ACM 算法模板

无敌爆哥

2023 年 10 月 19 日

目录

1 数学

1.1 康托展开

可以 O(nlogn) 复杂度求一个排列对于排列中的排名

```
const int N=1e6+10,mod=998244353;
long long fac[N];
int tr[N];
int n;
int a[N];
int lowbit(int x)
{
    return x&-x;
 }
void add(int x)
{
    for(int i=x;i<=n;i+=lowbit(i)) tr[i]+=1;
}
int qr(int x)
{
    int res=0;
    for(int i=x;i;i-=lowbit(i)) res+=tr[i];
    return res;
}
long long cantor(){
    fac[0]=1;
    for(int i=1;i<=n;i++) {
        fac[i]=fac[i-1]*i%mod;
    }
    long long ans=0;
    for(int i=1;i<=n;i++){
        ans=(ans+(a[i]-1-qr(a[i]))*fac[n-i]%mod)%mod;
        add(a[i]);
    }
    return (ans+1)%mod;
}
```

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

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

1.3 线性逆元

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

1.4 线性筛求因子数量

```cpp
int fac[N],q[N],top;
bool vis[N];
int cnt[N];
void init(int n=1e6)
{
    vis[1]=true;
    fac[1]=1;
    for(int i=2;i<=n;i++)
    {
        if(vis[i]==false)
        {
            q[top++]=i;
            fac[i]=2;
            cnt[i]=1;
        }
        for(int j=0;1ll*q[j]*i<=n;j++)
        {
            int nu=i*q[j];
            vis[nu]=true;
            if(i%q[j]==0)
            {
                fac[nu]=fac[i]/(cnt[i]+1)*(cnt[i]+2);
                cnt[nu]=cnt[i]+1;
                break;
            }
            fac[nu]=fac[i]*fac[q[j]];
            cnt[nu]=1;
        }
    }
}
```

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 $\sqrt{max}$)

```
const int M=1e6+10,T=1e3+10;//值域
bool vis[M];
int q[M],cnt;
int fac[M][3];
int gc[T][T];
int gcd(int a,int b)
{
    return b?gcd(b,a%b):a;
}
void init(int n=M-10)
{
    vis[1]=true;
    vis[0]=true;
    fac[1][0]=fac[1][1]=fac[1][2]=1;
    for(int i=2;i<=n;i++)
    {
        if(!vis[i])
        {
            q[cnt++]=i;
            fac[i][0]=fac[i][1]=1;
            fac[i][2]=i;
        }
        for(int j=0;1ll*q[j]*i<=n;j++)
        {
            int nu=q[j]*i;
            vis[nu]=true;
            fac[nu][0]=fac[i][0]*q[j];
            fac[nu][1]=fac[i][1];
            fac[nu][2]=fac[i][2];
            if(fac[nu][0]>fac[nu][1]) swap(fac[nu][0],fac[
               nu][1]);
            if(fac[nu][1]>fac[nu][2]) swap(fac[nu][1],fac[
               nu][2]);
            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。

```
int mod=998244353;
const int N=610;
long long mp[N][N];
int n;
long long gauss()//求矩阵行列式的值
{
    long long res=1;
    for(int i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            while(mp[i][i])
            {
                long long div=mp[j][i]/mp[i][i];
                for(int k=i;k<n&&div;k++) {
                    mp[j][k]=((mp[j][k]-div*mp[i][k])%mod+
                       mod)%mod;
                }
                for(int k=i;k<n;k++) swap(mp[i][k],mp[j][k
                   ]);res=-res;
            }
            for(int k=i;k<n;k++) swap(mp[i][k],mp[j][k]);
               res=-res;
        }
    }
    res=(res+mod)%mod;
    for(int i=0;i<n;i++) res=res*mp[i][i]%mod;
    return res;
}
```

1.8 线性基

```
struct Vec{
    vector<long long>d,g;
    int n;
    int le;
    Vec(){};
    Vec(int _n){n=_n;d.resize(n+1);g.resize(n+1);}
    bool insert(long long x){
        for(int i=n;i>=0;i--){
            if((x>>i)&1){
                if(d[i]==0){
                    d[i]=x;
                    return true;
                }else{
                    x^=d[i];
                }
            }
        }
        return false;
    }
    void re(){
        for(int i=0;i<=n;i++){
            for(int j=i-1;j>=0;j--){
                if((d[i]>>j)&1){
                    d[i]^=d[j];
                }
            }
        }
        le=0;
        for(int i=0;i<=n;i++){
            if(d[i]){
                g[le++]=d[i];
            }
        }
    }
    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=LCM(m1,m2,m3..) ,R=LCM*k+q.m1+r1 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

```
long long gcd(long long a,long long b)
{
    return b?gcd(b,a%b):a;
}
long long exgcd(long long a,long long b,long long &x,long
   long &y)
{
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    long long x1,y1,d;
    d=exgcd(b,a%b,x1,y1);
    x=y1;
    y=x1-a/b*y1;
    return d;
}
long long lcm(long long a,long long b)
{
    return a/gcd(a,b)*b;
}
long long m[20],r[20];
int tp;
long long CRT()
{
    for(int i=1;i<tp;i++)
    {
        long long d,x,y;
        d=exgcd(m[i-1],m[i],x,y);
        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

$$a^x = b(mod p)$$

```
long long gcd(long long a,long long b)
{
    return b?gcd(b,a%b):a;
}
long long BSGS(long long a,long long b,long long p)
{
    a%=p,b%=p;
    if(b==1||p==1){
        return 0;
    }
    int k=0;
    long long A=1;
    while(1)
    {
        long long d=gcd(a,p);
        if(b%d!=0) {
            return -1;
        }
        if(d==1) break;
        k++;b/=d;p/=d;A=A*(a/d)%p;b%=p;
        if(A==b) {
            return k;
        }
    }
    long long m=ceil(sqrt(p));
    map<long long,int>mp;
    long long t=b;
    for(int i=0;i<m;i++)
    {
        mp[t]=i;
        t=t*a%p;
    }
    t=1;
    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 mod 为质数

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

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

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

1.13 高精度

```
struct BigInt{
    int w=6;
    int tw=1e6;
    vector<int>v;
    int len=0;
    BigInt(){len=0;}
    BigInt(int _len){len=_len;v.resize(len);}
    BigInt operator +(const BigInt &w)const{
        BigInt res;
        res.v=v;
        res.len=len;
        long long tp=0;
        for(int i=0;i<(int)v.size();i++){
            if(i<(int)w.v.size()){
                tp+=w.v[i];
            }
            tp+=v[i];
            res.v[i]=(tp%tw);
            tp/=tw;
        }
        while(tp) res.v.push_back(tp%tw),tp/=tw,res.len++;
        return res;
    }
    BigInt operator +(const int &x)const{
        int tp=x;
        BigInt res;
        res.v=v;
        res.len=len;
        for(int i=0;i<(int)v.size();i++){
            tp+=v[i];
            res.v[i]=(tp%tw);
            tp/=tw;
            if(tp==0) break;
        }
        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

```
BigInt gcd(BigInt a,BigInt b){
    int tp=0;
    while(1){
        int t=a.equal(b);
        if(t==-1) swap(a,b);
        if(t==0||b.len==1&&b.v[0]==0) break;
        if(!(a.v[0]&1)&&!(b.v[0]&1)){
            tp++;
            a=a/2;
            b=b/2;
        }
        else if(!(a.v[0]&1)){
            a=a/2;
        }
        else if(!(b.v[0]&1)){
            b=b/2;
        }else{
            a=a-b;
        }
    }
    while(tp--){
        a=a*2;
    }
    return a;
}
```

1.15 拉格朗日插值

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

```
const int N=2e5+10,mod=998244353;
long long ksm(long long a,long long b){
    long long res=1;
    a%=mod;
    while(b){
        if(b&1) res=res*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return res;
}
struct pp{
    long long x,y;
}p[N];
int n;
long long lagrange(long long k){
    long long ans=0;
    for(int i=1;i<=n;i++){
        long long res=1;
        long long res1=1;
        for(int j=1;j<=n;j++){
            if(i==j) continue;
            res=res*(p[i].x-p[j].x+mod)%mod;
            res1=res1*(k-p[j].x+mod)%mod;
        }
        ans=(ans+p[i].y*res1%mod*ksm(res,mod-2)%mod)%mod;
    }
    return ans;
}
```

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：

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

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

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

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

设 w_n^k $(k < \frac{n}{2})$

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

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

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

```
#define double long double
const double PI=acos(-1);
struct complex_{
    double a,b;
    complex_(){a=0;b=0;}
    complex_(double _a,double _b){a=_a;b=_b;}
    complex_ operator +(const complex_ &w)const{
        return complex_(a+w.a,b+w.b);
    }
    complex_ operator -(const complex_ &w)const{
        return complex_(a-w.a,b-w.b);
    }
    complex_ operator *(const complex_ &w)const{
        return complex_(a*w.a-b*w.b,a*w.b+b*w.a);
    }
```

```
16      complex_ operator /(const double w){
17          return complex_(a/w,b/w);
18      }
19  };
20  namespace FFT{
21      using poly=vector<long long>;
22      using poly1=vector<complex_>;
23      int limit,top;
24      const int N=3e6+10;
25      int r[N];
26      void init(int n,int m){
27          limit=1;top=0;
28          while(limit<=n+m) limit<<=1,top++;
29          for(int i=0;i<limit;i++){
30              r[i]=(r[i>>1]>>1)|((i&1)<<(top-1));
31          }
32      }
33      void fft(poly1 &v,int type){
34          int len=v.size();
35          for(int i=0;i<len;i++){
36              if(i<r[i]) swap(v[i],v[r[i]]);
37          }
38          for(int mid=1;mid<len;mid<<=1){
39              complex_ wn(cos(PI/mid),type*sin(PI/mid));
40              for(int R=mid<<1,j=0;j<len;j+=R){
41                  complex_ w(1,0);
42                  for(int k=0;k<mid;k++,w=w*wn){
43                      complex_ x=v[j+k],y=w*v[j+k+mid];
44                      v[j+k]=x+y;
45                      v[j+k+mid]=x-y;
46                  }
47              }
48          }
49          if(type==-1){
50              for(int i=0;i<len;i++){
51                  v[i]=v[i]/(double)len;
52              }
```

```
53          }
54      }
55       poly Mul(poly A,poly B,int len=0){
56           int n=A.size(),m=B.size();
57          if(len){
58              if(n>len){
59                  A.resize(len),n=len;
60              }
61              if(m>len) B.resize(len),m=len;
62          }
63          if(n<=30||m<=30){
64              poly res(n+m-1,0);
65              for(int i=0;i<n;i++){
66                  for(int j=0;j<m;j++){
67                      res[i+j]=(res[i+j]+A[i]*B[j]);
68                  }
69              }
70              if(len){
71                  res.resize(len);
72              }
73              return res;
74          }
75          init(n,m);
76          poly1 a(limit),b(limit);
77          A.resize(limit);B.resize(limit);
78          for(int i=0;i<limit;i++){
79              a[i].a=a[i].b=0;
80              b[i].a=b[i].b=0;
81              if(i<n) a[i].a=A[i];
82              if(i<m) b[i].a=B[i];
83          }
84          fft(a,1);fft(b,1);
85          for(int i=0;i<limit;i++){
86              a[i]=a[i]*b[i];
87          }
88          fft(a,-1);
89          if(len==0){
```

```
90              A.resize(n+m-1);
91          }
92          else{
93              A.resize(len);
94          }
95          for(int i=0;i<(int)A.size();i++){
96              A[i]=(long long)(a[i].a+0.5);
97          }
98          return A;
99       }
100     //反转多项式系数
101     void Reverse(poly &a){
102         reverse(a.begin(),a.end());
103     }
104 }
105 using namespace FFT;
```

1.17 原根

若 $gcd(a,m)==1$ 同时 $a^x \equiv 1(modm)$ 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

```
vector<int>root(int n){
    vector<int>res;
    if(!check(n)){ //判断是否有原根条件 2 4 p^a 2p^a
        ;
    }else{
        int t=getol(n);//获得欧拉函数
        vector<int>pr=getpr(t);//获得质因子
        int fas=0;
        for(int g=1;;g++){
            int f=0;
            if(__gcd(g,n)!=1) continue;
            //因为阶是n的欧拉函数，所以因子都不能满足
            for(int v:pr){
                if(ksm(g,t/v,n)==1){
                    f=1;break;
                }
            }
            if(f==0){
                fas=g;break;//最小原根
            }
        }
        vector<int>res;
        int num=fas;
        //所有原根
        for(int i=1;i<=t;i++){
            if(__gcd(i,t)==1){
                res.push_back(num);
            }
            num=num*fas%n;
        }
        sort(res.begin(),res.end());
    }
```

```
33      return res;
34  }
```

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。

```
/*
65537
998244353
1004535809
4179340454199820289
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(len)要大）
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
*/
```

```
const int N=3e6+10;
int r[N];
int limit,L;
void init(int n){
    for(int i=0;i<n;i++){
        r[i]=(r[i>>1]>>1)|((i&1)<<(L-1));
    }
}
struct NTT{
    int len;
    int g=3,gi=332748118;
    long long mod=998244353;
    vector<long long>v;
    NTT(){g=3;gi=332748118;mod=998244353;}
    NTT(int _len){len=_len;v.resize(len);g=3;gi=332748118;
       mod=998244353;}
    void init(int _len){len=_len;v.resize(len);g=3;gi
       =332748118;mod=998244353;}
    void ntt(int type){
        for(int i=0;i<len;i++){
            if(i<r[i]) swap(v[i],v[r[i]]);
        }
        for(int mid=1;mid<len;mid<<=1){
```

```
22              long long wn=ksm(type==1?g:gi,(mod-1)/(mid<<1)
                    ,mod);
23              for(int j=0,R=mid<<1;j<limit;j+=R){
24                  long long w=1;
25                  for(int k=0;k<mid;k++,w=(w*wn)%mod){
26                      long long x=v[j+k],y=v[j+mid+k]*w%mod;
27                      v[j+k]=(x+y)%mod;
28                      v[j+k+mid]=(x-y+mod)%mod;
29                  }
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。

```
/*
如果要用ntt就是如下
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
使其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
千万记得改变模数时候改变原根和原根的逆元！
*/
LL md[3]={469762049,998244353,1004535809};//常用三模数
```

```
//这里是基于FFT拆系数的MTT
long long ksm(long long a,long long b,long long mod){
    long long res=1;
    a%=mod;
    while(b){
        if(b&1) res=res*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return res;
}
#define double long double
const double PI=acos(-1);
struct complex_{
    double a,b;
    complex_(){a=0;b=0;}
    complex_(double _a,double _b){a=_a;b=_b;}
    complex_ operator +(const complex_ &w)const{
        return complex_(a+w.a,b+w.b);
    }
    complex_ operator -(const complex_ &w)const{
        return complex_(a-w.a,b-w.b);
    }
    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

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

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

```
//通过区间前半段的f值来反馈贡献给后半段区间
void CDQ_NTT(int l,int r,poly &g,poly &f){
    if(l==r){
        return;
    }
    int mi=(l+r)>>1;
    CDQ_NTT(l,mi,g,f);
    poly a(mi-l+1,0),b(r-l+1,0);
    int n=mi-l+1,m=r-l+1;
    for(int i=l;i<=mi;i++){
        a[i-l]=f[i];
    }A
    for(int i=0;i<m;i++){
        b[i]=g[i];
    }
    init(m,m);
    a.resize(limit);b.resize(limit);
    for(int i=n;i<limit;i++){
        a[i]=0;
        if(i>=m) b[i]=0;
    }
    a=Mul(a,b);//根据具体卷积定义来调整
    for(int i=mi+1;i<=r;i++){
        f[i]=(f[i]+a[i-l])%mod;
    }
    CDQ_NTT(mi+1,r,g,f);
}
```

1.22 二次剩余

```
//二次剩余 2log
struct cipolla {
    long long n,p,w2,a; //w2 = a*a-n mod p

    struct cx {long long x,y;} ;//Fp2 = {x+yw | x,y belong to Fp}
    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};}
    cx cx_qpow(cx a,int n) {
        cx ans = (cx){1,0};
        for (;n;n>>=1,a=mul(a,a)) if (n&1) ans=mul(ans,a);
        return ans;
    }

    void getsol(long long _n,long long _p,long long&ans1,long long&ans2)
    {
        n = _n, p = _p;
        ans1=ans2=-1; //最多两个解,无解是-1
        if (n==0 || p==2) {ans1=n; return ;}

        if (ksm((long long)n,(long long)(p-1)/2,(long long)p) == p-1) return ;

        for (a=rand()%p;;a==p-1?(a=0):(++a)) {
            if (ksm((a*a-n+p)%p,(p-1)/2,p) == p-1) {
                w2 = (a*a-n+p)%p;
                break;
            }
        }
        cx b = (cx){a,1};
        b = cx_qpow(b,(p+1)/2);
        ans1 = b.x;
        if (ans1 != p-ans1) ans2 = p-ans1;
        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%
                p*n%p)%p,(a.x*b.y%p+a.y*b.x%p+a.z*b.z%p*n%p)%p
                ,(a.x*b.z%p+a.y*b.y%p+a.z*b.x%p)%p};
40      }
41      tri tri_qpow(tri a,int n) {
42          tri ans = (tri){1,0,0};
43          for (;n;n>>=1,a=mul(a,a)) if (n&1) ans=mul(ans,a);
44          return ans;
45      }
46      void getsol(long long _n,long long _p,long long &ans1,
            long long &ans2,long long &ans3)
47      {
48          n = _n, p = _p;
49          ans1=ans2=ans3=-1;
50          if (n==0 || p==2 || p==3) {ans1=n; return ;}
51          if (p%3 == 2) {ans1 = ksm(n,(2*p-1)/3,p); return
                ;}
52
53          if (ksm(n,(p-1)/3,p) != 1) return ;
54          cipolla cip;
55          long long tmp;
56          cip.getsol(p-3,p,epsi,tmp);
57          epsi = (epsi-1)*((p+1)/2)%p;
58
59          for (;;) {
60              tri a;
61              a.x = rand()%p, a.y = rand()%p+1, a.z = rand()
                    %p+1;
62              a = tri_qpow(a,(p-1)/3);
63              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）

```
namespace Poly{
    const int N=3e6+10;
    const int mod=998244353;
    const int g=3,gi=332748118;//根据模数不同改变原根
    using poly=vector<long long>;
    long long ksm(long long a,long long b,long long mod){
        long long res=1;
        a%=mod;
        while(b){
            if(b&1) res=res*a%mod;
            a=a*a%mod;
            b>>=1;
        }
        return res;
    }
    int r[N];
    int limit,L;
    void init(int n,int m){
        limit=1;L=0;
        while(limit<=n+m) limit<<=1,L++;
        for(int i=0;i<limit;i++){
            r[i]=(r[i>>1]>>1)|((i&1)<<(L-1));
        }
    }
    void ntt(poly &v,int type){
        int len=v.size();
        for(int i=0;i<len;i++){
            if(i<r[i]) swap(v[i],v[r[i]]);
        }
        for(int mid=1;mid<len;mid<<=1){
            long long wn=ksm(type==1?g:gi,(mod-1)/(mid<<1)
                ,mod);
            for(int j=0,R=mid<<1;j<limit;j+=R){
                long long w=1;
                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

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

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

```
const int mod=998244353;
using ll=long long;
const ll inv2=ksm(2,mod-2,mod),inv6=ksm(6,mod-2,mod);
struct Data
{
    ll f,g,h;
    Data(){
        f=g=h=0;
    }
};

Data cale(ll n,ll a,ll b,ll c){
    ll ac=a/c,bc=b/c,m=(a*n+b)/c,n1=n+1,n21=n*2+1;
    Data d;
    if(a==0){//边界
        d.f=bc*n1%mod;
        d.g=bc*n%mod*n1%mod*inv2%mod;
        d.h=bc*bc%mod*n1%mod;
        return d;
    }
    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

```
#include<bits/stdc++.h>
using namespace std;
/*
这是无离散化版本
离散化只需要将有下标的地方改成对应 id[下标]即可
*/
const int N=1e5+10;
int n,q;
int a[N];
struct pp
{
    long long a,d;
}tr[N<<2];

void build(int k,int l,int r)
{
    tr[k].a=tr[k].d=0;
    if(l==r)
    {
        tr[k].a=a[l];
        return;
    }
    int mi=(l+r)>>1;
    build(k<<1,l,mi);
    build(k<<1|1,mi+1,r);
}

void down(int k,int l,int r)
{
    int mi=(l+r)>>1;
    tr[k<<1].d+=tr[k].d;
    tr[k<<1|1].d+=tr[k].d;
    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--)
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 积性函数

常见的积性函数有：莫比乌斯函数，欧拉函数，一个数的约数和，一个数约数个数，$f(x)=x^k$ 当 $f(x)$ $g(x)$ 是积性函数时，他们的迪利克雷卷积也是积性函数 $h(x)=\sum_{d|x}f(d)g(\frac{x}{d})$

1.28.1 莫比乌斯反演

设 $g(x)$ $f(x)$

$f(d)$ 是 $x=d$ 时候的答案 $g(d)$ 是 $d|x$ 时候的答案则有 $g(x)=\sum_{x|n}f(n)$

反演得 $f(d)=\sum_{d|x}u(x)g(x)$

1.29 Pollard-Rho

```
#define LL long long
LL mul(LL a, LL b, LL p) {
    return (a * b - (LL)(a / (long double)p * b + 1e-3) *
       p + p) % p;
}
LL power(LL a, LL r, LL p) {
    LL res = 1;
    for (; r; a = mul(a, a, p), r >>= 1)
        if (r & 1)
            res = mul(res, a, p);
    return res;
};
bool miller_rabin(LL n) {
    static LL p[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    if (n == 1)
        return false;
    LL d = n - 1, r = 0;
    for (; not(d & 1); d >>= 1)
        r += 1;
    bool res = true;
    for (int i = 0; i < 9 and p[i] < n and res; i += 1) {
        LL x = power(p[i], d, n);
        if (x == 1 or x == n - 1)
            continue;
        for (int j = 1; j < r; j += 1) {
            x = mul(x, x, n);
            if (x == n - 1)
                break;
        }
        if (x != n - 1)
            res = false;
    }
    return res;
};
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()
                    % n; d == 1;
45                   k <<= 1, y = x, s = 1) {
46                  for (int i = 1; i <= k; i += 1) {
47                      x = (mul(x, x, n) + c) % n;
48                      s = mul(s, abs(x - y), n);
49                      if (not(i % 127) or i == k) {
50                          d = __gcd(s, n);
51                          if (d != 1)
52                              break;
53                      }
54                  }
55              }
56          }
57          rho(d);
58          rho(n / d);
59      };
60      rho(n);
61      return res;
62  }
```

1.30 杜教筛

能够在 $O(n^{\frac{2}{3}})$ 的时间负责度里求前 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 项目欧拉函数和莫比乌斯函数和

```
const int N=3e6+10;
int q[N],cnt;
long long u[N],o[N];
bool vis[N];
long long sumol[N],sumu[N];
void init(int n=N-10)
{
    vis[1]=true;
    o[1]=1;
    u[1]=1;
    for(int i=2;i<=n;i++)
    {
        if(vis[i]==false)
        {
            q[cnt++]=i;
            u[i]=-1;
            o[i]=i-1;
        }
        for(int j=0;1ll*q[j]*i<=n;j++)
        {
            int nu=q[j]*i;
            vis[nu]=true;
            if(i%q[j]==0)
            {
                u[nu]=0;
                o[nu]=o[i]*q[j];
                break;
            }
            u[nu]=-u[i];
            o[nu]=o[i]*(q[j]-1);
        }
    }
    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  }
```

1.31 min25 筛

俩个条件：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

```
long long f(long long x){
    long long res=0;
    return res;
}
int q[N],cnt;
bool vis[N];
long long sum[N];
void init(int n){
    q[0]=1;
    for(int i=2;i<=n;i++){
        if(!vis[i]){
            q[++cnt]=i;
            sum[cnt]=sum[cnt-1]+f(i);
        }
        for(int j=1;1ll*q[j]*i<=n;j++){
            int nu=q[j]*i;
            vis[nu]=true;
            if(i%q[j]==0){
                break;
            }
        }
    }
}
int id1[N],id2[N],m;
long long w[N];
long long n;
int T;
int find(long long x){
    return x<=T?id1[x]:id2[n/x];
}
long long g[N];
long long s(long long x){
    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  }
```

1.32 斯特林数

1.32.1 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 $\begin{bmatrix} n \\ m \end{bmatrix}$

递推式：$\begin{bmatrix} n \\ m \end{bmatrix}$=$\begin{bmatrix} n-1 \\ m-1 \end{bmatrix}$+$m * \begin{bmatrix} n-1 \\ m \end{bmatrix}$

1.32.2 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 个非空子集的方案数记作 $\begin{Bmatrix} n \\ m \end{Bmatrix}$

递推式：$\begin{Bmatrix} n \\ m \end{Bmatrix}$=$\begin{Bmatrix} n-1 \\ m-1 \end{Bmatrix}$+$m * \begin{Bmatrix} n-1 \\ m \end{Bmatrix}$

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

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

1.32.3 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

1.32.4 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

1.32.5 下降幂

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

1.32.6 变换

$$x^n = \sum_{k=0}^{n}\begin{Bmatrix} n \\ k \end{Bmatrix} x^{\underline{k}} = \sum_{k=0}^{n}\begin{Bmatrix} n \\ k \end{Bmatrix} C_x^k * k! \quad x^{\underline{n}} = \sum_{k=0}^{n}\begin{bmatrix} n \\ k \end{bmatrix}(-1)^{n-k}x^k$$

1.32.7 上升幂

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

1.32.8 变换

$$x^n = \sum_{k=0}^{n}\begin{Bmatrix} n \\ k \end{Bmatrix}(-1)^{n-k}x^{\overline{k}} \quad x^{\overline{n}} = \sum_{k=0}^{n}\begin{bmatrix} n \\ k \end{bmatrix}x^k$$

1.33 十二种球盒模型

```
using ll=long long;
//n个球,m个盒子
//球之间互不相同，盒子互不相同
ll cale1(ll n,ll m)//
{
    /*
        一个球可以选m个,n个就是(m)^n
    */
    if(n<0||m<0) return 0;
    ll ans=ksm(m,n,mod);
    return ans;
}
ll cale2(ll n,ll m)//每个盒子最多一个球
{
    /*
        让球从1-n被还没有选过的盒子选,便是m*(m-1)*...*(m-n+1)=p(m,n)
    */
    if(n<0||m<0) return 0;
    ll ans=P(m,n);
    return ans;
}
ll cale3(ll n,ll m)//每个盒子至少装一个球
{
    /*
        考虑容斥
    */
    if(n<0||m<0) return 0;
    ll ans=0;
    for(int i=0,f=1;i<=m;i++,f=(mod-f)%mod){
        ans=(ans+C(m,i)*f%mod*cale1(n,m-i)%mod)%mod;
    }
    return ans;
}
//球之间互不相同,盒子全部相同
```

```
ll cale4(ll n,ll m)//
{
    /*
        根据第二类斯特林数可以得到
        n个不同元素,放在x个不为空的集合里面方案数
        那么这个问题便可以枚举有几个非空集合得出答案
    */
    if(n<0||m<0) return 0;
    ll ans=0;
    poly a(m+1,0),b(m+1,0);
    for(int i=0,f=1;i<=m;i++,f=(mod-f)%mod){
        a[i]=f*inv[i]%mod;
        b[i]=ksm(i,n,mod)*inv[i]%mod;
    }
    a=Mul(a,b);
    for(int i=0;i<=m;i++){
        ans=(ans+a[i])%mod;
    }
    return ans;
}
ll cale5(ll n,ll m)//每个盒子最多装一个球
{
    /*
        n个球都要被装完,只需要看盒子数量够不够装完n个球
    */
    if(n<0||m<0) return 0;
    ll ans=(n<=m);
    return ans;
}
ll cale6(ll n,ll m)//每个盒子至少一个球
{
    /*
        n个不同元素,分成m个非空集合方案数
        第二类斯特林数通项直接可得
    */
    if(n<0||m<0) return 0;
    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]>=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]>=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]>=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...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)...=(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 }
```

1.34 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 $g(x)=e^{f(x)}$

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

1.36 小公式

1. $ln(1-x^k)=-\sum_{i=1}\frac{x^{ik}}{i}$
2. $a_0=1\ \ a_n=\frac{\sum_{i=0}^{n-1}C_{2n}^{i}(n-i)2^i}{n}$ 1 1 5 29 181 1181 7941 54573
3. $(a^n+b^n)==0(\bmod(a+b))$(n 为奇数)

1.37 常见泰勒展开

1. $e^x=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+...+\frac{x^n}{n!}...$
2. $sinx=x-\frac{x^3}{3!}+\frac{x^5}{5!}+...+(-1)^{m-1}\frac{x^{2m-1}}{2m-1!}...$
3. $cosx=1-\frac{x^2}{2!}+\frac{x^4}{4!}+...+(-1)^{m}\frac{x^{2m}}{2m!}...$
4. $ln(1+x)=x-\frac{x^2}{2!}+\frac{x^3}{3!}+...+(-1)^{n-1}\frac{x^n}{n!}...$
5. $(1+x)^a=1+ax+\frac{a*(a-1)}{2!}x^2+...+C_a^ix^n...$
6. $\frac{1}{1-x}=1+x+x^2+...+x^n..$ 为 a 次方时第 i 位系数为 C_{i+a-1}^{a-1}

1.38 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;
```

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

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

1.39 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

1.40 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

1.41 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 有两堆各若干的物品，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=（int）[（（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()//-pi,pi x轴正方向为0 逆时针
63      {
64          return atan2(y,x);
65      }
66  };
67  struct Line{
68      double a,b,c;     //直线一般式
```

```
69      Line(){a=0;b=0;c=0;}
70      Line(double _a,double _b,double _c)
71      {
72          a=_a;
73          b=_b;
74          c=_c;
75      }
76      Line(point &x,point &y)//俩点一线
77      {
78          a=y.y-x.y;
79          b=x.x-y.x;
80          c=x.y*y.x-x.x*y.y;
81      }
82      point cross(Line &x)   //线交点
83      {
84          point res;
85          res.x=(x.c*b-c*x.b)/(a*x.b-x.a*b);
86          res.y=(x.c*a-c*x.a)/(b*x.a-x.b*a);
87          return res;
88      }
89       bool check(point &x)//判断点是否在线上
90      {
91          return cmpb(x.x*a+x.y*b+c)==0;
92      }
93      Line getL(point &x){//获得得某点到这个条线的垂直线
94          double a1,b1,c1;
95          if(a==0) {
96              a1=1.0;
97              b1=0;
98              c1=-x.x;
99          }
100         else if(b==0)
101         {
102             a1=0;
103             b1=1.0;
104             c1=-x.y;
105         }
```

```
106         else{
107             a1=b;
108             b1=-a;
109             c1=a*x.y-b*x.x;
110         }
111         Line re(a1,b1,c1);
112         return re;
113     }
114     bool Px(Line &w)//判断平行
115     {
116         return cmpb(a*w.b-b*w.a)==0;
117     }
118     bool Li(Line &w)//判断垂直
119     {
120         return cmpb(a*w.b+b*w.a)==0;
121     }
122 };
123 struct Line //起点 终点版
124 {
125     point s,e;
126     Line(){}
127     Line(point _s,point _e){s=_s;e=_e;}
128     point cross(Line &W) {
129         point u=s-W.s,v=e-s,w=W.e-W.s;
130         double t=(u^w)/(w^v);
131         return s+v*t;
132     }
133     bool right(Line b,Line c)//判断b,c交点是否在当前线段右
        边
134     {
135         point p=b.cross(c);
136         return ((e-s)^(p-s))<0;
137     }
138     double arg() //极角 (-pi,pi]
139     {
140         return atan2(e.y-s.y,e.x-s.x);
141     }
```

```
142 };
143 double cro(point a,point b,point c)
144 {
145     return (b-a)^(c-a);
146 }
147
148 double dot(point a,point b,point c)
149 {
150     return (b-a)|(c-a);
151 }
152
153 struct point3{//三维
154     double x,y,z;
155     point3()
156     {
157         x=0.0;y=0.0;z=0.0;
158     }
159     point3(double _x,double _y,double _z)
160     {
161         x=_x;y=_y;z=_z;
162     }
163     point3 operator -(const point3 w)
164     {
165         return (point3){x-w.x,y-w.y,z-w.z};
166     }
167     point3 operator +(const point3 w)
168     {
169         return (point3){x+w.x,y+w.y,z+w.z};
170     }
171     point3 operator ^(const point3 w)
172     {
173         return (point3){(y*w.z-z*w.y),(z*w.x-x*w.z),(x*w.y
               -y*w.x)};
174     }
175     double operator |(const point3 w)
176     {
177         return x*w.x+y*w.y+z*w.z;
```

```
178     }
179     double rand01()
180     {
181         return (double)rand()/(double)RAND_MAX;
182     }
183     double randeps()
184     {
185         return (double)(rand01()-0.5)*eps;
186     }
187     void noise()//三维点需要抖动一下避免有一些影响 十分重
            要!
188     {
189         x+=randeps();y+=randeps(),z+=randeps();
190     }
191     double db(double x)
192     {
193         return x*x;
194     }
195     double dict(const point3 w)
196     {
197         return sqrt(db(x-w.x)+db(y-w.y)+db(z-w.z));
198     }
199     double mod()
200     {
201         return sqrt(x*x+y*y+z*z);
202     }
203 };
```

2.2 极角序

```
//排序cmp
struct argcmp{//极角排序
    int quad(const point&x) //象限排序 根据题目定义
        {
            if(x.y<-eps) return 1;
            else if(x.y>eps) return 3;
            else if(x.x>eps) return 4;
            else if(x.x<-eps) return 5;
            return 2;
        };
    bool operator()(const point &x,const point &y)
    {

        int xb=quad(x);
        int yb=quad(y);
        if(xb!=yb)
        {
            return xb<yb;
        }
        return (x^y)>eps; //右手定则
    }
};
```

2.3 凸包

2.3.1 二维凸包

```
const int N=4e5+10;
bool cmp(point x,point y)
{
    return (x.x==y.x?x.y<y.y:x.x<y.x);
}
point st[N],p[N];
int top,tp;
double Andrew()
{
    top=0;//多组清空
    sort(p+1,p+tp+1,cmp);
    for(int i=1;i<=tp;i++)
    {
        while(top>1&&((st[top]-st[top-1])^(p[i]-st[top-1]))<=0.0) top--;
        st[++top]=p[i];
    }
    int t=top;
    for(int i=tp-1;i>=1;i--)
    {
        while(top>t&&((st[top]-st[top-1])^(p[i]-st[top-1]))<=0.0) top--;
        st[++top]=p[i];
    }
    double ans=0.0;
    for(int i=1;i<top;i++)
    {
        ans=ans+st[i].dict(st[i+1]);
    }
    return ans;
}
```

2.4 三维凸包

2.4.1 暴力

```
const int N=110;
point3 p[N];
int n;
bool check(int id,int id1,int id2)
{
    double now=0;
    int c1=0,c2=0;
    point3 vr=(p[id1]-p[id])^(p[id2]-p[id]);
    for(int i=1;i<=n;i++)
    {
        if(i==id||i==id1||i==id2) continue;
        now=((p[i]-p[id])|vr);
        if(now>0)
        {
            c1++;
        }
        else if(now<0){
            c2++;
        }
        if(c1&&c2) return false;

    }
    return true;
}
void solve()
{
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%Lf%Lf%Lf",&p[i].x,&p[i].y,&p[i].z);
        p[i].noise();
    }
    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

```
const int N=2010;
point3 p[N];
struct face{
    int v[3];
    face(){v[0]=v[1]=v[2]=0;}
    face(int a,int b,int c){v[0]=a;v[1]=b;v[2]=c;}
    point3 normal(){return ((p[v[1]]-p[v[0]])^(p[v[2]]-p[v[0]]));}
    double area(){return normal().mod()/2.0;}
};
int cansee(face A,point3 b) {return ((b-p[A.v[0]])|A.normal())>0;}
face cv[N],h[N];
int n,th,cnt,vis[N][N];
void convex3() {
    cnt=th=0;
    cv[++cnt]=face(1,2,3);
    cv[++cnt]=face(3,2,1);
    for(int i=4;i<=n;++i) {
        for(int j=1,v;j<=cnt;++j) {
            if(!(v=cansee(cv[j],p[i])))h[++th]=cv[j];
            for(int k=0;k<3;++k)
                vis[cv[j].v[k]][cv[j].v[k>1?0:k+1]]=v;
        }
        for(int j=1;j<=cnt;++j)
            for(int k=0;k<3;++k) {
                int x=cv[j].v[k],y=cv[j].v[k>1?0:k+1];
                if(vis[x][y]&&!vis[y][x])h[++th]=face(x,y,i);
            }
        for(int j=1;j<=th;++j)cv[j]=h[j];
        cnt=th,th=0;
    }
}
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

```
double rotating_calingpers()
{
    top--;
    double ans=0;
    for(int i=0,j=1;i<top;i++)
    {
        while(((st[(i+1)%top]-st[i])^(st[j]-st[i]))<
              ((st[(i+1)%top]-st[i])^(st[(j+1)%top]-st[i]))){
            j=(j+1)%top;
        }
        ans=max(ans,st[i].dict(st[j]));
        ans=max(ans,st[(i+1)%top].dict(st[j]));
    }
    return ans;
}
```

2.7.2 俩个不相交凸包最小距离

```
double cro(point a,point b,point c)
{
    return ((b-a)^(c-a));
}

double dot(point p0,point p1,point p2){  //点积 p0为角点
    return (p1-p0)|(p2-p0);
}

double getDis(point p0,point p1,point p2){
     if(cmpb(p0.dict(p1))==0) return p0.dict(p2);//p0 p1共同点
     if(cmpb(dot(p0,p1,p2))==-1) return p2.dict(p0);//钝角
     if(cmpb(dot(p1,p0,p2))==-1) return p2.dict(p1);//钝角
     return fabs(cro(p0,p1,p2)/p0.dict(p1));  //点到线的距离
}

double minDis(point p1,point p2,point p3,point p4){//线段间最短距离
     return min(min(getDis(p1,p2,p3),getDis(p1,p2,p4)),min(getDis(p3,p4,p1),getDis(p3,p4,p2)));
}

double r_c(int f)
{
    int ymi=1,ymx=1;
    for(int i=1;i<=top[f];i++)
    {
        if(st[f][i].y<st[f][ymi].y) ymi=i;
    }
    for(int i=1;i<=top[f^1];i++)
    {
        if(st[f^1][i].y>st[f^1][ymx].y) ymx=i;
    }//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
              (| /) (\ /) (\ |) 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

```
double r_c()
{
    double ans=1e20;
    int a,b,c;
    a=b=c=2;
    int n=top-1;
    for(int i=2;i<=top;i++)
    {
        while(cro(st[i-1],st[i],st[a])<cro(st[i-1],st[i],
           st[a%n+1])) a=a%n+1;
        while(dot(st[i-1],st[i],st[b])<dot(st[i-1],st[i],
           st[b%n+1])) b=b%n+1;
        if(i==2) c=a;
        while(dot(st[i],st[i-1],st[c])<dot(st[i],st[i-1],
           st[c%n+1])) c=c%n+1;
        double H=cro(st[i-1],st[i],st[a]);
        double d=st[i].dict(st[i-1]);
        H/=d;
        double R=dot(st[i-1],st[i],st[b])/d,L=dot(st[i],st
           [i-1],st[c])/d;
        if(H*(R+L-d)<ans)
        {
            ans=H*(R+L-d);
            res[0]=st[i-1]+(st[i]-st[i-1])*R/d;
            res[3]=st[i]+(st[i-1]-st[i])*L/d;
            res[1]=res[3]-res[0];
            res[1].rotate(-pi/2.0);
            res[1]=res[1]*H/(L+R-d);
            res[1]=res[1]+res[0];
            res[2]=res[0]-res[3];
            res[2]=res[2]*H/(L+R-d);
            res[2].rotate(pi/2.0);
            res[2]=res[2]+res[3];
        }

```

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

2.8 闵可夫斯基和

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

```
const int N=4e5+10;
bool cmp(point x,point y)
{
    return (x.x==y.x?x.y<y.y:x.x<y.x);
}
point st[3][N],p[3][N];
int top[3],tp[3];
void Andrew(int id)
{
    top[id]=0;//多组清空
    sort(p[id]+1,p[id]+tp[id]+1,cmp);
    for(int i=1;i<=tp[id];i++)
    {
        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]--;
        st[id][++top[id]]=p[id][i];
    }
    int t=top[id];
    for(int i=tp[id]-1;i>=1;i--)
    {
        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]--;
        st[id][++top[id]]=p[id][i];
    }
    top[id]--;
}
point s1[N],s2[N];
void Minkovski(int id,int id1,int id2)//a={b+c}
{
    for(int i=1;i<top[id1];i++) s1[i]=st[id1][i+1]-st[id1
      ][i];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

```
const int N=4e5+10;
Line le[N],dq[N];
int pt;
bool cmphp(Line a,Line b)//极角+左侧
{
    double A=a.arg(),B=b.arg();
    return cmpb(A-B)!=0?A<B:((a.e-a.s)^(b.e-a.s))<0.0;
}
void half_plane()//半平面交
{
    sort(le+1,le+pt+1,cmphp);
    int h=1,t=1;dq[1]=le[1];
    for(int i=2;i<=pt;i++)
    {
        if(le[i].arg()-le[i-1].arg()<eps) continue;
        while(h<t&&le[i].right(dq[t],dq[t-1])) t--;//维护头尾
        while(h<t&&le[i].right(dq[h],dq[h+1])) h++;
        dq[++t]=le[i];
    }
    while(h<t&&dq[h].right(dq[t],dq[t-1])) t--; //割多余尾巴
//  dq[++t]=dq[h];//封口 看题目需要
}
```

2.10 最小圆覆盖

```
const int N=1e5+10;
int n;
point p[N];
point gauss(point x,point y,point z){
    double c1=x.db(x.dz())-y.db(y.dz());
    double c2=x.db(x.dz())-z.db(z.dz());
    point e=x-y,e1=x-z;
    point o;
    o.x=(c1*e1.y-c2*e.y)/(2.0*(e.x*e1.y-e1.x*e.y));
    o.y=(c1*e1.x-c2*e.x)/(2.0*(e.y*e1.x-e1.y*e.x));
    return o;
}
void getcircle(point &o,double &r){
    random_shuffle(p,p+n);
    o=p[0];
    for(int i=1;i<n;i++){
        if(cmpb(r-o.dict(p[i]))==-1){
            o=p[i];
            r=0.0;
            for(int j=0;j<i;j++){
                if(cmpb(r-o.dict(p[j]))==-1){
                    o=(p[i]+p[j])/2.0;
                    r=o.dict(p[j]);
                    for(int k=0;k<j;k++){
                        if(cmpb(r-o.dict(p[k]))==-1){
                            o=gauss(p[i],p[j],p[k]);
                            r=o.dict(p[k]);
                        }
                    }
                }
            }
        }
    }
}
```

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

```
double R;//半径
point o;//圆心
bool onSeg(point p,point a,point b){
    return cmpb((a-p)^(b-p))==0&&((a-p)|(b-p))<=eps;
}
point getNode(point a,point u,point b,point v){
    double t=((a-b)^v)/(v^u);
    return a+(u*t);
}
double getDP2(point a,point b,point &pa,point &pb){
    point i=(b-a);
    i.rotate(pi/2);
    point e=getNode(a,b-a,o,i);//垂足
    double d=o.dict(e);
    if(!onSeg(e,a,b)) d=min(o.dict(a),o.dict(b));
    if(R<=d) return d;
    double len=sqrt(R*R-o.dict(e)*o.dict(e));
    pa=e+(a-b).norm()*len;
    pb=e+(b-a).norm()*len;
    return d;
}
//圆上扇形面积
double sector(point a,point b){
    double angle=acos(((a)|(b))/a.dz()/b.dz());
    if(cmpb((a)^(b))==-1) angle=-angle;
    return R*R*angle/2.0;
}
double getArea(point a,point b){
    if(cmpb(a^b)==0) return 0;
    double da=a.dict(o),db=b.dict(o);
    if(R>=da&&R>=db) return (a^b)/2;//三角形在圆内
    point pa,pb;
    double d=getDP2(a,b,pa,pb);
    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

```
x1=x*cos(n)-y*sin(n);
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=2πb+4(a-b) 面积为 S=πab

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 浮点输入

```
string s;
cin>>s;
double d=stod(s);
```

2.16 浮点输出

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

2.17 浮点取模

```
fmod(a,b)==a%b
```

2.18 扫描线

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10;
struct Line
{
    int x,y1,y2,f,id1,id2;
    bool operator<(const Line &le1)
    {
        return x<le1.x;
    }
}le[N];
vector<int>b;
int n;
int find(int x)
{
    return lower_bound(b.begin(),b.end(),x)-b.begin()+1;
}
struct pp
{
    long long le;
    int f;
}tr[N<<4];
void up(int k,int l,int r)
{
    if(!tr[k].f){
        tr[k].le=tr[k<<1].le+tr[k<<1|1].le;
    }
    else{
        tr[k].le=b[r]-b[l-1];
    }
}
void mif(int k,int l,int r,int x,int y,int f)
{
    if(l==x&&r==y)
    {
```

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

```
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  }
```

2.19 平面最近点对

```
const int N=1e6+10;
int n;
point p[N],st[N];
bool cmpx(point &x,point &y){
    return x.x<y.x;
}
bool cmpy(point &x,point &y){
    return x.y<y.y;
}
double find(int l,int r){
    if(l==r) return 1e18;
    if(r-l==1){
        return p[l].dict(p[r]);
    }
    int mi=(l+r)>>1;
    double d=min(find(l,mi),find(mi+1,r));
    int tp=0;
    for(int i=l;i<=r;i++){
        if(cmpb(d-fabs(p[i].x-p[mi].x))==1){
            st[++tp]=p[i];
        }
    }
    sort(st+1,st+tp+1,cmpy);
    for(int i=1;i<tp;i++){
        for(int j=i+1;j<=tp&&st[j].y-st[i].y<d;j++){
            d=min(d,st[i].dict(st[j]));
        }
    }
    return d;
}
```

2.20 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

```
const double eps=1e-12;
int cmpb(double x)
{
    if(fabs(x)<eps) return 0;
    if(x>0) return 1;
    return -1;
}
double f(double x)//对应题目函数
{
    double ans;
    return ans;
}
double simpson(double l,double r){//辛普森公式
    return (r-l)*(f(r)+f(l)+4.0*f((r+l)/2.0))/6.0;
}
double asr(double l,double r,double ans){//自适应
    double mi=(l+r)/2.0;
    double a=simpson(l,mi),b=(simpson(mi,r));
    if(cmpb(a+b-ans)==0) return ans;
    return asr(l,mi,a)+asr(mi,r,b);
}
```

3 其它

3.1 快读快写开 02

```
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")

inline int read()
{
    char c=getchar();int x=0,f=1;
    while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}
    while(c>='0'&&c<='9'){x=x*10+c-'0';c=getchar();}
    return x*f;
}
inline void write(int x)
{
    static int t[25];int tp=0;
    if(x==0) return(void)(putchar('0'));else if(x<0)
       putchar('-'),x=-x;
    while(x) t[tp++]=x%10,x/=10;
    while(tp--) putchar(t[tp]+48);
}
```

3.2 bitset 处理多维偏序

```cpp
#include<bits/stdc++.h>
using namespace std;

const int N=5010,M=510;
int n,m;
int mp[M][N];
long long p[N],dp[N];
bitset<N>tp[N];
vector<int>tr[N];
int in[N];
typedef pair<int,int> pii;
bool cmp(pii x,pii y)
{
    if(x.first==y.first)
    {
        return x.second<y.second;
    }
    return x.first<y.first;
}
int main()
{
    ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        cin>>p[i];
        dp[i]=p[i];
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            cin>>mp[i][j];
        }
    }
```

```
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）

```
#include<bits/stdc++.h>
using namespace std;

const int N=2e5+10;
int n,k;
struct pp
{
    int a,b,c,cnt,ans;
    bool operator ==(const pp &x)
    {
        return a==x.a&&b==x.b&&c==x.c;
    }
    bool operator <(const pp &x)
    {
        if(a==x.a)
        {
            if(b==x.b)
            {
                return c<x.c;
            }
            return b<x.b;
        }
        return a<x.a;
    }
}p[N],nw[N];
bool cmp(pp x,pp y)
{
    return x.b<y.b;
}
int top;
int tr[N];
int lowbit(int x)
{
    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

```
#include<bits/stdc++.h>
using namespace std;

const int N=5e6+10;
int n,m;
int head[N],ne[N],to[N],ct=1;
void add(int v,int u)
{
    ne[++ct]=head[v];
    head[v]=ct;
    to[ct]=u;
}
int dfn[N],low[N],p;
int st[N],tp;
int tot[N],cnt;
bool in[N];
void tanjan(int now,int old)
{
    if(dfn[now]==0)
    {
        dfn[now]=++p;
        low[now]=dfn[now];
        st[++tp]=now;
        in[now]=true;
    }
    for(int i=head[now];~i;i=ne[i])
    {
        int v=to[i];
        if(!dfn[v])
        {
            tanjan(v,i);
            low[now]=min(low[now],low[v]);
        }
```

```
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

```
#include<bits/stdc++.h>
using namespace std;

const int N=2e5+10;
int head[N],ne[N],to[N],ct=1;
void add(int v,int u)
{
    ne[++ct]=head[v];
    head[v]=ct;
    to[ct]=u;
}
int n,m;
long long w[N],totw[N];
int tot[N];
int st[N],tp;
int dfn[N],low[N],cnt,p;
bool vis[N];
void tanjan(int now,int old)
{
    dfn[now]=++p;
    low[now]=p;
    st[++tp]=now;
    vis[now]=true;
    for(int i=head[now];~i;i=ne[i])
    {
        int v=to[i];
        if(!dfn[v])
        {
            tanjan(v,i);
            low[now]=min(low[now],low[v]);
        }
        else if(vis[v]){
            low[now]=min(low[now],dfn[v]);
        }
    }
```

```
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
              ].end());
90      }
91      for(int i=1;i<=cnt;i++)
92      {
93          for(int v:tr[i])
94          {
95              in[v]++;
96          }
97      }
98      for(int i=1;i<=cnt;i++) {
99          if(in[i]==0){
100           q.push(i);
101         }
102         dp[i]=totw[i];
103     }
104     long long ans=0;
105     while(q.size())
106     {
107         int u=q.front();
108         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

```
int ne[N];
int pr[N];
char st[N];
int tp;
void add(char c)
{
    int j=tp;
    while(j&&st[ne[j]+1]!=c) j=pr[j];
    st[++tp]=c;
    j=ne[j]+1;
    if(tp==1) ne[1]=pr[1]=0;
    else if(st[j]==c)
    {
        ne[tp]=j;
        if(st[ne[j]+1]==st[j+1]) pr[tp]=pr[j];
        else pr[tp]=j;
    }
    else ne[tp]=pr[tp]=0;

}
```

3.6 map 排序

可以根据 map key-val 来进行排序具体操作

```
map<（key类型）,(val类型),decltype(&cmp)>mp{&cmp}; //cmp为定义的排序规则
 for(map<point,pair<double,double>,decltype(&cmp)>::iterator it=mp.begin();it!=mp.end();it++)//11都能用
{
    ;
}

class cmp{//换struct一样
    public:
    bool operator()(const point &x,const point &y) const//排序方法
    {
        return (x^y)>0;
    }
};
map<point,pair<double,double>,cmp>mp;
for(map<point,pair<double,double>,cmp>::iterator it=mp.begin();it!=mp.end();it++)//迭代器遍历
{
    ;
}
```

3.7 更快的哈希表

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
const int RANDOM = chrono::high_resolution_clock::now().
    time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
typedef gp_hash_table<int, int, chash> hash_t;
```