     a%=p, b%=p;
8     if(b==1 || p==1){
9         return 0;
10    }
11    int k=0;
12    long long A=1;
13    while(1)
14    {
15        long long d=gcd(a,p);
16        if(b%d!=0) {
17            return -1;
18        }
19        if(d==1) break;
20        k++; b/=d; p/=d; A=A*(a/d)%p; b%=p;
21        if(A==b) {
22            return k;
23        }
24    }
25    long long m=ceil(sqrt(p));
26    map<long long, int> mp;
27    long long t=b;
28    for(int i=0; i<m; i++)
29    {
30        mp[t]=i;
31        t=t*a%p;
32    }
33    t=1;
34    for(int i=1; i<=m; i++)
```

```
35     {
36         t=t*a%p;
37     }
38     long long tt=A;
39     for(int i=1;i<=m;i++)
40     {
41         tt=tt*t%p;
42         if(mp.count(tt)&& i*m-mp[tt]>=0)
43         {
44             return i*m-mp[tt]+k;
45         }
46     }
47     }
48     return -1;
49 }
```

### 1.11 卢卡斯求组合数

满足  $\text{mod}$  为质数

```
1 long long lucas(long long n,long long m)
2 {
3     if(m==0) return 1;
4     return lucas(n/mod,m/mod)*C(n%mod,m%mod)%mod;
5 }
```

## 1.12 错排

$n$  个不相同盒子和  $n$  个不相同球，对应球不能放到下标相同盒子中

```
1 f[0]=1;  
2 f[1]=0;  
3 f[2]=1;  
4 f[i]=(f[i-1]+f[i-2])*(i-1);
```

## 1.13 高精度

```
1 struct BigInt{
2     int w=6;
3     int tw=1e6;
4     vector<int>v;
5     int len=0;
6     BigInt(){len=0;}
7     BigInt(int _len){len=_len;v.resize(len);}
8     BigInt operator +(const BigInt &w)const{
9         BigInt res;
10        res.v=v;
11        res.len=len;
12        long long tp=0;
13        for(int i=0;i<(int)v.size();i++){
14            if(i<(int)w.v.size()){
15                tp+=w.v[i];
16            }
17            tp+=v[i];
18            res.v[i]=(tp%tw);
19            tp/=tw;
20        }
21        while(tp) res.v.push_back(tp%tw),tp/=tw,res.len++;
22        return res;
23    }
24    BigInt operator +(const int &x)const{
25        int tp=x;
26        BigInt res;
27        res.v=v;
28        res.len=len;
29        for(int i=0;i<(int)v.size();i++){
30            tp+=v[i];
31            res.v[i]=(tp%tw);
32            tp/=tw;
33            if(tp==0) break;
34        }
35        while(tp) res.v.push_back(tp%tw),tp/=tw,res.len++;
```

```
36         return res;
37     }
38     //sub只有a>=b
39     BigInt operator -(const BigInt &w)const{
40         BigInt res;
41         res.v=v;
42         res.len=len;
43         for(int i=0;i<len;i++){
44             if(i<(int)w.len){
45                 res.v[i]-=w.v[i];
46             }
47             if(res.v[i]<0) res.v[i]+=tw,res.v[i+1]--;
48         }
49         while(res.len>1&&res.v.back()==0){
50             res.v.pop_back();
51             res.len--;
52         }
53         return res;
54     }
55     BigInt operator - (const int b)const{
56         BigInt res;
57         res.v=v;
58         res.len=len;
59         res.v[0]-=b;
60         for(int i=0;i<(int)v.size();i++){
61             while(res.v[i]<0) res.v[i]+=tw,res.v[i]--;
62         }
63         while(res.len>1&&res.v.back()==0){
64             res.v.pop_back();res.len--;
65         }
66         return res;
67     }
68     BigInt operator *(const BigInt & w)const{
69         BigInt res(len+w.len+2);
70         for(int i=0;i<len;i++){
71             for(int j=0;j<w.len;j++){
72                 long long tp=1ll*v[i]*w.v[j];
```

```
73             res.v[i+j+1]+=tp/tw;
74             res.v[i+j]+=tp%tw;
75             res.v[i+j+1]+=res.v[i+j]/tw;
76             res.v[i+j]%=tw;
77         }
78     }
79     while(res.len>1&&res.v.back()==0){
80         res.v.pop_back();res.len--;
81     }
82     return res;
83 }
84 BigInt operator *(const int w)const{
85     BigInt res;
86     res.v=v;
87     res.len=len;
88     long long tp=0;
89     for(int i=0;i<len;i++){
90         tp+=1ll*w*res.v[i];
91         res.v[i]=tp%tw;
92         tp/=tw;
93     }
94     while(tp){
95         res.v.push_back(tp%tw);res.len++;
96         tp/=tw;
97     }
98     return res;
99 }
100 BigInt operator /(const int w)const{
101     BigInt res;
102     res.v=v;
103     res.len=len;
104     long long tp=0;
105     for(int i=len-1;i>=0;i--){
106         tp=tp*tw+v[i];
107         res.v[i]=tp/w;
108         tp%=w;
109     }
```

```
110         while(res.len>1&&res.v.back()==0){
111             res.v.pop_back();res.len--;
112         }
113         return res;
114     }
115     int operator %(const int w)const{
116         long long res=0;
117         for(int i=len-1;i>=0;i--){
118             res=res*tw+v[i];
119             res%=w;
120         }
121         return res;
122     }
123     void read(string &a){
124         for(int i=(int)a.size();i-1>=0;i-=w){
125             int res=0;
126             for(int j=max(0,i-w);j<=i-1;j++){
127                 res=res*10+(a[j]-'0');
128             }
129             v.push_back(res);len++;
130         }
131     }
132     /*
133     1 a>b
134     -1 a<b
135     0 a=b
136     */
137     int equal(const BigInt &w){
138         if(len>w.len) return 1;
139         else if(len<w.len) return -1;
140         for(int i=len-1;i>=0;i--){
141             if(v[i]>w.v[i]) return 1;
142             else if(v[i]<w.v[i]) return -1;
143         }
144         return 0;
145     }
146     void pr(){
```



```
147         if(len==1){
148             cout<<v[0];
149         }
150         else{
151             int f=0;
152             for(int i=len-1;i>=0;i--){
153                 int x=v[i];
154                 int tww=tw;
155                 for(int j=0;j<w;j++){
156                     tww/=10;
157                     int y=x/tww;
158                     if(y) f=1;
159                     if(f) cout<<y;
160                     x%=tww;
161                 }
162             }
163         }
164         cout<<'\\n';
165     }
166 };
```

## 1.14 高精度 GCD

```
1 BigInt gcd(BigInt a,BigInt b){
2     int tp=0;
3     while(1){
4         int t=a.equal(b);
5         if(t== -1) swap(a,b);
6         if(t==0||b.len==1&&b.v[0]==0) break;
7         if(!(a.v[0]&1)&&!(b.v[0]&1)){
8             tp++;
9             a=a/2;
10            b=b/2;
11        }
12        else if(!(a.v[0]&1)){
13            a=a/2;
14        }
15        else if(!(b.v[0]&1)){
16            b=b/2;
17        }else{
18            a=a-b;
19        }
20    }
21    while(tp--){
22        a=a*2;
23    }
24    return a;
25 }
```

## 1.15 拉格朗日插值

$$f(x) = \sum_{i=1}^n y_i \prod_{j=1}^n \frac{x-x_j}{x_i-x_j} (x_i \neq x_j)$$

```
1  const int N=2e5+10,mod=998244353;
2  long long ksm(long long a,long long b){
3      long long res=1;
4      a%=mod;
5      while(b){
6          if(b&1) res=res*a%mod;
7          a=a*a%mod;
8          b>>=1;
9      }
10     return res;
11 }
12 struct pp{
13     long long x,y;
14 }p[N];
15 int n;
16 long long lagrange(long long k){
17     long long ans=0;
18     for(int i=1;i<=n;i++){
19         long long res=1;
20         long long res1=1;
21         for(int j=1;j<=n;j++){
22             if(i==j) continue;
23             res=res*(p[i].x-p[j].x+mod)%mod;
24             res1=res1*(k-p[j].x+mod)%mod;
25         }
26         ans=(ans+p[i].y*res1%mod*ksm(res,mod-2)%mod)%mod;
27     }
28     return ans;
29 }
```

## 1.16 FFT(快速傅里叶变换)

### 1.16.1 单位根

用来实现 DFT 的一个重要元素

前提:  $n = 2^k$  ( $k$  为非负整数)

形式:  $w_n^k = \cos(\frac{2\pi k}{n}) + i * \sin(\frac{2\pi k}{n})$

周期性:  $w_n^{n+k} = w_n^k$

对称性:  $w_n^{k+\frac{n}{2}} = -w_n^k$

折半性:  $w_n^{2k} = w_{\frac{n}{2}}^k$

在 DFT 中的体现是:

$$A(x) = \sum_{i=0}^{n-1} a_i x^i$$

$$A_1(x) = a_0 + a_2 x^2 \dots a_{n-2} x^{n-2}$$

$$A_2(x) = a_1 x^1 + a_3 x^3 \dots a_{n-1} x^{n-1}$$

$$A(x) = A_1(x^2) + x A_2(x^2)$$

设  $w_n^k$  ( $k < \frac{n}{2}$ )

$$A(w_n^k) = A_1(w_n^{2k}) + w_n^k A_2(w_n^{2k}) = A_1(w_{\frac{n}{2}}^k) + w_n^k A_2(w_{\frac{n}{2}}^k)$$

$$A(w_n^{k+\frac{n}{2}}) = A_1(w_n^{2k+n}) + w_n^{k+\frac{n}{2}} A_2(w_n^{2k+n}) = A_1(w_{\frac{n}{2}}^k) - w_n^k A_2(w_{\frac{n}{2}}^k)$$

### 1.16.2 代码实现

系数表示法—> 点值表示法—> 系数表示法

采用单位根, 运用负数表达, 系数需要小数表达可以选择 `fft`

```

1  const double PI=acos(-1);
2  struct complex_{
3      double a,b;
4      complex_(){a=0;b=0;}
5      complex_(double _a,double _b){a=_a;b=_b;}
6      complex_ operator +(const complex_ &w)const{
7          return complex_(a+w.a,b+w.b);
8      }
9      complex_ operator -(const complex_ &w)const{
10         return complex_(a-w.a,b-w.b);
11     }
12     complex_ operator *(const complex_ &w)const{
13         return complex_(a*w.a-b*w.b,a*w.b+b*w.a);
14     }
15 };

```

```
16 namespace FFT{
17     using poly=vector<complex_>;
18     int limit,top;
19     const int N=3e6+10;
20     int r[N];
21     void init(int n,int m){
22         limit=1;top=0;
23         while(limit<=n+m) limit<<=1,top++;
24         for(int i=0;i<limit;i++){
25             r[i]=(r[i>>1]>>1)|((i&1)<<(top-1));
26         }
27     }
28     void fft(poly &v,int type){
29         int len=v.size();
30         for(int i=0;i<len;i++){
31             if(i<r[i]) swap(v[i],v[r[i]]);
32         }
33         for(int mid=1;mid<len;mid<=1){
34             complex_ wn(cos(PI/mid),type*sin(PI/mid));
35             for(int R=mid<<1,j=0;j<len;j+=R){
36                 complex_ w(1,0);
37                 for(int k=0;k<mid;k++,w=w*wn){
38                     complex_ x=v[j+k],y=w*v[j+k+mid];
39                     v[j+k]=x+y;
40                     v[j+k+mid]=x-y;
41                 }
42             }
43         }
44         if(type== -1){
45             for(int i=0;i<len;i++){
46                 v[i].a/=(double)len;
47             }
48         }
49     }
50     poly Mul(poly A,poly B,int len=0){
51         int n=A.size(),m=B.size();
52         if(len){
```

```
53         if(n>len){
54             A.resize(len),n=len;
55         }
56         if(m>len) B.resize(len),m=len;
57     }
58     if(n<=30||m<=30){
59         poly res(n+m-1,{0,0});
60         for(int i=0;i<n;i++){
61             for(int j=0;j<m;j++){
62                 res[i+j].a=(res[i+j].a+A[i].a*B[j].a);
63             }
64         }
65         if(len){
66             res.resize(len);
67         }
68         return res;
69     }
70     init(n,m);
71     A.resize(limit);B.resize(limit);
72     for(int i=min(n,m);i<limit;i++){
73         if(i>=n) A[i].a=A[i].b=0;
74         if(i>=m) B[i].a=B[i].b=0;
75     }
76     fft(A,1);fft(B,1);
77     for(int i=0;i<limit;i++){
78         A[i]=A[i]*B[i];
79     }
80     fft(A,-1);
81     if(len==0){
82         A.resize(n+m-1);
83     }
84     else{
85         A.resize(len);
86     }
87     return A;
88 }
89 //反转多项式系数
```

```
90     void Reverse(poly &a){  
91         reverse(a.begin(),a.end());  
92     }  
93 }  
94 using namespace FFT;
```

## 1.17 原根

若  $\gcd(a, m) == 1$  同时  $a^x \equiv 1 \pmod{m}$  的最小阶为  $\phi(m)$  则表示  $a$  是  $m$  的原根  
 求解通常是暴力算，已经被证明原根渐近  $m^{\frac{1}{4}}$  所以时间还是比较可观  
 且只有  $2, 4, p^a, 2p^a$  ( $p$  为奇质数) 才有原根

```

1  int n,d;
2      cin>>n>>d;
3      if(!vvis[n]){ //通过线性筛预处理出 2 4 p^a 2p^a
4          cout<<"0\n";
5          cout<<' \n';
6      }else{
7          int t=getol(n); //获得欧拉函数
8          vector<int>pr=getpr(t); //获得质因子
9          int fas=0;
10         for(int g=1;;g++){
11             int f=0;
12             if(__gcd(g,n)!=1) continue;
13             //因为阶是n的欧拉函数，所以因子都不能满足
14             for(int v:pr){
15                 if(ksm(g,t/v,n)==1){
16                     f=1; break;
17                 }
18             }
19             if(f==0){
20                 fas=g; break; //最小原根
21             }
22         }
23         vector<int>ans;
24         int num=fas;
25         //所有原根
26         for(int i=1;i<=t;i++){
27             if(__gcd(i,t)==1){
28                 ans.push_back(num);
29             }
30             num=num*fas%n;
31         }
32     }

```



## 1.18 NTT（快速数论变换）

采用原根来表达，都为整数运算，可以相对 `fft` 减少精度的影响。

```

1  /*
2  65537
3  998244353
4  1004535809
5  4179340454199820289
6  因为ntt受限制模数需要 $p=r*2^l+1$  （其中 $l$ 需要比给定运算的多项
   式的 $\log(\text{len})$ 要大）
7  所有以上是常见的 且 $g=3$  如果遇到一个其余模数，可以先预处理
   看是否符合ntt要求在 选择ntt求解
8  */

```

```

1  const int N=3e6+10;
2  int r[N];
3  int limit,L;
4  void init(int n){
5      for(int i=0;i<n;i++){
6          r[i]=(r[i>>1]>>1)|((i&1)<<(L-1));
7      }
8  }
9  struct NTT{
10     int len;
11     int g=3,gi=332748118;
12     long long mod=998244353;
13     vector<long long>v;
14     NTT(){g=3;gi=332748118;mod=998244353;}
15     NTT(int _len){len=_len;v.resize(len);g=3;gi=332748118;
        mod=998244353;}
16     void init(int _len){len=_len;v.resize(len);g=3;gi
        =332748118;mod=998244353;}
17     void ntt(int type){
18         for(int i=0;i<len;i++){
19             if(i<r[i]) swap(v[i],v[r[i]]);
20         }
21         for(int mid=1;mid<len;mid<=<=1){

```

```
22         long long wn=ksm(type==1?g:gi,(mod-1)/(mid<<1)
23             ,mod);
24         for(int j=0,R=mid<<1;j<limit;j+=R){
25             long long w=1;
26             for(int k=0;k<mid;k++,w=(w*wn)%mod){
27                 long long x=v[j+k],y=v[j+mid+k]*w%mod;
28                 v[j+k]=(x+y)%mod;
29                 v[j+k+mid]=(x-y+mod)%mod;
30             }
31         }
32     }
33     void mul(const NTT &w){
34         for(int i=0;i<len;i++){
35             v[i]=(v[i]*w.v[i])%mod;
36         }
37     }
38 };
```

## 1.19 任意模数 MTT

采用原根来表达，都为整数运算，可以相对 `fft` 减少精度的影响。

```

1  /*
2  如果要用ntt就是如下
3  根据理论的乘法之后系数大小,通常最大 $10^{(25)}$ ，因此可以通过中
   国剩余定理来设置三模数且满足ntt形式
4  使其 $\text{lcm}(m1,m2,m3) >$  理论最大系数，来分别继续ntt操作，常数较
   大，需要9次ntt
5  千万记得改变模数时候改变原根和原根的逆元！
6  */
7  LL md[3]={469762049,998244353,1004535809}; //常用三模数

```

```

1  //这里是基于FFT拆系数的MTT
2  long long ksm(long long a,long long b,long long mod){
3      long long res=1;
4      a%=mod;
5      while(b){
6          if(b&1) res=res*a%mod;
7          a=a*a%mod;
8          b>>=1;
9      }
10     return res;
11 }
12 #define double long double
13 const double PI=acos(-1);
14 struct complex_{
15     double a,b;
16     complex_(){a=0;b=0;}
17     complex_(double _a,double _b){a=_a;b=_b;}
18     complex_ operator +(const complex_ &w)const{
19         return complex_(a+w.a,b+w.b);
20     }
21     complex_ operator -(const complex_ &w)const{
22         return complex_(a-w.a,b-w.b);
23     }
24     complex_ operator *(const complex_ &w)const{

```

```

25         return complex_(a*w.a-b*w.b,a*w.b+b*w.a);
26     }
27     complex_ operator /(const double w){
28         return complex_(a/w,b/w);
29     }
30 };
31 namespace MTT{
32     using poly=vector<long long>;
33     using poly1=vector<complex_>;
34     const int N=3e6+10;
35     const long long B=(1<<15);
36     const complex_ I(0,1);
37     int mod=1e9+7;
38     int r[N],limit,L;
39     void init(int n,int m){
40         limit=1;L=0;
41         while(limit<=n+m) limit<<=1,L++;
42         for(int i=0;i<limit;i++){
43             r[i]=(r[i>>1]>>1)|((i&1)<<(L-1));
44         }
45     }
46     void fft(poly1 &v,int type){
47         int len=v.size();
48         for(int i=0;i<len;i++){
49             if(i<r[i]) swap(v[i],v[r[i]]);
50         }
51         for(int mid=1;mid<len;mid<<=1){
52             complex_ wn(cos(PI/mid),type*sin(PI/mid));
53             for(int R=mid<<1,j=0;j<len;j+=R){
54                 complex_ w(1,0);
55                 for(int k=0;k<mid;k++,w=w*wn){
56                     complex_ x=v[j+k],y=w*v[j+k+mid];
57                     v[j+k]=x+y;
58                     v[j+k+mid]=x-y;
59                 }
60             }
61         }

```

```
62         if(type==-1){
63             for(int i=0;i<len;i++){
64                 v[i]=v[i]/(double)len;
65             }
66         }
67     }
68     void mtt(poly1 &a,poly1 &b){
69         for(int i=0;i<limit;i++){
70             a[i].b=b[i].a;
71         }
72         fft(a,1);
73         b[0]=complex_(a[0].a,-a[0].b);
74         for(int i=1;i<limit;i++){
75             b[i]=complex_(a[limit-i].a,-a[limit-i].b);
76         }
77         complex_ t1,t2;
78         for(int i=0;i<limit;i++){
79             t1=a[i],t2=b[i];
80             a[i]=(t1+t2)/2.0;
81             b[i]=(t2-t1)*I/2.0;
82         }
83     }
84     long long rd(const double x){
85         if(x>=0) return (long long)(x+0.5)%mod;
86         return (long long)(x-0.5)%mod;
87     }
88     poly Mul(poly a,poly b,int len=0){
89         int n=(int)a.size(),m=(int)b.size();
90         if(len){
91             if(n>=len){
92                 n=len;
93             }
94             if(m>=len){
95                 m=len;
96             }
97         }
98         if(n<=30||m<=30){
```

```

99         poly res(n+m-1);
100         for(int i=0;i<n;i++){
101             for(int j=0;j<m;j++){
102                 res[i+j]=(res[i+j]+a[i]*b[j]%mod)%mod;
103             }
104         }
105         if(len){
106             res.resize(len);
107         }
108         return res;
109     }
110     if(len){
111         init(len,len);
112     }else{
113         init(n,m);
114     }
115     a.resize(n+m-1);
116     poly1 a0(limit),b0(limit),a1(limit),b1(limit);
117     for(int i=0;i<limit;i++){
118         a0[i]=a1[i]=complex_(0,0);
119         if(i<n){
120             a1[i].a=a[i]%B;
121             a0[i].a=a[i]/B;
122         }
123     }
124     for(int i=0;i<limit;i++){
125         b0[i]=b1[i]=complex_(0,0);
126         if(i<m){
127             b1[i].a=b[i]%B;
128             b0[i].a=b[i]/B;
129         }
130     }
131     mtt(a0,a1);mtt(b0,b1);
132     complex_ t1,t2;
133     for(int i=0;i<limit;i++){
134         t1=a0[i]*b0[i]+I*a1[i]*b0[i];
135         t2=a0[i]*b1[i]+I*a1[i]*b1[i];

```

```

136         a0[i]=t1;b0[i]=t2;
137     }
138     fft(a0,-1);fft(b0,-1);
139     for(int i=0;i<n+m-1;i++){
140         long long ans=1ll*B*B%mod*rd(a0[i].a)%mod+
141             1ll*B*rd(b0[i].a+a0[i].b)%mod+rd(b0[i].b)%mod;
142         a[i]=(ans+mod)%mod;
143     }
144     if(len) a.resize(len);
145     return a;
146 }
147 //反转多项式系数
148 void Reverse(poly &a){
149     reverse(a.begin(),a.end());
150 }
151 //B(x)=2B'(x)-A(x)B'(x)B'(x)
152 poly GetInv(int n,poly &a){
153     if(n==1){
154         return {ksm(a[0],mod-2,mod)};
155     }
156     poly b=GetInv((n+1)>>1,a);
157     poly c=a;
158     c=Mul(c,b,n);
159     for(int i=0;i<n;i++){
160         c[i]=(mod-c[i])%mod;
161     }
162     c[0]=(c[0]+2)%mod;
163     c=Mul(c,b,n);
164     return c;
165 }
166 //多项式求逆
167 poly Inv(poly a){
168     int len=a.size();
169     return GetInv(len,a);
170 }
171 }
172 using namespace MTT;

```

## 1.20 牛顿迭代法

$$f(x) \equiv f_0(x) - \frac{g(f_0(x))}{g'(f_0(x))} \pmod{x}$$

多项式求逆，开方和 **eps** 都可以用此公式推出递推方程



## 1.21 分治 NTT

```
1 //通过区间前半段的f值来反馈贡献给后半段区间
2 void CDQ_NTT(int l,int r,poly &g,poly &f){
3     if(l==r){
4         return;
5     }
6     int mi=(l+r)>>1;
7     CDQ_NTT(l,mi,g,f);
8     poly a(mi-l+1,0),b(r-l+1,0);
9     int n=mi-l+1,m=r-l+1;
10    for(int i=l;i<=mi;i++){
11        a[i-l]=f[i];
12    }A
13    for(int i=0;i<m;i++){
14        b[i]=g[i];
15    }
16    init(m,m);
17    a.resize(limit);b.resize(limit);
18    for(int i=n;i<limit;i++){
19        a[i]=0;
20        if(i>=m) b[i]=0;
21    }
22    a=Mul(a,b);//根据具体卷积定义来调整
23    for(int i=mi+1;i<=r;i++){
24        f[i]=(f[i]+a[i-l])%mod;
25    }
26    CDQ_NTT(mi+1,r,g,f);
27 }
```

## 1.22 二次剩余

```

1 //二次剩余 2log
2 struct cipolla {
3     long long n,p,w2,a; //w2 = a*a-n mod p
4
5     struct cx {long long x,y;} ;//Fp2 = {x+yw | x,y belong
        to Fp}
6     cx mul(cx a,cx b) {return (cx){(a.x*b.x+a.y*b.y%p*w2)%
        p,(a.x*b.y+a.y*b.x)%p};}
7     cx cx_qpow(cx a,int n) {
8         cx ans = (cx){1,0};
9         for (;n;n>>=1,a=mul(a,a)) if (n&1) ans=mul(ans,a);
10        return ans;
11    }
12
13    void getsol(long long _n,long long _p,long long&ans1,
        long long&ans2)
14    {
15        n = _n, p = _p;
16        ans1=ans2=-1; //最多两个解,无解是-1
17        if (n==0 || p==2) {ans1=n; return ;}
18
19        if (ksm((long long)n,(long long)(p-1)/2,(long long)
            )p) == p-1) return ;
20
21        for (a=rand()%p;;a==p-1?(a=0):(++a)) {
22            if (ksm((a*a-n+p)%p,(p-1)/2,p) == p-1) {
23                w2 = (a*a-n+p)%p;
24                break;
25            }
26        }
27        cx b = (cx){a,1};
28        b = cx_qpow(b,(p+1)/2);
29        ans1 = b.x;
30        if (ans1 != p-ans1) ans2 = p-ans1;
31        return ;

```

```

32     }
33 };
34 //三次剩余 9log
35 struct cuberoot {
36     long long n,p,epsi;
37     struct tri {long long x,y,z; };
38     tri mul(tri a,tri b) {
39         return (tri){((a.x*b.x%p+a.y*b.z%p*n%p)%p+a.z*b.y%
40             p*n%p)%p,(a.x*b.y%p+a.y*b.x%p+a.z*b.z%p*n%p)%p
41             ,(a.x*b.z%p+a.y*b.y%p+a.z*b.x%p)%p};
42     }
43     tri tri_qpow(tri a,int n) {
44         tri ans = (tri){1,0,0};
45         for (;n>=1;a=mul(a,a)) if (n&1) ans=mul(ans,a);
46         return ans;
47     }
48     void getsol(long long _n,long long _p,long long &ans1,
49         long long &ans2,long long &ans3)
50     {
51         n = _n, p = _p;
52         ans1=ans2=ans3=-1;
53         if (n==0 || p==2 || p==3) {ans1=n; return ;}
54         if (p%3 == 2) {ans1 = ksm(n,(2*p-1)/3,p); return ;}
55
56         if (ksm(n,(p-1)/3,p) != 1) return ;
57         cipolla cip;
58         long long tmp;
59         cip.getsol(p-3,p,epsi,tmp);
60         epsi = (epsi-1)*((p+1)/2)%p;
61
62         for (;;) {
63             tri a;
64             a.x = rand()%p, a.y = rand()%p+1, a.z = rand()
65                 %p+1;
66             a = tri_qpow(a,(p-1)/3);
67             if (a.x==0&&a.z==0) {

```

```
64         ans1 = ksm(a.y,p-2,p);
65         ans2 = ans1*epsi % p;
66         ans3 = ans2*epsi % p;
67         return ;
68     }
69 }
70 }
71 };
```

## 1.23 多项式通用模板 (NTT)

```
1 namespace Poly{
2     const int N=3e6+10;
3     const int mod=998244353;
4     const int g=3,gi=332748118;//根据模数不同改变原根
5     using poly=vector<long long>;
6     long long ksm(long long a,long long b,long long mod){
7         long long res=1;
8         a%=mod;
9         while(b){
10             if(b&1) res=res*a%mod;
11             a=a*a%mod;
12             b>>=1;
13         }
14         return res;
15     }
16     int r[N];
17     int limit,L;
18     void init(int n,int m){
19         limit=1;L=0;
20         while(limit<=n+m) limit<<=1,L++;
21         for(int i=0;i<limit;i++){
22             r[i]=(r[i>>1]>>1)|((i&1)<<(L-1));
23         }
24     }
25     void ntt(poly &v,int type){
26         int len=v.size();
27         for(int i=0;i<len;i++){
28             if(i<r[i]) swap(v[i],v[r[i]]);
29         }
30         for(int mid=1;mid<len;mid<<=1){
31             long long wn=ksm(type==1?g:gi,(mod-1)/(mid<<1),mod);
32             for(int j=0,R=mid<<1;j<limit;j+=R){
33                 long long w=1;
34                 for(int k=0;k<mid;k++,w=(w*wn)%mod){
```

```
35         long long x=v[j+k],y=v[j+mid+k]*w%mod;
36         v[j+k]=(x+y)%mod;
37         v[j+k+mid]=(x-y+mod)%mod;
38     }
39 }
40 }
41 if(type==-1){
42     long long inv=ksm(len,mod-2,mod);
43     for(int i=0;i<len;i++){
44         v[i]=v[i]*inv%mod;
45     }
46 }
47 }
48 //反转多项式系数
49 void Reverse(poly &a){
50     reverse(a.begin(),a.end());
51 }
52 //多项式乘法
53 poly Mul(poly A,poly B,int len=0){
54     int n=A.size(),m=B.size();
55     if(len){
56         if(n>len){
57             A.resize(len),n=len;
58         }
59         if(m>len) B.resize(len),m=len;
60     }
61     if(n<=30||m<=30){
62         poly res(n+m-1);
63         for(int i=0;i<n;i++){
64             for(int j=0;j<m;j++){
65                 res[i+j]=(res[i+j]+A[i]*B[j]%mod)%mod;
66             }
67         }
68         if(len){
69             res.resize(len);
70         }
71         return res;
```

```
72     }
73     init(n,m);
74     A.resize(limit);B.resize(limit);
75     for(int i=min(n,m);i<limit;i++){
76         if(i>=n) A[i]=0;
77         if(i>=m) B[i]=0;
78     }
79     ntt(A,1);ntt(B,1);
80     for(int i=0;i<limit;i++){
81         A[i]=A[i]*B[i]%mod;
82     }
83     ntt(A,-1);
84     if(len==0){
85         A.resize(n+m-1);
86     }
87     else{
88         A.resize(len);
89     }
90     return A;
91 }
92 //求导
93 poly Direv(poly res){
94     for(int i=1;i<(int)res.size();i++){
95         res[i-1]=res[i]*i%mod;
96     }
97     res.pop_back();
98     return res;
99 }
100 //积分
101 poly Inter(poly res){
102     int len=res.size();
103     for(int i=len-1;i>=1;i--){
104         res[i]=res[i-1]*ksm(i,mod-2,mod)%mod;
105     }
106     res[0]=0;
107     return res;
108 }
```

```

109 //B(x)=2B'(x)-A(x)B'(x)B'(x)
110 poly GetInv(int n,poly &a){
111     if(n==1){
112         return {ksm(a[0],mod-2,mod)};
113     }
114     poly b=GetInv((n+1)>>1,a);
115     poly c=a;
116     int m=b.size();
117     init(n,n);
118     c.resize(limit);b.resize(limit);
119     for(int i=m;i<limit;i++){
120         b[i]=0;
121     }
122     for(int i=n;i<limit;i++){
123         c[i]=0;
124     }
125     ntt(b,1);ntt(c,1);
126     for(int i=0;i<limit;i++){
127         c[i]=(2-c[i]*b[i]%mod+mod)%mod;
128         c[i]=c[i]*b[i]%mod;
129     }
130     ntt(c,-1);
131     c.resize(n);
132     return c;
133 }
134 //多项式求逆
135 poly Inv(poly a){
136     int len=a.size();
137     return GetInv(len,a);
138 }
139 //b'x=a'x/ax
140 poly Ln(poly a){
141     int n=a.size();
142     a=Mul(Direv(a),Inv(a));
143     a.resize(n);
144     return Inter(a);
145 }

```



```

146 //gx=g'x(1-ln(g'x)+ax)
147 poly Exp(int n,poly &a){
148     if(n==1){
149         return {1};
150     }
151     poly b=Exp((n+1)>>1,a);
152     b.resize(n);
153     for(int i=(n+1)>>1;i<n;i++) {
154         b[i]=0;
155     }
156     poly e=Ln(b);
157     poly c=a;
158     init(n,n);
159     b.resize(limit);e.resize(limit);c.resize(limit);
160     for(int i=n;i<limit;i++){
161         c[i]=e[i]=b[i]=0;
162     }
163     ntt(b,1);ntt(c,1);ntt(e,1);
164     for(int i=0;i<limit;i++){
165         c[i]=b[i]*((1-e[i]+mod+c[i])%mod)%mod;
166     }
167     ntt(c,-1);
168     c.resize(n);
169     return c;
170 }
171 poly Exp(poly a){
172     return Exp((int)a.size(),a);
173 }
174 const int inv2=ksm(2,mod-2,mod);
175 //g(x)=(f(x)+g'x*g'x)/(2*g'x)
176 poly Sqrt(int n,poly &a){
177     if(n==1){
178         if(a[0]==1)
179             return {1};
180         else{
181             long long ans1,ans2;
182             cipolla res;

```

```

183         res.getsol(a[0],mod,ans1,ans2);
184         return {min(ans1,ans2)};
185     }
186 }
187 poly b=Sqrt((n+1)>>1,a);
188 b.resize(n);
189 for(int i=(n+1)>>1;i<n;i++) {
190     b[i]=0;
191 }
192 poly e=Inv(b);
193 poly c=a;
194 init(n,n);
195 b.resize(limit);c.resize(limit);e.resize(limit);
196 for(int i=n;i<limit;i++){
197     c[i]=b[i]=e[i]=0;
198 }
199 ntt(b,1);ntt(c,1);ntt(e,1);
200 for(int i=0;i<limit;i++){
201     c[i]=(c[i]+b[i]*b[i]%mod)%mod*inv2%mod*e[i]%
        mod;
202 }
203 ntt(c,-1);
204 c.resize(n);
205 return c;
206 }
207 poly Sqrt(poly a){
208     int n=(int)a.size();
209     return Sqrt(n,a);
210 }
211 //f(x)^k=e^(kln(f(x)))
212 poly Pow_1(poly a,long long k){
213     a=Ln(a);
214     for(int i=0;i<(int)a.size();i++){
215         a[i]=a[i]*k%mod;
216     }
217     return Exp(a);
218 }

```

```

219 //f(x)^k=(f(x)/f(t))^k*(f(t)^k) t为第一个不为0元素
220 poly Pow_2(poly a,long long k,long long k1){
221     int x=-1;
222     for(int i=0;i<(int)a.size();i++){
223         if(a[i]) {
224             x=i;
225             break;
226         }
227     }
228     int n=(int)a.size();
229     if(x==-1||x*k>n)
230     return vector<long long>(n,0);
231     poly res(n-x);
232     long long val=a[x];
233     long long inv=ksm(val,mod-2,mod);
234     for(int i=x;i<n;i++){
235         res[i-x]=a[i]*inv%mod;
236     }
237     res=Pow_1(res,k);
238     val=ksm(val,k1,mod);
239     x=x*k;
240     x=min(x,n);
241     for(int i=0;i<x;i++){
242         a[i]=0;
243     }
244     for(int i=x;i<n;i++){
245         a[i]=res[i-x]*val%mod;
246     }
247     return a;
248 }
249 poly Pow(poly a,long long k,long long k1){
250     return a[0]==1?Pow_1(a,k):Pow_2(a,k,k1);
251 }
252 //多项式除法 y=kx+b
253 void Div(poly y,poly x,poly &k,poly &b){
254     int n=y.size(),m=x.size();
255     Reverse(y);Reverse(x);

```

```
256     poly o=x;o.resize(n-m+1);
257     k=Mul(y,Inv(o),n-m+1);
258     Reverse(k);Reverse(y);Reverse(x);
259     b=Mul(k,x,m-1);
260     for(int i=0;i<m-1;i++){
261         b[i]=(mod-b[i]+y[i])%mod;
262     }
263 }
264 long long w4=ksm(g,(mod-1)/4,mod);
265 long long invw4=ksm(w4,mod-2,mod);
266 //欧拉公式变换得出
267 poly Cos(poly a){
268     int n=a.size();
269     poly b;
270     for(int i=0;i<n;i++){
271         a[i]=a[i]*w4%mod;
272     }
273     a=Exp(a);
274     b=Inv(a);
275     for(int i=0;i<n;i++){
276         a[i]=(a[i]+b[i])%mod*inv2%mod;
277     }
278     return a;
279 }
280 poly Sin(poly a){
281     int n=a.size();
282     poly b;
283     for(int i=0;i<n;i++){
284         a[i]=a[i]*w4%mod;
285     }
286     a=Exp(a);
287     b=Inv(a);
288     for(int i=0;i<n;i++){
289         a[i]=(a[i]+mod-b[i])%mod*inv2%mod*invw4%mod;
290     }
291     return a;
292 }
```

```
293 }  
294 using namespace Poly;
```

## 1.24 类欧几里得

定义  $f(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor$

在  $\log(n)$  时间内得出答案

```
1 #define ll long long
2 ll f(ll a, ll b, ll c, ll n) {
3     if(n <= 0) return 0;
4     return n * (n - 1) / 2 * (a / c) + n * (b / c) + f(c,
5         (a * n + b) % c, a % c, (a % c * n + b % c) / c);
6 }
```

## 1.24.1 拓展

再定义  $g(a, b, c, n) = \sum_{i=0}^n i \lfloor \frac{ai+b}{c} \rfloor$ ,  $h(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor^2$

可以一起得出来

```
1 const int mod=998244353;
2 using ll=long long;
3 const ll inv2=ksm(2,mod-2,mod),inv6=ksm(6,mod-2,mod);
4 struct Data
5 {
6     ll f,g,h;
7     Data(){
8         f=g=h=0;
9     }
10 };
11
12 Data cale(ll n,ll a,ll b,ll c){
13     ll ac=a/c,bc=b/c,m=(a*n+b)/c,n1=n+1,n21=n*2+1;
14     Data d;
15     if(a==0){//边界
16         d.f=bc*n1%mod;
17         d.g=bc*n%mod*n1%mod*inv2%mod;
18         d.h=bc*bc%mod*n1%mod;
19         return d;
20     }
21     if(a>=c||b>=c){//取模
```

```
22     d.f=n*n1%mod*inv2%mod*ac%mod+bc*n1%mod;
23     d.g=ac*n%mod*n1%mod*n21%mod*inv6%mod+bc*n%mod*n1%
        mod*inv2%mod;
24     d.h=ac*ac%mod*n%mod*n1%mod*n21%mod*inv6%mod+
25         bc*bc%mod*n1%mod+ac*bc%mod*n%mod*n1%mod;
26     d.f%=mod;d.g%=mod;d.h%=mod;
27     Data e=cale(n,a%c,b%c,c);//迭代
28     d.h+=e.h+2*bc%mod*e.f%mod+2*ac%mod*e.g%mod;
29     d.g+=e.g;d.f+=e.f;
30     d.f%=mod;d.g%=mod;d.h%=mod;
31     return d;
32 }
33 Data e=cale(m-1,c,c-b-1,a);
34 d.f=n*m%mod-e.f,d.f=(d.f%mod+mod)%mod;
35 d.g=m*n%mod*n1%mod-e.h-e.f,d.g=(d.g*inv2%mod+mod)%mod;
36 d.h=n*m%mod*(m+1)%mod-2*e.g-2*e.f-d.f;
37 d.h=(d.h%mod+mod)%mod;
38 return d;
39 }
```

## 1.25 区间维护等差数列

当有修改操作等需要给某一段区间加上等差数列，可以通过如下操作来进行维护，最后多用于求某单点值

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 /*
4 这是无离散化版本
5 离散化只需要将下有标的地方改成对应 id[下标]即可
6 */
7 const int N=1e5+10;
8 int n,q;
9 int a[N];
10 struct pp
11 {
12     long long a,d;
13 }tr[N<<2];
14
15 void build(int k,int l,int r)
16 {
17     tr[k].a=tr[k].d=0;
18     if(l==r)
19     {
20         tr[k].a=a[l];
21         return;
22     }
23     int mi=(l+r)>>1;
24     build(k<<1,l,mi);
25     build(k<<1|1,mi+1,r);
26 }
27
28 void down(int k,int l,int r)
29 {
30     int mi=(l+r)>>1;
31     tr[k<<1].d+=tr[k].d;
32     tr[k<<1|1].d+=tr[k].d;
33     tr[k<<1].a+=tr[k].a;
```



```
34     tr[k<<1|1].a+=tr[k].a+(mi+1-l)*tr[k].d;
35     tr[k].a=tr[k].d=0;
36 }
37
38 void mif(int k,int l,int r,int x,int y,int tp,long long a,
    long long d)
39 {
40     if(l==x&&r==y)
41     {
42         tr[k].d+=d;
43         tr[k].a+=a+(l-tp)*d;
44         return;
45     }
46     int mi=(l+r)>>1;
47     down(k,l,r);
48     if(y<=mi) mif(k<<1,l,mi,x,y,tp,a,d);
49     else if(mi<x) mif(k<<1|1,mi+1,r,x,y,tp,a,d);
50     else mif(k<<1,l,mi,x,mi,tp,a,d),mif(k<<1|1,mi+1,r,mi
        +1,y,tp,a,d);
51 }
52
53 long long qr(int k,int l,int r,int x)
54 {
55     if(l==x&&r==x)
56     {
57         return tr[k].a;
58     }
59     int mi=(l+r)>>1;
60     down(k,l,r);
61     if(x<=mi) return qr(k<<1,l,mi,x);
62     else return qr(k<<1|1,mi+1,r,x);
63 }
64
65 int main()
66 {
67     ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
68     cin>>n>>q;
```

```
69     for(int i=1;i<=n;i++)
70     {
71         cin>>a[i];
72     }
73     build(1,1,n);
74     while(q-->0)
75     {
76         int op;
77         cin>>op;
78         if(op==1)
79         {
80             int l,r;
81             long long a,d;
82             cin>>l>>r>>a>>d;
83             mif(1,1,n,l,r,l,a,d);
84         }
85         else{
86             int x;
87             cin>>x;
88             cout<<qr(1,1,n,x)<<'\n';
89         }
90     }
91     return 0;
92 }
```

## 1.26 随机化

### 1.26.1 随机数据生成器

```
1 mt19937 rng(chrono::steady_clock::now().time_since_epoch()  
    .count()); //uint  
2 mt19937_64 rng(chrono::steady_clock::now().  
    time_since_epoch().count()); //ull 大概率生成不相同数据  
3 int getRand(int l, int r){  
4     uniform_int_distribution < int > uid(l,r);  
5     return uid(rng);  
6 }
```

### 1.26.2 随机排列数组

```
1 random_shuffle(a,a+n);
```

## 1.27 二项式反演

可以总结归结为俩个式子:

$g_n$  表示至多有  $n$  个/种方案数量,  $f_n$  恰好  $n$  个/种方案数量

$$g_n = \sum_{i=0}^n C_n^i f_i \Leftrightarrow f_n = \sum_{i=0}^n (-1)^{n-i} C_n^i g_i$$

形式 2

$g_k$  表示至少有  $n$  个/种方案数量,  $f_k$  恰好  $n$  个/种方案数量

$$g_k = \sum_{i=k}^n C_i^k f_i \Leftrightarrow f_k = \sum_{i=k}^n (-1)^{i-k} C_i^k g_i$$

## 1.28 Pollard-Rho

```
1 #define LL long long
2 LL mul(LL a, LL b, LL p) {
3     return (a * b - (LL)(a / (long double)p * b + 1e-3) *
4         p + p) % p;
5 }
6 LL power(LL a, LL r, LL p) {
7     LL res = 1;
8     for (; r; a = mul(a, a, p), r >>= 1)
9         if (r & 1)
10             res = mul(res, a, p);
11 return res;
12 };
13 bool miller_rabin(LL n) {
14     static LL p[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
15     if (n == 1)
16         return false;
17     LL d = n - 1, r = 0;
18     for (; not(d & 1); d >>= 1)
19         r += 1;
20     bool res = true;
21     for (int i = 0; i < 9 and p[i] < n and res; i += 1) {
22         LL x = power(p[i], d, n);
23         if (x == 1 or x == n - 1)
24             continue;
25         for (int j = 1; j < r; j += 1) {
26             x = mul(x, x, n);
27             if (x == n - 1)
28                 break;
29         }
30         if (x != n - 1)
31             res = false;
32     }
33     return res;
34 };
35 vector<LL> pollard_rho(LL n) {
```

```
35     vector<LL> res;
36     function<void(LL)> rho = [&](LL n) {
37         if (n == 1)
38             return;
39         if (miller_rabin(n))
40             return res.push_back(n), void();
41         LL d = n;
42         while (d == n) {
43             d = 1;
44             for (LL k = 1, y = 0, x = 0, s = 1, c = rand()
45                 % n; d == 1;
46                 k <= 1, y = x, s = 1) {
47                 for (int i = 1; i <= k; i += 1) {
48                     x = (mul(x, x, n) + c) % n;
49                     s = mul(s, abs(x - y), n);
50                     if (not(i % 127) or i == k) {
51                         d = __gcd(s, n);
52                         if (d != 1)
53                             break;
54                     }
55                 }
56             }
57             rho(d);
58             rho(n / d);
59         };
60         rho(n);
61         return res;
62     }
```

## 1.29 杜教筛

能够在  $O(n^{\frac{2}{3}})$  的时间复杂度里求前  $n$  项积性函数的和

$g(1)s(n) = \sum_{i=1}^n (f * g)(i) - \sum_{i=2}^n g(i)s(\lfloor \frac{n}{i} \rfloor)$  前  $n$  项欧拉函数和莫比乌斯函数和

```

1  const int N=3e6+10;
2  int q[N],cnt;
3  long long u[N],o[N];
4  bool vis[N];
5  long long sumo1[N],sumu[N];
6  void init(int n=N-10)
7  {
8      vis[1]=true;
9      o[1]=1;
10     u[1]=1;
11     for(int i=2;i<=n;i++)
12     {
13         if(vis[i]==false)
14         {
15             q[cnt++]=i;
16             u[i]=-1;
17             o[i]=i-1;
18         }
19         for(int j=0;1ll*q[j]*i<=n;j++)
20         {
21             int nu=q[j]*i;
22             vis[nu]=true;
23             if(i%q[j]==0)
24             {
25                 u[nu]=0;
26                 o[nu]=o[i]*q[j];
27                 break;
28             }
29             u[nu]=-u[i];
30             o[nu]=o[i]*(q[j]-1);
31         }
32     }
33     for(int i=1;i<=n;i++)

```

```
34     {
35         sumol[i]=sumol[i-1]+o[i];
36         sumu[i]=sumu[i-1]+u[i];
37     }
38 }
39 map<long long,long long>mpo,mpu;
40 long long get_o(long long n)//前n项欧拉函数的和
41 {
42     if(n<=N-10) return sumol[n];
43     if(mpo.count(n)) return mpo[n];
44     long long ans=1ll*n*(n+1)/2;
45     for(long long int l=2,r;l<=n;l=r+1)
46     {
47         r=n/(n/l);
48         ans-=get_o(n/l)*(r-l+1);
49     }
50     return mpo[n]=ans;
51 }
52
53 long long get_u(long long n)//前n项莫比乌斯函数的和
54 {
55     if(n<=N-10) return sumu[n];
56     if(mpu.count(n)) return mpu[n];
57     long long ans=1;
58     for(long long int l=2,r;l<=n;l=r+1)
59     {
60         r=n/(n/l);
61         ans-=get_u(n/l)*(r-l+1);
62     }
63     return mpu[n]=ans;
64 }
```



## 1.30 min25 筛

两个条件：1.  $f(p^k)$   $p$  为质数可以用多项式表示，且项数比较少  
2.  $f(p^k)$  可以常数时间计算出这个值

```
1 long long f(long long x){
2     long long res=0;
3     return res;
4 }
5 int q[N],cnt;
6 bool vis[N];
7 long long sum[N];
8 void init(int n){
9     q[0]=1;
10    for(int i=2;i<=n;i++){
11        if(!vis[i]){
12            q[++cnt]=i;
13            sum[cnt]=sum[cnt-1]+f(i);
14        }
15        for(int j=1;1ll*q[j]*i<=n;j++){
16            int nu=q[j]*i;
17            vis[nu]=true;
18            if(i%q[j]==0){
19                break;
20            }
21        }
22    }
23 }
24 int id1[N],id2[N],m;
25 long long w[N];
26 long long n;
27 int T;
28 int find(long long x){
29     return x<=T?id1[x]:id2[n/x];
30 }
31 long long g[N];
32 long long s(long long x){
33     long long res=0;
```

```
34     return res;
35 }
36 //min25第一步质数积性函数和
37 void G(){
38     m=0;cnt=0;
39     T=sqrtl(n);
40     init(T);
41     for(long long l=1,r;l<=n;l=r+1){
42         long long k=n/l;r=n/k;
43         w[++m]=k;
44         if(k<=T) id1[k]=m;
45         else id2[n/k]=m;
46         g[m]=s(k);
47     }
48     for(int i=1;i<=cnt;i++){
49         for(int j=1;j<=m&&1ll*q[i]*q[i]<=w[j];j++){
50             int id=find(w[j]/q[i]);
51             g[j]=g[j]-f(q[i])*(g[id]-sum[i-1]);
52         }
53     }
54 }
55 //min25第二步除一之外数的积性函数的和
56 long long S(long long n,int k){
57     if(q[k]>=n) return 0;
58     int id=find(n);
59     long long ans=g[id]-sum[k];
60     for(int i=k+1;i<=cnt&&1ll*q[i]*q[i]<=n;i++){
61         long long p=q[i],pp=1ll*q[i]*q[i];
62         while(pp<=n){
63             long long F1=f(p),F2=f(pp);
64             ans=ans+F1*S(n/p,i)+F2;
65             p=pp;pp*=q[i];
66         }
67     }
68     return ans;
69 }
```

## 1.31 斯特林数

### 1.31.1 第一类斯特林数（斯特林轮换数）

意义：将  $n$  个不同元素，划分成  $m$  个非空圆排列的方案数记作  $\left[ \begin{smallmatrix} n \\ m \end{smallmatrix} \right]$

$$\text{递推式: } \left[ \begin{smallmatrix} n \\ m \end{smallmatrix} \right] = \left[ \begin{smallmatrix} n-1 \\ m-1 \end{smallmatrix} \right] + m * \left[ \begin{smallmatrix} n-1 \\ m \end{smallmatrix} \right]$$

### 1.31.2 第二类斯特林数（斯特林轮换数）

意义：将  $n$  个不同元素，划分成  $m$  个非空子集的方案数记作  $\left\{ \begin{smallmatrix} n \\ m \end{smallmatrix} \right\}$

$$\text{递推式: } \left\{ \begin{smallmatrix} n \\ m \end{smallmatrix} \right\} = \left\{ \begin{smallmatrix} n-1 \\ m-1 \end{smallmatrix} \right\} + m * \left\{ \begin{smallmatrix} n-1 \\ m \end{smallmatrix} \right\}$$

$$\text{通项: } s(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^i C_k^i (k-i)^n$$

$$\text{生成函数形式: } \sum_{n=0}^{\infty} s(n, k) \frac{x^n}{n!} = \frac{1}{k!} (\exp(x) - 1)^k$$

### 1.31.3 行

可以运用通项式得出： $\sum_{i=0}^k \frac{(-1)^i}{i!} \frac{(k-i)^n}{(k-i)!}$  的卷积形式，运用 NTT 快速求值

### 1.31.4 列

可以运用生成函数形式直接得出相对应系数，不过要注意最后的答案形式依旧是指数生成函数形式，所以对于相关系数记得乘对应系数的阶乘

### 1.31.5 下降幂

$$x^{\underline{k}} = C_x^k * k!$$

### 1.31.6 变换

$$x^n = \sum_{k=0}^n \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} x^{\underline{k}} = \sum_{k=0}^n \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} C_x^k * k! \quad x^{\underline{n}} = \sum_{k=0}^n \left[ \begin{smallmatrix} n \\ k \end{smallmatrix} \right] (-1)^{n-k} x^k$$

### 1.31.7 上升幂

$$x^{\overline{k}} = C_x^k * k!$$

### 1.31.8 变换

$$x^n = \sum_{k=0}^n \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} (-1)^{n-k} x^{\overline{k}} \quad x^{\overline{n}} = \sum_{k=0}^n \left[ \begin{smallmatrix} n \\ k \end{smallmatrix} \right] x^k$$

## 1.32 十二种球盒模型

```
1 using ll=long long;
2 //n个球,m个盒子
3 //球之间互不相同,盒子互不相同
4 ll cale1(ll n,ll m)//
5 {
6     /*
7         一个球可以选m个,n个就是(m)^n
8     */
9     if(n<0||m<0) return 0;
10    ll ans=ksm(m,n,mod);
11    return ans;
12 }
13 ll cale2(ll n,ll m)//每个盒子最多一个球
14 {
15     /*
16         让球从1-n被还没有选过的盒子选,便是m*(m-1)*...*(m-n
17         +1)=p(m,n)
18     */
19     if(n<0||m<0) return 0;
20    ll ans=P(m,n);
21    return ans;
22 }
23 ll cale3(ll n,ll m)//每个盒子至少装一个球
24 {
25     /*
26         考虑容斥
27     */
28     if(n<0||m<0) return 0;
29    ll ans=0;
30    for(int i=0,f=1;i<=m;i++,f=(mod-f)%mod){
31        ans=(ans+C(m,i)*f%mod*cale1(n,m-i)%mod)%mod;
32    }
33    return ans;
34 }
35 //球之间互不相同,盒子全部相同
```

```
35 ll cale4(ll n,ll m)//
36 {
37     /*
38         根据第二类斯特林数可以得到
39         n个不同元素,放在x个不为空的集合里面方案数
40         那么这个问题便可以枚举有几个非空集合得出答案
41     */
42     if(n<0||m<0) return 0;
43     ll ans=0;
44     poly a(m+1,0),b(m+1,0);
45     for(int i=0,f=1;i<=m;i++,f=(mod-f)%mod){
46         a[i]=f*inv[i]%mod;
47         b[i]=ksm(i,n,mod)*inv[i]%mod;
48     }
49     a=Mul(a,b);
50     for(int i=0;i<=m;i++){
51         ans=(ans+a[i])%mod;
52     }
53     return ans;
54 }
55 ll cale5(ll n,ll m)//每个盒子最多装一个球
56 {
57     /*
58         n个球都要被装完,只需要看盒子数量够不够装完n个球
59     */
60     if(n<0||m<0) return 0;
61     ll ans=(n<=m);
62     return ans;
63 }
64 ll cale6(ll n,ll m)//每个盒子至少一个球
65 {
66     /*
67         n个不同元素,分成m个非空集合方案数
68         第二类斯特林数通项直接可得
69     */
70     if(n<0||m<0) return 0;
71     ll ans=0;
```

```
72     for(int i=0,f=1;i<=m;i++,f=(mod-f)%mod){
73         ans=(ans+f*C(m,i)%mod*ksm(m-i,n,mod)%mod)%mod;
74     }
75     ans=ans*inv[m]%mod;
76     return ans;
77 }
78 //球全部相同,盒子之间互不相同
79 ll cale7(ll n,ll m)//
80 {
81     /*
82         隔板法,相当于 $a[1]+a[2]+...+a[m]=n$  条件( $a[i]\geq 0$ )
83         那么只要让 $(a[1]+1)+(a[2]+1)+...+(a[m]+1)=n+m$ 
84         便和 $a[1]+a[2]+...+a[m]=n$  条件( $a[i]\geq 1$ ) 形式一样
85     */
86     if(n<0||m<0) return 0;
87     ll ans=C(n+m-1,m-1);
88     return ans;
89 }
90 ll cale8(ll n,ll m)//每个盒子最多装一个球
91 {
92     /*
93         那么就可以转换成,从m个盒子里选n个出来放球
94     */
95     if(n<0||m<0) return 0;
96     ll ans=C(m,n);
97     return ans;
98 }
99 ll cale9(ll n,ll m)//每个盒子至少一个球
100 {
101     /*
102          $a[1]+a[2]+...+a[m]=n$  条件( $a[i]\geq 1$ )
103     */
104     if(n<0||m<0) return 0;
105     ll ans=C(n-1,m-1);
106     return ans;
107 }
108 //球全部相同,盒子全部相同
```

```

109 ll cale10(ll n,ll m)//
110 {
111     /*
112         便是构造一重无序且大小为m,集合元素和为n的方案数
113         就通过生成函数可以得到
114         当集合可能有v的时候
115         有  $0,1,2,3,\dots,\min(n/x),m$  这个集合可能有这么几种可能包含x
116         那么生成函数为  $x+x^v+x^{2v}\dots=(1-x^{(k+1)})/(1-x^v)$ 
117         去掉上界  $g(v)=1/(1-x^v)$ 
118         便是 $(1-n)$   $g(i)$ 的累乘,通过ln把乘法变加分之后再exp回来的第n项便是答案
119     */
120     if(n<0||m<0) return 0;
121     ll ans=0;
122     poly a(n+1,0);
123     for(int i=1;i<=m;i++){
124         for(int j=i;j<=n;j+=i){
125             a[j]=(a[j]+in[j/i])%mod;
126         }
127     }
128     a=Exp(a);
129     ans=a[n];
130     return ans;
131 }
132 ll cale11(ll n,ll m)//每个盒子最多装一个球
133 {
134     /*
135         取决于盒子能不能被放完
136     */
137     if(n<0||m<0) return 0;
138     ll ans=(n<=m);
139     return ans;
140 }
141 ll cale12(ll n,ll m)//每个盒子至少一个球
142 {
143     /*

```

```
144         先让每个盒子有一个球,然后问题同cale10一样
145     */
146     if(n<0||m<0) return 0;
147     ll ans=cale10(n-m,m);
148     return ans;
149 }
```



### 1.33 贝尔数

$bell(n)$  是将  $n$  个不同的球分成若干的集合的方案数

设:  $f(x)$  为非空集合 EGF,  $f(x) = e^x - 1$

$g(x)$  为贝尔数的 EGF, 那么因为多项式的复合性, 可得  $g(x) = e^{f(x)}$

### 1.34 欧拉数

具体意义: 一个大小为  $n$  排列中有  $k$  个位置使得  $p_i < p_{i+1} (i < n)$  递推式:

$$f(n, m) = f(n-1, m-1) * (n-k) + f(n-1, k) * (k+1)$$

$$\text{通项式: } f(n, m) = \sum_{k=0}^m C_{n+1}^k (-1)^k (m-k+1)^k$$

### 1.35 小公式

$$1. \ln(1-x^k) = -\sum_{i=1}^{\infty} \frac{x^{ik}}{i}$$

### 1.36 佩尔 (pell) 方程

$x^2 - d * y^2 = 1$  其中  $d$  为正整数且不为完全平方数时候, 可以借助本方程特性构建矩阵

```
1 mt[0][0]=x0;mt[0][1]=d1*y0;
2 mt[1][0]=y0;mt[1][1]=x0;
```

$$x_n = x_{n-1} * x_0 + d * y_0 * y_{n-1}$$

$$y_n = y_{n-1} * x_0 + y_0 * x_{n-1}$$

### 1.37 LGV 引理

结论: 当有一个起始点的子集  $A$ , 和一个终点子集  $B$ , 大小为  $m$  且一一对应那么  $A$  到  $B$  且路径俩俩互不相交的方案数为: 一个大小为  $m \times m$  的二维矩阵, 第  $i$  行第  $j$  列的值为  $a_i$  到  $b_j$  的方案数, 的矩阵行列式的值

### 1.38 伯特兰-切比雪夫定理

若整数  $n > 3$ , 则至少存在一个质数  $P$ , 符合  $n < P < 2 * n - 2$ , 一个稍弱说法, 一个大于 1 的整数, 至少存在一个质数  $P$ , 符合  $n < P < 2 * n$

## 1.39 基础博弈论

## 1 一. 巴什博弈 (Bash Game) :

2

3 A和B一块报数, 每人每次报最少1个, 最多报4个, 看谁先报到30。  
这应该是最古老的关于巴什博弈的游戏了吧。

4

5 其实如果知道原理, 这游戏一点运气成分都没有, 只和先手后手有关, 比如第一次报数, A报k个数, 那么B报 $5-k$ 个数, 那么B报数之后问题就变为, A和B一块报数, 看谁先报到25了, 进而变为20,15,10,5, 当到5的时候, 不管A怎么报数, 最后一个数肯定是B报的, 可以看出, 作为后手的B在这个游戏中是不会输的。

6

7 那么如果我们要报n个数, 每次最少报一个, 最多报m个, 我们可以找到这么一个整数k和r, 使 $n=k*(m+1)+r$ , 代入上面的例子我们就可以知道, 如果 $r=0$ , 那么先手必败; 否则, 先手必胜。

8

## 9 二. 威佐夫博弈 (Wythoff Game) :

10

11 有两堆各若千的物品, 两人轮流从其中一堆取至少一件物品, 至多不限, 或从两堆中同时取相同件物品, 规定最后取完者胜利。

12

13 直接说结论了, 若两堆物品的初始值为 $(x, y)$ , 且 $x < y$ , 则另 $z = y - x$ ;

14

15 记 $w = (\text{int}) [ ((\text{sqrt}(5) + 1) / 2) * z ]$ ;

16

17 若 $w = x$ , 则先手必败, 否则先手必胜。

18

## 19 三. 尼姆博弈 (Nimm Game) :

20

21 尼姆博弈指的是这样一个博弈游戏: 有任意堆物品, 每堆物品的个数是任意的, 双方轮流从中取物品, 每一次只能从一堆物品中取部分或全部物品, 最少取一件, 取到最后一件物品的人获胜。

22

23 结论就是: 把每堆物品数全部异或起来, 如果得到的值为0, 那么

先手必败，否则先手必胜。

24

25 四．斐波那契博弈：

26

27 有一堆物品，两人轮流取物品，先手最少取一个，至多无上限，但不能把物品取完，之后每次取的物品数不能超过上一次取的物品数的二倍且至少为一件，取走最后一件物品的人获胜。

28

29 结论是：先手胜当且仅当 $n$ 不是斐波那契数（ $n$ 为物品总数）

## 2 几何

能不用浮点就不用

### 2.1 几何通用

```
1 // #define double long double // wa 了就试一下
2 const double eps=1e-12;
3 const double pi=acos(-1);
4 int cmpb(double x)
5 {
6     if(fabs(x)<eps) return 0;
7     if(x>0) return 1;
8     return -1;
9 }
10 struct point{
11     double x,y;
12     point(){x=0;y=0;}
13     point(double _x,double _y){x=_x;y=_y;};
14     double operator |(const point &w) const // 点乘
15     {
16         return x*w.x+y*w.y;
17     }
18     double operator ^(const point &w) const // 叉乘
19     {
20         return x*w.y-y*w.x;
21     }
22     point operator -(const point &w) const
23     {
24         point res;
25         res.x=x-w.x;res.y=y-w.y;
26         return res;
27     }
28     point operator +(const point &w) const
29     {
30         point res;
31         res.x=x+w.x;res.y=y+w.y;
```

```
32         return res;
33     }
34     point operator /(double w) const
35     {
36         point res(x/w,y/w);
37         return res;
38     }
39     point operator *(double w) const
40     {
41         point res(x*w,y*w);
42         return res;
43     }
44     double db(double x)    //平方
45     {
46         return x*x;
47     }
48     double dict(point &to)//点距
49     {
50         return sqrt(db(x-to.x)+db(y-to.y));
51     }
52     double dz()
53     {
54         return sqrt(db(x)+db(y));
55     }
56     void rotate(double _o) //旋转都是逆时针
57     {
58         double _x=x,_y=y;
59         x=_x*cos(_o)-_y*sin(_o);
60         y=_x*sin(_o)+_y*cos(_o);
61     }
62     double arg()//绕正x轴旋转多少度到达
63     {
64         int xb=cmpb(x),yb=cmpb(y);
65         if(xb==0&&yb==0) return 0.0;
66         if(yb==0)
67         {
68             if(xb==1) return 0.0;
```

```
69         else return pi;
70     }
71     if(xb==0)
72     {
73         if(yb==1) return pi/2.0;
74         else return 1.5*pi;
75     }
76     double ta=fabs(atan(y/x));
77     if(xb==1&&yb==1) ta+=1.5*pi;
78     else if(xb==1&&yb==0) ta+=pi;
79     else if(xb==0&&yb==1) ta+=0.5*pi;
80     return ta;
81 }
82 };
83 struct Line{
84     double a,b,c;    //直线一般式
85     Line(){a=0;b=0;c=0;}
86     Line(double _a,double _b,double _c)
87     {
88         a=_a;
89         b=_b;
90         c=_c;
91     }
92     Line(point &x,point &y)//俩点一线
93     {
94         a=y.y-x.y;
95         b=x.x-y.x;
96         c=x.y*y.x-x.x*y.y;
97     }
98     point cross(Line &x)    //线交点
99     {
100         point res;
101         res.x=(x.c*b-c*x.b)/(a*x.b-x.a*b);
102         res.y=(x.c*a-c*x.a)/(b*x.a-x.b*a);
103         return res;
104     }
105     bool check(point &x)//判断点是否在线上
```

```
106     {
107         return cmpb(x.x*a+x.y*b+c)==0;
108     }
109     Line getL(point &x){//获得得某点到这个条线的垂直线
110         double a1,b1,c1;
111         if(a==0) {
112             a1=1.0;
113             b1=0;
114             c1=-x.x;
115         }
116         else if(b==0)
117         {
118             a1=0;
119             b1=1.0;
120             c1=-x.y;
121         }
122         else{
123             a1=b;
124             b1=-a;
125             c1=a*x.y-b*x.x;
126         }
127         Line re(a1,b1,c1);
128         return re;
129     }
130     bool Px(Line &w)//判断平行
131     {
132         return cmpb(a*w.b-b*w.a)==0;
133     }
134     bool Li(Line &w)//判断垂直
135     {
136         return cmpb(a*w.b+b*w.a)==0;
137     }
138 };
139 struct Line //起点 终点版
140 {
141     point s,e;
142     Line(){}

```

```

143     Line(point _s,point _e){s=_s;e=_e;}
144     point cross(Line &W) {
145         point u=s-W.s,v=e-s,w=W.e-W.s;
146         double t=(u^w)/(w^v);
147         return s+v*t;
148     }
149     bool right(Line b,Line c)//判断b,c交点是否在当前线段右
        边
150     {
151         point p=b.cross(c);
152         return ((e-s)^(p-s))<0;
153     }
154     double arg() //极角 [-pi,pi]
155     {
156         return atan2(e.y-s.y,e.x-s.x);
157     }
158 };
159 double cro(point a,point b,point c)
160 {
161     return (b-a)^(c-a);
162 }
163
164 double dot(point a,point b,point c)
165 {
166     return (b-a)!(c-a);
167 }
168
169 struct point3{//三维
170     double x,y,z;
171     point3()
172     {
173         x=0.0;y=0.0;z=0.0;
174     }
175     point3(double _x,double _y,double _z)
176     {
177         x=_x;y=_y;z=_z;
178     }

```



```
179 point3 operator -(const point3 w)
180 {
181     return (point3){x-w.x,y-w.y,z-w.z};
182 }
183 point3 operator +(const point3 w)
184 {
185     return (point3){x+w.x,y+w.y,z+w.z};
186 }
187 point3 operator ^(const point3 w)
188 {
189     return (point3){(y*w.z-z*w.y),(z*w.x-x*w.z),(x*w.y
        -y*w.x)};
190 }
191 double operator |(const point3 w)
192 {
193     return x*w.x+y*w.y+z*w.z;
194 }
195 double rand01()
196 {
197     return (double)rand()/(double)RAND_MAX;
198 }
199 double randeps()
200 {
201     return (double)(rand01()-0.5)*eps;
202 }
203 void noise()//三维点需要抖动一下避免有一些影响 十分重
    要！
204 {
205     x+=randeps();y+=randeps(),z+=randeps();
206 }
207 double db(double x)
208 {
209     return x*x;
210 }
211 double dict(const point3 w)
212 {
213     return sqrt(db(x-w.x)+db(y-w.y)+db(z-w.z));
```

```
214     }  
215     double mod()  
216     {  
217         return sqrt(x*x+y*y+z*z);  
218     }  
219 };
```

## 2.2 极角序

```
1 //排序cmp
2 struct argcmp{//极角排序
3     int quad(const point&x) //象限排序 根据题目定义
4     {
5         if(x.y<=-eps) return 1;
6         else if(x.y>eps) return 3;
7         else if(x.x>eps) return 4;
8         else if(x.x<=-eps) return 5;
9         return 2;
10    };
11    bool operator()(const point &x,const point &y)
12    {
13
14        int xb=quad(x);
15        int yb=quad(y);
16        if(xb!=yb)
17        {
18            return xb<yb;
19        }
20        return (x^y)>eps; //右手定则
21    }
22 };
```

## 2.3 凸包

### 2.3.1 二维凸包

```
1  const int N=4e5+10;
2  bool cmp(point x,point y)
3  {
4      return (x.x==y.x?x.y<y.y:x.x<y.x);
5  }
6  point st[N],p[N];
7  int top,tp;
8  double Andrew()
9  {
10     top=0; //多组清空
11     sort(p+1,p+tp+1,cmp);
12     for(int i=1;i<=tp;i++)
13     {
14         while(top>1&&((st[top]-st[top-1])^(p[i]-st[top-1]))
15             <=0.0) top--;
16         st[++top]=p[i];
17     }
18     int t=top;
19     for(int i=tp-1;i>=1;i--)
20     {
21         while(top>t&&((st[top]-st[top-1])^(p[i]-st[top-1]))
22             <=0.0) top--;
23         st[++top]=p[i];
24     }
25     double ans=0.0;
26     for(int i=1;i<top;i++)
27     {
28         ans=ans+st[i].dict(st[i+1]);
29     }
30     return ans;
31 }
```

## 2.4 三维凸包

### 2.4.1 暴力

```
1  const int N=110;
2  point3 p[N];
3  int n;
4  bool check(int id,int id1,int id2)
5  {
6      double now=0;
7      int c1=0,c2=0;
8      point3 vr=(p[id1]-p[id])^(p[id2]-p[id]);
9      for(int i=1;i<=n;i++)
10     {
11         if(i==id||i==id1||i==id2) continue;
12         now=((p[i]-p[id])lvr);
13         if(now>0)
14         {
15             c1++;
16         }
17         else if(now<0){
18             c2++;
19         }
20         if(c1&& c2) return false;
21     }
22     return true;
23 }
24 void solve()
25 {
26     scanf("%d",&n);
27     for(int i=1;i<=n;i++)
28     {
29         scanf("%Lf%Lf%Lf",&p[i].x,&p[i].y,&p[i].z);
30         p[i].noise();
31     }
32     double ans=0;
```

```
34     for(int i=1;i<=n;i++)
35     {
36         for(int j=i+1;j<=n;j++)
37         {
38             for(int k=j+1;k<=n;k++)
39             {
40                 if(check(i,j,k))
41                 {
42                     ans+=((p[j]-p[i])^(p[k]-p[i])).mod()
                        /2.0; //通过向量积的模长一半求得出
43                 }
44             }
45         }
46     }
47     printf("%Lf",ans);
48 }
```

## 2.5 增量法

```

1  const int N=2010;
2  point3 p[N];
3  struct face{
4      int v[3];
5      face(){v[0]=v[1]=v[2]=0;}
6      face(int a,int b,int c){v[0]=a;v[1]=b;v[2]=c;}
7      point3 normal(){return ((p[v[1]]-p[v[0]])^(p[v[2]]-p[v[0]]));}
8      double area(){return normal().mod()/2.0;}
9  };
10 int cansee(face A,point3 b) {return ((b-p[A.v[0]])|A.
    normal())>0;}
11 face cv[N],h[N];
12 int n,th,cnt,vis[N][N];
13 void convex3() {
14     cnt=th=0;
15     cv[++cnt]=face(1,2,3);
16     cv[++cnt]=face(3,2,1);
17     for(int i=4;i<=n;++i) {
18         for(int j=1;j<=cnt;++j) {
19             if(!(v=cansee(cv[j],p[i])))h[++th]=cv[j];
20             for(int k=0;k<3;++k)
21                 vis[cv[j].v[k]][cv[j].v[k>1?0:k+1]]=v;
22         }
23         for(int j=1;j<=cnt;++j)
24             for(int k=0;k<3;++k) {
25                 int x=cv[j].v[k],y=cv[j].v[k>1?0:k+1];
26                 if(vis[x][y]&&!vis[y][x])h[++th]=face(x,y,
                    i);
27             }
28         for(int j=1;j<=th;++j)cv[j]=h[j];
29         cnt=th,th=0;
30     }
31 }
32 double calc(){

```

```
33     double res=0;
34     for(int i=1;i<=cnt;i++)
35     {
36         res+=cv[i].area();
37     }
38     return res;
39 }
40 double vol6(point3 a,point3 b,point3 c,point3 d) {
41     return ((b-a)^(c-a))|(d-a);
42 } //计算体积的6倍。理解：先算出底的有向面积的2倍，向量的方向变为高，然后点乘就是另一条边在高方向的投影，由体积公式( $V=1/3*Sh$ )，这个值就是体积的6倍
43 double calcV() { //体积
44     double res=0;
45     for(int i=1;i<=cnt;++i)
46         res+=fabs(vol6(p[1],p[cv[i].v[0]],p[cv[i].v[1]],p[cv[i].v[2]]));
47     res/=6.0;
48     return res;
49 }
```



## 2.6 线段到线段最小距离

```
1 double cro(point a,point b,point c)
2 {
3     return ((b-a)^(c-a));
4 }
5
6 double dot(point p0,point p1,point p2){ //点积 p0为角点
7     return (p1-p0)!(p2-p0);
8 }
9
10 double getDis(point p0,point p1,point p2){
11     if(cmpb(p0.dict(p1))==0) return p0.dict(p2); //p0 p1共
        同点
12     if(cmpb(dot(p0,p1,p2))== -1) return p2.dict(p0); //钝角
13     if(cmpb(dot(p1,p0,p2))== -1) return p2.dict(p1); //钝角
14     return fabs(cro(p0,p1,p2)/p0.dict(p1)); //点到线的距
        离
15 }
16
17 double minDis(point p1,point p2,point p3,point p4){ //线段
        间最短距离
18     return min(min(getDis(p1,p2,p3),getDis(p1,p2,p4)),min
        (getDis(p3,p4,p1),getDis(p3,p4,p2)));
19 }
```

## 2.7 旋转卡壳

### 2.7.1 凸包直径

```
1 double rotating_calipers()
2 {
3     top--;
4     double ans=0;
5     for(int i=0,j=1;i<top;i++)
6     {
7         while(((st[(i+1)%top]-st[i])^(st[j]-st[i]))<
8             ((st[(i+1)%top]-st[i])^(st[(j+1)%top]-st[i]))){
9             j=(j+1)%top;
10        }
11        ans=max(ans,st[i].dict(st[j]));
12        ans=max(ans,st[(i+1)%top].dict(st[j]));
13    }
14    return ans;
15 }
```

## 2.7.2 俩个不相交凸包最小距离

```

1  double cro(point a,point b,point c)
2  {
3      return ((b-a)^(c-a));
4  }
5
6  double dot(point p0,point p1,point p2){ //点积 p0为角点
7      return (p1-p0)!(p2-p0);
8  }
9
10 double getDis(point p0,point p1,point p2){
11     if(cmpb(p0.dict(p1))==0) return p0.dict(p2); //p0 p1共
        同点
12     if(cmpb(dot(p0,p1,p2))== -1) return p2.dict(p0); //钝角
13     if(cmpb(dot(p1,p0,p2))== -1) return p2.dict(p1); //钝角
14     return fabs(cro(p0,p1,p2)/p0.dict(p1)); //点到线的距
        离
15 }
16
17 double minDis(point p1,point p2,point p3,point p4){ //线段
        间最短距离
18     return min(min(getDis(p1,p2,p3),getDis(p1,p2,p4)),min
        (getDis(p3,p4,p1),getDis(p3,p4,p2)));
19 }
20
21 double r_c(int f)
22 {
23     int ymi=1,ymx=1;
24     for(int i=1;i<=top[f];i++)
25     {
26         if(st[f][i].y<st[f][ymi].y) ymi=i;
27     }
28     for(int i=1;i<=top[f^1];i++)
29     {
30         if(st[f^1][i].y>st[f^1][ymx].y) ymx=i;
31     } //分别最小和最大的开始找就保证的下面那些情况保证在凸

```

多边形靠近的那附件来旋转

```

32  double t=0,ans=1e20;
33  for(int i=1;i<=top[f];i++){
34      //因为是逆时针 所以所以旋转到 $acb>acd$ 时候 就相当于
        (l /) (\ /) (\ l) 这种情况
35      while(t=cro(st[f][ymi],st[f][ymi+1],st[f^1][ymx])-
        cro(st[f][ymi],st[f][ymi+1],st[f^1][ymx+1])<=
        eps)
36      {
37          ymx=ymx%top[f^1]+1;
38      }
39      if(t>eps) ans=min(ans,getDis(st[f][ymi],st[f][ymi
        +1],st[f^1][ymx]));
40      else ans=min(ans,minDis(st[f][ymi],st[f][ymi+1],st
        [f^1][ymx],st[f^1][ymx+1]));
41      ymi=ymi%top[f]+1;
42  }
43  return ans;
44  }
```

## 2.7.3 最小矩阵覆盖

```

1  double r_c()
2  {
3      double ans=1e20;
4      int a,b,c;
5      a=b=c=2;
6      int n=top-1;
7      for(int i=2;i<=top;i++)
8      {
9          while(cro(st[i-1],st[i],st[a])<cro(st[i-1],st[i],
              st[a%n+1])) a=a%n+1;
10         while(dot(st[i-1],st[i],st[b])<dot(st[i-1],st[i],
              st[b%n+1])) b=b%n+1;
11         if(i==2) c=a;
12         while(dot(st[i],st[i-1],st[c])<dot(st[i],st[i-1],
              st[c%n+1])) c=c%n+1;
13         double H=cro(st[i-1],st[i],st[a]);
14         double d=st[i].dict(st[i-1]);
15         H/=d;
16         double R=dot(st[i-1],st[i],st[b])/d,L=dot(st[i],st
              [i-1],st[c])/d;
17         if(H*(R+L-d)<ans)
18         {
19             ans=H*(R+L-d);
20             res[0]=st[i-1]+(st[i]-st[i-1])*R/d;
21             res[3]=st[i]+(st[i-1]-st[i])*L/d;
22             res[1]=res[3]-res[0];
23             res[1].rotate(-pi/2.0);
24             res[1]=res[1]*H/(L+R-d);
25             res[1]=res[1]+res[0];
26             res[2]=res[0]-res[3];
27             res[2]=res[2]*H/(L+R-d);
28             res[2].rotate(pi/2.0);
29             res[2]=res[2]+res[3];
30         }
31     }

```

```
32     }  
33     return ans;  
34 }
```

## 2.8 闵可夫斯基和

$A, B$  两个凸包的闵可夫斯基和为  $C = a + b | a \in A, b \in B$

```

1  const int N=4e5+10;
2  bool cmp(point x,point y)
3  {
4      return (x.x==y.x?x.y<y.y:x.x<y.x);
5  }
6  point st[3][N],p[3][N];
7  int top[3],tp[3];
8  void Andrew(int id)
9  {
10     top[id]=0;//多组清空
11     sort(p[id]+1,p[id]+tp[id]+1,cmp);
12     for(int i=1;i<=tp[id];i++)
13     {
14         while(top[id]>1&&((st[id][top[id]]-st[id][top[id]-1])^(p[id][i]-st[id][top[id]-1]))<=0.0) top[id]--;
15         st[id][++top[id]]=p[id][i];
16     }
17     int t=top[id];
18     for(int i=tp[id]-1;i>=1;i--)
19     {
20         while(top[id]>t&&((st[id][top[id]]-st[id][top[id]-1])^(p[id][i]-st[id][top[id]-1]))<=0.0) top[id]--;
21         st[id][++top[id]]=p[id][i];
22     }
23     top[id]--;
24 }
25 point s1[N],s2[N];
26 void Minkovski(int id,int id1,int id2)//a={b+c}
27 {
28     for(int i=1;i<top[id1];i++) s1[i]=st[id1][i+1]-st[id1][1];s1[top[id1]]=st[id1][1]-st[id1][top[id1]];//s1
    是id1凸包

```

```

29     for(int i=1;i<top[id2];i++) s2[i]=st[id2][i+1]-st[id2
        ][i];s2[top[id2]]=st[id2][1]-st[id2][top[id2]];//s2
        是id2凸包
30     int p1=1,p2=1;
31     tp[id]=1;
32     p[id][tp[id]]=st[id1][1]+st[id2][1];//归并加入一下，p[
        id]数组就是闵可夫斯基和
33     while(p1<=top[id1]&& p2<=top[id2]) tp[id]++,p[id][tp[id]
        ]=p[id][tp[id]-1]+((s1[p1]^s2[p2])>=0?s1[p1++]:s2[
        p2++]);//极角序从小到大并从p[id][top[id]-1]开始往后
        接
34     while(p1<=top[id1]) tp[id]++,p[id][tp[id]]=p[id][tp[id]
        ]-1]+s1[p1++];
35     while(p2<=top[id2]) tp[id]++,p[id][tp[id]]=p[id][tp[id]
        ]-1]+s2[p2++];
36 }
37 bool cmp2(point a,point b)//判断
38 {
39     return (a^b)>0||(a^b)==0&& a.dz(< b.dz());
40 }
41 int in(point a,int id) //通过极角序来判断点是否在凸包中
42 {
43     if((a^st[id][1])>0||(st[id][top[id]]^a)>0) return 0;
44     int to=lower_bound(st[id]+1,st[id]+top[id]+1,a,cmp2)-
        st[id]-1;
45     return ((a-st[id][to])^(st[id][to%top[id]+1]-st[id][to
        ]))<=0;
46 }

```



## 2.9 半平面交

```

1  const int N=4e5+10;
2  Line le[N],dq[N];
3  int pt;
4  bool cmphp(Line a,Line b)//极角+左侧
5  {
6      double A=a.arg(),B=b.arg();
7      return cmpb(A-B)!=0?A<B:((a.e-a.s)^(b.e-a.s))<0.0;
8  }
9  void half_plane()//半平面交
10 {
11     sort(le+1,le+pt+1,cmphp);
12     int h=1,t=1;dq[1]=le[1];
13     for(int i=2;i<=pt;i++)
14     {
15         if(le[i].arg()-le[i-1].arg()<eps) continue;
16         while(h<t&&le[i].right(dq[t],dq[t-1])) t--;//维护
            头尾
17         while(h<t&&le[i].right(dq[h],dq[h+1])) h++;
18         dq[++t]=le[i];
19     }
20     while(h<t&&dq[h].right(dq[t],dq[t-1])) t--; //割多余尾
        巴
21 // dq[++t]=dq[h];//封口 看题目需要
22 }

```

## 2.10 最小圆覆盖

```
1  const int N=1e5+10;
2  int n;
3  point p[N];
4  point gauss(point x,point y,point z){
5      double c1=x.db(x.dz())-y.db(y.dz());
6      double c2=x.db(x.dz())-z.db(z.dz());
7      point e=x-y,e1=x-z;
8      point o;
9      o.x=(c1*e1.y-c2*e.y)/(2.0*(e.x*e1.y-e1.x*e.y));
10     o.y=(c1*e1.x-c2*e.x)/(2.0*(e.y*e1.x-e1.y*e.x));
11     return o;
12 }
13 void getcircle(point &o,double &r){
14     random_shuffle(p,p+n);
15     o=p[0];
16     for(int i=1;i<n;i++){
17         if(cmpb(r-o.dict(p[i]))== -1){
18             o=p[i];
19             r=0.0;
20             for(int j=0;j<i;j++){
21                 if(cmpb(r-o.dict(p[j]))== -1){
22                     o=(p[i]+p[j])/2.0;
23                     r=o.dict(p[j]);
24                     for(int k=0;k<j;k++){
25                         if(cmpb(r-o.dict(p[k]))== -1){
26                             o=gauss(p[i],p[j],p[k]);
27                             r=o.dict(p[k]);
28                     }
29                 }
30             }
31         }
32     }
33 }
34 }
```

## 2.11 三角形和圆交面积

拓展：多边形和圆的相交面积可以使多边形三角剖分之后逐步求面积

```

1  double R;//半径
2  point o;//圆心
3  bool onSeg(point p,point a,point b){
4      return cmpb((a-p)^(b-p))==0&&((a-p)|(b-p))<=eps;
5  }
6  point getNode(point a,point u,point b,point v){
7      double t=((a-b)^v)/(v^u);
8      return a+(u*t);
9  }
10 double getDP2(point a,point b,point &pa,point &pb){
11     point i=(b-a);
12     i.rotate(pi/2);
13     point e=getNode(a,b-a,o,i);//垂足
14     double d=o.dict(e);
15     if(!onSeg(e,a,b)) d=min(o.dict(a),o.dict(b));
16     if(R<=d) return d;
17     double len=sqrt(R*R-o.dict(e)*o.dict(e));
18     pa=e+(a-b).norm()*len;
19     pb=e+(b-a).norm()*len;
20     return d;
21 }
22 //圆上扇形面积
23 double sector(point a,point b){
24     double angle=acos(((a)|(b))/a.dz()/b.dz());
25     if(cmpb((a)^(b))== -1) angle=-angle;
26     return R*R*angle/2.0;
27 }
28 double getArea(point a,point b){
29     if(cmpb(a^b)==0) return 0;
30     double da=a.dict(o),db=b.dict(o);
31     if(R>=da&&R>=db) return (a^b)/2;//三角形在圆内
32     point pa,pb;
33     double d=getDP2(a,b,pa,pb);
34     if(R<=d) return sector(a,b);//ab在圆外

```

```
35     if(R>=da) return (a^pb)/2+sector(pb,b); //a在圆内
36     if(R>=db) return sector(a,pa)+(pa^b)/2;
37     return (pa^pb)/2+sector(a,pa)+sector(pb,b); //ab是割线
38 }
39 double GetArea(point a,point b,point c){
40     return fabs(getArea(a,b)+getArea(b,c)+getArea(c,a));
41 }
```

### 2.12 二维向量绕 0 点旋转 n 度

```
1 x1=x*cos(n)-y*sin(n);  
2 y1=x*sin(n)+y*cos(n);
```

### 2.13 椭圆

a 为长半轴 b 为短半轴 周长为  $L=2\pi b+4(a-b)$  面积为  $S=\pi ab$

### 2.14 皮克定理

格点三角形的面积  $S = n + m/2 - 1$   $n$  表示三角形内部点的格点数量  $m$  表示三角形边上的顶点数

### 2.15 浮点输出

```
1 cout<<fixed<<setprecision(2)<<ans;
```

## 2.16 扫描线

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=2e5+10;
4  struct Line
5  {
6      int x,y1,y2,f,id1,id2;
7      bool operator<(const Line &le1)
8      {
9          return x<le1.x;
10     }
11 }le[N];
12 vector<int>b;
13 int n;
14 int find(int x)
15 {
16     return lower_bound(b.begin(),b.end(),x)-b.begin()+1;
17 }
18 struct pp
19 {
20     long long le;
21     int f;
22 }tr[N<<4];
23 void up(int k,int l,int r)
24 {
25     if(!tr[k].f){
26         tr[k].le=tr[k<<1].le+tr[k<<1|1].le;
27     }
28     else{
29         tr[k].le=b[r]-b[l-1];
30     }
31 }
32 void mif(int k,int l,int r,int x,int y,int f)
33 {
34     if(l==x&&r==y)
35     {
```

```
36         tr[k].f+=f;
37         up(k,l,r);
38         return;
39     }
40     int mi=(l+r)>>1;
41     if(y<=mi) mif(k<<1,l,mi,x,y,f);
42     else if(mi<x) mif(k<<1|1,mi+1,r,x,y,f);
43     else mif(k<<1,l,mi,x,mi,f),mif(k<<1|1,mi+1,r,mi+1,y,f)
44         ;
45     up(k,l,r);
46 }
47 int main()
48 {
49     ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
50     cin>>n;
51     for(int i=1;i<=n;i++)
52     {
53         int x,y,x1,y1;
54         cin>>x>>y>>x1>>y1;
55         le[i]={x,y,y1,1};
56         le[n+i]={x1,y,y1,-1};
57         b.push_back(y);
58         b.push_back(y1);
59     }
60     sort(b.begin(),b.end());
61     b.erase(unique(b.begin(),b.end()),b.end());
62     n<=1;
63     sort(le+1,le+n+1);
64     for(int i=1;i<=n;i++)
65     {
66         le[i].id1=find(le[i].y1);
67         le[i].id2=find(le[i].y2);
68     }
69     long long ans=0;
70     int m=(int)b.size();
71     for(int i=1;i<n;i++)
72     {
```

```
72         mif(1,1,m,le[i].id1,le[i].id2-1,le[i].f);
73         ans+=1ll*tr[1].le*(le[i+1].x-le[i].x);
74     }
75     cout<<ans;
76     return 0;
77 }
```



## 2.17 平面最近点对

```
1  const int N=1e6+10;
2  int n;
3  point p[N],st[N];
4  bool cmpx(point &x,point &y){
5      return x.x<y.x;
6  }
7  bool cmpy(point &x,point &y){
8      return x.y<y.y;
9  }
10 double find(int l,int r){
11     if(l==r) return 1e18;
12     if(r-l==1){
13         return p[l].dict(p[r]);
14     }
15     int mi=(l+r)>>1;
16     double d=min(find(l,mi),find(mi+1,r));
17     int tp=0;
18     for(int i=l;i<=r;i++){
19         if(cmpb(d-fabs(p[i].x-p[mi].x))==1){
20             st[++tp]=p[i];
21         }
22     }
23     sort(st+1,st+tp+1,cmpy);
24     for(int i=1;i<tp;i++){
25         for(int j=i+1;j<=tp&&st[j].y-st[i].y<d;j++){
26             d=min(d,st[i].dict(st[j]));
27         }
28     }
29     return d;
30 }
```

## 2.18 自适应辛普森公式

分成很多小段把一小段当成二次函数，通过分治进行求解

```
1  const double eps=1e-12;
2  int cmpb(double x)
3  {
4      if(fabs(x)<eps) return 0;
5      if(x>0) return 1;
6      return -1;
7  }
8  double f(double x)//对应题目函数
9  {
10     double ans;
11     return ans;
12 }
13 double simpson(double l,double r){//辛普森公式
14     return (r-l)*(f(r)+f(l)+4.0*f((r+l)/2.0))/6.0;
15 }
16 double asr(double l,double r,double ans){//自适应
17     double mi=(l+r)/2.0;
18     double a=simpson(l,mi),b=(simpson(mi,r));
19     if(cmpb(a+b-ans)==0) return ans;
20     return asr(l,mi,a)+asr(mi,r,b);
21 }
```

## 3 其它

### 3.1 快读快写开 02

```
1 #pragma GCC optimize(2)
2 #pragma GCC optimize(3,"Ofast","inline")
3
4 inline int read()
5 {
6     char c=getchar();int x=0,f=1;
7     while(c<'0' || c>'9'){if(c=='-')f=-1;c=getchar();}
8     while(c>='0'&&c<='9'){x=x*10+c-'0';c=getchar();}
9     return x*f;
10 }
11 inline void write(int x)
12 {
13     static int t[25];int tp=0;
14     if(x==0) return(void)(putchar('0'));else if(x<0)
        putchar('-'),x=-x;
15     while(x) t[tp++]=x%10,x/=10;
16     while(tp--) putchar(t[tp]+48);
17 }
```

## 3.2 bitset 处理多维偏序

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=5010,M=510;
5  int n,m;
6  int mp[M][N];
7  long long p[N],dp[N];
8  bitset<N>tp[N];
9  vector<int>tr[N];
10 int in[N];
11 typedef pair<int,int> pii;
12 bool cmp(pii x,pii y)
13 {
14     if(x.first==y.first)
15     {
16         return x.second<y.second;
17     }
18     return x.first<y.first;
19 }
20 int main()
21 {
22     ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
23     cin>>n>>m;
24     for(int i=1;i<=m;i++)
25     {
26         cin>>p[i];
27         dp[i]=p[i];
28     }
29     for(int i=1;i<=n;i++)
30     {
31         for(int j=1;j<=m;j++)
32         {
33             cin>>mp[i][j];
34         }
35     }
```

```
36     for(int i=1;i<=m;i++)
37     {
38         for(int j=1;j<=m;j++){
39             tp[i][j]=1;
40         }
41     }
42     for(int i=1;i<=n;i++)
43     {
44         vector<pii>v;
45         for(int j=1;j<=m;j++)
46         {
47             v.push_back({mp[i][j],j});
48         }
49         sort(v.begin(),v.end(),cmp);
50         bitset<N>tpp;
51         int l=0;
52         for(int j=0;j<(int)v.size();j++)
53         {
54             if(v[l].first!=v[j].first)
55             {
56                 while(l<j){
57                     tpp[v[l].second]=1;
58                     l++;
59                 }
60             }
61             tp[v[j].second]&=tpp;
62         }
63     }
64     long long ans=0;
65     for(int i=1;i<=m;i++)
66     {
67         for(int j=1;j<=m;j++)
68         {
69             if(tp[i][j]){
70                 tr[j].push_back(i);
71                 in[i]++;
72             }
```

```
73     }
74 }
75 queue<int>q;
76 for(int i=1;i<=m;i++)
77 {
78     if(in[i]==0) q.push(i);
79 }
80 while(q.size())
81 {
82     int v=q.front();
83     q.pop();
84     ans=max(ans,dp[v]);
85     for(int u:tr[v])
86     {
87         dp[u]=max(dp[u],dp[v]+p[u]);
88         if(--in[u]==0) q.push(u);
89     }
90 }
91 cout<<ans;
92 return 0;
93 }
```

### 3.3 CDQ 分治

三维偏序（陌上开花）

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=2e5+10;
5  int n,k;
6  struct pp
7  {
8      int a,b,c,cnt,ans;
9      bool operator ==(const pp &x)
10     {
11         return a==x.a&&b==x.b&&c==x.c;
12     }
13     bool operator <(const pp &x)
14     {
15         if(a==x.a)
16         {
17             if(b==x.b)
18             {
19                 return c<x.c;
20             }
21             return b<x.b;
22         }
23         return a<x.a;
24     }
25 }p[N],nw[N];
26 bool cmp(pp x,pp y)
27 {
28     return x.b<y.b;
29 }
30 int top;
31 int tr[N];
32 int lowbit(int x)
33 {
34     return x&-x;
```

```
35 }
36 void add(int x,int val)
37 {
38     for(int i=x;i<=k;i+=lowbit(i)) tr[i]+=val;
39 }
40 int qr(int x)
41 {
42     int res=0;
43     for(int i=x;i; i-=lowbit(i)) res+=tr[i];
44     return res;
45 }
46 void CDQ(int l,int r)
47 {
48     if(l==r) return;
49     int mi=(l+r)>>1;
50     CDQ(l,mi);
51     CDQ(mi+1,r);
52     sort(nw+l,nw+mi+1,cmp);
53     sort(nw+mi+1,nw+r+1,cmp);
54     int j=l;
55     for(int i=mi+1;i<=r;i++)
56     {
57         while(j<=mi&&nw[j].b<=nw[i].b){
58             add(nw[j].c,nw[j].cnt);
59             j++;
60         }
61         nw[i].ans+=qr(nw[i].c);
62     }
63     for(int i=l;i<j;i++) add(nw[i].c,-nw[i].cnt);
64 }
65 int ans[N];
66 int main()
67 {
68     ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
69     cin>>n>>k;
70     for(int i=1;i<=n;i++)
71     {
```



```
72         cin>>p[i].a>>p[i].b>>p[i].c;p[i].cnt=1;
73     }
74     sort(p+1,p+n+1);
75     for(int i=1,r=1;i<=n;i=r)
76     {
77         top++;
78         int re=1;
79         while(re+i<=n&&p[i]==p[re+i]){
80             re++;
81         }
82         nw[top]=p[i];nw[top].cnt=re;
83         r=i+re;
84     }
85     CDQ(1,top);
86     for(int i=1;i<=top;i++)
87     {
88         ans[nw[i].cnt+nw[i].ans-1]+=nw[i].cnt;
89     }
90     for(int i=0;i<n;i++) {
91         cout<<ans[i]<<'\\n';
92     }
93     return 0;
94 }
```

## 3.4 Tanjan

### 3.4.1 边双连通分量

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=5e6+10;
5  int n,m;
6  int head[N],ne[N],to[N],ct=1;
7  void add(int v,int u)
8  {
9      ne[++ct]=head[v];
10     head[v]=ct;
11     to[ct]=u;
12 }
13 int dfn[N],low[N],p;
14 int st[N],tp;
15 int tot[N],cnt;
16 bool in[N];
17 void tanjan(int now,int old)
18 {
19     if(dfn[now]==0)
20     {
21         dfn[now]=++p;
22         low[now]=dfn[now];
23         st[++tp]=now;
24         in[now]=true;
25     }
26     for(int i=head[now];~i;i=ne[i])
27     {
28         int v=to[i];
29         if(!dfn[v])
30         {
31             tanjan(v,i);
32             low[now]=min(low[now],low[v]);
33         }
```

```
34         else if(i!=(old^1)) {
35             low[now]=min(low[now],dfn[v]);
36         }
37     }
38     if(low[now]==dfn[now])
39     {
40         ++cnt;
41         int x;
42         do
43         {
44             x=st[tp--];
45             tot[x]=cnt;
46         }while(x!=now);
47     }
48 }
49
50 int main()
51 {
52     ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
53     cin>>n>>m;
54     for(int i=1;i<=n;i++)
55     {
56         head[i]=-1;
57     }
58     for(int i=1;i<=m;i++)
59     {
60         int v,u;
61         cin>>v>>u;
62         if(v==u) continue;
63         add(v,u);
64         add(u,v);
65     }
66     vector<vector<int>>>ans(n+1);
67     for(int i=1;i<=n;i++)
68     {
69         if(!dfn[i])
70         {
```

```
71         tanjan(i,0);
72     }
73 }
74 for(int i=1;i<=n;i++) ans[tot[i]].push_back(i);
75 cout<<cnt<<'\n';
76 for(int i=1;i<=cnt;i++)
77 {
78     cout<<(int)ans[i].size()<<"_";
79     for(int j=0;j<(int)ans[i].size();j++)
80     {
81         cout<<ans[i][j]<<"_";
82     }
83     cout<<'\n';
84 }
85 return 0;
86 }
```

## 3.4.2 缩点

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=2e5+10;
5  int head[N],ne[N],to[N],ct=1;
6  void add(int v,int u)
7  {
8      ne[++ct]=head[v];
9      head[v]=ct;
10     to[ct]=u;
11 }
12 int n,m;
13 long long w[N],totw[N];
14 int tot[N];
15 int st[N],tp;
16 int dfn[N],low[N],cnt,p;
17 bool vis[N];
18 void tanjan(int now,int old)
19 {
20     dfn[now]=++p;
21     low[now]=p;
22     st[++tp]=now;
23     vis[now]=true;
24     for(int i=head[now];~i;i=ne[i])
25     {
26         int v=to[i];
27         if(!dfn[v])
28         {
29             tanjan(v,i);
30             low[now]=min(low[now],low[v]);
31         }
32         else if(vis[v]){
33             low[now]=min(low[now],dfn[v]);
34         }
35     }
```

```
36     if(dfn[now]==low[now])
37     {
38         ++cnt;
39         int x;
40         do{
41             x=st[tp--];
42             totw[cnt]+=w[x];
43             vis[x]=false;
44             tot[x]=cnt;
45         }while(x!=now);
46     }
47 }
48 vector<int>tr[N];
49 int in[N];
50 long long dp[N];
51 int main()
52 {
53     ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
54     cin>>n>>m;
55     for(int i=1;i<=n;i++)
56     {
57         cin>>w[i];
58         head[i]=-1;
59     }
60     for(int i=1;i<=m;i++)
61     {
62         int v,u;
63         cin>>v>>u;
64         if(v==u) continue;
65         add(v,u);
66     }
67     for(int i=1;i<=n;i++)
68     {
69         if(!dfn[i])
70         {
71             tanjan(i,0);
72         }
```

```
73     }
74     for(int i=1;i<=n;i++)
75     {
76         for(int j=head[i];~j;j=ne[j])
77         {
78             int v=to[j];
79             if(tot[i]!=tot[v])
80             {
81                 tr[tot[i]].push_back(tot[v]);
82             }
83         }
84     }
85     queue<int>q;
86     for(int i=1;i<=cnt;i++)
87     {
88         sort(tr[i].begin(),tr[i].end());
89         tr[i].erase(unique(tr[i].begin(),tr[i].end()),tr[i]
90             ].end());
91     }
92     for(int i=1;i<=cnt;i++)
93     {
94         for(int v:tr[i])
95         {
96             in[v]++;
97         }
98     }
99     for(int i=1;i<=cnt;i++) {
100         if(in[i]==0){
101             q.push(i);
102         }
103         dp[i]=totw[i];
104     }
105     long long ans=0;
106     while(q.size())
107     {
108         int u=q.front();
109         q.pop();
```

```
109         ans=max(ans,dp[u]);
110         for(int i=0;i<(int)tr[u].size();i++)
111         {
112             int v=tr[u][i];
113             dp[v]=max(dp[u]+totw[v],dp[v]);
114             if(--in[v]==0) q.push(v);
115         }
116     }
117     cout<<ans;
118     return 0;
119 }
```



### 3.5 可持久化 kmp

在原来朴素 kmp 的基础上提供了删除维护操作

```
1  int ne[N];
2  int pr[N];
3  char st[N];
4  int tp;
5  void add(char c)
6  {
7      int j=tp;
8      while(j&&st[ne[j]+1]!=c) j=pr[j];
9      st[++tp]=c;
10     j=ne[j]+1;
11     if(tp==1) ne[1]=pr[1]=0;
12     else if(st[j]==c)
13     {
14         ne[tp]=j;
15         if(st[ne[j]+1]==st[j+1]) pr[tp]=pr[j];
16         else pr[tp]=j;
17     }
18     else ne[tp]=pr[tp]=0;
19
20 }
```

### 3.6 map 排序

可以根据 map key-val 来进行排序具体操作

```
1 map<(key类型),(val类型),decltype(&cmp)>mp{&cmp}; //cmp为
    定义的排序规则
2 for(map<point,pair<double,double>,decltype(&cmp)>::
    iterator it=mp.begin();it!=mp.end();it++)//11都能用
3 {
4     ;
5 }
6
7 class cmp{//换struct一样
8     public:
9     bool operator()(const point &x,const point &y) const//
        排序方法
10    {
11        return (x^y)>0;
12    }
13 };
14 map<point,pair<double,double>,cmp>mp;
15 for(map<point,pair<double,double>,cmp>::iterator it=mp.
    begin();it!=mp.end();it++)//迭代器遍历
16 {
17     ;
18 }
```