



ACM-ICPC 模板

HDU2020

Team : HDUNchar66

应以周

舒飘扬

郑豪杰

目录

1	数论	6
1.1	埃拉托斯特尼筛	6
1.2	线性筛	6
1.3	扩展欧几里得算法	8
1.4	整除分块	8
1.5	快速幂、费马小定理	8
1.6	线性递推逆元	9
1.7	欧拉函数	9
1.8	欧拉定理/欧拉降幂	10
1.9	莫比乌斯函数	10
1.10	狄利克雷卷积, 莫比乌斯反演	10
1.11	中国剩余定理	12
1.12	BSGS/exBSGS	14
1.13	Lucas 定理	15
1.14	杜教筛	16
1.15	Pollar-Rho	18
1.16	N 以内素数个数	20
2	数学	21
2.1	快速傅里叶变换 FFT	21
2.2	快速数论变换 NTT	24
2.2.1	原根表	24
2.2.2	普通 NTT	25
2.2.3	分治 NTT	30
2.2.4	三模 NTT	30
2.2.5	任意模数 NTT	35
2.3	容斥定理	38
2.4	广义容斥定理	38
2.5	Min-Max 容斥	38
2.6	高斯消元	39
2.7	线性基	41
2.8	矩阵快速幂	42
2.9	康托展开	44
2.10	卡特兰数	44
2.11	斯特林数	44
2.12	小球放入盒子模型	45
2.13	快速沃尔什变换 FWT	45
2.14	克莱姆法则	46
2.15	自适应辛普森积分	46
2.16	曼哈顿距离	46

2.17	高次方求和	47
2.18	常用求和, 组合数公式	47
3	数据结构	49
3.1	树状数组	49
3.2	线段树	50
3.2.1	懒惰标记线段树	50
3.2.2	暴力更新线段树	51
3.2.3	线段树上二分	52
3.2.4	线段树合并	52
3.2.5	线段树分裂	53
3.2.6	扫描线	56
3.2.7	segment tree beats	57
3.3	单调栈	58
3.4	单调队列	60
3.5	ST 表	61
3.6	并查集	61
3.6.1	路径压缩并查集	61
3.6.2	按秩合并并查集	61
3.6.3	可撤销并查集	62
3.6.4	可持久化并查集	62
3.7	主席树	64
3.8	树套树	66
3.9	Trie 树	67
3.10	可持久化 01Trie	67
3.11	莫队	68
3.11.1	单点修改莫队	68
3.11.2	回滚莫队	69
3.11.3	树上莫队	71
3.12	平衡树	74
3.12.1	Treap	74
3.12.2	splay	77
3.13	动态树	79
3.14	左偏树 (可并堆)	82
3.15	笛卡尔树	83
4	图论	83
4.1	最小生成树	83
4.1.1	Kruskal	83
4.1.2	Prim	84
4.2	最短路	85
4.2.1	Floyd	85

4.2.2	Floyd 求最小环	85
4.2.3	Dijkstra	86
4.2.4	SPFA	87
4.3	二分图最大匹配	88
4.4	拓扑排序	88
4.5	Tarjan	89
4.5.1	有向图 scc	89
4.5.2	无向图 edcc(割边)	91
4.5.3	无向图 vdcc(割点)	92
4.5.4	仙人掌建圆方树	94
4.6	差分约束	97
4.7	2-SAT	98
4.8	Dinic 最大流	101
4.9	Edmonds Karp 最小费用最大流	103
4.10	Kruskal 重构树	104
4.11	二分图最大权匹配 KM 算法	105
4.12	矩阵树定理	106
4.13	LGV 引理	108
5	树上问题	108
5.1	树上倍增	108
5.2	轻重链剖分	109
5.3	欧拉序 ST 表求 LCA	111
5.4	点分治	112
5.5	启发式合并	113
5.6	长链剖分 $O(1)k$ 级祖先	115
5.7	虚树	116
6	动态规划	117
6.1	树形背包	117
6.2	区间 DP	118
6.3	数位 DP	118
6.4	最长上升子序列	119
6.5	所有字段和	120
6.6	公共子序列	120
6.7	错位排列数	121
6.8	基环树直径	121
6.9	高维前缀和	123
7	字符串	124
7.1	KMP 算法	124
7.2	失配树	125

7.3	字符串哈希	127
7.4	Manacher	128
7.5	AC 自动机	128
7.6	后缀数组 SA	130
7.7	后缀自动机 SAM	131
7.8	广义 SAM	133
7.9	子序列自动机	135
7.10	回文自动机 PAM	136
8	计算几何	137
8.1	实数运算	137
8.2	点/向量类	137
8.3	平面最近点对	138
8.4	点线关系	139
8.5	线线关系	140
8.6	极角排序	140
8.7	多边形、凸包	141
8.8	旋转卡壳	142
8.9	半平面交	143
8.10	圆	144
8.11	公式	147
9	博弈	147
9.1	必败态	147
9.2	SG 函数	147
9.3	反 SG 游戏	148
9.4	巴什博弈	148
9.5	Nim 游戏	148
9.6	阶梯 Nim	148
9.7	Nim-k	148
9.8	威佐夫博弈	149
9.9	树上删边游戏	149
9.10	无向图删边游戏	149
10	基础算法	149
10.1	二分查找	149
10.2	去重, 离散化	150
10.3	取出第 k 小/中位数	150
10.4	bitset	150
10.5	builtin 函数	150
10.6	归并排序求逆序对	150
10.7	计数排序	151

10.8 基数排序	151
10.9 维护区间并集	152
10.10 计算天数	153
10.11 表达式求值	153
11 模拟游戏	156
11.1 算 24 点	156
11.2 德州扑克	158
12 其他技巧	162
12.1 C++ 编译命令	162
12.2 快读快写	162
12.3 对拍	164

1 数论

1.1 埃拉托斯特尼筛

时间复杂度为调和级数 $O(n \ln n)$

```

1 bool vis[maxn+5];
2 void sieve(){
3     for(int i=1;i<=maxn;++i){
4         if(!vis[i]){//i是质数
5             for(int j=i+i;j<=maxn;j+=i)vis[j]=1;
6         }
7     }
8 }
```

1.2 线性筛

$[1, n]$ 的正整数中质数个数大约有 $\frac{n}{\ln n}$ 个

积性函数: $f(xy) = f(x)f(y) \iff \gcd(x, y) = 1$, 积性函数可以线性筛, 每个数被最小质因子递筛一次
常见积性函数有:

欧拉函数: $\varphi(n) = n \times \prod_{p_i | n} \frac{p_i - 1}{p_i}$

莫比乌斯函数: $\mu(n) = \begin{cases} 1, n = 1 \\ (-1)^k, n = p_1 p_2 \dots p_k \\ 0, \text{other}(\text{has square factor}) \end{cases}$

因数个数: $d(n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}) = (1 + a_1)(1 + a_2) \dots (1 + a_k)$

因数和: $\sigma(n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}) = \frac{1 - p_1^{a_1 + 1}}{1 - p_1} \frac{1 - p_2^{a_2 + 1}}{1 - p_2} \dots \frac{1 - p_k^{a_k + 1}}{1 - p_k}$

因数幂和: $\sigma_k(n) = \sum_{d|n} d^k$

线性筛积性函数 $f(n)$ 时, 需要分三类情况递推函数值

1、 i 是质数: 直接得出 $f(i)$

2、 $i \% p_j \neq 0, p_j$ 是 $i \times p_j$ 的最小质因数, $f(i \times p_j) = f(i)f(p_j)$

3、 $i \% p_j = 0, p_j$ 是 $i \times p_j$ 的最小质因数, 且存在 p_j 的平方因子, 需要根据函数定义计算贡献

有时需要维护最小质因子的幂次 $mpfpow, mpfcnt$

$\varphi(i) = i \prod_{p_k | i} \frac{p_k - 1}{p_k} \rightarrow \varphi(i \times p_j) = i \times p_j \prod_{p_k | i \times p_j} \frac{p_k - 1}{p_k} = \varphi(i) \times p_j$

$\mu(i \times p_j) = 0$, 因为有平方因子 p_j^2

$d(p^k) = k + 1, \sigma(p^k) = \frac{1 - p^{k+1}}{1 - p}$

$d(i \times p_j) = d(mpfpow(i \times p_j))d(i \times p_j / mpfpow(i \times p_j))$

```

1 const int maxn=1e6,N=maxn+4;
2 int p[N],hp;//p[1~hp]是质数
3 int mpf[N],mpfpow[N],mpfcnt[N];//最小质因子,mpf的幂,mpf的指数大小
4 int phi[N],mu[N];//欧拉函数、莫比乌斯函数
5 int d[N];//约数个数
```

```

6  ll sigma[N]; //约数和
7  void sieve(){
8      static bool vis[maxn+5];
9      phi[1]=mu[1]=d[1]=sigma[1]=1;
10     for(int i=2;i<=maxn;++i){
11         if(!vis[i]){ //1、i是质数
12             p[++hp]=i; mu[i]=-1; phi[i]=i-1;
13             mpf[i]=i; mpfpow[i]=i; mpfcnt[i]=1;
14             d[i]=2; sigma[i]=i+1;
15         }
16         for(int j=1;j<=hp&&i*p[j]<=maxn;++j){ //maxn<=2e8不会爆int
17             int x=i*p[j]; vis[x]=1;
18             if(i%p[j]==0){ //2、p[j]是i的最小质因子,且i*p[j]有p[j]*p[j]这个因子
19                 mpf[x]=mpf[i];
20                 mpfpow[x]=mpfpow[i]*p[j];
21                 mpfcnt[x]=mpfcnt[i]+1;
22                 phi[x]=phi[i]*p[j]; mu[x]=0;
23                 if(mpfpow[x]==x){
24                     d[x]=mpfcnt[x]+1;
25                     sigma[x]=(1-1ll*x*p[j])/(1-p[j]);
26                 }
27                 else{
28                     d[x]=d[mpfpow[x]]*d[x/mpfpow[x]];
29                     sigma[x]=sigma[mpfpow[x]]*sigma[x/mpfpow[x]];
30                 }
31                 break;
32             }
33             mpf[x]=p[j]; //3、p[j]是i*p[j]的最小质因子
34             mpfcnt[x]=1;
35             mpfpow[x]=p[j];
36             phi[x]=phi[i]*phi[p[j]];
37             mu[x]=-mu[i];
38             d[x]=d[i]*d[p[j]];
39             sigma[x]=sigma[i]*sigma[p[j]];
40         }
41     }
42 }
43 void f(int x){ //对x分解质因数
44     vector<int>v;
45     while(x>1){
46         int t=mpf[x];

```



```

47     v.push_back(t);
48     while(x%t==0)x/=t;
49 }
50 }

```

1.3 扩展欧几里得算法

裴蜀定理：对任何整数 a, b, x, y ， $ax + by$ 是 $\gcd(a, b)$ 的倍数，且不定方程 $ax + by = \gcd(a, b)$ 一定有解。若 $a < 0$ ，则把 $ax + by = \gcd(a, b)$ 转化为 $|a|(-x') + by = \gcd(|a|, b)$

求 $ax \equiv 1 \pmod{b}$ ，即求 $ax + by = 1$ 的解中 x 的最小正整数值，有解的必要条件是 $\gcd(a, b) = 1$

```

1  ll gcd(ll a, ll b){
2      return b?gcd(b, a%b):a;
3  }
4  ll exgcd(ll a, ll b, ll &x, ll &y){
5      if(!b){
6          x=1; y=0; return a;
7      }
8      ll g=exgcd(b, a%b, x, y);
9      ll t=x; x=y; y=t-a/b*y;
10     return g;
11 }
12 int main(){
13     ll a=10, b=13; ll x, y; //x, y传引用
14     ll g=exgcd(a, b, x, y);
15     //g=gcd(a, b), ax+by=gcd(a, b), a(x+kb)+b(y-ka)=gcd(a, b)
16     ll inv=(x%b+b)%b; //a*inv%b=0;
17 }

```

1.4 整除分块

```

1  for(ll l=1, r; l<=n; l=r+1){ //O(n根号n)
2      ll t=n/l; r=n/t; //对n和m同步整除分块时, r=min(n/(n/l), m/(m/l));
3      //任意i属于[l, r], n/i==t
4  }

```

1.5 快速幂、费马小定理

费马小定理： $a^{p-1} \equiv 1 \pmod{p}$

```

1  ll qpow(ll a, ll b){
2      ll s=1;

```

```

3     for(;b;b>>=1){
4         if(b&a)s=s*a%mod;
5         a=a*a%mod;
6     }
7     return s;
8 }
9 ll inva = qpow(a,mod-2);//a的逆元
    
```

1.6 线性递推逆元

```

1 int inv[N];
2 void invwork(){//递推逆元
3     inv[1]=1;
4     for(int i=2;i<N;++i){
5         inv[i]=(1ll*mod-mod/i)*inv[mod%i]%mod;
6     }
7 }
    
```

```

1 ll fac[N],ifac[N],inv[N];
2 ll C(int n,int m){
3     return fac[n]*ifac[m]%mod*ifac[n-m]%mod;
4 }
5 int main(){
6     fac[0]=ifac[0]=1;
7     for(int i=1;i<N;++i)fac[i]=fac[i-1]*i%mod;
8     ifac[N-1]=qpow(fac[N-1],mod-2);
9     for(int i=N-2;i--i)ifac[i]=ifac[i+1]*(i+1)%mod;
10    for(int i=1;i<N;++i)inv[i]=ifac[i]*fac[i-1]%mod;
11 }
    
```

1.7 欧拉函数

$\varphi(n)$ 为 $[1, n]$ 内与 n 互质的正整数个数 (包括 1), $\varphi(n) = n * \prod_{p_i|n} \frac{p_i-1}{p_i}$

```

1 ll phi(ll x){//1~x与x互质的个数
2     ll s=x;
3     for(ll i=2;i*i<=x;++i){
4         if(x%i==0){
5             s=s-s/i;
6             for(;x%i==0;x/=i);
7         }
8     }
    
```

```

9      if(x>1)s=s-s/x;
10     return s;
11 }
    
```

$[x]$ 表示方括号内若为真, 值为 1, 否则为 0

推公式通常可以考虑枚举因数

$$\sum_{i=1}^n gcd(i, n) = \sum_{d=1, d|n}^n d\varphi\left(\frac{n}{d}\right)$$

$$\sum_{i=1}^n \sum_{j=1}^n [gcd(i, j) = 1] = 2 \sum_{i=1}^n \sum_{j=1}^i [gcd(i, j) = 1] - 1 = 2 \sum_{i=1}^n \varphi(i) - 1$$

$$\sum_{i=1}^n gcd(i, n) = \sum_{d=1, d|n}^n d \sum_{i=1}^n [gcd(\frac{i}{d}, \frac{n}{d} = 1)] = \sum_{d=1, d|n}^n d \sum_{i=1}^{\frac{n}{d}} [gcd(i, \frac{n}{d} = 1)] = \sum_{d=1, d|n}^n d\varphi\left(\frac{n}{d}\right)$$

1.8 欧拉定理/欧拉降幂

$$a^b \equiv \begin{cases} a^{b \% \varphi(p)}, gcd(a, b) = 1 \\ a^{b \% \varphi(p) + \varphi(p)}, b \geq \varphi(p) \end{cases} \pmod{p}$$

求 $a^{a^{\dots^a}} \pmod{p}$

```

1  ll qpow(ll a, ll b, ll mod){
2      ll s=1;
3      for(;b>=1){
4          if(b&1)s=s*a%mod;
5          a=a*a%mod;
6      }
7      return s;
8  }
9  ll f(ll a, int mod){
10     if(mod==1)return 0;
11     return qpow(a, f(a, phi[mod])+phi[mod], mod);
12 }
    
```

1.9 莫比乌斯函数

$$\mu(n) = \begin{cases} 1, n = 1 \\ (-1)^k, n = p_1 p_2 \dots p_k \\ 0, other(has\ square\ factor) \end{cases}$$

1.10 狄利克雷卷积, 莫比乌斯反演

狄利克雷卷积: 若 h 为 f 和 g 的狄利克雷卷积, 记作 $h = f * g$, 则 $h(n) = \sum_{d|n} f(d)g\left(\frac{n}{d}\right)$

常见数论函数: $id(n) = n, 1(n) = 1, \varepsilon(n) = [n = 1]$ (方括号表示若括号内条件成立为 1, 若不成立为 0)

莫比乌斯反演公式: 若 $f = g * 1$, 则 $g = f * \mu$, 即在狄利克雷卷积运算中 μ 是 1 的逆元

当式子无法继续推时, 可以设 $f = g * 1, g = f * \mu$, 然后推出 g

常见狄利克雷卷积公式:

$$\varepsilon = \mu * 1, id = \varphi * 1$$

例题, 保证 $n \leq m$

$$\begin{aligned} \sum_{i=1}^n \sum_{j=1}^m [gcd(i, j) = 1] &= \sum_{i=1}^n \sum_{j=1}^m \sum_{d|gcd(i, j)} \mu(d) = \sum_{d=1}^n \mu(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor \\ \sum_{i=1}^n \sum_{j=1}^m lcm(i, j) &= \sum_{i=1}^n \sum_{j=1}^m \frac{ij}{gcd(i, j)} = \sum_{i=1}^n \sum_{j=1}^m \sum_{g=1}^{min(i, j)} \frac{ij}{g} [gcd(i, j) = g] \\ &= \sum_{g=1}^n \sum_{i=1}^{\lfloor \frac{n}{g} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{g} \rfloor} ijg [gcd(i, j) = 1] \quad (\text{原先的 } i, j \text{ 被替换为枚举 } g \text{ 的 } i, j \text{ 倍}) \\ &= \sum_{g=1}^n \sum_{i=1}^{\lfloor \frac{n}{g} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{g} \rfloor} ijg \sum_{d|gcd(i, j)} \mu(d) = \sum_{g=1}^n g \sum_{d=1}^{\lfloor \frac{n}{g} \rfloor} \mu(d) d^2 \sum_{i=1}^{\lfloor \frac{n}{gd} \rfloor} i \sum_{j=1}^{\lfloor \frac{m}{gd} \rfloor} j \\ &= (T = gd, f(n) = \frac{n(n+1)}{2}) \sum_{g=1}^n g \sum_{d=1}^{\lfloor \frac{n}{g} \rfloor} \mu(d) d^2 f(\lfloor \frac{n}{T} \rfloor) f(\lfloor \frac{m}{T} \rfloor) \\ &= \sum_{T=1}^n f(\lfloor \frac{n}{T} \rfloor) f(\lfloor \frac{m}{T} \rfloor) \sum_{g|T} g \mu(\frac{T}{g}) (\frac{T}{g})^2 = \sum_{T=1}^n f(\lfloor \frac{n}{T} \rfloor) f(\lfloor \frac{m}{T} \rfloor) \sum_{d|T} d \mu(d) T \\ \text{线性筛出 } f(T) &= \sum_{d|T} d \mu(d) \text{ 并求前缀和后, 对 } n, m \text{ 整除分块} \\ \sum_{i=1}^n \sum_{j=1}^m [gcd(i, j) \text{ is prime}] &= \sum_{k=2, k \in \text{prime}}^n \sum_{i=1}^n \sum_{j=1}^m [gcd(i, j) = k] = \sum_{k \in \text{prime}}^n \sum_{i=1}^{\lfloor \frac{n}{k} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{k} \rfloor} [gcd(i, j) = 1] \\ &= \sum_{k \in \text{prime}}^n \sum_{i=1}^{\lfloor \frac{n}{k} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{k} \rfloor} \sum_{d|gcd(i, j)} \mu(d) = \sum_{k \in \text{prime}}^n \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) \lfloor \frac{n}{kd} \rfloor \lfloor \frac{m}{kd} \rfloor \\ &= (T = kd) \sum_{k \in \text{prime}}^n \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) \lfloor \frac{n}{T} \rfloor \lfloor \frac{m}{T} \rfloor = \sum_{T=1}^n \lfloor \frac{n}{T} \rfloor \lfloor \frac{m}{T} \rfloor \sum_{k|T, k \in \text{prime}} \mu(\frac{T}{k}) \quad \text{预处理 } \mu(\frac{T}{k}) \text{ 前缀和, 对 } T \text{ 整除分块} \\ d(n) = n \text{ 的约数个数} &= \sum_{d|n} 1 \quad d(ij) = \sum_{x|i} \sum_{y|j} [gcd(x, y) = 1] \\ \sum_{i=1}^n \sum_{j=1}^m d(ij) &= \sum_{i=1}^n \sum_{j=1}^m \sum_{x|i} \sum_{y|j} [gcd(x, y) = 1] = \sum_{i=1}^n \sum_{j=1}^m \lfloor \frac{n}{i} \rfloor \lfloor \frac{m}{j} \rfloor [gcd(i, j) = 1] \quad (xy \text{ 提到前面, 再把 } xy \text{ 写成 } ij) \\ &= \sum_{i=1}^n \sum_{j=1}^m \lfloor \frac{n}{i} \rfloor \lfloor \frac{m}{j} \rfloor \sum_{d|gcd(i, j)} \mu(d) = \sum_{d=1}^n \mu(d) \lfloor \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{d} \rfloor} \lfloor \frac{n}{id} \rfloor \lfloor \frac{m}{jd} \rfloor \rfloor \quad \text{预处理 } \sum_{i=1}^n \lfloor \frac{n}{i} \rfloor, \text{ 对 } \mu \text{ 求前缀和, 对 } n \text{ 和 } m \text{ 整除分块} \end{aligned}$$

```

1  const int N=5e4+5,maxn=5e4;
2  int p[N],hp,mu[N];
3  bool vis[N];
4  ll sum[N];
5  void sieve(){
6      mu[1]=1;
7      for(int i=2;i<=maxn;++i){
8          if(!vis[i])p[++hp]=i,mu[i]=-1;
9          for(int j=1;j<=hp&&i*p[j]<=maxn;++j){
10             vis[i*p[j]]=1;

```

```

11         if(i%p[j]==0)break;
12         mu[i*p[j]]=-mu[i];
13     }
14 }
15 }
16 int main(){
17     sieve();
18     for(int i=1;i<=maxn;++i)mu[i]+=mu[i-1];
19     for(int i=1;i<=maxn;++i){
20         for(int l=1,r;l<=i;l=r+1){
21             r=i/(i/l);
22             sum[i]+=1ll*(i/l)*(r-l+1);
23         }
24     }
25     int T;scanf("%d",&T);
26     while(T--){
27         int n,m;scanf("%d%d",&n,&m);
28         if(n>m)swap(n,m);
29         ll ans=0;
30         for(int l=1,r;l<=n;l=r+1){
31             r=min(n/(n/l),m/(m/l));
32             ans+=1ll*(mu[r]-mu[l-1])*sum[n/l]*sum[m/l];
33         }
34         printf("%lld\n",ans);
35     }
36 }

```

1.11 中国剩余定理

$$\begin{cases} x \equiv a[1] \pmod{m[1]} \\ x \equiv a[2] \pmod{m[2]} \\ x \equiv a[3] \pmod{m[3]} \\ \dots \\ x \equiv a[n] \pmod{m[n]} \end{cases}$$

满足对任意 $i, j : \gcd(m[i], m[j]) = 1$

设 $s = m[1] * m[2] * m[3] * \dots * m[n]$

则解为 $x = (\sum a[i] * s/m[i] * \text{inv}(s/m[i])(s/m[i] \text{ 对 } m[i] \text{ 的逆元})) \% s$

```

1  typedef __int128 ll; // 可能会爆ll
2  ll n, m[15], a[15], s;

```

```

3  ll ans;
4  void exgcd(ll a,ll b,ll &x,ll &y){
5      if(!b)x=1,y=0;
6      else exgcd(b,a%b,y,x),y-=a/b*x;
7  }
8  int main(){
9      cin>>n;s=1;
10     for(int i=1;i<=n;++i){
11         cin>>m[i]>>a[i];
12         s*=m[i];//需要保证m[i]之积在__int128范围之内
13     }
14     for(int i=1;i<=n;++i){
15         ll x,y,t=s/m[i];
16         exgcd(t,m[i],x,y);
17         x=(x+m[i])%m[i];
18         ans+=a[i]*t*x;
19     }
20     printf("%lld",ans%s);
21 }

```

扩展中国剩余定理:

$$\begin{cases} x \equiv a[1] \pmod{m[1]} \\ x \equiv a[2] \pmod{m[2]} \\ x \equiv a[3] \pmod{m[3]} \\ \dots \\ x \equiv a[n] \pmod{m[n]} \end{cases}$$

其中 $m[i]$ 不一定两两互质

设 $lcm[i] = lcm(m[1], m[2] \dots m[i])$ $lcm[1] = m[1]$

设 $ans[i] =$ 方程 $1 \sim i$ 的解 $ans[1] = a[1]$

方程 $1 \sim i-1$ 的通解为 $ans[i-1] + t * lcm[i-1]$

解同余方程 $ans[i-1] + t * lcm[i-1] \equiv a[i] \pmod{m[i]}$

即 $t * lcm[i-1] \equiv a[i] - ans[i-1] \pmod{m[i]}$

即 $lcm[i-1] * t + m[i] * k = a[i] - ans[i-1]$

若 $a[i] - ans[i-1]$ 是 $gcd(lcm[i-1], m[i])$ 的整数倍, 则 t 有解 (反之无解)

设方程 $lcm[i-1] * x + m[i] * y = gcd(lcm[i-1], m[i])$ 的解为 x

$t = x * (a[i] - ans[i-1]) / gcd(lcm[i-1], m[i])$

递推出 $ans[i] = ans[i-1] + t * lcm[i-1]$, 再递推出 $lcm[i]$

$ans[i]$ 可表示为 $ans1[i] + t1 * lcm[i]$

将 $ans[i]$ 化为最小正整数 $(ans[i] \% lcm[i] + lcm[i]) \% lcm[i]$

其中 ans 和 lcm 都可以用单个变量递推, 不需要数组

```

1  typedef __int128 ll;//可能会爆long long
2  ll n,m[100],a[100],s;
3  ll ans;
4  void exgcd(ll a,ll b,ll &x,ll &y){
5      if(!b)x=1,y=0;
6      else exgcd(b,a%b,y,x),y-=a/b*x;
7  }
8  int main(){
9      cin>>n;s=1;
10     for(int i=1;i<=n;++i){
11         cin>>m[i]>>a[i];
12         s*=m[i];//需要保证m[i]之积在__int128范围之内
13     }
14     for(int i=1;i<=n;++i){
15         ll x,y,t=s/m[i];
16         exgcd(t,m[i],x,y);
17         x=(x+m[i])%m[i];
18         ans+=a[i]*t*x;
19     }
20     printf("%lld",ans%s);
21 }

```

1.12 BSGS/exBSGS

在 $O(\sqrt{\varphi(p)})$ 的时间复杂度内求 $a^x \equiv b \pmod{p}$

```

1  ll qpow(ll a,ll b,ll p){
2      ll s=1;
3      for(;b;b>>=1){
4          if(b&1)s=s*a%p;
5          a=a*a%p;
6      }
7      return s;
8  }
9  ll bsgs(ll a,ll b,ll p){//p是质数,求a^x=b(mod p)最小自然数x
10     unordered_map<ll,ll>mp;
11     b%=p; ll t=sqrt(p)+1;
12     for(ll i=1;i<=t;++i){
13         b=b*a%p;
14         mp[b]=i;
15     }

```

```

16     a=qpow(a,t,p);
17     if(!a)return b==0?1:-1;
18     ll f=1;
19     for(ll i=1;i<=t;++i){
20         f=f*a%p;
21         int j=mp.find(f)==mp.end()?-1:mp[f];
22         if(j>=0&&i*t-j>=0)return i*t-j;
23     }
24     return -1;//无解
25 }
26 //-----exBSGS
27 ll exgcd(ll a,ll b,ll &x,ll &y){
28     if(!b){x=1;y=0;return a;}
29     ll g=exgcd(b,a%b,x,y);
30     ll t=x;x=y;y=t-a/b*y;
31     return g;
32 }
33 ll exbsgs(ll a,ll b,ll p) {// a^x=b(mod p)最小自然数x
34     if(b==1||p==1)return 0;
35     b%=p;ll f=1;
36     for (int i = 0; i < 30; i++) {
37         if(f==b)return i;
38         f=f*a%p;
39     }
40     ll x,y;
41     ll g=exgcd(f,p,x,y);
42     if(x<0)x+=p;
43     if(b%g)return -1;
44     f/=g,b/=g,p/=g;
45     b=b*x%p;
46     ll ret=bsgs(a,b,p);
47     return ~ret?ret+30:-1;//-1表示无解
48 }

```

1.13 Lucas 定理

```

1 ll C(ll n,ll m){
2     return fac[n]*qpow(fac[m],mod-2)%mod*qpow(fac[n-m],mod-2)%mod;
3 }
4 ll C(ll n,ll m){
5     return fac[n]*ifac[m]%mod*ifac[n-m]%mod;//预处理阶乘逆元后

```



```

6 }
7 ll lucas(ll n,ll m){//C(n,m)%mod
8     if(!m)return 1;
9     return C(n%mod,m%mod)*lucas(n/mod,m/mod)%mod;
10 }
    
```

1.14 杜教筛

再 $O(n^{3/4})$ 时间复杂度内求积性函数前缀和

$$\sum_{i=1}^n f(i) = S(n)$$

构造一个积性函数 g , 使得 $f * g$ 的前缀和很好求

$$\sum_{i=1}^n (f * g)(i) = \sum_{i=1}^n \sum_{d|i} f(d)g(\frac{i}{d}) = \sum_{d=1}^n g(d) \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} f(i) = \sum_{d=1}^n g(d)S(\lfloor \frac{n}{d} \rfloor)$$

$$g(1)S(n) = \sum_{i=1}^n g(i)S(\lfloor \frac{n}{i} \rfloor) - \sum_{i=2}^n g(i)S(\lfloor \frac{n}{i} \rfloor) = \sum_{i=1}^n (f * g)(i) - \sum_{i=2}^n g(i)S(\lfloor \frac{n}{i} \rfloor)$$

```

1 ll getS(int n){
2     if(n<=N)return sum[n]; //N以内线性筛
3     if(mp[n])return mp[n]; //N以外unordered_map记忆化
4     ll ans=f_g_sum(n); //f*g的前缀和要能O(1)算出
5     for(int l=2,r;l<=n;l=r+1){
6         int t=n/l; r=n/t;
7         ret+=(sumg(r)-sumg(l-1))*getS(t); //sum(g)的前缀和要能O(1)算出
8     }
9     return mp[n]=ans; // 记忆化
10 }
    
```

求 $\sum_{i=1}^n \varphi(i)$, 需要构造 $g(n) = 1(n) = 1, \varphi * 1 = id$

求 $\sum_{i=1}^n \mu(i)$, 需要构造 $g(n) = 1(n) = 1, \mu * 1 = \varepsilon$

```

1 const int N=2e6; //N > pow(maxn,2.0/3)
2 int p[N+5],hp;
3 bool vis[N+5];
4 ll phi[N+5],mu[N+5];
5 void seive(){
6     phi[1]=1;mu[1]=1;
7     for(int i=2;i<=N;++i){
8         if(!vis[i]){
9             p[++hp]=i;
10            mu[i]=-1;
11            phi[i]=i-1;
        }
    }
    
```

```

12     }
13     for(int j=1;j<=hp&& i*p[j]<=N;++j){
14         vis[i*p[j]]=1;
15         if(i%p[j]==0){
16             phi[i*p[j]]=phi[i]*p[j];
17             break;
18         }
19         phi[i*p[j]]=phi[i]*(p[j]-1);
20         mu[i*p[j]]=-mu[i];
21     }
22 }
23 }
24 unordered_map<int,ll>mp_phi,mp_mu;
25 ll sumphi(ll n){
26     if(n<=N)return phi[n];
27     if(mp_phi.count(n))return mp_phi[n];
28     ll ans=n*(n+1)/2;
29     for(ll l=2,r;l<=n;l=r+1){
30         ll t=n/l;r=n/t;
31         ans-=(r-l+1)*sumphi(t);
32     }
33     return mp_phi[n]=ans;
34 }
35 ll summu(ll n){
36     if(n<=N)return mu[n];
37     if(mp_mu.count(n))return mp_mu[n];
38     ll ans=1;
39     for(ll l=2,r;l<=n;l=r+1){
40         ll t=n/l;r=n/t;
41         ans-=(r-l+1)*summu(t);
42     }
43     return mp_mu[n]=ans;
44 }
45 int main(){
46     seive();
47     for(int i=1;i<=N;++i){
48         phi[i]+=phi[i-1];
49         mu[i]+=mu[i-1];
50     }
51     int T;scanf("%d",&T);
52     while(T--){

```

```

53         ll n;scanf("%lld",&n);
54         printf("%lld %lld\n",sumphi(n),summu(n));
55     }
56 }

```

1.15 Pollar-Rho

```

1  typedef long long ll;
2  typedef unsigned long long ull;
3  typedef long double db;
4  const int prime[]={2,3,5,7,11,13,17,19,23,29,31,37};
5  ll p[105];int hp;
6  mt19937_64 rnd(19260817);
7  ull add(ull x,ull y,ull mod){return x+y>=mod?x+y-mod:x+y;}
8  ull mul(ull x,ull y,ull mod){//x*y%mod,卡常用,不要怀疑这个函数
9      ll s=x*y-ull(db(1)/mod*x*y)*mod;
10     return s<0?s+=mod:(s>=mod?s-=mod:s);
11 }
12 ll qpow(ll a,ll b,const ll mod){
13     ll s=1;
14     while(b){
15         if(b&1)s=mul(s,a,mod);
16         a=mul(a,a,mod);
17         b>>=1;
18     }
19     return s;
20 }
21 bool check(ll p,ll n){
22     int k=0;ll d=n-1,x,s;
23     while(d&1^1)d>>=1,++k;
24     x=qpow(p,d,n);
25     while(k){
26         s=mul(x,x,n);
27         if(s==1)return x==1||x==n-1;
28         --k;x=s;
29     }
30     return false;
31 }
32 bool miller_rabin(ll n){
33     for (int i=0;i<12;++i){
34         if (n==prime[i]) return true;

```

```
35     if (!(n%prime[i])) return false;
36     if (!check(prime[i],n)) return false;
37 }
38 return true;
39 }
40 ll gcd(ll a,ll b){
41     if(!a)return b;if(!b)return a;
42     int t=__builtin_ctzll(a|b);
43     a>>=__builtin_ctzll(a);
44     do{
45         b>>=__builtin_ctzll(b);
46         if(a>b)swap(a,b);
47         b-=a;
48     }while (b);
49     return a<<t;
50 }
51 #define stp(x) (add(mul(x,x,n),r,n))
52 ll find_div(ll n){
53     if(n&1^1) return 2;
54     static const int S=128;
55     ll x=rnd()%n,r=rnd()%(n-1)+1;
56     for(int l=1;l<=1){
57         ll y=x,p=1;
58         for(int i=0;i<l;i++) x=stp(x);
59         for(int i=0;i<l;i+=S){
60             ll z=x;
61             for(int j=0;j<S&&j<l-i;++j)x=stp(x),p=mul(p,x+n-y,n);
62             p=gcd(p,n);
63             if(p==1)continue;
64             if(p<n)return p;
65             for (p=1,x=z;p==1;)x=stp(x),p=gcd(x+n-y,n);
66             return p;
67         }
68     }
69 }
70 void pollar_rho(ll n){
71     if(n==1) return;
72     if(miller_rabin(n)) {
73         p[++hp]=n; return;
74     }
75     ll d=n;
```

```

76     while(d==n)d=find_div(n);
77     pollar_rho(d);
78     pollar_rho(n/d);
79 }
80 int main(){
81     ll n;scanf("%lld",&n);
82     hp=0;pollar_rho(n);//p[1~hp]为质因数
83 }

```

1.16 N 以内素数个数

```

1  #define N 216000
2  #define ll long long
3  using namespace std;
4  int mn[N],pri[N/10],fl[N];
5  int tot,cnt,num,n;
6  int f[20005][55];
7  int inf=2e9;
8  int dp(int x,int y){
9      if (x<=20000&&y<=50) return f[x][y];
10     if (x==0||y==0) return x;
11     if (1ll*pri[y]*pri[y]>=x&&x<N) return max(0,mn[x]-y);
12     return dp(x,y-1)-dp(x/pri[y],y-1);
13 }
14 void pre(){
15     for (int i=2;i<N;i++){
16         if (!fl[i]) pri[++tot]=i;
17         for (int j=1;i*pri[j]<N&&j<=tot;j++){
18             fl[i*pri[j]]=1;
19             if (i%pri[j]==0) break;
20         }
21     }
22     for (int i=1;i<N;i++)
23         mn[i]=(cnt+=1-fl[i]);
24     for (int i=1;i<=20000;i++) f[i][0]=i;
25     for (int i=1;i<=20000;i++)
26         for (int j=1;j<=50;j++)
27             f[i][j]=f[i][j-1]-f[i/pri[j]][j-1];
28 }
29 int power(int x,int y){
30     int s=1;

```

```

31     while (y!=0){
32         if (y&1){
33             if (s>=inf/x) s=inf;
34             else s=s*x;
35         }
36         y/=2;
37         if (x>=inf/x) x=inf;
38         else x=x*x;
39     }
40     return s;
41 }
42 int yroot(ll x,int y){
43     int l=2,r=6666,ans=1;
44     while (l<=r){
45         int mid=(l+r)/2;
46         if (power(mid,y)<=x) ans=mid,l=mid+1;
47         else r=mid-1;
48     }
49     return ans;
50 }
51 int work(int m){
52     if (m<N) return mn[m]-1;
53     int y=yroot(m,3),n=mn[y];
54     int ans=dp(m,n)+n-1;
55     for (n++;pri[n]*pri[n]<=m;n++)
56         ans-=mn[m/pri[n]]-mn[pri[n]]+1;
57     return ans;
58 }
59 int main(){
60     pre(); scanf("%d",&n);
61     printf("%d\n",work(n));
62 }

```

2 数学

2.1 快速傅里叶变换 FFT

```

1  const int N = 2e6 + 100;
2  const long double pi = acos(-1.0);
3  typedef std::vector<int> Poly;
4  namespace Pol {

```

```

5    //const int N = 4200000; // 200MB
6    const int N = 2100000; // 100MB
7    const double pi = acos(-1);
8    struct Complex {
9        double x, y;
10       Complex(double x = 0, double y = 0) : x(x), y(y) {}
11       friend Complex operator + (const Complex &u, const Complex &v) { return
           Complex(u.x + v.x, u.y + v.y); }
12       friend Complex operator - (const Complex &u, const Complex &v) { return
           Complex(u.x - v.x, u.y - v.y); }
13       friend Complex operator * (const Complex &u, const Complex &v) {
14           return Complex(u.x * v.x - u.y * v.y, u.x * v.y + u.y * v.x);
15       }
16   } a[N], b[N];
17   int P[N];
18   void init_P(int n) {
19       int l = 0; while ((1 << l) < n) ++l;
20       for (int i = 0; i < n; ++i) P[i] = (P[i >> 1] >> 1) | ((i & 1) << l - 1);
21   }
22   vector<Complex> init_W(int n) {
23       vector<Complex> w(n); w[1] = 1;
24       for (int i = 2; i < n; i <= 1) {
25           auto w0 = w.begin() + i / 2, w1 = w.begin() + i;
26           Complex wn(cos(pi / i), sin(pi / i));
27           for (int j = 0; j < i; j += 2)
28               w1[j] = w0[j >> 1], w1[j + 1] = w1[j] * wn;
29       }
30       return w;
31   } auto w = init_W(1 << 21);
32   void DIT(Complex *a, int n) {
33       for (int k = n >> 1; k; k >= 1)
34           for (int i = 0; i < n; i += k << 1)
35               for (int j = 0; j < k; ++j) {
36                   Complex x = a[i + j], y = a[i + j + k];
37                   a[i + j + k] = (x - y) * w[k + j], a[i + j] = x + y;
38               }
39   }
40   void DIF(Complex *a, int n) {
41       for (int k = 1; k < n; k <= 1)
42           for (int i = 0; i < n; i += k << 1)
43               for (int j = 0; j < k; ++j) {

```

```

44         Complex x = a[i + j], y = a[i + j + k] * w[k + j];
45         a[i + j + k] = x - y, a[i + j] = x + y;
46     }
47     //         for (int i = 0; i < n; ++i) a[i].x /= 4 * n, a[i].y /= 4 * n; //
        两次
48     for (int i = 0; i < n; ++i) a[i].x /= n, a[i].y /= n; // 三次
49     reverse(a + 1, a + n);
50 }
51 /*
52 Poly Mul(const Poly &A, const Poly &B, int n1, int n2) { // 三次变两次更快精
        度差一点
53     int n = 1; while (n < n1 + n2 - 1) n <= 1; init_P(n);
54     fill(a, a + n, Complex(0, 0));
55     for (int i = 0; i < n1; ++i) a[i].x += A[i], a[i].y += A[i];
56     for (int i = 0; i < n2; ++i) a[i].x += B[i], a[i].y -= B[i];
57     //for (int i = 0; i < max(n1, n2); ++i) a[i] = Complex(A[i] + B[i], A[i]
        - B[i]);
58     DIT(a, n);
59     for (int i = 0; i < n; ++i) a[i] = a[i] * a[i];
60     DIF(a, n); Poly ans(n1 + n2 - 1); for (int i = 0; i < n1 + n2 - 1; ++i)
        ans[i] = (int) (a[i].x + 0.5);
61     return ans;
62 }*/
63 Poly Mul(const Poly &A, const Poly &B, int n1, int n2) {
64     int n = 1; while (n < n1 + n2 - 1) n <= 1; init_P(n);
65     for (int i = 0; i < n1; ++i) a[i] = Complex(A[i], 0);
66     for (int i = 0; i < n2; ++i) b[i] = Complex(B[i], 0);
67     fill(a + n1, a + n, Complex(0, 0)); fill(b + n2, b + n, Complex(0, 0));
68     DIT(a, n); DIT(b, n);
69     for (int i = 0; i < n; ++i) a[i] = a[i] * b[i];
70     DIF(a, n); Poly ans(n1 + n2 - 1); for (int i = 0; i < n1 + n2 - 1; ++i)
        ans[i] = (int) (a[i].x + 0.5);
71     return ans;
72 }
73 Poly MMul(const Poly &A, const Poly &B, int len) { // 减法卷积
74     int n = 1; while (n < 2 * len - 1) n <= 1; init_P(n);
75     for (int i = 0; i < len; ++i) a[i] = Complex(A[i], 0);
76     for (int i = 0; i < len; ++i) b[i] = Complex(B[i], 0);
77     fill(a + len, a + n, Complex(0, 0)); fill(b + len, b + n, Complex(0, 0));
        reverse(b, b + len);
78     DIT(a, n); DIT(b, n);

```



```

79     for (int i = 0; i < n; ++i) a[i] = a[i] * b[i];
80     DIF(a, n); Poly ans(len); for (int i = 0; i < len; ++i) ans[i] = (int) (a
      [i].x + 0.5);
81     reverse(ans.begin(), ans.end()); return ans;
82 }
83 } // namespace Pol
84 int main() {
85     int n, m; cin >> n >> m;
86     Poly A(n+1); Poly B(m+1);
87     for (int i = 0; i <= n; i++) cin >> A[i];
88     for (int i = 0; i <= m; i++) cin >> B[i];
89     A = Pol::Mul(A, B, n+1, m+1);
90     for (int i = 0; i <= n+m; i++) cout << A[i] << " ";
91 }

```

2.2 快速数论变换 NTT

2.2.1 原根表

原根表: g 是 $\text{mod} = r \times 2^k + 1$ 的原根

mod	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3

3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

2.2.2 普通 NTT

```

1  #define ll long long
2  const int mod = 998244353;
3  int add(int x, int y) { return (x += y) >= mod ? x - mod : x; }
4  int mul(int x, int y) { return 1ll * x * y % mod; }
5  int add(initializer_list<int> lst) { int s = 0; for (auto t : lst) s = add(s, t);
   return s; }
6  int mul(initializer_list<int> lst) { int s = 1; for (auto t : lst) s = mul(s, t);
   return s; }
7  ll qpow(ll a, ll b) {
8      ll res = 1;
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 #define Poly vector<int>
17 #define len(A) ((int) A.size())

```

```

18 namespace Pol {
19     int add(int a, int b) { return (a += b) >= mod ? a -= mod : a; }
20     int mul(int a, int b) { return 1ll * a * b % mod; }
21     Poly operator - (const int &v, const Poly &a) {
22         Poly res(a);
23         for (int i = 0; i < len(res); ++i) res[i] = mod - res[i];
24         res[0] = add(res[0], v); return res;
25     }
26     Poly operator - (const Poly &a, const int &v) {
27         Poly res(a); res[0] = add(res[0], mod - v); return res;
28     }
29     Poly operator * (const Poly &a, const int &v) {
30         Poly res(a);
31         for (int i = 0; i < len(res); ++i) res[i] = mul(res[i], v);
32         return res;
33     }
34
35     const int N = 4200000;
36     const int G = 3;
37
38     int P[N], inv[N], fac[N];
39     void init_P(int n) {
40         int l = 0; while ((1 << l) < n) ++l;
41         for (int i = 0; i < n; ++i) P[i] = (P[i >> 1] >> 1) | ((i & 1) << l - 1);
42     }
43     void init_C() {
44         if (fac[0]) return;
45         fac[0] = 1; for (int i = 1; i < N; ++i) fac[i] = mul(fac[i - 1], i);
46         inv[N - 1] = qpow(fac[N - 1], mod - 2); for (int i = N - 2; ~i; --i) inv[
            i] = mul(inv[i + 1], i + 1);
47     }
48     vector<int> init_W(int n) {
49         vector<int> w(n); w[1] = 1;
50         for (int i = 2; i < n; i <= 1) {
51             auto w0 = w.begin() + i / 2, w1 = w.begin() + i;
52             int wn = qpow(G, (mod - 1) / (i << 1));
53             for (int j = 0; j < i; j += 2)
54                 w1[j] = w0[j >> 1], w1[j + 1] = mul(w1[j], wn);
55         }
56         return w;
57     } auto w = init_W(1 << 21);

```

```

58 void DIT(Poly &a) {
59     int n = len(a);
60     for (int k = n >> 1; k; k >>= 1)
61         for (int i = 0; i < n; i += k << 1)
62             for (int j = 0; j < k; ++j) {
63                 int x = a[i + j], y = a[i + j + k];
64                 a[i + j + k] = mul(add(x, mod - y), w[k + j]), a[i + j] = add(x, y);
65             }
66 }
67 void DIF(Poly &a) {
68     int n = len(a);
69     for (int k = 1; k < n; k <= 1)
70         for (int i = 0; i < n; i += k << 1)
71             for (int j = 0; j < k; ++j) {
72                 int x = a[i + j], y = mul(a[i + j + k], w[k + j]);
73                 a[i + j + k] = add(x, mod - y), a[i + j] = add(x, y);
74             }
75     int inv = qpow(n, mod - 2);
76     for (int i = 0; i < n; ++i) a[i] = mul(a[i], inv);
77     reverse(a.begin() + 1, a.end());
78 }
79 Poly operator * (const Poly &A, const Poly &B) {
80     int n = 1, n1 = len(A), n2 = len(B); while (n < n1 + n2 - 1) n <= 1;
81     init_P(n);
82     Poly a(n), b(n);
83     for (int i = 0; i < n1; ++i) a[i] = add(A[i], mod);
84     for (int i = 0; i < n2; ++i) b[i] = add(B[i], mod);
85     DIT(a); DIT(b);
86     for (int i = 0; i < n; ++i) a[i] = mul(a[i], b[i]);
87     DIF(a); a.resize(n1 + n2 - 1); return a;
88 }
89 Poly MMul(const Poly &A, const Poly &B) { // 差卷积, 默认 A 和 B 的长度相同
90     //C[k]=(i=k~n)A[i-k]B[i]=(i=0~n-k)A[n-k-i]B'[i],B'[k]=B[n-k]
91     int n = 1, L = len(A); while (n < 2 * L - 1) n <= 1; init_P(n);
92     Poly a(n), b(n);
93     for (int i = 0; i < L; ++i) a[i] = add(A[i], mod);
94     for (int i = 0; i < L; ++i) b[i] = add(B[i], mod);
95     reverse(b.begin(), b.begin() + L);
96     DIT(a); DIT(b);
97     for (int i = 0; i < n; ++i) a[i] = mul(a[i], b[i]);
98     DIF(a); a.resize(L); reverse(a.begin(), a.end()); return a;

```

```

98     }
99     Poly Der(const Poly &a) {
100         Poly res(a);
101         for (int i = 0; i < len(a) - 1; ++i) res[i] = mul(i + 1, res[i + 1]);
102         res[len(a) - 1] = 0; return res;
103     }
104     Poly Int(const Poly &a) {
105         static int inv[N];
106         if (!inv[1]) {
107             inv[1] = 1;
108             for (int i = 2; i < N; ++i) inv[i] = mul(mod - mod / i, inv[mod % i])
109                 ;
110         }
111         Poly res(a); res.resize(len(a) + 1);
112         for (int i = len(a); i --i) res[i] = mul(res[i - 1], inv[i]);
113         res[0] = 0; return res;
114     }
115     Poly Inv(const Poly &a) {
116         Poly res(1, qpow(a[0], mod - 2));
117         int n = 1; while (n < len(a)) n <<= 1;
118         for (int k = 2; k <= n; k <<= 1) {
119             int L = 2 * k; init_P(L); Poly t(L);
120             copy_n(a.begin(), min(k, len(a)), t.begin());
121             t.resize(L); res.resize(L);
122             DIT(res); DIT(t);
123             for (int i = 0; i < L; ++i) res[i] = mul(res[i], add(2, mod - mul(t[i]
124                 ], res[i])));
125             DIF(res); res.resize(k);
126         } res.resize(len(a)); return res;
127     }
128     Poly Offset(const Poly &a, int c) { //多项式平移,  $G[x] = F(x+c)(\text{mod } x^n)$  c为常
129         数
130         // 令  $F[x] = (i=0-n) f_i x^i$ 
131         int n = len(a); init_C();
132         Poly t1(n), t2(n);
133         for (int i = 0; i < n; ++i) t1[i] = mul(qpow(c, i), inv[i]);
134         for (int i = 0; i < n; ++i) t2[i] = mul(a[i], fac[i]);
135         t1 = MMul(t1, t2);
136         for (int i = 0; i < n; ++i) t1[i] = mul(t1[i], inv[i]);
137         return t1;
138     }

```

```

136 pair<Poly, Poly> Divide(const Poly &a, const Poly &b) { // 已知n次F[x]和m次G[x
    ], 求出一个n-m+1次Q[x]和一个次数小于m的R[x]满足F[x]=Q[x]G[x]+R[x]
137     int n = len(a), m = len(b);
138     Poly t1(a.rbegin(), a.rbegin() + n - m + 1), t2(b.rbegin(), b.rend()); t2
        .resize(n - m + 1);
139     Poly Q = Inv(t2) * t1; Q.resize(n - m + 1); reverse(Q.begin(), Q.end());
140     Poly R = Q * b; R.resize(m - 1); for (int i = 0; i < len(R); ++i) R[i] =
        add(a[i], mod - R[i]);
141     return make_pair(Q, R);
142 }
143 Poly Ln(const Poly &a) {
144     Poly res = Int(Der(a) * Inv(a));
145     res.resize(len(a)); return res;
146 }
147 Poly Exp(const Poly &a) {
148     Poly res(1, 1);
149     int n = 1; while (n < len(a)) n <<= 1;
150     for (int k = 2; k <= n; k <<= 1) {
151         Poly t(res.begin(), res.end()); t.resize(k); t = Ln(t);
152         for (int i = 0; i < min(len(a), k); ++i) t[i] = add(a[i], mod - t[i])
            ; t[0] = add(t[0], 1);
153         res = res * t; res.resize(k);
154     } res.resize(len(a)); return res;
155 }
156 Poly Sqrt(const Poly &a) { // a[0] = 1
157     Poly res(1, 1); ll inv2 = qpow(2, mod - 2);
158     int n = 1; while (n < len(a)) n <<= 1;
159     for (int k = 2; k <= n; k <<= 1) {
160         Poly t(res.begin(), res.end()), ta(a.begin(), a.begin() + min(len(a),
            k));
161         t.resize(k); t = Inv(t) * ta;
162         res.resize(k); for (int i = 0; i < k; ++i) res[i] = mul(add(res[i], t
            [i]), inv2);
163     } res.resize(len(a)); return res;
164 }
165 Poly Pow(const Poly &a, int k) { // a[0] = 1
166     return Exp(Ln(a) * k);
167 }
168 Poly Pow(const Poly &a, int k, int kk) {
169     int n = len(a), t = n, m, v, inv, powv; Poly res(n);
170     for (int i = n - 1; ~i; --i) if (a[i]) t = i, v = a[i];

```

```

171     if (k && t >= (n + k - 1) / k) return res;
172     if (t == n) { if (!k) res[0] = 1; return res; }
173     m = n - t * k; res.resize(m);
174     inv = qpow(v, mod - 2); powv = qpow(v, kk);
175     for (int i = 0; i < m; ++i) res[i] = mul(a[i + (k > 0) * t], inv);
176     res = Exp(Ln(res) * k); res.resize(n);
177     for (int i = m - 1; ~i; --i) {
178         int tmp = mul(res[i], powv);
179         res[i] = 0, res[i + t * k] = tmp;
180     }
181     return res;
182 }
183 } // namespace Pol

```

2.2.3 分治 NTT

```

1 // fi = (j = 1-i)f[i-j]g[j] g[j]已知 f[0]=1;
2 void solve(Poly &A, const Poly &B, int l, int r) {
3     if (l + 1 == r) return ;
4     int m = l + r >> 1; solve(A,B,l, m);
5     Poly t1(A.begin() + l, A.begin() + m), t2(B.begin(), B.begin() + r - 1);
6     Poly t = Pol::operator*(t1, t2);
7     for (int i = m; i < r; ++i) {
8         // if (i-l>t[i].size()) break;//注意有些题可能没有两倍长转移
9         A[i] = (A[i] + t[i - l]) % p;
10    }
11    solve(A,B,m, r);
12 }
13 int main() {
14     cin >> n;
15     Poly A(n), B(n); A[0] = 1;
16     for (int i = 1; i < n; ++i) cin >> B[i];
17     solve(A,B,0,n);
18     for (int i = 0; i < n; ++i) cout << A[i] << " ";
19 }

```

2.2.4 三模 NTT

```

1 #define ll long long
2 int p;
3 const int mod1 = 998244353, mod2 = 1004535809, mod3 = 469762049, G = 3;

```

```

4  const ll mod12 = 1ll * mod1 * mod2;
5  ll pow_mod(ll x, ll n, int p) {
6      ll s = 1;
7      for (; n; n >>= 1, x = x * x % p)
8          if (n & 1) s = s * x % p;
9      return s;
10 }
11 const int inv1 = pow_mod(mod1, mod2 - 2, mod2), inv2 = pow_mod(mod12 % mod3, mod3
    - 2, mod3);
12 inline int mod(int x, int p) { return x >= p ? x - p : x; }
13 struct Int {
14     int x, y, z;
15
16     Int() { x = y = z = 0; }
17     Int(int x, int y, int z) : x(x % mod1), y(y % mod2), z(z % mod3) {}
18     Int(int v) : x(v % mod1), y(v % mod2), z(v % mod3) {}
19
20     static inline int add(int x, int y, int p) { return (x += y) >= p ? x - p : x
        ; }
21     inline friend Int operator + (const Int &u, const Int &v) {
22         return Int(add(u.x, v.x, mod1), add(u.y, v.y, mod2), add(u.z, v.z, mod3))
            ;
23     }
24     inline friend Int operator - (const Int &u, const Int &v) {
25         return Int(add(u.x, mod1 - v.x, mod1), add(u.y, mod2 - v.y, mod2), add(u.
            z, mod3 - v.z, mod3));
26     }
27     inline friend Int operator * (const Int &u, const Int &v) {
28         return Int(1ll * u.x * v.x % mod1, 1ll * u.y * v.y % mod2, 1ll * u.z * v.
            z % mod3);
29     }
30     inline int get() const {
31         ll v = 1ll * add(y, mod2 - x, mod2) * inv1 % mod2 * mod1 + x;
32         return (1ll * add(z, mod3 - v % mod3, mod3) * inv2 % mod3 * (mod12 % p) %
            p + v) % p;
33     }
34 };
35
36 typedef vector<Int> Poly;
37 #define len(a) ((int) a.size())
38 namespace Pol {

```



```

39 Poly operator - (const Int &v, const Poly &a) {
40     Poly res(a);
41     for (int i = 0; i < len(res); ++i) res[i] = p - res[i];
42     res[0] = res[0] + v; return res;
43 }
44 Poly operator - (const Poly &a, const Int &v) {
45     Poly res(a); res[0] = res[0] + (p - v); return res;
46 }
47 Poly operator * (const Poly &a, const Int &v) {
48     Poly res(a);
49     for (int i = 0; i < len(res) ; ++i) res[i] = res[i] * v;
50     return res;
51 }
52
53 const int N = 4200000;
54
55 int P[N];
56 void init_P(int n) {
57     int l = 0; while ((1 << l) < n) ++l;
58     for (int i = 0; i < n; ++i) P[i] = (P[i >> 1] >> 1) | ((i & 1) << l - 1);
59 }
60 vector<Int> init_W(int n) {
61     vector<Int> w(n); w[1] = 1;
62     for (int i = 2; i < n; i <= 1) {
63         auto w0 = w.begin() + i / 2, w1 = w.begin() + i;
64         Int wn = Int(pow_mod(G, (mod1 - 1) / (i << 1), mod1),
65             pow_mod(G, (mod2 - 1) / (i << 1), mod2),
66             pow_mod(G, (mod3 - 1) / (i << 1), mod3));
67         for (int j = 0; j < i; j += 2)
68             w1[j] = w0[j >> 1], w1[j + 1] = w1[j] * wn;
69     }
70     return w;
71 } auto w = init_W(1 << 21);
72 void DIT(Poly &a) {
73     int n = len(a);
74     for (int k = n >> 1; k >= 1)
75         for (int i = 0; i < n; i += k << 1)
76             for (int j = 0; j < k; ++j) {
77                 Int x = a[i + j], y = a[i + j + k];
78                 a[i + j + k] = (x - y) * w[k + j], a[i + j] = x + y;
79             }

```

```

80     }
81     void DIF(Poly &a) {
82         int n = len(a);
83         for (int k = 1; k < n; k <= 1)
84             for (int i = 0; i < n; i += k < 1)
85                 for (int j = 0; j < k; ++j) {
86                     Int x = a[i + j], y = a[i + j + k] * w[k + j];
87                     a[i + j + k] = x - y, a[i + j] = x + y;
88                 }
89         Int inv = Int(pow_mod(n, mod1 - 2, mod1), pow_mod(n, mod2 - 2, mod2),
90                     pow_mod(n, mod3 - 2, mod3));
91         for (int i = 0; i < n; ++i) a[i] = a[i] * inv;
92         reverse(a.begin() + 1, a.end());
93     }
94     Poly restore(Poly &a) {
95         for (int i = 0; i < len(a); ++i) {
96             int x = a[i].get();
97             a[i] = Int(x % mod1, x % mod2, x % mod3);
98         }
99         return a;
100     }
101     Poly operator * (const Poly &A, const Poly &B) {
102         int n = 1, n1 = len(A), n2 = len(B); while (n < n1 + n2 - 1) n <= 1;
103         init_P(n);
104         Poly a(n), b(n); copy_n(A.begin(), n1, a.begin()); copy_n(B.begin(), n2,
105                             b.begin());
106         DIT(a); DIT(b);
107         for (int i = 0; i < n; ++i) a[i] = a[i] * b[i];
108         DIF(a); Poly ans(n1 + n2 - 1);
109         for (int i = 0; i < n1 + n2 - 1; ++i) ans[i] = a[i].get();
110         return ans;
111     }
112     Poly Der(const Poly &a) {
113         Poly res(a);
114         for (int i = 0; i < len(a) - 1; ++i) res[i] = (i + 1) * res[i + 1];
115         res[len(a) - 1] = 0; restore(res); return res;
116     }
117     Poly Inte(const Poly &a) {
118         static int inv[N];
119         if (!inv[1]) {
120             inv[1] = 1;

```

```

118         for (int i = 2; i < N; ++i) inv[i] = 1ll * (p - p / i) * inv[p % i] %
           p;
119     }
120     Poly res(a); res.resize(len(a) + 1);
121     for (int i = len(a); i; --i) res[i] = res[i - 1] * inv[i];
122     res[0] = 0; restore(res); return res;
123 }
124 Poly Inv(const Poly &a) {
125     Poly res(1, Int(pow_mod(a[0].get(), p - 2, p)));
126     int n = 1; while (n < len(a)) n <<= 1;
127     for (int k = 2; k <= n; k <<= 1) {
128         Poly t(a); t.resize(k);
129         t = t * res; t.resize(k);
130         res = res * (2 - t); res.resize(k);
131     } res.resize(len(a)); restore(res); return res;
132 }
133 Poly Ln(const Poly &a) {
134     Poly res = Inte(Der(a) * Inv(a));
135     res.resize(len(a)); restore(res); return res;
136 }
137 Poly Exp(const Poly &a) {
138     Poly res(1, 1);
139     int n = 1; while (n < len(a)) n <<= 1;
140     for (int k = 2; k <= n; k <<= 1) {
141         Poly t(res.begin(), res.end()); t.resize(k); t = Ln(t);
142         for (int i = 0; i < min(len(a), k); ++i) t[i] = a[i] + (p - t[i]); t
           [0] = t[0] + 1; restore(t);
143         res = res * t; res.resize(k);
144     } res.resize(len(a)); restore(res); return res;
145 }
146 } // namespace Pol
147 int main() {
148     int n, m; cin >> n >> m >> p;
149     Poly A(n+1);
150     for (int i = 0; i <= n; i++) {
151         int x; cin>>x; Int tmp(x); A[i]=tmp;
152     }
153     Poly B(m+1);
154     for (int i = 0; i <= m; i++) {
155         int x; cin>>x; Int tmp(x); B[i]=tmp;
156     }

```

```

157     Poly C = Pol::operator*(A,B);
158     for (int i = 0; i <= n+m; i++) cout << C[i].get() << " ";
159 }

```

2.2.5 任意模数 NTT

```

1  namespace Pol {
2      int add(int a, int b) { return (a += b) >= p ? a -= p : a; }
3      int mul(int a, int b) { return 1ll * a * b % p; }
4      Poly operator - (const int &v, const Poly &a) {
5          Poly res(a);
6          for (int i = 0; i < len(res); ++i) res[i] = p - res[i];
7          res[0] = add(res[0], v); return res;
8      }
9      Poly operator - (const Poly &a, const int &v) {
10         Poly res(a); res[0] = add(res[0], p - v); return res;
11     }
12     Poly operator * (const Poly &a, const int &v) {
13         Poly res(a);
14         for (int i = 0; i < len(res); ++i) res[i] = mul(res[i], v);
15         return res;
16     }
17
18     const int N = 4200000;
19     const double pi = acos(-1);
20     struct Complex {
21         double x, y;
22         Complex(double x = 0, double y = 0) : x(x), y(y) {}
23         friend Complex operator + (const Complex &u, const Complex &v) { return
24             Complex(u.x + v.x, u.y + v.y); }
25         friend Complex operator - (const Complex &u, const Complex &v) { return
26             Complex(u.x - v.x, u.y - v.y); }
27         friend Complex operator * (const Complex &u, const Complex &v) {
28             return Complex(u.x * v.x - u.y * v.y, u.x * v.y + u.y * v.x);
29         }
30     }; typedef vector<Complex> Vcp;
31     int P[N];
32     void init_P(int n) {
33         int l = 0; while ((1 << l) < n) ++l;
34         for (int i = 0; i < n; ++i) P[i] = (P[i >> 1] >> 1) | ((i & 1) << l - 1);
35     }

```

```

34     vector<Complex> init_W(int n) {
35         vector<Complex> w(n); w[1] = 1;
36         for (int i = 2; i < n; i <= 1) {
37             auto w0 = w.begin() + i / 2, w1 = w.begin() + i;
38             Complex wn(cos(pi / i), sin(pi / i));
39             for (int j = 0; j < i; j += 2)
40                 w1[j] = w0[j >> 1], w1[j + 1] = w1[j] * wn;
41         }
42         return w;
43     } auto w = init_W(1 << 21);
44     void DIT(Vcp& a) {
45         int n = len(a);
46         for (int k = n >> 1; k; k >= 1)
47             for (int i = 0; i < n; i += k << 1)
48                 for (int j = 0; j < k; ++j) {
49                     Complex x = a[i + j], y = a[i + j + k];
50                     a[i + j + k] = (x - y) * w[k + j], a[i + j] = x + y;
51                 }
52     }
53     void DIF(Vcp &a) {
54         int n = len(a);
55         for (int k = 1; k < n; k <= 1)
56             for (int i = 0; i < n; i += k << 1)
57                 for (int j = 0; j < k; ++j) {
58                     Complex x = a[i + j], y = a[i + j + k] * w[k + j];
59                     a[i + j + k] = x - y, a[i + j] = x + y;
60                 }
61         const double inv = 1. / n;
62         for (int i = 0; i < n; ++i) a[i].x *= inv, a[i].y *= inv;
63         reverse(a.begin() + 1, a.end());
64     }
65     Poly operator *(const Poly &A, const Poly &B) {
66         int n = 1, n1 = len(A), n2 = len(B); while (n < n1 + n2 - 1) n <= 1;
67         init_P(n);
68         Vcp a(n), b(n), c0(n), c1(n);
69         for (int i = 0; i < n1; ++i) a[i] = Complex(A[i] & 0x7fff, A[i] >> 15);
70         for (int i = 0; i < n2; ++i) b[i] = Complex(B[i] & 0x7fff, B[i] >> 15);
71         DIT(a), DIT(b);
72         for (int k = 1, i = 0, j; k < n; k <= 1)
73             for (; i < k * 2; ++i) {
74                 j = i ^ k - 1;

```

```

74         c0[i] = Complex(a[i].x + a[j].x, a[i].y - a[j].y) * b[i] * 0.5;
75         c1[i] = Complex(a[i].y + a[j].y, -a[i].x + a[j].x) * b[i] * 0.5;
76     }
77     DIF(c0), DIF(c1); Poly ans(n1 + n2 - 1);
78     for (int i = 0; i < n1 + n2 - 1; i++) {
79         ll c00 = c0[i].x + 0.5, c01 = c0[i].y + 0.5, c10 = c1[i].x + 0.5, c11
            = c1[i].y + 0.5;
80         ans[i] = (c00 + ((c01 + c10) % p << 15) + (c11 % p << 30)) % p;
81     } return ans;
82 }
83 Poly Der(const Poly &a) {
84     Poly res(a);
85     for (int i = 0; i < len(a) - 1; ++i) res[i] = mul(i + 1, res[i + 1]);
86     res[len(a) - 1] = 0; return res;
87 }
88 Poly Int(const Poly &a) {
89     static int inv[N];
90     if (!inv[1]) {
91         inv[1] = 1;
92         for (int i = 2; i < N; ++i) inv[i] = mul(p - p / i, inv[p % i]);
93     }
94     Poly res(a); res.resize(len(a) + 1);
95     for (int i = len(a); i --; res[i] = mul(res[i - 1], inv[i]);
96     res[0] = 0; return res;
97 }
98 Poly Inv(const Poly &a) {
99     Poly res(1, pow_mod(a[0], p - 2));
100    int n = 1; while (n < len(a)) n <<= 1;
101    for (int k = 2; k <= n; k <<= 1) {
102        int L = 2 * k; init_P(L); Poly t(k);
103        copy_n(a.begin(), min(k, len(a)), t.begin());
104        t = 2 - t * res; res = res * t; res.resize(k);
105    } res.resize(len(a)); return res;
106 }
107 Poly Ln(const Poly &a) {
108     Poly res = Int(Der(a) * Inv(a));
109     res.resize(len(a)); return res;
110 }
111 Poly Exp(const Poly &a) {
112     Poly res(1, 1);
113     int n = 1; while (n < len(a)) n <<= 1;

```

```

114     for (int k = 2; k <= n; k <= 1) {
115         Poly t(res.begin(), res.end()); t.resize(k); t = Ln(t);
116         for (int i = 0; i < min(len(a), k); ++i) t[i] = add(a[i], p - t[i]);
117         t[0] = add(t[0], 1);
118         res = res * t; res.resize(k);
119     } res.resize(len(a)); return res;
120 } // namespace Pol
    
```

2.3 容斥定理

设有 n 个不同性质, $A_i =$ 具有性质 i 的元素集合, $|A_i| = A_i$ 集合中元素个数
容斥定理用于求 $|A_1 \cup A_2 \cup \dots \cup A_n|$ 和 $|\overline{A_1} \cap \overline{A_2} \cap \dots \cap \overline{A_n}|$

$$\begin{aligned}
 \left| \bigcup_{i=1}^n A_i \right| &= \sum_{m=1}^n (-1)^{m+1} \sum_{1 \leq i_1 < \dots < i_m \leq n} \left| \bigcap_{j=1}^m A_{i_j} \right| \\
 \left| \bigcap_{i=1}^n \overline{A_i} \right| &= |U| - \left| \bigcup_{i=1}^n A_i \right|
 \end{aligned}$$

2.4 广义容斥定理

设有 n 种不同的性质

$P_k =$ 至少有 k 个性质的元素个数, $Q_k =$ 恰好有 k 个性质的元素个数

$$Q_k = \sum_{i=0}^{n-k} (-1)^i C_{k+i}^k P_{k+i}$$

特别地, 不具有所有性质的元素个数 $Q_0 = P_0 - P_1 + P_2 - P_3 + \dots$

计数时不一定要计算所有状态, 只需要递推至少拥有奇数个/偶数个性质的个数

二项式反演:

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

2.5 Min-Max 容斥

$$\max(S) = \sum_{T \subseteq S} (-1)^{|T|-1} \min(T)$$

$$\min(S) = \sum_{T \subseteq S} (-1)^{|T|} \max(T)$$

$$\text{第 } k \text{ 大 } k\max(S) = \sum_{T \subseteq S} (-1)^{|T|-k} C_{|T|-1}^{k-1} \min(T)$$

$$\text{第 } k \text{ 小 } k\min(S) = \sum_{T \subseteq S} (-1)^{|T|-k} C_{|T|-1}^{k-1} \max(T)$$

2.6 高斯消元

实数

```

1  int n,m,r;double a[N][M];
2  void gauss(double a[N][M],int n,int m){
3      r=0;//矩阵的秩
4      int lim=min(n,m);
5      for(int i=1;i<=lim;++i){//当前到第几列
6          bool f=0;
7          for(int j=r+1;j<=n;++j){
8              if(abs(a[j][i])>1e-8){
9                  swap(a[r+1],a[j]);
10                 f=1;break;
11             }
12         }
13         if(!f)continue;
14         ++r;double t=a[r][i];
15         for(int j=i;j<=m;++j)a[r][j]/=t;
16         for(int j=1;j<=n;++j){
17             if(j==r)continue;
18             t=a[j][i];
19             for(int k=i;k<=m;++k)a[j][k]-=t*a[r][k];
20         }
21     }
22 }
```

整数取模

```

1  ll a[N][M];const int mod=1e9+7;//先写好快速幂qpow
2  void gauss(ll a[N][M],int n,int m){//注意化简后有负数未取模为正
3      r=0;//矩阵的秩
4      int lim=min(n,m);
5      for(int i=1;i<=lim;++i){
6          bool f=0;
7          for(int j=r+1;j<=n;++j){
8              if(a[j][i]!=0){
9                  swap(a[r+1],a[j]);
10                 f=1;break;
11             }
12         }
13         if(!f)continue;
14         ++r;ll t=qpow(a[r][i],mod-2);
15         for(int j=i;j<=m;++j)a[r][j]=a[r][j]*t%mod;
```



```

16         for(int j=1;j<=n;++j){
17             if(j==r)continue;
18             t=a[j][i];
19             for(int k=i;k<=m;++k)a[j][k]=(a[j][k]-t*a[r][k])%mod;
20         }
21     }
22 }

```

异或 bitset

```

1  int n,m,r,a[N][M];
2  bitset<M>g[N];//转为int可以g[i][j]?1:0或int(g[i][j])
3  void gauss(bitset<M>a[N],int n,int m){
4      r=0;int lim=min(n,m);
5      for(int i=1;i<=lim;++i){
6          bool f=0;
7          for(int j=r;j<n;++j){
8              if(a[j][i]){
9                  swap(a[r+1],a[j]);f=1;break;
10             }
11         }
12         if(!f)continue;
13         ++r;
14         for(int j=1;j<=n;++j){
15             if(j==r)continue;
16             if(a[j][i])a[j]^=a[r];
17         }
18     }
19 }

```

行列式

```

1  ll det(ll a[N][N],int n){
2      ll s=1;
3      for(int i=1;i<=n;++i){
4          for(int j=i;j<=n;++j){
5              if(a[j][i]){
6                  if(i!=j){
7                      swap(a[j],a[i]);s=-s;
8                  }
9                  break;
10             }
11         }
12     }
13 }

```

```

12     if(!a[i][i])return 0;
13     s=s*a[i][i]%mod;
14     ll inv=qpow(a[i][i],mod-2);
15     for(int j=i;j<=n;++j)a[i][j]=a[i][j]*inv%mod;
16     for(int j=i+1;j<=n;++j){
17         ll t=a[j][i];
18         for(int k=i;k<=n;++k)a[j][k]-=t*a[i][k],a[j][k]%mod;
19     }
20 }
21 return s;
22 }

```

辗转相减行列式 (模数不是质数时)

```

1 ll det(ll a[N][N],int n){
2     ll ans=1;
3     for(int i=1;i<=n;++i){
4         for(int j=i+1;j<=n;++j){
5             while(a[j][i]){
6                 ll t=a[i][i]/a[j][i];
7                 for(int k=i;k<=n;++k){
8                     a[i][k]-=t*a[j][k];
9                     a[i][k]%mod;
10                }
11                swap(a[i],a[j]);ans=-ans;
12            }
13        }
14        ans=ans*a[i][i]%mod;
15    }
16    return ans;
17 }

```

2.7 线性基

```

1 for(int i=1;i<=n;++i){//n个数的线性基
2     ll x;scanf("%lld",&x);
3     for(int j=62;j>=0;--j){
4         if(x>>j&1){
5             if(g[j])x^=g[j];
6             else{
7                 g[j]=x;
8                 break;

```

```

9         }
10    }
11 }
12 }
13 ll ans=0;
14 for(int i=62;i>=0;--i){//选几个数的最大异或
15     if(ans>>i&1)continue;
16     ans^=g[i];
17 }
    
```

2.8 矩阵快速幂

$$\begin{cases} f_1 = 1, f_2 = 1 \\ f_n = f_{n-1} + f_{n-2} \end{cases} \Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f_{n-1} \\ f_{n-2} \end{bmatrix} = \begin{bmatrix} f_n \\ f_{n-1} \end{bmatrix} \Rightarrow \begin{bmatrix} f_n \\ f_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2} \begin{bmatrix} f_2 \\ f_1 \end{bmatrix}$$

$$\begin{cases} a_1 = 7, a_2 = 12 \\ a_n = 2a_{n-1} - 3a_{n-2} + n^2, n \geq 2 \end{cases} \Rightarrow \begin{bmatrix} 2 & -3 & 1 & 2 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ (n-1)^2 \\ n-1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_n \\ a_{n-1} \\ n^2 \\ n \\ 1 \end{bmatrix}$$

```

1  const int mod=998244353;
2  const int n1=4;//有时需要是变量
3  struct matrix{
4      ll a[n1][n1];//此处过大可能导致本地RE
5      inline void clear(){
6          for(int i=0;i<n1;++i)memset(a[i],0,n1<<3);
7      }
8      inline void I(){
9          clear();for(int i=0;i<n1;++i)a[i][i]=1;
10     }
11     inline ll* operator [] (int i) {
12         return a[i];
13     }
14     matrix(const initializer_list<ll>&l={}){
15         int now=0;
16         for(auto x:l){
17             a[now/n1][now%n1]=x;
18             ++now;
19         }
    
```

```

20     }
21     matrix operator + (const matrix &b){
22         static matrix c;
23         for(int i=0;i<n1;++i){
24             for(int j=0;j<n1;++j){
25                 c[i][j]=(a[i][j]+b.a[i][j])%mod;
26             }
27         }
28         return c;
29     }
30     matrix operator * (const matrix &b) {
31         static matrix c;c.clear();
32         for(int i=0;i<n1;++i){
33             for(int j=0;j<n1;++j){
34                 for(int k=0;k<n1;++k){
35                     c[i][j]+=a[i][k]*b.a[k][j];
36                     if(k&3)c[i][j]%=mod;
37                 }
38                 c[i][j]%=mod;
39             }
40         }
41         return c;
42     }
43 };
44 matrix qpow(matrix a,ll b){
45     static matrix s;s.I();
46     for(;b;b>>=1){
47         if(b&1)s=s*a;
48         a=a*a;
49     }
50     return s;
51 }
52 matrix powsum(matrix a,ll b){//幂和:  $A^0+A+A^2+A^3+\dots+A^b$ 
53     matrix I;I.I();
54     if(!b)return I;
55     if(b&1)return I+a*powsum(a,b-1);
56     return I+(I+qpow(a,b/2))*a*powsum(a,b/2-1);
57 }
58 int main(){
59     matrix A({//需要定义列向量, 可以使用矩阵的第一列
60         1,1,1,2,

```

```

61         1,2,1,2,
62         3,4,0,3,
63         2,4,1,4
64     });
65 }
```

2.9 康托展开

排列 $P = \{p_1, p_2, \dots, p_n\}$ 是第 $f(P)$ 小的排列 (12345 是第 0 小)

$$f(P) = \sum_{i=1}^n (n-i)! \times r(i) \quad r(i) = p_i \text{ 右边比 } p_i \text{ 小的个数}$$

逆康托展开: 已知排列的排名 $rank$ (从 0 开始), 求出该排列

```

1  for(int i=1;i<=n;++i){
2      int t=rank/fac[n-i];
3      //p[i]右边比p[i]小的有t个,p[i]应该是剩余数字第t小
4      rank%=fac[n-i];int cnt=0;
5      for(int j=1;j<=n;++j){
6          if(!vis[j])++cnt;
7          if(cnt==t+1){
8              vis[j]=1;p[i]=j;break;
9          }
10     }
11 }
```

2.10 卡特兰数

n 个数的出栈序列有多少种

n 个结点可构造多少个不同的二叉树

$$\text{卡特兰数 } H_n = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1} = \frac{H_{n-1}(4n-2)}{n+1}$$

2.11 斯特林数

一、第一类斯特林数

第一类 Stirling 数 $S_1(p, k)$ 的组合意义是: 将 p 个物体排成 k 个非空循环排列的方案数

递推公式: $S_1(p, k) = (p-1)S_1(p-1, k) + S_1(p-1, k-1)$, $1 \leq k \leq p-1$

边界条件: $p \geq 1$ 时 $S_1(p, 0) = 0$, $p \geq 0$ 时 $S_1(p, p) = 1$

二、第二类斯特林数

第二类 Stirling 数 $S_2(p, k)$ 的组合意义: 将 p 个物体划分成 k 个非空的无编号集合的方案数

递推公式: $S_2(p, k) = kS_2(p-1, k) + S_2(p-1, k-1)$, $1 \leq k \leq p-1$

边界条件: $p \geq 1$ 时 $S_2(p, 0) = 0$, $p \geq 0$ 时 $S_2(p, p) = 1$

卷积形式: $S_2(n, m) = \frac{1}{m!} \sum_{k=0}^m (-1)^k C_m^k (m-k)^n = \sum_{k=0}^m \frac{(-1)^k (m-k)^n}{k!(m-k)!} = \sum_{k=0}^m \frac{(-1)^k}{k!} \times \frac{(m-k)^n}{(m-k)!}$

2.12 小球放入盒子模型

n 个球	m 个盒子	是否允许有空盒子	方案数
不同	不同	是	m^k
不同	不同	否	$m!S_2(n, m)$
不同	相同	是	$\sum_{i=1}^m S_2(n, i)$
不同	相同	否	$S_2(n, m)$
相同	不同	是	C_{n+m-1}^{m-1}
相同	不同	否	C_{n-1}^{m-1}
相同	相同	是	$\frac{1}{(1-x)(1-x^2)\dots(1-x^m)}$ 的 x^n 项系数
相同	相同	否	$\frac{x^m}{(1-x)(1-x^2)\dots(1-x^m)}$ 的 x^n 项系数

2.13 快速沃尔什变换 FWT

```

1  using ll = long long;
2  const int N=1<<17,mod=998244353;
3  int a[N], b[N], c[N];
4  void fwt_or(int *arr, int n, int flag) {
5      for (int i = 1; i << 1 <= n; i <= 1)
6          for (int j = 0; j < n; j += i << 1)
7              for (int k = 0; k < i; ++k)
8                  arr[i + j + k] = (arr[i + j + k] + (ll) arr[j + k] * flag) % mod;
9  }
10 void fwt_and(int *arr, int n, int flag) {
11     for (int i = 1; i << 1 <= n; i <= 1)
12         for (int j = 0; j < n; j += i << 1)
13             for (int k = 0; k < i; ++k)
14                 arr[j + k] = (arr[j + k] + (ll) arr[i + j + k] * flag) % mod;
15 }
16 void fwt_xor(int *arr, int n, int flag) {
17     for (int i = 1; i << 1 <= n; i <= 1)
18         for (int j = 0; j < n; j += i << 1)
19             for (int k = 0; k < i; ++k) {
20                 arr[j + k] += arr[i + j + k];
21                 arr[i + j + k] = arr[j + k] - arr[i + j + k] - arr[i + j + k];
22                 arr[j + k] = ((ll) arr[j + k] * flag % mod + mod) % mod;
23                 arr[i + j + k] = ((ll) arr[i + j + k] * flag % mod + mod) % mod;
            }
    }
    
```

```

24         }
25     }
    
```

2.14 克莱姆法则

$$\text{解方程组} \begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases}$$

$$\text{令 } D = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}, \text{ 令 } D_i = D \text{ 的第 } i \text{ 列变为 } b_1 \sim b_n, \text{ 则 } x_i = \frac{D_i}{D}$$

```

1 //a11*x + a12*y = b1,  a21*x + a22*y = b2, 返回pair(x,y)
2 pair<db,db>solve(db a11,db a12,db b1,db a21,db a22,db b2){
3     db z=(a11*a22-a12*a21);
4     db x=(b1*a22-a12*b2)/z;
5     db y=(b2*a11-b1*a21)/z;
6     return {x,y};
7 }
    
```

2.15 自适应辛普森积分

```

1 double simpson(double l,double r){
2     return (f(l)+f(r)+4*f((l+r)/2.0))*(r-l)/6.0;
3 }
4 double rsimpson(double l,double r){
5     double mid=(l+r)/2.0;
6     if(abs(simpson(l,r)-simpson(l,mid)-simpson(mid,r))<eps)
7         return simpson(l,mid)+simpson(mid,r);
8     return rsimpson(l,mid)+rsimpson(mid,r);
9 }
    
```

2.16 曼哈顿距离

$$|x_1 - x_2| + |y_1 - y_2| = \max(|(x_1 + y_1) - (x_2 + y_2)|, |(x_1 - y_1) - (x_2 - y_2)|)$$

2.17 高次方求和

```

1    ll fac[N],pl[N],pr[N];
2    ll calc(ll n,int k){//1^k+2^k+3^k+...+n^k O(k)
3        ll y=0,ans=0;
4        pl[0]=pr[k+3]=fac[0]=1;
5        for(int i=1;i<=k+2;++i){
6            fac[i]=fac[i-1]*i%mod;
7            pl[i]=(n-i)%mod*pl[i-1]%mod;
8        }
9        for(int i=k+2;i--i)pr[i]=(n-i)%mod*pr[i+1]%mod;
10       for(int i=1;i<=k+2;++i){
11           y+=qpow(i,k); y%=mod;
12           ll a=pl[i-1]*pr[i+1]%mod;
13           ll b=(k-i&1?-1:1)*fac[i-1]*fac[k+2-i]%mod;
14           ans+=y*a%mod*qpow(b,mod-2); ans%=mod;
15       }
16       return ans;
17   }
```

2.18 常用求和, 组合数公式

$$C_n^m = C_{n-1}^{m-1} + C_{n-1}^m \quad C_n^m = \frac{n!}{m!(n-m)!}$$

隔板法: x 个相同的球放入 y 个不同盒子里, 要求每个盒子至少一个:

相当于在 $x-1$ 个间隔里插入 $y-1$ 个隔板, C_{x-1}^{y-1}

$\sum_{i=0}^m C_n^i C_m^i = C_{n+m}^m$ ($n \geq m$) (所有从这 $n+m$ 个里面选 m 个的情况)

$$C_n^k + C_{n+1}^k + \dots + C_{n+m}^k = [C_{n+1}^{k+1} - C_n^{k+1}] + [C_{n+2}^{k+1} - C_{n+1}^{k+1}] + \dots + [C_{n+m+1}^{k+1} - C_{n+m}^{k+1}]$$

$$= C_{n+m+1}^{k+1} - C_n^{k+1} \quad (\text{注意 } C_n^{n+1} = 0)$$

高次方求和公式

$$1 \text{ 次方求和公式: } \sum_{i=1}^n i = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$$

$$2 \text{ 次方求和公式: } \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$$

$$3 \text{ 次方求和公式: } \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4} = \frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2$$

$$4 \text{ 次方求和公式: } \sum_{i=1}^n i^4 = \frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n$$

$$5 \text{ 次方求和公式: } \sum_{i=1}^n i^5 = \frac{1}{6}n^6 + \frac{1}{2}n^5 + \frac{5}{12}n^4 - \frac{1}{12}n^2$$

$$6 \text{ 次方求和公式: } \sum_{i=1}^n i^6 = \frac{1}{7}n^7 + \frac{1}{2}n^6 + \frac{1}{2}n^5 - \frac{1}{6}n^3 + \frac{1}{42}n$$

$$7 \text{ 次方求和公式: } \sum_{i=1}^n i^7 = \frac{1}{8}n^8 + \frac{1}{2}n^7 + \frac{7}{12}n^6 - \frac{7}{24}n^4 + \frac{1}{12}n^2$$

$$8 \text{ 次方求和公式: } \sum_{i=1}^n i^8 = \frac{1}{9}n^9 + \frac{1}{2}n^8 + \frac{2}{3}n^7 - \frac{7}{15}n^5 + \frac{2}{9}n^3 - \frac{1}{30}n$$

$$9 \text{ 次方求和公式: } \sum_{i=1}^n i^9 = \frac{1}{10}n^{10} + \frac{1}{2}n^9 + \frac{3}{4}n^8 - \frac{7}{10}n^6 + \frac{1}{2}n^4 - \frac{3}{20}n^2$$

$$10 \text{ 次方求和公式: } \sum_{i=1}^n i^{10} = \frac{1}{11}n^{11} + \frac{1}{2}n^{10} + \frac{5}{6}n^9 - n^7 + n^5 - \frac{1}{2}n^3 + \frac{5}{66}n$$

$$11 \text{ 次方求和公式: } \sum_{i=1}^n i^{11} = \frac{1}{12}n^{12} + \frac{1}{2}n^{11} + \frac{11}{12}n^{10} - \frac{11}{8}n^8 + \frac{11}{6}n^6 - \frac{11}{8}n^4 + \frac{5}{12}n^2$$

$$12 \text{ 次方求和公式: } \sum_{i=1}^n i^{12} = \frac{1}{13}n^{13} + \frac{1}{2}n^{12} + n^{11} - \frac{11}{6}n^9 + \frac{22}{7}n^7 - \frac{33}{10}n^5 + \frac{5}{3}n^3 - \frac{691}{2730}n$$

$$13 \text{ 次方求和公式: } \sum_{i=1}^n i^{13} = \frac{1}{14}n^{14} + \frac{1}{2}n^{13} + \frac{13}{12}n^{12} - \frac{143}{60}n^{10} + \frac{143}{28}n^8 - \frac{143}{20}n^6 + \frac{65}{12}n^4 - \frac{691}{420}n^2$$

$$14 \text{ 次方求和公式: } \sum_{i=1}^n i^{14} = \frac{1}{15}n^{15} + \frac{1}{2}n^{14} + \frac{7}{6}n^{13} - \frac{91}{30}n^{11} + \frac{143}{18}n^9 - \frac{143}{10}n^7 + \frac{91}{6}n^5 - \frac{691}{90}n^3 + \frac{7}{6}n$$

$$15 \text{ 次方求和公式: } \sum_{i=1}^n i^{15} = \frac{1}{16}n^{16} + \frac{1}{2}n^{15} + \frac{5}{4}n^{14} - \frac{91}{24}n^{12} + \frac{143}{12}n^{10} - \frac{429}{16}n^8 + \frac{455}{12}n^6 - \frac{691}{24}n^4 + \frac{35}{4}n^2$$

$$16 \text{ 次方求和公式: } \sum_{i=1}^n i^{16} = \frac{1}{17}n^{17} + \frac{1}{2}n^{16} + \frac{4}{3}n^{15} - \frac{14}{3}n^{13} + \frac{52}{3}n^{11} - \frac{143}{3}n^9 + \frac{260}{3}n^7 - \frac{1382}{15}n^5 + \frac{140}{3}n^3 - \frac{3617}{510}n$$

$$17 \text{ 次方求和公式: } \sum_{i=1}^n i^{17} = \frac{1}{18}n^{18} + \frac{1}{2}n^{17} + \frac{17}{12}n^{16} - \frac{17}{3}n^{14} + \frac{221}{9}n^{12} - \frac{2431}{30}n^{10} + \frac{1105}{6}n^8 - \frac{11747}{45}n^6 + \frac{595}{3}n^4 - \frac{3617}{60}n^2$$

$$18 \text{ 次方求和公式: } \sum_{i=1}^n i^{18} = \frac{1}{19}n^{19} + \frac{1}{2}n^{18} + \frac{3}{2}n^{17} - \frac{34}{5}n^{15} + 34n^{13} - \frac{663}{5}n^{11} + \frac{1105}{3}n^9 - \frac{23494}{35}n^7 + 714n^5 - \frac{3617}{10}n^3 + \frac{43867}{798}n$$

$$19 \text{ 次方求和公式: } \sum_{i=1}^n i^{19} = \frac{1}{20}n^{20} + \frac{1}{2}n^{19} + \frac{19}{12}n^{18} - \frac{323}{40}n^{16} + \frac{323}{7}n^{14} - \frac{4199}{20}n^{12} + \frac{4199}{6}n^{10} - \frac{223193}{140}n^8 + 2261n^6 - \frac{68723}{40}n^4 + \frac{43867}{84}n^2$$

$$20 \text{ 次方求和公式: } \sum_{i=1}^n i^{20} = \frac{1}{21}n^{21} + \frac{1}{2}n^{20} + \frac{5}{3}n^{19} - \frac{19}{2}n^{17} + \frac{1292}{21}n^{15} - 323n^{13} + \frac{41990}{33}n^{11} - \frac{223193}{63}n^9 + 6460n^7 - \frac{68723}{10}n^5 + \frac{219335}{63}n^3 - \frac{174611}{330}n$$

$$21 \text{ 次方求和公式: } \sum_{i=1}^n i^{21} = \frac{1}{22}n^{22} + \frac{1}{2}n^{21} + \frac{7}{4}n^{20} - \frac{133}{12}n^{18} + \frac{323}{4}n^{16} - \frac{969}{2}n^{14} + \frac{146965}{66}n^{12} - \frac{223193}{30}n^{10} + \frac{33915}{2}n^8 - \frac{481061}{20}n^6 + \frac{219335}{12}n^4 - \frac{1222277}{220}n^2$$

$$22 \text{ 次方求和公式: } \sum_{i=1}^n i^{22} = \frac{1}{23}n^{23} + \frac{1}{2}n^{22} + \frac{11}{6}n^{21} - \frac{77}{6}n^{19} + \frac{209}{2}n^{17} - \frac{3553}{5}n^{15} + \frac{11305}{3}n^{13} - \frac{223193}{15}n^{11} + \frac{124355}{3}n^9 - \frac{755953}{10}n^7 + \frac{482537}{6}n^5 - \frac{1222277}{30}n^3 + \frac{854513}{138}n$$

$$23 \text{ 次方求和公式: } \sum_{i=1}^n i^{23} = \frac{1}{24}n^{24} + \frac{1}{2}n^{23} + \frac{23}{12}n^{22} - \frac{1771}{120}n^{20} + \frac{4807}{36}n^{18} - \frac{81719}{80}n^{16} + \frac{37145}{6}n^{14} - \frac{5133439}{180}n^{12} + \frac{572033}{6}n^{10} - \frac{17386919}{80}n^8 + \frac{11098351}{36}n^6 - \frac{28112371}{120}n^4 + \frac{854513}{12}n^2$$

$$24 \text{ 次方求和公式: } \sum_{i=1}^n i^{24} = \frac{1}{25}n^{25} + \frac{1}{2}n^{24} + 2n^{23} - \frac{253}{15}n^{21} + \frac{506}{3}n^{19} - \frac{14421}{10}n^{17} + \frac{29716}{3}n^{15} - \frac{10266878}{195}n^{13} + 208012n^{11} - \frac{17386919}{30}n^9 + \frac{22196702}{21}n^7 - \frac{28112371}{25}n^5 + \frac{1709026}{3}n^3 - \frac{236364091}{2730}n$$

$$25 \text{ 次方求和公式: } \sum_{i=1}^n i^{25} = \frac{1}{26}n^{26} + \frac{1}{2}n^{25} + \frac{25}{12}n^{24} - \frac{115}{6}n^{22} + \frac{1265}{6}n^{20} - \frac{24035}{12}n^{18} + \frac{185725}{12}n^{16} - \frac{25667195}{273}n^{14} + \frac{1300075}{3}n^{12} - \frac{17386919}{12}n^{10} + \frac{277458775}{84}n^8 - \frac{28112371}{6}n^6 + \frac{21362825}{6}n^4 - \frac{1181820455}{1092}n^2$$

$$26 \text{ 次方求和公式: } \sum_{i=1}^n i^{26} = \frac{1}{27}n^{27} + \frac{1}{2}n^{26} + \frac{13}{6}n^{25} - \frac{65}{3}n^{23} + \frac{16445}{63}n^{21} - \frac{16445}{6}n^{19} + \frac{142025}{6}n^{17} -$$

$$\begin{aligned}
 & \frac{10266878}{63}n^{15} + \frac{2600150}{3}n^{13} - \frac{20548177}{6}n^{11} + \frac{3606964075}{378}n^9 - \frac{52208689}{3}n^7 + \frac{55543345}{3}n^5 - \frac{1181820455}{126}n^3 + \frac{8553103}{6}n \\
 & \quad 27 \text{ 次方求和公式: } \sum_{i=1}^n i^{27} = \frac{1}{28}n^{28} + \frac{1}{2}n^{27} + \frac{9}{4}n^{26} - \frac{195}{8}n^{24} + \frac{4485}{14}n^{22} - \frac{29601}{8}n^{20} + \frac{142025}{4}n^{18} - \\
 & \quad \frac{15400317}{56}n^{16} + 1671525n^{14} - \frac{61644531}{8}n^{12} + \frac{721392815}{28}n^{10} - \frac{469878201}{8}n^8 + \frac{166630035}{2}n^6 - \frac{3545461365}{56}n^4 + \frac{76977927}{4}n^2 \\
 & \quad 28 \text{ 次方求和公式: } \sum_{i=1}^n i^{28} = \frac{1}{29}n^{29} + \frac{1}{2}n^{28} + \frac{7}{3}n^{27} - \frac{273}{10}n^{25} + 390n^{23} - \frac{9867}{2}n^{21} + 52325n^{19} - \frac{905901}{2}n^{17} + \\
 & \quad 3120180n^{15} - \frac{33193209}{2}n^{13} + 65581165n^{11} - \frac{365460823}{2}n^9 + 333260070n^7 - \frac{709092273}{2}n^5 + 179615163n^3 - \\
 & \quad \frac{23749461029}{870}n \\
 & \quad 29 \text{ 次方求和公式: } \sum_{i=1}^n i^{29} = \frac{1}{30}n^{30} + \frac{1}{2}n^{29} + \frac{29}{12}n^{28} - \frac{609}{20}n^{26} + \frac{1885}{4}n^{24} - \frac{26013}{4}n^{22} + \frac{303485}{4}n^{20} - \frac{8757043}{12}n^{18} + \\
 & \quad \frac{22621305}{4}n^{16} - \frac{137514723}{4}n^{14} + \frac{1901853785}{12}n^{12} - \frac{10598363867}{20}n^{10} + \frac{4832271015}{4}n^8 - \frac{6854558639}{4}n^6 + \frac{5208839727}{4}n^4 - \\
 & \quad \frac{23749461029}{60}n^2 \\
 & \quad 30 \text{ 次方求和公式: } \sum_{i=1}^n i^{30} = \frac{1}{31}n^{31} + \frac{1}{2}n^{30} + \frac{5}{2}n^{29} - \frac{203}{6}n^{27} + \frac{1131}{2}n^{25} - \frac{16965}{2}n^{23} + \frac{216775}{2}n^{21} - \frac{2304485}{2}n^{19} + \\
 & \quad \frac{19959975}{2}n^{17} - \frac{137514723}{2}n^{15} + \frac{731482225}{2}n^{13} - \frac{31795091601}{22}n^{11} + \frac{8053785025}{2}n^9 - \frac{102818379585}{14}n^7 + \frac{15626519181}{2}n^5 - \\
 & \quad \frac{23749461029}{6}n^3 + \frac{8615841276005}{14322}n
 \end{aligned}$$

3 数据结构

3.1 树状数组

```

1 ll tr[N]; // tr[i] = [i-lowbit(i)+1,i] 区间的信息
2 void add(int x, ll k){for(;x<=n;x+=x&-x)tr[x]+=k;} // 需要保证x!=0
3 ll qry(int x){ll s=0;for(;x;x-=x&-x)s+=tr[x];return s;}
    
```

树状数组维护二阶前缀和

要实现区间加区间求和，需要维护差分数组的二阶前缀和

设 d, a, s 分别为差分数组，原数组，前缀和数组

$$s[n] = \sum_{i=1}^n a[i] = \sum_{i=1}^n (n-i+1)d[i]$$

接下来分别维护 $d[i]$ 和 $i \times d[i]$ 的前缀和即可

```

1 ll t1[N], t2[N];
2 void add(int x, ll k){
3     for(int i=x; i<=n; i+=i&-i){
4         t1[i]+=k; t2[i]+=x*k;
5     }
6 }
7 ll qry(int x){
8     ll s=0;
9     for(int i=x; i; i-=i&-i)s+=(x+1)*t1[i]-t2[i];
10    return s;
11 }
    
```

树状数组上二分

```

1  int bs(ll x){//最大的sum(i)>=x的下标i,若求最小的i,需要bs(x-1)+1
2      int pos=0;ll sum=0;
3      for(int i=20;i>=0;--i){
4          if(pos+(1<<i)<=n&&sum+t[pos+(1<<i)]<=x){
5              pos+=1<<i;
6              sum+=t[pos];
7          }
8      }
9      return pos;
10 }

```

3.2 线段树

3.2.1 懒惰标记线段树

```

1  int n,a[N];
2  struct sgt{
3      int l,r;
4      ll v,tag;
5  }t[N<<2];
6  #define ls (p<<1)
7  #define rs (p<<1|1)
8  void pushup(int p){
9      t[p].v=t[ls].v+t[rs].v;
10 }
11 void up1(int p,ll x){//结点打上标记
12     t[p].tag+=x;
13     t[p].v+=x*(t[p].r-t[p].l+1);
14 }
15 void pushdown(int p){
16     if(t[p].tag){
17         up1(ls,t[p].tag);
18         up1(rs,t[p].tag);
19         t[p].tag=0;
20     }
21 }
22 void build(int l,int r,int p){
23     t[p].l=l;t[p].r=r;
24     if(l==r){
25         t[p].v=a[l];

```

```

26         return;
27     }
28     int mid=l+r>>1;
29     build(l,mid,ls);build(mid+1,r,rs);
30     pushup(p);
31 }
32 void upd(int l,int r,int p,ll c){
33     if(l<=t[p].l&&t[p].r<=r){
34         up1(p,c);return;
35     }
36     pushdown(p);
37     int mid=t[p].l+t[p].r>>1;
38     if(l<=mid)upd(l,r,ls,c);
39     if(r>mid)upd(l,r,rs,c);
40     pushup(p);
41 }
42 ll qry(int l,int r,int p){
43     if(l<=t[p].l&&t[p].r<=r)return t[p].v;
44     pushdown(p);
45     int mid=t[p].l+t[p].r>>1;
46     ll s=0;
47     if(l<=mid)s=qry(l,r,ls);
48     if(r>mid)s+=qry(l,r,rs);
49     return s;
50 }

```

3.2.2 暴力更新线段树

```

1 void upd(int l,int r,int p,ll c){//少量暴力更新子树
2     if(l<=t[p].l&&t[p].r<=r&&t[p].flag){//该子树可以剪枝
3         up1(p,c);//打上标记
4         return;
5     }
6     if(t[p].l==t[p].r){ //叶子节点更新
7         return;
8     }
9     pushdown(p);
10    int mid=t[p].l+t[p].r>>1;
11    if(l<=mid)upd(l,r,ls,c);
12    if(r>mid)upde(l,r,rs,c);
13    pushup(p);

```

```
14 }
```

3.2.3 线段树上二分

```
1 int qry(int p,int l,int x){//[1,n]第一个<=x的下标
2     if(t[p].l==t[p].r)return t[p].l;
3     pushdown(p);
4     int mid=t[p].l+t[p].r>>1;
5     if(l<=t[p].l){
6         if(t[ls].mn<=x)return qry(ls,l,x);
7         if(t[rs].mn>x)return -1;
8         return qry(rs,l,x);
9     }
10    else if(l<=mid){
11        int s=qry(ls,l,x);
12        if(s!=-1)return s;
13        if(t[rs].mn>x)return -1;
14        return qry(rs,l,x);
15    }
16    else{
17        if(t[rs].mn>x)return -1;
18        return qry(rs,l,x);
19    }
20 }
```

3.2.4 线段树合并

```
1 void merge(int &p,int q){
2     if(!p||!q){p=p|q;return;}
3     t[p].v+=t[q].v;
4     merge(t[p].ls,t[q].ls);
5     merge(t[p].rs,t[q].rs);
6 }
7 int merge(int p,int q){//树形DP保留树上所有结点答案时，线段树合并需要新建结点
8     if(!p||!q)return p|q;
9     int now=++tot;
10    t[now].v=t[p].v+t[q].v;
11    t[now].ls=merge(t[p].ls,t[q].ls);
12    t[now].rs=merge(t[p].rs,t[q].rs);
13    return now;
14 }
```

3.2.5 线段树分裂

```

1  int n,m,q,rt[N],tot,a[N];
2  map<int,int>mp;//mp[区间左端点]=区间右端点
3  int vis[N];//0:升序 1:降序
4  struct sgt{
5      int ls,rs;
6      int v,rev,rtag;//tag=1:需要翻转
7  }t[N<<5];
8  int newnode(int op){
9      int p=++tot;
10     t[p].rev=op;
11     return p;
12 }
13 void pushup(int p){
14     t[p].v=t[t[p].ls].v+t[t[p].rs].v;
15 }
16 void upd(int &p,int l,int r,int x){
17     if(!p)p=newnode(0);
18     if(l==r){
19         ++t[p].v;
20         return;
21     }
22     int mid=l+r>>1;
23     if(x<=mid)upd(t[p].ls,l,mid,x);
24     else upd(t[p].rs,mid+1,r,x);
25     pushup(p);
26 }
27 void reverse1(int p){
28     if(!p)return;
29     t[p].rtag^=1;
30     t[p].rev^=1;
31     swap(t[p].ls,t[p].rs);
32 }
33 void pushdown(int p){
34     if(t[p].rtag){
35         reverse1(t[p].ls);
36         reverse1(t[p].rs);
37         t[p].rtag=0;

```

```

38     }
39 }
40 void merge(int &p,int q){
41     if(!p||!q){
42         p=p|q;return;
43     }
44     pushdown(p);pushdown(q);
45     merge(t[p].ls,t[q].ls);
46     merge(t[p].rs,t[q].rs);
47     pushup(p);
48 }
49 int kth(int p,int l,int r,int k){//从左往右第k个
50     if(l==r)return l;
51     pushdown(p);
52     int mid=l+r>>1;
53     if(!t[p].rev){
54         if(t[t[p].ls].v>=k)return kth(t[p].ls,l,mid,k);
55         else return kth(t[p].rs,mid+1,r,k-t[t[p].ls].v);
56     }
57     else{
58         if(t[t[p].ls].v>=k)return kth(t[p].ls,mid+1,r,k);
59         else return kth(t[p].rs,l,mid,k-t[t[p].ls].v);
60     }
61 }
62 void splitL(int p,int &q,ll k){
63     if(!p)return;
64     q=newnode(t[p].rev);//新结点翻转状态一致
65     pushdown(p);
66     ll s=t[t[p].ls].v;
67     if(k>=s){
68         t[q].ls=t[p].ls;t[p].ls=0;
69         if(k>s)splitL(t[p].rs,t[q].rs,k-s);
70     }
71     else splitL(t[p].ls,t[q].ls,k);
72     pushup(p);pushup(q);
73 }
74 void split(int L,int R,int l,int r,int op){//从[L,R]中分裂出[l,r]
75     if(l!=L){
76         rt[l]=rt[L];
77         splitL(rt[l],rt[L],l-L);
78         mp[L]=l-1;mp[l]=R;

```

```
79     vis[l]=vis[L];
80 }
81 if(r!=R){
82     rt[r+1]=rt[l];
83     splitL(rt[r+1],rt[l],r-l+1);
84     mp[l]=r;mp[r+1]=R;
85     vis[r+1]=vis[L];
86 }
87 if(vis[l]!=op){
88     vis[l]=op;
89     reverse1(rt[l]);
90 }
91 }
92 int main(){
93     scanf("%d%d",&n,&m);
94     for(int i=1;i<=n;++i){
95         scanf("%d",&a[i]);
96         upd(rt[i],1,n,a[i]);
97         mp[i]=i;vis[i]=0;
98     }
99     while(m--){
100         int op,l,r;scanf("%d%d%d",&op,&l,&r);
101         auto pr1=*--mp.upper_bound(l);
102         auto pr2=*--mp.upper_bound(r);
103         int a=pr1.first,b=pr1.second;
104         int c=pr2.first,d=pr2.second;
105         if(pr1==pr2){
106             split(a,b,l,r,op);
107         }
108         else{
109             split(a,b,l,b,op); split(c,d,c,r,op);
110             for(auto it=mp.upper_bound(l);it!=mp.end()&&it->second<=r;it=mp.erase
111                 (it)){
112                 if (vis[it->first]!=op){
113                     reverse1(rt[it->first]),vis[it->first]=op;
114                 }
115                 merge(rt[l],rt[it->first]);
116             }
117             mp[l]=r;
118         }
119     }
```



```

119     scanf("%d",&q);
120     int pos=(--mp.upper_bound(q))->first;
121     int ans=kth(rt[pos],1,n,q-pos+1);
122     printf("%d\n",ans);
123 }

```

3.2.6 扫描线

```

1 //求n个矩形面积并
2 struct rectangle{int x1,y1,x2,y2;}a[N];
3 int n,b[N<<1],h,hb;
4 struct line{int x1,x2,y,c;}c[N<<1];
5 bool cmp(line a,line b){return a.y<b.y;}
6 struct sgt{
7     int l,r,v,cnt;
8 }t[N<<3]; //N个矩形有2N个x坐标, 要8N线段树结点
9 #define ls (p<<1)
10 #define rs (p<<1|1)
11 void build(int l,int r,int p){
12     t[p].l=l;t[p].r=r;
13     if(l==r)return;
14     int mid=l+r>>1;
15     build(l,mid,ls);build(mid+1,r,rs);
16 }
17 void pushup(int p){
18     if(t[p].cnt)t[p].v=b[t[p].r+1]-b[t[p].l]; //整个区间都有
19     else{
20         if(t[p].l==t[p].r)t[p].v=0; //叶子结点特判避免RE
21         else t[p].v=t[ls].v+t[rs].v; //根据儿子更新
22     }
23 }
24 void upd(int l,int r,int p,int c){ //[l,r]被覆盖次数+=c
25     if(l<=t[p].l&&t[p].r<=r){
26         t[p].cnt+=c; pushup(p); //pushup来更新,比较方便
27         return;
28     }
29     int mid=t[p].l+t[p].r>>1;
30     if(l<=mid)upd(l,r,ls,c);
31     if(r>mid)upd(l,r,rs,c);
32     pushup(p);
33 }

```

```

34 int main(){
35     scanf("%d",&n);
36     for(int i=1;i<=n;++i)scanf("%d%d%d%d",&a[i].x1,&a[i].y1,&a[i].x2,&a[i].y2);
37     for(int i=1;i<=n;++i){b[++hb]=a[i].x1;b[++hb]=a[i].x2;}
38     sort(b+1,b+1+hb);hb=unique(b+1,b+1+hb)-b-1;
39     for(int i=1;i<=n;++i){//对x离散化
40         a[i].x1=lower_bound(b+1,b+1+hb,a[i].x1)-b;
41         a[i].x2=lower_bound(b+1,b+1+hb,a[i].x2)-b;
42     }
43     for(int i=1;i<=n;++i){//在y1对x的区间+1,在y2对x的区间-1,大于0就是有的
44         c[++h]={a[i].x1,a[i].x2,a[i].y1,1};
45         c[++h]={a[i].x1,a[i].x2,a[i].y2,-1};
46     }
47     sort(c+1,c+1+h,cmp);
48     build(1,hb-1,1);//化点为线,x坐标下标:x1--x2--x3--x4--x5--x6--x7
49     long long ans=0;//          线段下标:  s1  s2  s3  s4  s5  s6
50     for(int i=1;i<h;++i){
51         upd(c[i].x1,c[i].x2-1,1,c[i].c);
52         ans+=1ll*t[1].v*(c[i+1].y-c[i].y);//线段之和*高度(重合的扫描线高度差为0)
53     }
54     printf("%lld\n",ans);
55 }

```

3.2.7 segment tree beats

实现区间加, 区间对一个数取 min, 区间和

维护区间最大值 mx1 和区间次大值 mx2, 当区间对 c 取 min 时, c 小于 mx2 就暴力, 否则打标记

```

1 void pushup(int p){
2     t[p].v=t[ls].v+t[rs].v;
3     if(t[ls].mx1>t[rs].mx1){
4         t[p].mx1=t[ls].mx1;
5         t[p].cnt=t[ls].cnt;
6         t[p].mx2=max(t[ls].mx2,t[rs].mx1);
7     }
8     else if(t[ls].mx1==t[rs].mx1){
9         t[p].mx1=t[ls].mx1;
10        t[p].cnt=t[ls].cnt+t[rs].cnt;
11        t[p].mx2=max(t[ls].mx2,t[rs].mx2);
12    }
13    else{

```

```

14         t[p].mx1=t[rs].mx1;
15         t[p].cnt=t[rs].cnt;
16         t[p].mx2=max(t[rs].mx2,t[ls].mx1);
17     }
18 }
19 void up1(int p,int c){
20     if(c<t[p].mx1){
21         t[p].tag=c;
22         t[p].v-=1ll*(t[p].mx1-c)*t[p].cnt;
23         t[p].mx1=c;
24     }
25 }
26 void upd(int l,int r,int p,int c){
27     if(l<=t[p].l&& t[p].r<=r&&c>=t[p].mx2){
28         up1(p,c);return;
29     }
30     if(t[p].l==t[p].r){
31         t[p].cnt=1;
32         t[p].mx1=c;
33         t[p].mx2=-1;
34         t[p].tag=c;
35         t[p].v=c;
36         return;
37     }
38     pushdown(p);
39     int mid=t[p].l+t[p].r>>1;
40     if(l<=mid)upd(l,r,ls,c);
41     if(r>mid)upd(l,r,rs,c);
42     pushup(p);
43 }

```

3.3 单调栈

左边最近的小于 $a[i]$ 的下标

```

1 int a[N],L[N],n;//L[i]=左边最近的小于a[i]的下标
2 int sta[N],top;//单调递增的栈
3 int main(){
4     for(int i=1;i<=n;++i){
5         while(top&&a[sta[top]]>=a[i]){
6             //sta[top]左边第一个小于它的是sta[top-1]
7             //sta[top]右边第一个小于等于它的是i

```

```

8         --top;
9     }
10    L[i]=sta[top];sta[++top]=i;
11 }
12 }

```

最大子矩形

```

1 //给定一个只有A和B的矩阵,求最大的全是A的矩阵
2 int n,m,a[N][N],ans;//a[i][j]=(i,j)最大向上扩展(全A)的长度
3 for(int i=1;i<=n;++i){//对每一行向上扩展取max
4     sta[0]=0;
5     for(int j=1;j<=m+1;++j){
6         if(s[i][j]=='B' || j==m+1){//该位置是非法位置
7             while(top){//清空栈并更新答案
8                 ans=max(ans,(j-sta[top-1]-1)*a[i][sta[top]]);
9                 --top;
10            }
11            sta[0]=j;//更新栈低下标
12        }
13        else{//该位置是合法位置,需要在单调栈中对最大的矩形取max
14            while(top&& a[i][sta[top]]>a[i][j]){
15                ans=max(ans,(j-sta[top-1]-1)*a[i][sta[top]]);
16                --top;
17            }
18            sta[++top]=j;
19        }
20    }
21 }

```

计算矩形个数

```

1 struct P{
2     int x,y;
3     inline bool operator < (const P &b) const{
4         return x<b.x || x==b.x&& y<b.y;
5     }
6 };
7 ll calc(int h[],int l,int r){//[l,r]区间有几个矩形,第i个位置高度为h[i]
8     static int sta[N],top;
9     //vl[i]=左边最近的小于h[i]的位置,不存在时为l-1
10    //vr[i]=右边最近的小于等于h[i]的位置,不存在时为r+1
11    //ans=sum(vr[i]-i)*(i-vl[i])*h[i],实际不需要储存vl和vr数组

```

```

12     ll ans=0;    sta[top=0]=l-1;
13     for(int i=l;i<=r;++i){
14         while(top&&h[sta[top]]>=h[i]){
15             ans+=ll(sta[top]-sta[top-1])*(i-sta[top])*h[sta[top]];
16             --top;
17         }
18         sta[++top]=i;
19     }
20     while(top){
21         ans+=ll(sta[top]-sta[top-1])*(r+1-sta[top])*h[sta[top]];
22         --top;
23     }
24     return ans;
25 }
26 ll cnt_rectangle(vector<P>&v){
27     static int h[N][N]; //h[i][j]=(i,j)位置往上最多延申几个(包括自身)
28     int top=0;
29     sort(v.begin(),v.end());
30     for(auto &p:v)h[p.x][p.y]=1; //记录询问的点
31     ll ans=0;
32     for(int i=0;i<v.size();++i){
33         int j=i,x=v[i].x;
34         while(j+1<v.size()&&v[j+1].x==x)++j; //提取同一行
35         for(int k=i;k<=j;++k)h[x][v[k].y]=h[x-1][v[k].y]+1;
36         for(int k=i;k<=j;++k){
37             int t=k;
38             while(t+1<=j&&v[t+1].y==v[t].y+1)++t; //提取相连的
39             ans+=calc(h[x],v[k].y,v[t].y);
40             k=t;
41         }
42         i=j;
43     }
44     for(auto &p:v)h[p.x][p.y]=0; //删除询问的点
45     return ans;
46 }

```

3.4 单调队列

```

1 //dp[i]=min(dp[i-k],dp[i-k+1],...,dp[i-1])+a[i]
2 int q[N],fr=0;bc=-1; //单增队列,保证队首为区间最小值
3 q[++bc]=0; //先加入一个dp[0]=a[0]=0

```

```

4  for(int i=1;i<=n;++i){
5      dp[i]=dp[q[fr]]+a[i];
6      while(bc>=fr&&dp[q[bc]]>=dp[i])--bc;
7      q[++bc]=i;
8      while(bc>=fr&&q[fr]<=i-k)++fr;
9  }

```

3.5 ST 表

```

1  int st[22][N];
2  for(int i=2;i<N;++i)Log2[i]=Log2[i>>1]+1;//注意i从2开始
3  for(int i=1;i<=n;++i)st[0][i]=a[i];
4  for(int i=1;1<i<=n;++i){
5      for(int j=1;j+(1<i)-1<=n;++j){
6          st[i][j]=max(st[i-1][j],st[i-1][j+(1<i)-1]);
7      }
8  }
9  int qry(int l,int r){
10     int k=Log2[r-l+1];//__lg(r-l+1);
11     return max(st[k][l],st[k][r-(1<k)+1]);
12 }

```

3.6 并查集

3.6.1 路径压缩并查集

```

1  int n,fa[N];
2  int find(int x){
3      return x==fa[x]?x:fa[x]=find(fa[x]);
4  }
5  int main(){
6      for(int i=1;i<=n;++i)fa[i]=i;//初始化
7      fa[find(x)]=find(y);//连接x,y
8  }

```

划分两个集合，可以令 x 为选择 x ， $x+n$ 为不选择 x ，若 x 和 $y+n$ 在同一个并查集，代表某一个集合选 x 并且不选 y

3.6.2 按秩合并并查集

```

1  int fa[N],sz[N];//也可以按dep合并
2  int find(int x){return x==fa[x]?x:find(fa[x]);}
3  void merge(int x,int y){
4      x=find(x);y=find(y);
5      if(sz[x]>sz[y])swap(x,y);
6      fa[x]=y;sz[y]+=sz[x];
7  }

```

3.6.3 可撤销并查集

```

1  int n,fa[N],sz[N],top;
2  pair<int,int>sta[N];
3  int find(int x){
4      return x==fa[x]?x:find(fa[x]);
5  }
6  void merge(int x,int y){
7      int x=find(x),y=find(y);
8      if(sz[x]>sz[y])swap(x,y);
9      fa[x]=y;sz[y]+=sz[x];
10     sta[++top]=make_pair(x,y);
11 }
12 void del(){//后连接的必须先撤销
13     pair<int,int>pr=sta[top--];
14     int x=pr.first,y=pr.second;
15     fa[x]=x;sz[y]-=sz[x];
16 }

```

3.6.4 可持久化并查集

```

1  const int N=1e5+5,Q=2e5+5,LOGN=log2(N)+1;
2  int n,rt[N<<1],tot,q;
3  struct sgt{
4      int ls,rs;
5      int fa,d;
6  }t[(N+Q)*LOGN];
7  int build(int l,int r){
8      int p=++tot;
9      if(l==r){
10         t[p].fa=l;t[p].d=1;
11         return p;

```

```
12     }
13     int mid=l+r>>1;
14     t[p].ls=build(l,mid);
15     t[p].rs=build(mid+1,r);
16     return p;
17 }
18 int upd_fa(int l,int r,int last,int x,int fa){
19     int p=++tot;
20     t[p]=t[last];
21     if(l==r){t[p].fa=fa;return p;}
22     int mid=l+r>>1;
23     if(x<=mid)t[p].ls=upd_fa(l,mid,t[last].ls,x,fa);
24     else t[p].rs=upd_fa(mid+1,r,t[last].rs,x,fa);
25     return p;
26 }
27 void upd_dep(int l,int r,int p,int x){
28     if(l==r){++t[p].d;return;}
29     int mid=l+r>>1;
30     if(x<=mid)upd_dep(l,mid,t[p].ls,x);
31     else upd_dep(mid+1,r,t[p].rs,x);
32 }
33 int qry(int l,int r,int x,int p){
34     if(l==r)return p;
35     int mid=l+r>>1;
36     if(x<=mid)return qry(l,mid,x,t[p].ls);
37     else return qry(mid+1,r,x,t[p].rs);
38 }
39 int find(int x,int p){
40     int id=qry(1,n,x,p);
41     int f=t[id].fa;
42     return x==f?id:find(f,p);
43 }
44 void merge(int x,int y,int &p){
45     int fxid=find(x,p);
46     int fyid=find(y,p);
47     int fx=t[fxid].fa,fy=t[fyid].fa;
48     int dx=t[fxid].d,dy=t[fyid].d;
49     if(dx>dy)swap(fx,fy);
50     p=upd_fa(1,n,p,fx,fy);
51     if(dx==dy)upd_dep(1,n,p,fy);
52 }
```



```
53 int main(){rt[0]=build(1,n);}
```

3.7 主席树

区间第 k 小

```
1  const int N=2e5+10;
2  int n,q,a[N],b[N],rt[N],tot;
3  struct sgt{
4      int v;
5      int ls,rs;
6  }t[N*33]; //nlogC ,离散化后C=n
7  int insert(int l,int r,int last,int x,int c){
8      int p=++tot;
9      t[p]=t[last];
10     t[p].v+=c;
11     if(l!=r){
12         int mid=l+r>>1;
13         if(x<=mid)t[p].ls=insert(l,mid,t[p].ls,x,c);
14         else t[p].rs=insert(mid+1,r,t[p].rs,x,c);
15     }
16     return p;
17 }
18 int qry(int l,int r,int p1,int p2,int k){
19     if(l==r)return l;
20     int s=t[t[p2].ls].v-t[t[p1].ls].v;
21     int mid=l+r>>1;
22     if(k<=s)return qry(l,mid,t[p1].ls,t[p2].ls,k);
23     else return qry(mid+1,r,t[p1].rs,t[p2].rs,k-s);
24 }
25 int main(){
26     scanf("%d%d",&n,&q);
27     for(int i=1;i<=n;++i)scanf("%d",&a[i]),b[i]=a[i];
28     sort(b+1,b+1+n);
29     int h=unique(b+1,b+1+n)-b-1;
30     for(int i=1;i<=n;++i)a[i]=lower_bound(b+1,b+1+h,a[i])-b;
31     for(int i=1;i<=n;++i)rt[i]=insert(1,n,rt[i-1],a[i],1);
32     while(q--){
33         int l,r,k;scanf("%d%d%d",&l,&r,&k);
34         int ans=qry(1,n,rt[l-1],rt[r],k);
35         printf("%d\n",b[ans]);
36     }
```

37 }

区间 mex

```

1  const int N=2e5+5;
2  int n,q,a[N];
3  struct sgt{
4      int ls,rs;
5      int v;//权值区间最后一次出现位置的最小值
6  }t[N*33]; //nlogC
7  int rt[N],tot;
8  int insert(int l,int r,int last,int x,int pos){
9      int p=++tot;
10     t[p]=t[last];
11     if(l==r){
12         t[p].v=pos;return p;
13     }
14     int mid=l+r>>1;
15     if(x<=mid)t[p].ls=insert(l,mid,t[last].ls,x,pos);
16     else t[p].rs=insert(mid+1,r,t[last].rs,x,pos);
17     t[p].v=min(t[t[p].ls].v,t[t[p].rs].v);
18     return p;
19 }
20 int qry(int l,int r,int lim,int p){
21     if(l==r)return l;
22     int mid=l+r>>1;
23     if(t[t[p].ls].v<lim)return qry(l,mid,lim,t[p].ls);
24     else return qry(mid+1,r,lim,t[p].rs);
25 }
26 int main(){
27     scanf("%d%d",&n,&q);
28     for(int i=1;i<=n;++i)scanf("%d",&a[i]);
29     for(int i=1;i<=n;++i)rt[i]=insert(0,n,rt[i-1],a[i],i);
30     while(q--){
31         int l,r;scanf("%d%d",&l,&r);
32         int ans=qry(0,n,l,rt[r]);
33         printf("%d\n",ans);
34     }
35 }

```

区间 lcm:

对于一个数，将其分解质因数，若有因子 p^k ，那么拆分出 k 个数 p, p^2, \dots, p^k ，权值都为 p ，

$lcm([l, r]) = \text{区间权值之积}$

3.8 树套树

树状数组套动态开点线段树

```

1  struct sgt{
2      int ls,rs;
3      int v;
4  }t[N*200]; //nlognlogC
5  int rt[N],tot;
6  void upd(int l,int r,int &p,int x,int c){
7      if(!p)p=++tot;
8      t[p].v+=c;
9      if(l==r)return;
10     int mid=l+r>>1;
11     if(x<=m)upd(l,mid,t[p].ls,x,c);
12     else upd(mid+1,r,t[p].rs,x,c);
13 }
14 int qry(int L,int R,int l,int r,int p){
15     if(!p)return 0;
16     if(L<=l&&r<=R)return t[p].v;
17     int mid=l+r>>1,s=0;
18     if(L<=m)s+=qry(L,R,l,mid,t[p].ls);
19     if(R>m)s+=qry(L,R,mid+1,r,t[p].rs);
20     return s;
21 }
22 void add(int i,int x,int k){ //第i棵线段树在x的位置+k
23     for(;i<=n;i+=i&-i)upd(1,n,rt[i],x,k);
24 }
25 int query(int x,int l,int r){ //询问[1,x]区间值域[l,r]的个数
26     if(l>r)return 0;
27     int s=0;
28     for(;x;x-=x&-x)s+=qry(1,r,1,n,rt[x]);
29     return s;
30 }

```

线段树套动态开点线段树

```

1  //考虑权值套区间，还是区间套权值
2  //外层线段树应当只进行单点更新，且不需要pushup
3  //内存线段树动态开点，且可以进行区间打标记

```

3.9 Trie 树

```

1  const int N=1e5+10,M=N*30;
2  int t[M][26],cnt[M],tot;//M为单词字母个数和
3  void insert(string &s){
4      int p=0;
5      for(int i=0;i<s.length();++i){
6          int c=s[i]-'a';
7          if(!t[p][c])t[p][c]=++tot;
8          p=t[p][c];
9      }
10     ++cnt[p];
11 }
12 int qry(string &s){//s有几个
13     int p=0;
14     for(int i=0;i<s.length();++i){
15         int c=s[i]-'a';
16         if(!t[p][c])return 0;
17         p=t[p][c];
18     }
19     return cnt[p];
20 }

```

3.10 可持久化 01Trie

```

1  //求 $\max[1 \leq i \leq r] a[i]^a[i+1]^{\dots}^a[n]^x$ ,即求与 $(x^s[n])$ 异或起来最大的 $s[i-1]$ 
2  int n,q,a[N],s[N],rt[N];
3  int t[N*30][2],tot,lastvis[N*30];
4  int insert(int last,int i,int k){
5      int p=++tot;
6      lastvis[p]=i;
7      if(k<0)return p;
8      int c=s[i]>>k&1;
9      t[p][c^1]=t[last][c^1];
10     t[p][c]=insert(t[last][c],i,k-1);
11     return p;
12 }
13 int qry(int l,int p,int x){//[l,r]上 $\max(s[i]^x)$ 
14     for(int i=30;i>=0;--i){
15         int c=x>>i&1;
16         if(lastvis[t[p][c^1]]>=l)p=t[p][c^1];

```

```

17     else p=t[p][c];
18 }
19     return s[lastvis[p]]^x;
20 }
21 void debug(int p,int k,string s){
22     if(k<0){
23         cout<<s<<endl;
24         return;
25     }
26     if(t[p][0])debug(t[p][0],k-1,s+'0');
27     if(t[p][1])debug(t[p][1],k-1,s+'1');
28 }
29 int main(){
30     scanf("%d%d",&n,&q);
31     for(int i=1;i<=n;++i)scanf("%d",&a[i]);
32     for(int i=1;i<=n;++i)s[i]=s[i-1]^a[i];
33     lastvis[0]=-1;//0代表不存在的结点,必须赋值为-1
34     rt[0]=insert(0,0,30);//建立前缀和数组时s[0]存在,需要插入
35     for(int i=1;i<=n;++i)rt[i]=insert(rt[i-1],i,30);
36     char op[3];
37     for(;q--){
38         scanf("%s",op);
39         if(op[0]=='A'){//数组尾部插入一个数字
40             int x;scanf("%d",&x);
41             a[++n]=x;s[n]=s[n-1]^a[n];
42             rt[n]=insert(rt[n-1],n,30);
43         }
44         else{
45             int l,r,x;scanf("%d%d%d",&l,&r,&x);
46             int ans=qry(l-1,rt[r-1],x^s[n]);
47             printf("%d\n",ans);
48         }
49     }
50 }

```

3.11 莫队

3.11.1 单点修改莫队

```

1 int n,m,a[N],pos[N];//pos[i]=i所在块
2 struct QQ{ int l,r,id; }q[N];

```

```

3 inline bool cmp(QQ a,QQ b){//询问按块排序
4     return pos[a.l]<pos[b.l]||pos[a.l]==pos[b.l]&& a.r<b.r;
5 }
6 ll cnt[M],ans[N],now;//cnt[i]=数字出现次数,now=当前答案
7 void add(int x){
8     now+=cnt[s[x]]; ++cnt[s[x]];
9 }
10 void sub(int x){
11     now-=cnt[s[x]]-1; --cnt[s[x]];
12 }
13 int main(){
14     scanf("%d%d",&n,&m);int sz=sqrt(n);
15     for(int i=1;i<=n;++i)scanf("%d",&a[i]),pos[i]=i/sz;
16     for(int i=1;i<=m;++i){scanf("%d%d",&q[i].l,&q[i].r);q[i].id=i;}
17     sort(q+1,q+1+m,cmp);
18     int L=1,R=0; now=0;
19     for(int i=1;i<=m;++i){
20         while(q[i].l<L)add(--L);
21         while(q[i].r>R)add(++R);
22         while(q[i].l>L)sub(L++);
23         while(q[i].r<R)sub(R--);
24         ans[q[i].id]=now;
25     }
26     for(int i=1;i<=m;++i)printf("%lld\n",ans[i]);
27 }

```

3.11.2 回滚莫队

```

1 const int maxn=3e5+25,INF=0x3f3f3f3f;
2 int block_size;
3 int ans[maxn],color[maxn],cntl[maxn],cntr[maxn],s[maxn];
4 int l=1,r=0,sum=0,n,m,mx;
5 struct query {
6     int l,r,id;
7     bool operator<(const query& sec)const {
8         if(l/block_size!=sec.l/block_size)return l<sec.l;
9         return r<sec.r;
10    }
11 } q[maxn];
12 int tmp1[maxn], tmpr[maxn];
13 int __l[maxn], __r[maxn];

```

```

14 int main() {
15     memset(tmp1,INF,sizeof(tmp1));
16     memset(cnt1,INF,sizeof(cnt1));
17     n=read();
18     block_size=sqrt(n);
19     for (int i = 1; i <= n; i++)s[i]=color[i]=read();
20     sort(s+1,s+n+1);
21     int sz=unique(s+1,s+n+1)-s-1;
22     for(int i = 1; i <= n; i++)color[i]=lower_bound(s+1,s+1+sz,color[i])-s;
23     m=read();
24     for (int i = 1; i <= m; i++) {
25         q[i].l=read();q[i].r=read();q[i].id=i;
26     }
27     sort(q+1,q+1+m);
28     for (int i = 1; i <= m; i++) {
29         int curmx = 0;
30         if(q[i].l/block_size==q[i].r/block_size) { // l,r在通一块,暴力
31             for (int j = q[i].l; j <= q[i].r; j++) {
32                 if (!__l[color[j]]) __l[color[j]] = j;
33                 __r[color[j]] = j;
34                 curmx = max(curmx, __r[color[j]] - __l[color[j]]);
35             }
36             for (int j = q[i].l; j <= q[i].r; j++) {
37                 __l[color[j]] = 0;
38                 __r[color[j]] = 0;
39             }
40             ans[q[i].id] = curmx;
41             continue;
42         }
43         if((q[i].l/block_size+1)*block_size!=1) { // l在另一块
44             mx=0;
45             while(r>=1)cnt1[color[r]]=0x3f3f3f3f,cntr[color[r]]=0,r--; // reset
46             l=(q[i].l/block_size+1)*block_size;
47             r=l-1;
48         }
49         while(r<q[i].r) {
50             ++r;
51             cnt1[color[r]] = min(r,cnt1[color[r]]);
52             cntr[color[r]] = r;
53             mx=max(mx,cntr[color[r]] - cnt1[color[r]]);
54         }

```

```

55     curmx = mx;
56     for (int j = q[i].l; j <= l - 1; j++) {
57         tmp1[color[j]] = min(j, tmp1[color[j]]);
58         tmpr[color[j]] = max(j, tmpr[color[j]]);
59         curmx = max(curmx, max(tmpr[color[j]], cnt1[color[j]]) - min(tmp1[color[j]], cnt1[color[j]]));
60     }
61     for (int j = q[i].l; j <= l - 1; j++) {
62         tmp1[color[j]] = 0x3f3f3f3f;
63         tmpr[color[j]] = 0;
64     }
65     ans[q[i].id] = curmx;
66 }
67 for (int i = 1; i <= m; i++) printf("%d\n", ans[i]);
68 }

```

3.11.3 树上莫队

```

1  struct treenode {
2      int fa, hson, sz, depth, top;
3      vector<int> nxt;
4  } t[maxn];
5  struct Query {
6      int l, r, id, version;
7      int posl, posr;
8      bool operator<(const Query& sec) const {
9          if(posl != sec.posl) return l < sec.l;
10         if(posr != sec.posr) return r < sec.r;
11         return version < sec.version;
12     }
13 } q[maxn];
14 int block_size, Q;
15 int color[maxn], cnt[maxn], ver[maxn], save[maxn], v[maxn], w[maxn], c[maxn], vis[maxn];
16 int in[maxn], out[maxn], id[maxn]; // tree_to_array
17 int l=1, r=0, n, m, version=0, tot=0, pos, newval, x, y, tmp1, tmp2, trans, stamp=1;
18 ll sum=0, ans[maxn];
19 void dfs1(int pos) {
20     in[pos] = ++stamp;
21     id[stamp] = pos;
22     t[pos].hson = -1; // heavyson should be -1
23     t[pos].sz = 1;

```



```

24     for(int k:t[pos].nxt) {
25         if(!t[k].depth) {
26             t[k].depth=t[pos].depth+1;
27             t[k].fa=pos;
28             dfs1(k);
29             t[pos].sz+=t[k].sz;
30             if(t[pos].hson==-1||t[k].sz>t[t[pos].hson].sz) {
31                 t[pos].hson=k;
32             }
33         }
34     }
35     out[pos]=++stamp;
36     id[stamp]=pos;
37 }
38 void dfs2(int pos,int top) {
39     t[pos].top=top;
40     if(t[pos].hson==-1)return;
41     dfs2(t[pos].hson,top); //to visit heavy_son first
42     for(int k:t[pos].nxt) {
43         if(k!=t[pos].fa&&k!=t[pos].hson)dfs2(k,k);
44     }
45 }
46 int query_LCA(int a,int b) {
47     while(t[a].top!=t[b].top) {
48         if(t[t[a].top].depth<t[t[b].top].depth)swap(a,b);
49         a=t[t[a].top].fa;
50     }
51     return t[a].depth<t[b].depth?a:b;
52 }
53 inline void add(int cur) { //cur: the number of node, in xor, add and del are
    equivalent
54     if(vis[cur]==0)sum+=1ll*v[color[cur]]*w[++cnt[color[cur]]];
55     else sum-=1ll*v[color[cur]]*w[cnt[color[cur]]--];
56     vis[cur]^=1;
57 }
58 inline void work(int cur) { //cur: the version
59     int num=ver[cur],clr=save[cur]; // clr:color
60     if((l<=in[num]&&in[num]<=r)^(l<=out[num]&&out[num]<=r)) {
61         sum+=1ll*v[clr]*w[++cnt[clr]];
62         sum-=1ll*v[color[num]]*w[cnt[color[num]]--];
63     }

```

```

64     swap(save[cur],color[num]);
65 }
66 int main() {
67     n=read(),m=read(),Q=read();
68     block_size=pow((n<<1),0.667);
69     for(int i=1; i<=m; i++)v[i]=read();
70     for(int i=1; i<=n; i++)w[i]=read();
71     for(int i=0; i<n-1; i++) {
72         tmp1=read(),tmp2=read();
73         t[tmp1].nxt.push_back(tmp2);
74         t[tmp2].nxt.push_back(tmp1);
75     }
76     t[1].depth=1; dfs1(1); dfs2(1,1);
77     for(int i=1; i<=n; i++) color[i]=read();
78     for(int i=0; i<Q; i++) {
79         trans=read();
80         if(trans==1) {
81             tmp1=read(),tmp2=read();
82             if(in[tmp1]>in[tmp2])swap(tmp1,tmp2);
83             if(out[tmp1]<out[tmp2])q[tot].l=out[tmp1];
84             else q[tot].l=in[tmp1];
85             q[tot].r=in[tmp2];
86             q[tot].posl=q[tot].l/block_size;
87             q[tot].posr=q[tot].r/block_size;
88             q[tot].id=tot;
89             q[tot].version=version;
90             tot++;
91         } else {
92             version++;
93             ver[version]=read();
94             save[version]=read();
95         }
96     }
97     sort(q,q+tot);
98     version=0;
99     for(int i=0; i<tot; i++) {
100         while(l>q[i].l)add(id[--l]);
101         while(r<q[i].r)add(id[++r]);
102         while(l<q[i].l)add(id[l++]);
103         while(r>q[i].r)add(id[r--]);
104         while(version<q[i].version)work(++version);

```

```

105     while(version>q[i].version)work(version--);
106     x=id[l],y=id[r];
107     int f=query_LCA(x,y);
108     if(f!=x&&f!=y) {
109         add(f);
110         ans[q[i].id]=sum;
111         add(f);
112     } else ans[q[i].id]=sum;
113 }
114 for(int i=0; i<tot; i++)printf("%lld\n",ans[i]);
115 }

```

3.12 平衡树

3.12.1 Treap

```

1  struct treap{
2      int l,r;
3      int v,k;
4      int sz,cnt;
5  }t[N];
6  int rt,tot;
7  int New(int x){
8      t[++tot].v=x;
9      t[tot].k=rand();
10     t[tot].sz=t[tot].cnt=1;
11     t[tot].l=t[tot].r=0;
12     return tot;
13 }
14 void upd(int p){
15     t[p].sz=t[t[p].l].sz+t[t[p].r].sz+t[p].cnt;
16 }
17 void build(){
18     srand(time(0));
19     New(-inf);New(inf);
20     rt=1;t[1].r=2;
21     upd(rt);
22 }
23 void zig(int &p){
24     int q=t[p].l;
25     t[p].l=t[q].r;

```

```
26     t[q].r=p;p=q;
27     upd(t[p].r);upd(p);
28 }
29 void zag(int &p){
30     int q=t[p].r;
31     t[p].r=t[q].l;
32     t[q].l=p;p=q;
33     upd(t[p].l);upd(p);
34 }
35 void insert(int &p,int x){
36     if(!p){
37         p=New(x);
38         return;
39     }
40     if(x==t[p].v){
41         ++t[p].cnt;
42         ++t[p].sz;
43         return;
44     }
45     if(x<t[p].v){
46         insert(t[p].l,x);
47         if(t[t[p].l].k>t[p].k)zig(p);
48     }
49     else{
50         insert(t[p].r,x);
51         if(t[t[p].r].k>t[p].k)zag(p);
52     }
53     upd(p);
54 }
55 void del(int &p,int x){
56     if(!p)return;
57     if(x==t[p].v){
58         if(t[p].cnt>1){
59             --t[p].cnt;
60             --t[p].sz;
61             return;
62         }
63         if(t[p].l||t[p].r){
64             if(!t[p].r||t[t[p].l].k>t[t[p].r].k){
65                 zig(p);
66                 del(t[p].r,x);
```

```
67         }
68         else{
69             zag(p);
70             del(t[p].l,x);
71         }
72         upd(p);
73     }
74     else p=0;
75     return;
76 }
77 if(x<t[p].v)del(t[p].l,x);
78 else del(t[p].r,x);
79 upd(p);
80 }
81 int getrk(int p,int x){
82     if(!p)return 0;
83     if(x==t[p].v)return t[t[p].l].sz+1;
84     if(x<t[p].v)return getrk(t[p].l,x);
85     return getrk(t[p].r,x)+t[t[p].l].sz+t[p].cnt;
86 }
87 int getv(int p,int rk){
88     if(!p)return inf;
89     if(t[t[p].l].sz>=rk)return getv(t[p].l,rk);
90     if(t[t[p].l].sz+t[p].cnt>=rk)return t[p].v;
91     return getv(t[p].r,rk-t[t[p].l].sz-t[p].cnt);
92 }
93 int getpre(int x){
94     int p=rt,s=1;
95     while(p){
96         if(x==t[p].v){
97             if(t[p].l){
98                 p=t[p].l;
99                 while(t[p].r)p=t[p].r;
100                 s=p;
101             }
102             break;
103         }
104         if(t[p].v<x&& t[p].v>t[s].v)s=p;
105         if(x<t[p].v)p=t[p].l;
106         else p=t[p].r;
107     }
```

```

108     return t[s].v;
109 }
110 int getnext(int x){
111     int p=rt,s=2;
112     while(p){
113         if(x==t[p].v){
114             if(t[p].r){
115                 p=t[p].r;
116                 while(t[p].l)p=t[p].l;
117                 s=p;
118             }
119             break;
120         }
121         if(t[p].v>x&& t[p].v<t[s].v)s=p;
122         if(x<t[p].v)p=t[p].l;
123         else p=t[p].r;
124     }
125     return t[s].v;
126 }
127 int main(){
128     build();memset(t,0,sizeof t);//多测要清空
129     for(int n=rd();n;--n){
130         int op=rd(),x=rd();
131         if(op==1)insert(rt,x);
132         else if(op==2)del(rt,x);
133         else if(op==3)printf("%d\n",getrk(rt,x)-1);
134         //注意平衡树初始有-inf和inf,所以这里要+1/-1
135         else if(op==4)printf("%d\n",getv(rt,x+1));
136         else if(op==5)printf("%d\n",getpre(x));
137         else printf("%d\n",getnext(x));
138     }
139 }

```

3.12.2 splay

```

1  const int N=5e5+10,M=4e6+N,inf=1<<30;
2  int n,a[N],rt,tot;
3  struct Splay{
4      int v,tag;
5      int sz;
6      int c[2],f;

```

```

7  }t[M];
8  inline int New(int x){
9      t[++tot].v=x;t[tot].sz=1;
10     return tot;
11 }
12 inline void pushup(int p){
13     t[p].sz=t[t[p].c[0]].sz+t[t[p].c[1]].sz+1;
14 }
15 inline void pushdown(int p){}
16 int build(int l,int r,int f){//建树
17     if(l>r)return 0;
18     int m=l+r>>1;
19     int p=New(a[m]);
20     // if(a[m]!=-inf)idx[a[m]]=p;
21     t[p].f=f;
22     t[p].c[0]=build(l,m-1,p);
23     t[p].c[1]=build(m+1,r,p);
24     pushup(p); return p;
25 }
26 void rotate(int x){
27     int y=t[x].f,z=t[y].f;
28     int c=t[y].c[1]==x;
29     t[z].c[t[z].c[1]==y]=x;t[x].f=z;
30     t[y].c[c]=t[x].c[c^1];t[t[x].c[c^1]].f=y;
31     t[x].c[c^1]=y;t[y].f=x;
32     pushup(y);
33 }
34 void splay(int x,int g){
35     while(t[x].f!=g){
36         int y=t[x].f,z=t[y].f;
37         if(z!=g)rotate((t[y].c[1]==x)^(t[z].c[1]==y)?x:y);
38         rotate(x);
39     }
40     pushup(x);if(!g)rt=x;
41 }
42 int find(int k){
43     int p=rt;
44     while(1){
45         pushdown(p);
46         if(t[t[p].c[0]].sz>=k)p=t[p].c[0];
47         else if(t[t[p].c[0]].sz+1>=k)return p;

```

```

48         else{
49             k--=t[t[p].c[0]].sz+1;
50             p=t[p].c[1];
51         }
52     }
53 }
54 void gettree(int l,int r,int &x,int &y){//伸展[l,r]区间,t[y].c[0]即为区间的树根
55     x=find(l);y=find(r+2);
56     splay(x,0);splay(y,x);
57 }
58 void getpos(int pos,int &x,int &y){//在pos和pos+1之间的位置t[y].c[0]
59     x=find(pos+1);y=find(pos+2);
60     splay(x,0);splay(y,x);
61 }
62 void insert(int x,int y,int p){//已经getpos得到x,y之后,在y的左子树插入结点p(p为结
    点下标)
63     t[y].c[0]=p;t[p].f=y;
64     pushup(y);pushup(x);
65 }
66 int del(int l,int r){//摘下[l,r]区间的子树,返回子树的根结点
67     int x,y;
68     gettree(l,r,x,y);
69     int p=t[y].c[0];
70     t[y].c[0]=0;
71     t[p].f=0;
72     pushup(y);pushup(x);
73     return p;
74 }
75 int main(){
76     scanf("%d%d",&n,&q);
77     tot=0;memset(t,0,(n+5)*sizeof(Splay));
78     for(int i=1;i<=n;++i)scanf("%d",&a[i]);
79     a[0]=a[n+1]=-inf;
80     rt=build(0,n+1,0);
81 }

```

3.13 动态树

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int N=1e5+10;

```



```

4  int n,q;
5  struct LCT{
6      int c[2],f,rtag;
7      int val,sum;//val=点权,sum=链和
8      int sz,vsz;//sz=子树和,vsz=虚儿子和
9  }t[N];
10 #define notroot(x) (t[t[x].f].c[0]==x||t[t[x].f].c[1]==x)
11 void pushup(int x){
12     t[x].sum=t[t[x].c[0]].sum^t[t[x].c[1]].sum^t[x].val;
13     t[x].sz=t[t[x].c[0]].sz+t[t[x].c[1]].sz+1+t[x].vsz;
14 }
15 void rev1(int x){
16     t[x].rtag^=1; swap(t[x].c[0],t[x].c[1]);
17 }
18 void pushdown(int x){
19     if(t[x].rtag){
20         t[x].rtag=0;
21         if(t[x].c[0])rev1(t[x].c[0]);
22         if(t[x].c[1])rev1(t[x].c[1]);
23     }
24 }
25 void rotate(int x){
26     int y=t[x].f,z=t[y].f;
27     int c=t[y].c[1]==x;
28     if(notroot(y))t[z].c[t[z].c[1]==y]=x;t[x].f=z;
29     t[y].c[c]=t[x].c[c^1];t[t[x].c[c^1]].f=y;
30     t[x].c[c^1]=y;t[y].f=x;
31     pushup(y);pushup(x);
32 }
33 void splay(int x){
34     static int sta[N],top;//注意修改此处的N
35     int y=x; sta[top]=y;
36     while(notroot(y))sta[++top]=y=t[y].f;
37     while(top)pushdown(sta[top--]);
38     while(notroot(x)){
39         int y=t[x].f,z=t[y].f;
40         if(notroot(y))rotate(t[y].c[1]==x?t[z].c[1]==y?x:y);
41         rotate(x);
42     }
43 }
44 void access(int x){

```

```
45     for(int f=0;x;f=x,x=t[x].f){
46         splay(x);
47         t[x].vsz+=t[t[x].c[1]].sz-t[f].sz;
48         t[x].c[1]=f; pushup(x);
49     }
50 }
51 void makeroot(int x){
52     access(x);splay(x);rev1(x);
53 }
54 int findroot(int x){
55     access(x);splay(x);
56     while(t[x].c[0])pushdown(x),x=t[x].c[0];
57     splay(x);return x;
58 }
59 void split(int x,int y){
60     makeroot(x);access(y);splay(y);
61 }
62 void link(int x,int y){
63     if(findroot(x)==findroot(y))return;
64     split(x,y); t[x].f=y;
65     t[y].vsz+=t[x].sz;
66     pushup(y);
67 }
68 void cut(int x,int y){
69     if(findroot(x)!=findroot(y))return;
70     split(x,y);
71     if(t[y].c[0]==x&&!t[x].c[1]){
72         t[y].c[0]=t[x].f=0;pushup(y);
73     }
74 }
75 void upd(int x,int c){
76     splay(x);t[x].val=c;pushup(x);
77 }
78 int main(){
79     scanf("%d%d",&n,&q);
80     for(int i=1;i<=n;++i)scanf("%d",&t[i].val);
81     int op,x,y;
82     while(q--){
83         scanf("%d%d%d",&op,&x,&y);
84         if(op==0){
85             split(x,y);printf("%d\n",t[y].sum);
```

```

86         }
87         else if(op==1)link(x,y);
88         else if(op==2)cut(x,y);
89         else upd(x,y);
90     }
91 }

```

3.14 左偏树 (可并堆)

```

1  struct node{
2      int ls,rs,fa;//fa用来维护并查集
3      int val,dis;//val是关键字,dis是结点到最近不完整结点(至多一个儿子)的距离
4  }t[N];
5  int rt[N],tot;
6  int merge(int x,int y){//合并以x为根的堆,和以y为根的堆
7      if(!x||!y)return x+y;
8      if(t[x].val>t[y].val||t[x].val==t[y].val&&x>y)swap(x,y);
9      t[x].rs=merge(t[x].rs,y);
10     t[t[x].rs].fa=x;
11     if(t[t[x].ls].dis<t[t[x].rs].dis)swap(t[x].ls,t[x].rs);
12     t[x].dis=t[t[x].rs].dis+1;
13     return x;
14 }
15 int find(int x){//找到x结点所在堆的堆顶结点下标
16     return x==t[x].fa?x:t[x].fa=find(t[x].fa);
17 }
18 void pop(int x){//x为根的堆pop堆顶
19     t[x].val=-1;//val=-1表示被删除
20     t[t[x].ls].fa=t[x].ls;
21     t[t[x].rs].fa=t[x].rs;
22     t[x].fa=merge(t[x].ls,t[x].rs);
23 }
24 int main(){
25     int n,q;scanf("%d%d",&n,&q);
26     t[0].dis=-1;
27     for(int i=1;i<=n;++i)scanf("%d",&t[i].val),t[i].fa=i;
28     while(q--){
29         int op,x,y;scanf("%d%d",&op,&x);
30         if(op==1){//合并x和y结点所在的堆
31             scanf("%d",&y);
32             if(t[x].val== -1||t[y].val== -1)continue;

```

```

33         int fx=find(x),fy=find(y);
34         if(fx!=fy)t[fx].fa=t[fy].fa=merge(fx,fy);
35     }
36     else{//找到x结点所在的堆的堆顶并pop
37         if(t[x].val==-1)puts("-1");//x结点已被删除
38         else{
39             int fx=find(x);
40             printf("%d\n",t[fx].val);
41             pop(fx);
42         }
43     }
44 }
45 }

```

3.15 笛卡尔树

```

1 for(int i=1;i<=n;++i){
2     while(top&&a[sta[top]]>a[i])t[i].ls=sta[top--];
3     t[sta[top]].rs=i;
4     sta[++top]=i;
5 }//单调栈建二叉树,入度为0是根

```

4 图论

4.1 最小生成树

4.1.1 Kruskal

```

1 int n,m,fa[N],sum;
2 int find(int x){
3     return x==fa[x]?x:fa[x]=find(fa[x]);
4 }
5 struct node{
6     int x,y,z;
7 }b[M];
8 bool cmp(node a,node b){
9     return a.z<b.z;
10 }
11 int main(){
12     scanf("%d%d",&n,&m);
13     for(int i=1;i<=n;++i)fa[i]=i;

```

```

14     for(int i=1;i<=m;++i)scanf("%d%d%d",&b[i].x,&b[i].y,&b[i].z);
15     sort(b+1,b+1+m,cmp);
16     for(int i=1;i<=m;++i){
17         int x=b[i].x,y=b[i].y;
18         if(find(x)!=find(y)){
19             sum+=b[i].z;
20             fa[find(x)]=find(y);
21         }
22     }
23 }

```

4.1.2 Prim

```

1  int n,m;
2  int e[N][N],dis[N];//e[i][j]:边i->j的长度,dis[i]连通图到i的最短边
3  bool vis[N];
4  inline void prim(){
5      memset(dis,0x3f,n+1<<2);
6      memset(vis+1,0,n);
7      dis[1]=0;
8      for(int i=1;i<n;++i){
9          int x=0;
10         for(int j=1;j<=n;++j)if(!vis[j]&&dis[j]<dis[x])x=j;
11         vis[x]=1;
12         for(int j=1;j<=n;++j)if(!vis[j])dis[j]=min(dis[j],e[x][j]);
13     }
14 }
15 int main(){
16     scanf("%d%d",&n,&m);
17     memset(e,0x3f,sizeof e);
18     for(int i=1;i<=n;++i)e[i][i]=0;
19     for(int i=1;i<=m;++i){
20         int x,y,z;scanf("%d%d%d",&x,&y,&z);
21         e[y][x]=e[x][y]=min(e[x][y],z);
22     }
23     prim();int sum=0;
24     for(int i=1;i<=n;++i)sum+=dis[i];
25 }

```

4.2 最短路

4.2.1 Floyd

```

1  int n,m,d[N][N];
2  int main(){
3      scanf("%d%d",&n,&m);
4      memset(d,0x3f,sizeof d);
5      for(int i=1;i<=n;++i)d[i][i]=0;
6      for(int i=1;i<=m;++i){
7          int x,y,z;scanf("%d%d%d",&x,&y,&z);
8          d[x][y]=min(d[x][y],z);
9          d[y][x]=min(d[y][x],z);
10     }
11     for(int k=1;k<=n;++k)//先枚举中转点
12         for(int i=1;i<=n;++i)
13             for(int j=1;j<=n;++j)
14                 d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
15 }
```

4.2.2 Floyd 求最小环

<http://acm.hdu.edu.cn/showproblem.php?pid=1599>

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=105,inf=1e8;
4  int n,m,a[N][N],dis[N][N];
5  int main(){
6      while(scanf("%d%d",&n,&m)!=EOF){
7          for(int i=1;i<=n;++i){
8              for(int j=1;j<=n;++j){
9                  dis[i][j]=a[i][j]=inf;
10             }
11         }
12         for(int i=1;i<=n;++i)dis[i][i]=a[i][i]=0;
13         for(int i=1;i<=m;++i){
14             int x,y,z;scanf("%d%d%d",&x,&y,&z);
15             a[x][y]=a[y][x]=min(a[x][y],z);
16             dis[x][y]=dis[y][x]=min(dis[x][y],z);
17         }
18         int ans=inf;
19         for(int k=1;k<=n;++k){
```

```

20     for(int i=1;i<=n;++i){
21         for(int j=1;j<=n;++j){
22             if(i!=j&&j!=k&&i!=k){
23                 ans=min(ans,dis[i][j]+a[j][k]+a[k][i]);
24             }
25         }
26     }
27     for(int i=1;i<=n;++i){
28         for(int j=1;j<=n;++j){
29             dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
30         }
31     }
32 }
33 if(ans==inf)puts("It's impossible.");//没有环
34 else printf("%d\n",ans);
35 }
36 }

```

4.2.3 Dijkstra

```

1  priority_queue<pair<ll,int> >q;
2  ll d[N];bool vis[N];
3  void dij(int s){
4      memset(d,0x3f,sizeof d);
5      memset(vis,0,sizeof vis);
6      d[s]=0;q.push(make_pair(0,s));
7      for(;!q.empty();){
8          int x=q.top().second;
9          q.pop();
10         if(vis[x])continue;
11         vis[x]=1;
12         for(int i=hd[x];i;i=e[i].nx){
13             int y=e[i].to;
14             if(d[y]>d[x]+e[i].v){
15                 d[y]=d[x]+e[i].v;
16                 q.push(make_pair(-d[y],y));
17             }
18         }
19     }
20 }

```

4.2.4 SPFA

按点判断负环

```

1  bool spfa(int s){
2      memset(dis,0x3f,sizeof dis);
3      //push_time和in_queue要清空
4      dis[s]=0;in_queue[s]=1;
5      queue<int>q({s});
6      while(q.size()){
7          int x=q.front();q.pop();
8          in_queue[x]=0;
9          for(int i=hd[x];i;i=e[i].nx){
10             int y=e[i].to;
11             if(dis[x]+e[i].v<dis[y]){
12                 dis[y]=dis[x]+e[i].v;
13                 if(!in_queue[y]){
14                     if(push_time[y]>n)return 1;//有负环
15                     q.push(y);in_queue[y]=1;++push_time[y];
16                     //if(dis[q.back()]<dis[q.front()])swap(q.back(),q.front());
17                 }
18             }
19         }
20     }
21     return 0;//没有负环
22 }

```

按边判断负环

```

1  bool in_queue[N];int edge_cnt[N];
2  bool spfa(int s){
3      memset(dis,0x3f,sizeof dis);
4      dis[s]=0;in_queue[s]=1;
5      queue<int>q({s});
6      while(q.size()){
7          int x=q.front();q.pop();
8          in_queue[x]=0;
9          for(int i=hd[x];i;i=e[i].nx){
10             int y=e[i].to;
11             if(dis[x]+e[i].v<dis[y]){
12                 dis[y]=dis[x]+e[i].v;
13                 edge_cnt[y]=edge_cnt[x]+1;
14                 if(edge_cnt[y]>=n+1)return 1;//有负环
15                 if(!in_queue[y]){

```



```

16         q.push(y);
17         in_queue[y]=1;
18     }
19 }
20 }
21 }
22 return 0;//没有负环
23 }

```

4.3 二分图最大匹配

```

1 //二分图两个部分的点编号分别为1到n和1到m,边(i,j)表示左边的i到右边的j
2 vector<int>e[N];
3 int n,m,t,match[N],v[N],ans;
4 bool dfs(int x,int t){
5     if(v[x]==t)return 0;
6     v[x]=t;
7     for(auto y:e[x]){
8         if(!match[y]||dfs(match[y],t)){
9             match[y]=x;
10            return 1;
11        }
12    }
13    return 0;
14 }
15 int main(){
16     scanf("%d%d%d",&n,&m,&t);
17     for(int i=1;i<=t;++i){
18         int x,y;scanf("%d%d",&x,&y);
19         e[x].push_back(y);
20     }
21     for(int i=1;i<=n;++i)if(dfs(i,i))++ans;
22     printf("%d\n",ans);
23 }

```

最小路径覆盖 = 最大独立集 = 总节点数 - 二分图最大匹配数

最小点覆盖 = 二分图最大匹配数

最大独立集 + 最小点覆盖集 = V , 最大团 = 补图的最大独立集。

4.4 拓扑排序

```

1  int n,in[N],rk[N];
2  vector<int>e[N];
3  int main(){
4      queue<int>q;
5      for(int i=1;i<=n;++i)if(!in[i])q.push(i);
6      while(q.size()){
7          int x=q.front();q.pop();
8          for(auto y:e[x]){
9              rk[y]=max(rk[y],rk[x]+1);//注意取max
10             if(!--in[y])q.push(y);
11         }
12     }
13 }

```

4.5 Tarjan

4.5.1 有向图 scc

```

1  struct edge{
2      int to,nx;
3  }e[M],e1[M];
4  int hd[N],hd1[N],tot,tot1;
5  void add(int x,int y){
6      e[++tot].to=y;e[tot].nx=hd[x];hd[x]=tot;
7  }
8  void add1(int x,int y){
9      e1[++tot1].to=y;e1[tot1].nx=hd1[x];hd1[x]=tot1;
10 }
11 int n,m,a[N];//a[i]点权
12 int dfn[N],low[N],sta[N];
13 bool vis[N];//在栈内
14 int idx[N],tscc;//i号点所在的scc的下标
15 int cnt[N];//SCC中点的个数
16 ll sum[N];//SCC点权和
17 void tarjan(int x){
18     low[x]=dfn[x]=++dfn[0];
19     vis[x]=1;sta[++sta[0]]=x;
20     for(int i=hd[x];i;i=e[i].nx){
21         int y=e[i].to;
22         if(!dfn[y]){
23             tarjan(y);

```

```
24         low[x]=min(low[x],low[y]);
25     }
26     else if(vis[y]){
27         low[x]=min(low[x],dfn[y]);
28     }
29 }
30 if(low[x]==dfn[x]){
31     int y; ++tscc;
32     do{
33         y=sta[sta[0]--]; vis[y]=0;
34         // 接下来添加SCC信息
35         ++cnt[tscc]; sum[tscc] += a[y];
36         idx[y]=tscc;
37     }while(y!=x);
38 }
39 }
40 void init(){
41     memset(vis+1,0,n);
42     memset(hd+1,0,n<<2);
43     memset(hd1+1,0,n<<2);
44     memset(low+1,0,n<<2);
45     memset(dfn+1,0,n<<2);
46     tot=tot1=dfn[0]=sta[0]=tscc=0;
47     // 接下来添加初始化
48     memset(cnt+1,0,n<<2);
49     memset(sum+1,0,n<<3);
50 }
51 void getscc(){
52     for(int i=1;i<=n;++i)if(!dfn[i])tarjan(i);
53     for(int x=1;x<=n;++x){
54         for(int i=hd[x];i;i=e[i].nx){
55             int y=e[i].to;
56             if(idx[x]!=idx[y])add1(idx[x],idx[y]); // 不属于同一SCC就连边
57         }
58     }
59 }
60 int main(){
61     n=rd();m=rd();
62     init();// 初始化要在输入n之后
63     for(int i=1;i<=m;++i){
64         int x=rd(),y=rd(); add(x,y);
```

```

65     }
66     getscc();
67     //缩点后的图用hd1[],e1[]表示
68 }

```

4.5.2 无向图 edcc(割边)

```

1  struct edge{
2      int to,nx;
3  }e[M<<1],e1[M<<1];
4  int n,m,hd[N],tot,hd1[N],tot1;
5  void add(int x,int y){
6      e[++tot].to=y;e[tot].nx=hd[x];hd[x]=tot;
7  }
8  void add1(int x,int y){//缩点后的图
9      e1[++tot1].to=y;e1[tot1].nx=hd1[x];hd1[x]=tot1;
10 }
11 int a[N];//点权
12 int low[N],dfn[N];
13 int idx[N],tedcc,cnt[N];//idx[x]=x所属edcc,cnt[i]=第i个edcc里有几个点
14 ll sum[N];//sum[i]=第i个edcc里的点权之和
15 bool bridge[M<<1];//割边
16 void tarjan(int x,int eg){//eg=边下标
17     low[x]=dfn[x]=++dfn[0];
18     for(int i=hd[x];i;i=e[i].nx){
19         int y=e[i].to;
20         if(!dfn[y]){
21             tarjan(y,i);
22             low[x]=min(low[x],low[y]);
23             if(dfn[x]<low[y])bridge[i]=bridge[i^1]=1;//是割边
24         }
25         else if(i!=(eg^1))low[x]=min(low[x],dfn[y]);//(eg^1)一定要加括号
26     }
27 }
28 void getedcc(int x){//缩点
29     //接下来添加edcc信息
30     idx[x]=tedcc;sum[tedcc]+=a[x];++cnt[tedcc];
31     for(int i=hd[x];i;i=e[i].nx){
32         int y=e[i].to;
33         if(idx[y]||bridge[i])continue;//走过||是割边的话不走
34         getedcc(y);

```

```

35     }
36 }
37 void getgraph(){//求出缩点后的图
38     for(int i=1;i<=n;++i)if(!dfn[i])tarjan(i,0);
39     for(int i=1;i<=n;++i)if(!idx[i])++tedcc,getedcc(i);
40     for(int i=2;i<=tot;i+=2){//枚举所有边
41         int x=e[i].to,y=e[i^1].to;
42         if(idx[x]!=idx[y]){//所在edcc不同
43             add1(idx[x],idx[y]);
44             add1(idx[y],idx[x]);
45         }
46     }
47 }
48 void init(){
49     tot=tot1=1;tedcc=dfn[0]=0;
50     memset(hd+1,0,n<<2);
51     memset(hd1+1,0,n<<2);
52     memset(low+1,0,n<<2);
53     memset(dfn+1,0,n<<2);
54     memset(bridge+1,0,m<<1);
55     memset(idx+1,0,n<<2);
56     //接下来添加初始化
57     memset(cnt+1,0,n<<2);
58     memset(sum+1,0,n<<3);
59 }
60 int main(){
61     n=rd();m=rd();
62     init();//初始化要在输入n之后
63     for(int i=1;i<=n;++i)a[i]=rd();
64     for(int i=1;i<=m;++i){
65         int x=rd(),y=rd();
66         if(x==y)continue;
67         add(x,y);add(y,x);
68     }
69     getgraph();
70     //缩点后的图用hd1[],e1[]表示
71 }

```

4.5.3 无向图 vdcc(割点)

```

1  const int N=1e5+10,N1=N<<1;

```

```

2  struct edge{
3      int to,nx;
4  }e[N<<1],e1[N1<<1];
5  int n,m,a[N],hd[N],tot,hd1[N<<1],tot1;
6  void add(int x,int y){
7      e[++tot].to=y;e[tot].nx=hd[x];hd[x]=tot;
8  }
9  void add1(int x,int y){
10     e1[++tot1].to=y;e1[tot1].nx=hd1[x];hd1[x]=tot1;
11 }
12 int rt,low[N],dfn[N],sta[N],tvdcc;
13 vector<int>vdcc[N];
14 bool cut[N];//是割点
15 void tarjan(int x){
16     low[x]=dfn[x]=++dfn[0];
17     sta[++sta[0]]=x;
18     int flag=0;
19     for(int i=hd[x];i;i=e[i].nx){
20         int y=e[i].to;
21         if(!dfn[y]){
22             tarjan(y);
23             low[x]=min(low[x],low[y]);
24             if(dfn[x]<=low[y]){
25                 ++flag;
26                 if(x!=rt||flag>1)cut[x]=1;
27                 vdcc[++tvdcc].push_back(x);
28                 int z;
29                 do{
30                     z=sta[sta[0]--];
31                     vdcc[tvdcc].push_back(z);
32                 }while(z!=y);
33             }
34         }
35         else low[x]=min(low[x],dfn[y]);
36     }
37 }
38 void get_round_square_tree(){
39     for(int i=1;i<=n;++i)if(!dfn[i])rt=i,tarjan(i);
40     for(int i=1;i<=tvdcc;++i){
41         for(auto x:vdcc[i]){
42             add1(i+n,x);add1(x,i+n);

```

```

43         //添加方点信息
44     }
45 }
46 }
47 void init(){
48     memset(dfn+1,0,n<<2);
49     memset(low+1,0,n<<2);
50     memset(hd+1,0,n<<2);
51     memset(hd1+1,0,n<<3);
52     dfn[0]=sta[0]=tvdcc=0;tot=tot1=1;
53     for(int i=1;i<=n;++i)vdcc[i].clear();
54     //接下来添加初始化
55     memset(cut+1,0,n);
56 }
57 int main(){
58     n=rd();m=rd();
59     init();//初始化要在读入n之后
60     for(int i=1;i<=m;++i){
61         int x=rd(),y=rd();
62         if(x==y)continue;
63         add(x,y);add(y,x);
64     }
65     get_round_square_tree();
66     //1~n是圆点下标,n+1 ~ n+tvdcc是方点下标,圆方树存在hd1[],e1[]
67     //注意圆方树节点数最多为2N
68 }

```

4.5.4 仙人掌建圆方树

```

1  const int N=1e4+5,N1=N<<1,M=2e5+5;
2  struct edge{
3      int to,nx,v;
4  }e[M<<1],e1[N1<<1];
5  int hd[N],tot,hd1[N1],tot1;
6  void add(int x,int y,int z){
7      e[++tot].to=y;e[tot].v=z;
8      e[tot].nx=hd[x];hd[x]=tot;
9  }
10 void add1(int x,int y,int z){
11     e1[++tot1].to=y;e1[tot1].v=z;
12     e1[tot1].nx=hd1[x];hd1[x]=tot1;

```

```

13 }
14 int n,m,q,vercnt;
15 int low[N],dfn[N],tdfn,fa[N],b[N],sum[N];
16 void tarjan(int x,int f){
17     low[x]=dfn[x]=++tdfn;
18     for(int i=hd[x];i;i=e[i].nx){
19         int y=e[i].to;
20         if(y==f)continue;
21         int z=e[i].v;
22         if(!dfn[y]){
23             fa[y]=x;b[y]=z;
24             tarjan(y,x);
25             low[x]=min(low[x],low[y]);
26         }
27         else
28             low[x]=min(low[x],dfn[y]);
29         if(dfn[x]>=low[y])continue;
30         add1(x,y,z);add1(y,x,z);
31     }
32     for(int i=hd[x];i;i=e[i].nx){
33         int y=e[i].to;
34         if(fa[y]==x||dfn[x]>=dfn[y])continue;
35         int s=e[i].v;
36         for(int u=y;u!=fa[x];u=fa[u]){
37             sum[u]=s;s+=b[u];
38         }
39         sum[++vercnt]=sum[x];sum[x]=0;
40         for(int u=y;u!=fa[x];u=fa[u]){
41             int w=min(sum[u],sum[vercnt]-sum[u]);
42             add1(vercnt,u,w);add1(u,vercnt,w);
43         }
44     }
45 }
46 int dis[N1],son[N1],d[N1],top[N1],dfn1[N1],sz[N1],ff[N1];
47 void dfs1(int x,int f){
48     sz[x]=1;
49     for(int i=hd1[x];i;i=e1[i].nx){
50         int y=e1[i].to;
51         if(y==f)continue;
52         ff[y]=x;
53         d[y]=d[x]+1;

```



```

54         dis[y]=dis[x]+e1[i].v;
55         dfs1(y,x);
56         sz[x]+=sz[y];
57         if(sz[y]>sz[son[x]])son[x]=y;
58     }
59 }
60 void dfs2(int x,int t){
61     dfn1[x]=++dfn1[0];top[x]=t;
62     if(!son[x])return;
63     dfs2(son[x],t);
64     for(int i=hd1[x];i;i=e1[i].nx){
65         int y=e1[i].to;
66         if(!dfn1[y])dfs2(y,y);
67     }
68 }
69 int lca(int x,int y){
70     while(top[x]!=top[y]){
71         if(d[top[x]]<d[top[y]])swap(x,y);
72         x=ff[top[x]];
73     }
74     return d[x]<d[y]?x:y;
75 }
76 int find(int x,int f){//方点lca下的圆点
77     int s;
78     while(top[x]!=top[f]){
79         s=top[x];x=ff[top[x]];
80     }
81     return x==f?s:son[f];
82 }
83 int main(){
84     scanf("%d%d%d",&n,&m,&q);
85     vercnt=n;
86     for(int i=1;i<=m;++i){
87         int x,y,z;
88         scanf("%d%d%d",&x,&y,&z);
89         add(x,y,z);add(y,x,z);
90     }
91     tarjan(1,0);
92     dfs1(1,0);dfs2(1,1);
93     for(;q--){
94         int x,y;

```

```

95     scanf("%d%d",&x,&y);
96     int f=lca(x,y);
97     int ans;
98     if(f<=n)ans=dis[x]+dis[y]-2*dis[f];
99     else{
100         int u=find(x,f),v=find(y,f);
101         ans=dis[x]+dis[y]-dis[u]-dis[v];
102         if(sum[u]<sum[v])swap(sum[u],sum[v]);
103         ans+=min(sum[u]-sum[v],sum[f]-(sum[u]-sum[v]));
104     }
105     printf("%d\n",ans);
106 }
107 }
    
```

4.6 差分约束

$$\begin{cases} x_{a_1} - x_{b_1} \leq dis_1 \\ x_{a_2} - x_{b_2} \leq dis_2 \\ \vdots \\ x_{a_n} - x_{b_n} \leq dis_n \end{cases}$$

$x_i - x_j \leq dis$ 相当于一条 $x_j \rightarrow x_i$ 距离为 dis 的边

$a \rightarrow b$ 的最短路即为 $x_b - x_a$ 的最大值

要求一组解，可以用超级起点连接所有点 (长为 0, 用于处理部分点相互独立的情况)，以超级起点为源点跑 SPFA, $x_i = dis[i]$

若存在负环, 则意义为 $x_i - x_i \leq -5$, 即无解

下面的程序用于求出任意一个满足所有不等式的解

```

1  int n,m,hd[N],tot;
2  struct edge{
3      int to,nx,v;
4  }e[N<<1];
5  inline void add(int x,int y,int z){
6      e[++tot]={y,hd[x],z};hd[x]=tot;
7  }
8  bool inq[N];//在队列内
9  int dis[N],push_time[N];
10 bool spfa(int s){
11     queue<int>q({s});
12     memset(dis,0x3f,sizeof dis);
    
```

```

13     dis[s]=0;inq[s]=1;
14     while(q.size()){
15         int x=q.front();q.pop();
16         inq[x]=0;
17         for(int i=hd[x];i;i=e[i].nx){
18             int y=e[i].to;
19             if(dis[x]+e[i].v<dis[y]){
20                 dis[y]=dis[x]+e[i].v;
21                 if(!inq[y]){
22                     inq[y]=1;
23                     ++push_time[y];
24                     if(push_time[y]>n)return 1; //有负环,无解
25                     q.push(y);
26                 }
27             }
28         }
29     }
30     return 0;//这里不要漏写
31 }
32 int main(){
33     scanf("%d%d",&n,&m);
34     for(int i=1;i<=n;++i)add(0,i,0);//超级源点,连通整张图
35     for(int i=1;i<=m;++i){
36         int x,y,z;scanf("%d%d%d",&x,&y,&z);
37         add(y,x,z);
38     }
39     if(spfa(0))puts("NO");//有负环
40     else for(int i=1;i<=n;++i)printf("%d ",dis[i]);
41 }

```

4.7 2-SAT

有 n 个布尔变量 x_1, x_2, \dots, x_n , 有多组限制条件 $x_i \rightarrow x_j$, 是否存在满足所有条件的解

每个变量在图上被抽象为 2 个点, i 点表示 $x_i = 1$, $i + n$ 点表示 $x_i = 0$

对于限制条件 $x_i \rightarrow x_j$, 你需要加一条 $i \rightarrow j$ 的边, 类似地, $x_i \rightarrow \neg x_j$ 需要加一条 $i \rightarrow j + n$ 的边

对于限制条件 $x_i \vee x_j$, 根据等价式 $\neg p \vee q \iff p \rightarrow q$, 可以转化为 $i + n \rightarrow j$ 和 $j + n \rightarrow i$

对于限制条件 $x_i = 1$, 相当于 $\neg x_i \rightarrow x_i$, 需要添加一条 $i + n \rightarrow i$ 的边

设 $idx[i] = i$ 所在强连通分量的编号, 若 $\forall i : idx[i] \neq idx[i + n]$, 则有解, 否则无解

因为 $idx[i] = idx[i + n]$ 意义是 $x_i = 1$ 能推出 $x_i = 0$, 且 $x_i = 0$ 能推出 $x_i = 1$, 即 x_i 无解

Tarjan 判断有解/求一种方案

```

1  int n,m,dfn[N],low[N],sta[N],idx[N],tscc;
2  bool vis[N]; //在栈内
3  void tarjan(int x){
4      low[x]=dfn[x]=++dfn[0];
5      vis[x]=1; sta[++sta[0]]=x;
6      for(int i=hd[x]; i; i=e[i].nx){
7          int y=e[i].to;
8          if(!dfn[y]){
9              tarjan(y);
10             low[x]=min(low[x],low[y]);
11         }
12         else if(vis[y]){
13             low[x]=min(low[x],dfn[y]);
14         }
15     }
16     if(low[x]==dfn[x]){
17         int y; ++tscc;
18         do{
19             y=sta[sta[0]--]; vis[y]=0;
20             idx[y]=tscc;
21         }while(y!=x);
22     }
23 }
24 int main(){
25     scanf("%d%d",&n,&m);
26     for(; m--; ){ //对于约束条件x or y, 需要加边!x->y和!y->x
27         int x,y,a,b; //x=a or y=b (a,b=0或1)
28         scanf("%d%d%d%d",&x,&a,&y,&b);
29         if(a&&b) add(x+n,y), add(y+n,x);
30         else if(a&&!b) add(y,x), add(x+n,y+n);
31         else if(!a&&b) add(x,y), add(y+n,x+n);
32         else add(x,y+n), add(y,x+n);
33     } //x号结点x=1, x+n表示x=0
34     for(int i=1; i<=n*2; ++i) if(!dfn[i]) tarjan(i); //注意n*2
35     for(int i=1; i<=n; ++i){ //判断不合法
36         if(idx[i]==idx[i+n]){
37             puts("IMPOSSIBLE");
38             return 0;
39         }
40     }

```

```

41     puts("POSSIBLE");
42     for(int i=1;i<=n;++i){//输出一种方案
43         int x=0;
44         if(idx[i]<idx[i+n])x=1;//用Tarjan算法需要选择scc编号小的
45         printf("%d ",x);
46     }
47 }

```

求出最小字典序

```

1  const int N=8005<<1,M=20005<<1;
2  struct edge{
3      int to,nx;
4  }e[M];
5  int n,m,hd[N],tot;
6  void add(int x,int y){
7      e[++tot]={y,hd[x]};hd[x]=tot;
8  }
9  int sta[N],top,choose[N];//choose=1:选, 2:不选, 0:没访问过
10 bool dfs(int x){
11     if(choose[x]==1)return 1;
12     if(choose[x]==2)return 0;
13     choose[x]=1;choose[x^1]=2;
14     sta[++top]=x;
15     for(int i=hd[x];i;i=e[i].nx){
16         int y=e[i].to;
17         if(!dfs(y))return 0;
18     }
19     return 1;
20 }
21 bool twosat(){
22     memset(choose,0,n<<3);//2n个点清空
23     for(int i=0;i<n<<1;++i){
24         if(choose[i])continue;
25         top=0;
26         if(!dfs(i)){
27             for(int j=1;j<=top;++j)choose[sta[j]]=choose[sta[j]^1]=0;
28             if(!dfs(i^1))return 0;
29         }
30     }
31     return 1;
32 }

```

```

33 int main(){
34     while(scanf("%d%d",&n,&m)!=EOF){
35         memset(hd,0,n<<3);tot=0;
36         for(int i=1;i<=m;++i){
37             int x,y;scanf("%d%d",&x,&y);
38             --x;--y;
39             add(x,y^1);add(y,x^1);
40         }
41         if(!twosat())puts("NIE");
42         else{
43             for(int i=0;i<n<<1;++i)
44                 if(choose[i]==1)printf("%d\n",i+1);
45         }
46     }
47 }

```

4.8 Dinic 最大流

```

1  const int N=5e4+10,M=1e5+10;
2  const ll inf=1ll<<60;
3  struct edge{
4      int to,nx;ll v;
5  }e[M<<1];//注意有反边,边数要乘以2
6  int n,m,s,t,vercnt,d[N],now[N],hd[N],tot=1;
7  ll maxflow; queue<int>q;
8  void add(int x,int y,ll z){//一次加好正向和反向的边
9      e[++tot]={y,hd[x],z};hd[x]=tot;
10     e[++tot]={x,hd[y],0};hd[y]=tot;
11 }
12 bool bfs(){
13     for(;!q.empty();q.pop());
14     q.push(s);
15     memset(d,0,vercnt+1<<2);
16     d[s]=1; now[s]=hd[s];
17     while(q.size()){
18         int x=q.front();
19         q.pop();
20         for(int i=hd[x];i;i=e[i].nx){
21             int y=e[i].to;
22             if(e[i].v&&!d[y]){
23                 q.push(y);

```

```

24         now[y]=hd[y];
25         d[y]=d[x]+1;
26         if(y==t)return 1;
27     }
28 }
29 }
30 return 0;
31 }
32 ll dinic(int x,ll flow){
33     if(x==t)return flow;
34     int i;
35     ll k,rest=flow;
36     for(i=now[x];i&&rest;i=e[i].nx){
37         int y=e[i].to;
38         if(e[i].v&&d[y]==d[x]+1){
39             k=dinic(y,min(rest,e[i].v));
40             if(!k)d[y]=0;
41             e[i].v-=k;
42             e[i^1].v+=k;
43             rest-=k;
44         }
45     }
46     now[x]=i;
47     return flow-rest;
48 }
49 void init(){
50     tot=1;maxflow=0;
51     memset(hd,0,vercnt+1<<2);
52 }
53 int main(){
54     scanf("%d%d",&n,&m);
55     s=0;//源点建议为0
56     t=n+1;//汇点编号根据实际情况调整
57     vercnt=t+1;//点的个数 要求所有点编号在[0,vercnt]范围
58     init();//初始化tot=1
59     //开始加边
60
61     //加边结束
62     ll flow=0;
63     while(bfs()){
64         while(flow=dinic(s,inf)){

```

```

65         maxflow+=flow;
66     }
67 }
68 printf("%lld\n",maxflow);
69 return 0;
70 }

```

4.9 Edmonds Karp 最小费用最大流

```

1  const int N=5e4+10,M=1e5+10;
2  const ll inf=1ll<<60;
3  struct edge{
4      int to,nx;ll f,c;
5  }e[M<<1];//注意有反边,边数要乘以2
6  int n,m,s,t,hd[N],pre[N],tot=1,vercnt;
7  ll cost[N],flow[N],maxflow,mincost;
8  bool vis[N]; queue<int>q;
9  void add(int x,int y,ll f,ll c){//一次加好正向和反向的边
10     e[++tot].to=y;e[tot].f=f;e[tot].c=c;
11     e[tot].nx=hd[x];hd[x]=tot;
12     e[++tot].to=x;e[tot].f=0;e[tot].c=-c;
13     e[tot].nx=hd[y];hd[y]=tot;
14 }
15 bool spfa(){
16     for(;!q.empty();)q.pop();
17     q.push(s);
18     for(int i=0;i<=vercnt+1;++i)cost[i]=inf;
19     memset(vis,0,vercnt+1);
20     cost[s]=0;vis[s]=1;
21     flow[s]=inf;
22     while(q.size()){
23         int x=q.front();q.pop();
24         vis[x]=0;
25         for(int i=hd[x];i;i=e[i].nx){
26             int y=e[i].to;
27             if(!e[i].f)continue;
28             if(cost[x]+e[i].c<cost[y]){
29                 cost[y]=cost[x]+e[i].c;
30                 flow[y]=min(flow[x],e[i].f);
31                 pre[y]=i;
32                 if(!vis[y]){

```



```

33         vis[y]=1;
34         q.push(y);
35     }
36 }
37 }
38 }
39 return cost[t]!=inf;
40 }
41 void mincostmaxflow(){
42     while(spfa()){
43         int x=t;
44         maxflow+=flow[t];
45         mincost+=cost[t]*flow[t];
46         while(x!=s){
47             int i=pre[x];
48             e[i].f-=flow[t];
49             e[i^1].f+=flow[t];
50             x=e[i^1].to;
51         }
52     }
53 }
54 void init(){
55     tot=1;maxflow=mincost=0;
56     memset(hd,0,vercnt+1<<2);
57 }
58 int main(){
59     scanf("%d%d",&n,&m);
60     s=0;//源点必须为0
61     t=n+1;//汇点编号根据实际情况调整
62     vercnt=t+1;//点的个数,要求所有结点编号为[0,vercnt]
63     init();//初始化tot=1
64     //开始加边
65
66     //加边结束
67     mincostmaxflow();
68     printf("%lld %lld\n",maxflow,mincost);
69 }

```

4.10 Kruskal 重构树

```

1 int main(){

```

```

2   vercnt=n;//1-n是原来的点,n+1~2n-1是新建的点,边抽象为点
3   sort(b+1,b+1+m,cmp);//边按边权排序
4   for(int i=1;i<=m;++i){
5       int x=b[i].u,y=b[i].v;
6       int x1=find(x),y1=find(y);
7       if(x1!=y1){
8           ++vercnt;
9           h[vercnt]=b[i].h;//边的信息存在新建的点上
10          add(vercnt,x1); add(vercnt,y1);
11          fa[x1]=vercnt; fa[y1]=vercnt;
12          dis[vercnt]=min(dis[x1],dis[y1]);
13      }
14  }
15 }

```

4.11 二分图最大权匹配 KM 算法

```

1  typedef long long ll;
2  const int N=505;
3  const ll inf=0x3f3f3f3f3f3f3f3f;
4  int match[N];//右边的i匹配左边的match[i]
5  ll e[N][N];//边,e[i][j]=左边的i匹配右边的j的边权
6  ll dbx[N],dby[N],slack[N];
7  int pre[N]; bool vis[N];
8  int n,m;
9  void bfs(int k) {
10     int px,py=0,yy=0;ll d;
11     memset(pre,0,sizeof pre);
12     memset(slack,0x3f,sizeof slack);
13     match[py]=k;
14     do {
15         px=match[py],d=inf,vis[py]=1;
16         for(int i=1; i<=n; i++)
17             if(vis[i]==0) {
18                 if(slack[i]>dbx[px]+dby[i]-e[px][i])
19                     slack[i]=dbx[px]+dby[i]-e[px][i],pre[i]=py;
20                 if(slack[i]<d) d=slack[i],yy=i;
21             }
22         for(int i=0; i<=n; i++)
23             if(vis[i]) dbx[match[i]]-=d,dby[i]+=d;
24             else slack[i]-=d;

```

```

25     py=yy;
26 } while(match[py]!=0);
27 while(py) match[py]=match[pre[py]],py=pre[py];
28 }
29 void km() { //二分图最大权匹配算法
30     memset(dbx,0,sizeof dbx);
31     memset(dby,0,sizeof dby);
32     memset(match,0,sizeof match);
33     for(int i=1; i<=n; i++)
34         memset(vis,0,sizeof vis),bfs(i);
35 }
36 int main() {
37     memset(e,0xcf,sizeof e);//初始化为-inf
38     scanf("%d%d",&n,&m);
39     for(int i=1; i<=m; ++i) {
40         int x,y,z; scanf("%d%d%d",&x,&y,&z);
41         e[x][y]=z;//左边的x匹配右边的y,价值为z
42     }
43     km();    ll ans=0;//最大权匹配sum
44     for(int i=1; i<=n; ++i)ans+=e[match[i]][i];
45     printf("%lld\n",ans);
46     for(int i=1; i<=n; ++i)printf("%d ",match[i]);
47 }

```

4.12 矩阵树定理

一、无向无环图

A 为邻接矩阵, $A[i][j] = i \rightarrow j$ 的边数

D 为度数矩阵, $D[i][i] = \sum_{j=1}^n A[i][j] = i$ 的度数, 其他位置为 0

基尔霍夫矩阵 $K = D - A$, 令 $K' = K$ 的去掉第 k 行第 k 列 (k 任意) 的 $n-1$ 阶主子式

$\det(K') =$ 该无向图生成树个数

特别地, 完全图生成树个数是 n^{n-2}

二、加权

求所有生成树边权的乘积之和, 需要把邻接矩阵中边的条数改为边权和

度数矩阵改为 $D[i][i] = \sum_{j=1}^n A[i][j]$

三、有向图

对于有根外向树，需要把度数矩阵改为入度和， $D[i][i] = \sum_{j=1}^n A[j][i]$

对于有根内向树，需要把度数矩阵改为出度和， $D[i][i] = \sum_{j=1}^n A[i][j]$

类似地，求所有有向生成树边权的乘积之和，需要把邻接矩阵改为入/出边边权和

四、变形

求所有生成树边权和的和，给原先边权为 w 的边赋值为一次多项式 $wx + 1$ ，多项式乘法对 x^2 取模， $\prod(w_i x + 1)$ 的一次项系数即为 w_i 之和

```

1 struct P {
2     ll x,y;//x是一次项系数，y是常数项
3     P (ll x=0,ll y=0):x(x),y(y){}
4     friend P operator + (const P &u, const P &v) {
5         return P(add(u.x, v.x), add(u.y, v.y));
6     }
7     friend P operator - (const P &u, const P &v) {
8         return P(add(u.x, mod - v.x), add(u.y, mod - v.y));
9     }
10    friend P operator * (const P &u, const P &v) {
11        return P(add(mul(u.x, v.y), mul(u.y, v.x)), mul(u.y, v.y));
12    }
13    friend P operator / (const P &u, const P &v) {
14        ll inv=qpow(v.y, mod-2);
15        return P(add(mul(u.x, v.y), mod - mul(u.y, v.x)) * inv % mod * inv % mod,
16            mul(u.y, inv));
17    }
18 };
19 P g[N][N];
20 ll gauss(P g[N][N],int n){
21     P res(0,1);
22     for(int i=1;i<=n;++i) {
23         int pos=-1;
24         for(int j=i;j<=n;++j){
25             if(g[j][i].y){
26                 pos=j;break;
27             }
28         }
29         if(pos==-1)return 0;
30         swap(g[i],g[pos]);
31         if(pos!=i)res=res*P(0,mod-1);
32         res=res*g[i][i];
33         P inv=P(0,1)/g[i][i];

```

```

33     for(int j=i+1;j<=n;++j){
34         P t=g[j][i]*inv;
35         for(int k=n;k>=i;--k)g[j][k]=g[j][k]-t*g[i][k];
36     }
37 }
38 return res.x;
39 }

```

4.13 LGV 引理

G 是一个有限的带权有向无环图。每个顶点的度是有限的，不存在有向环。

起点 $A = \{a_1, a_2, \dots, a_n\}$ ，终点 $B = \{b_1, b_2, \dots, b_n\}$ ，每条边 e 有边权 w_e

对于一个有向路径 P ，定义 $\omega(P)$ 为路径上所有边权的积。

对任意顶点 (a, b) ，定义 $e(a, b) = \sum_{P: a \rightarrow b} \omega(P)$ = 所有 $a \rightarrow b$ 的路径边权乘积之和

$$M = \begin{bmatrix} e(a_1, b_1) & e(a_1, b_2) & \cdots & e(a_1, b_n) \\ e(a_2, b_1) & e(a_2, b_2) & \cdots & e(a_2, b_n) \\ \vdots & \vdots & \ddots & \vdots \\ e(a_n, b_1) & e(a_n, b_2) & \cdots & e(a_n, b_n) \end{bmatrix}$$

一个排列 σ 对应 $a_i \rightarrow b_{\sigma(i)}$ 的一种路径方案

$inv(\sigma)$ = 排列 σ 的逆序对， $a_i \rightarrow b_{\sigma(i)}$ 的路径交点数至少为 $inv(\sigma)$

设 P_i 表示从 a_i 到 $b_{\sigma(i)}$ 的一条路径， $det(M) = \sum_{P: a \rightarrow b} (-1)^{inv(\sigma(P))} \prod_{i=1}^n \omega(P_i)$

即路径交点数至少为 0 个的边权乘积和，减去路径焦点数至少为 1 个的边权乘积和，加上路径交点数至少为 2 个的边权乘积和...

根据广义容斥定理， $det(M) = \sum_{P: a \rightarrow b, P_i \cap P_j = \emptyset} \prod_{i=1}^n \omega(P_i)$ ，即所有方案中所有路径边权积的积的和

5 树上问题

5.1 树上倍增

```

1 int d[N]; // d[i]=i 的深度, d[rt]=1, (不能为 0)
2 int f[N][21], mx[N][21];
3 void dfs(int x, int fa) {
4     for(int i=hd[x]; i; i=e[i].nx) {
5         int y=e[i].to;
6         if(y==fa) continue;
7         mx[y][0]=e[i].v;
8         d[y]=d[x]+1;
9         f[y][0]=x;

```

```

10     for(int j=1;j<=logn;++j){
11         f[y][j]=f[f[y][j-1]][j-1];
12         mx[y][j]=max(mx[y][j-1],mx[f[y][j-1]][j-1]);
13     }
14     dfs(y,x);
15 }
16 }
17 int lca(int x,int y){
18     if(d[x]<d[y])swap(x,y);
19     for(int i=logn;i>=0;--i){
20         if(d[f[x][i]]>=d[y])x=f[x][i];//注意令根的深度=1，否则会跳错
21     }
22     if(x==y)return x;
23     for(int i=logn;i>=0;--i){
24         if(f[x][i]!=f[y][i]){
25             x=f[x][i];y=f[y][i];
26         }
27     }
28     return f[x][0];
29 }
30 int chianmax(int x,int y){
31     int s=0;
32     if(d[x]<d[y])swap(x,y);
33     for(int i=logn;i>=0;--i){
34         if(d[f[x][i]]>=d[y])x=max(x,mx[x][i]),x=f[x][i];
35     }
36     if(x==y)return x;
37     for(int i=logn;i>=0;--i){
38         if(f[x][i]!=f[y][i]){
39             s=max({s,mx[x][i],mx[y][i]});
40             x=f[x][i];y=f[y][i];
41         }
42     }
43     s=max({s,mx[x][0],mx[y][0]});
44     return f[x][0];
45 }

```

5.2 轻重链剖分

```

1 //假设线段树已经写好
2 int dfn[N],sz[N],son[N],fa[N],top[N],d[N],tdfn;

```

```

3 void dfs1(int x,int f){
4     sz[x]=1;
5     for(int i=hd[x];i;i=e[i].nx){
6         int y=e[i].to;
7         if(y==f)continue;
8         fa[y]=x;
9         d[y]=d[x]+1;
10        dfs1(y,x);
11        sz[x]+=sz[y];
12        if(sz[son[x]]>sz[y])son[x]=y;
13    }
14 }
15 void dfs2(int x,int t){
16     top[x]=t;dfn[x]=++tdfn;
17     if(son[x])dfs2(son[x],t);
18     for(int i=hd[x];i;i=e[i].nx){
19         int y=e[i].to;
20         if(dfn[y])continue;//if(fa[x]==y||y==son[x])continue;
21         dfs2(y,y);
22     }
23 }
24 int lca(int x,int y){
25     while(top[x]!=top[y]){
26         if(d[top[x]]>d[top[y]])swap(x,y);
27         x=fa[top[x]];
28     }
29     return d[x]<d[y]?x:y;
30 }
31 void updchain(int x,int y,int c){//更新树链,询问也类似
32     while(top[x]!=top[y]){
33         if(d[top[x]]<d[top[y]])swap(x,y);
34         upd(dfn[top[x]],dfn[x],1,c);
35         x=fa[top[x]];
36     }
37     if(d[x]>d[y])swap(x,y);
38     upd(dfn[x],dfn[y],1,c);//若维护边的信息,最后需要改为:
39                             //if(x!=y)upd(dfn[x]+1,dfn[y],1,c);
40 }
41 int main(){
42     //输入点/边信息
43     dfs1(1,0);dfs2(1,1);

```

```

44     for(int i=1;i<=n;++i)v[dfn[i]]=a[i];
45     build(1,n,1);
46     upd(dfnt[x],dfnt[x]+sz[x]-1,1,c);//子树更新
47 }

```

5.3 欧拉序 ST 表求 LCA

```

1  int Log2[N<<1],st[N<<1][21],euler[N<<1],cnt;//欧拉序长度为2n-1
2  int idx[N],d[N];//idx[i]=i号结点欧拉序中第一次出现的下标,d表示深度
3  void dfs(int x){//获得欧拉序
4      euler[++cnt]=x;
5      idx[x]=cnt;
6      for(int i=hd[x];i;i=e[i].nx){
7          int y=e[i].to;
8          if(d[y])continue;
9          d[y]=d[x]+1;
10         dfs(y);
11         euler[++cnt]=x;
12     }
13 }
14 int qrymin(int l,int r){//ST表查询深度最小值的结点编号
15     int k=Log2[r-l+1];
16     return d[st[l][k]]<d[st[r-(1<<k)+1][k]]?st[l][k]:st[r-(1<<k)+1][k];
17 }
18 int lca(int x,int y){//求x,y的lca
19     if(idx[x]>idx[y])swap(x,y);
20     return qrymin(idx[x],idx[y]);
21 }
22 int main(){
23     //输入点边信息
24     for(int i=2;i<=(N<<1)-1;++i)Log2[i]=Log2[i>>1]+1;
25     d[1]=1;dfs(1);
26     for(int i=1;i<=cnt;++i)st[i][0]=euler[i];
27     for(int j=1;1<<j<=cnt;++j){
28         for(int i=1;i+(1<<j)-1<=cnt;++i){
29             st[i][j]=d[st[i][j-1]]<d[st[i+(1<<j-1)][j-1]]?
30                 st[i][j-1]:st[i+(1<<j-1)][j-1];
31         }
32     }
33 }

```


5.4 点分治

```

1  int n,rt,hd[N],tot;
2  struct edge{
3      int to,nx,v;
4  }e[N<<1];
5  inline void add(int x,int y,int z){
6      e[++tot]={y,hd[x],v};hd[x]=tot;
7  }
8  int maxson[N],sz[N],sum;
9  bool vis[N];
10 void getroot(int x,int fa){
11     maxson[x]=0;sz[x]=1;//初始化最大子树大小以及size
12     for(int i=hd[x];i;i=e[i].nx){
13         int y=e[i].to;
14         if(vis[y]||y==fa)continue;
15         getroot(y,x);
16         sz[x]+=sz[y];
17         maxson[x]=max(maxson[x],sz[y]);
18     }
19     maxson[x]=max(maxson[x],sum-sz[x]);
20     if(!rt||maxson[x]<maxson[rt])rt=x;
21 }
22 int b[N],c[N];
23 void getsubtree(int x,int fa,ll len){//统计子树信息
24     c[++c[0]]=len;
25     for(int i=hd[x];i;i=e[i].nx){
26         int y=e[i].to;
27         if(vis[y]||y==fa)continue;
28         getsubtree(y,x,len+e[i].v);
29     }
30 }
31 void cclt(int x){
32     b[b[0]=1]=0;//将根加入数组
33     for(int i=hd[x];i;i=e[i].nx){
34         int y=e[i].to;
35         if(vis[y])continue;
36         c[0]=0;//记录子树的数组清空
37         getsubtree(y,x,e[i].v);//枚举所有子树
38         //新的子树和已访问子树连成的所有链
39         for(int j=1;j<=c[0];++j){
40             for(int k=1;k<=b[0];++k){

```

```

41
42     }
43 }
44 //将新的子树加入数组
45 for(int j=1;j<=c[0];++j)b[++b[0]]=c[i];
46 }
47 for(int i=1;i<=b[0];++i){//删除子树的贡献
48 }
49 }
50 void divide(int x){
51     vis[x]=1;//分割该点
52     cclt(x);//统计该重心的子树
53     for(int i=hd[x];i;i=e[i].nx){//枚举所有子树
54         int y=e[i].to;
55         if(vis[y])continue;
56         sum=sz[y];rt=0;//sum=子树大小,初始化重心
57         getroot(y,0);//找到子树的重心
58         divide(rt);//递归子树
59     }
60 }
61 int main(){
62     n=rd();
63     for(int i=1;i<n;++i){
64         int x=rd(),y=rd(),z=rd();
65         add(x,y,z);add(y,x,z);
66     }
67     sum=n;rt=0;
68     getroot(1,0);
69     divide(rt);
70 }

```

5.5 启发式合并

```

1 int sz[N],son[N],Son;
2 ll ans[N];
3 int cnt[N];//color i 出现的次数
4 ll sum,maxcnt;
5 void dfs1(int x,int fa){
6     int mx=0;sz[x]=1;
7     for(auto y:e[x]){
8         if(y==fa)continue;

```

```
9         dfs1(y,x);
10        sz[x]+=sz[y];
11        if(sz[y]>mx){
12            mx=sz[y];
13            son[x]=y;
14        }
15    }
16 }
17 void add(int x,int fa){
18     ++cnt[a[x]];
19     if(cnt[a[x]]>maxcnt){
20         maxcnt=cnt[a[x]];
21         sum=a[x];
22     }
23     else if(cnt[a[x]]==maxcnt)
24         sum+=a[x];
25     for(auto y:e[x]){
26         if(y==fa||y==son)continue;
27         add(y,x);
28     }
29 }
30 void del(int x,int fa){
31     --cnt[a[x]];
32     for(auto y:e[x]){
33         if(y==fa)continue;
34         del(y,x);
35     }
36 }
37 void dfs2(int x,int fa,bool f){
38     for(auto y:e[x]){
39         if(y==fa||y==son)continue;
40         dfs2(y,x,1);
41     }
42     if(son[x])dfs2(son[x],x,0);
43     Son=son[x];add(x,fa);
44     ans[x]=sum;
45     if(f){
46         del(x,fa);
47         sum=0;maxcnt=0;
48     }
49 }
```

```
50 int main(){dfs1(1,0);dfs2(1,0,0);}
```

5.6 长链剖分 $O(1)$ k 级祖先

```
1  int f[N][20],son[N],d[N],maxd[N],top[N],dfn[N],len[N],highbit[N];
2  vector<int>Up[N],Down[N];
3  void dfs1(int x,int fa){
4      maxd[x]=d[x]=d[fa]+1;f[x][0]=fa;
5      for(int i=1;i<20;++i){
6          if(!f[x][i-1])break;
7          f[x][i]=f[f[x][i-1]][i-1];
8      }
9      for(int i=hd[x];i;i=e[i].nx){
10         int y=e[i].to;
11         if(y==fa)continue;
12         dfs1(y,x);
13         if(maxd[y]>maxd[son[x]])son[x]=y,maxd[x]=maxd[y];
14     }
15 }
16 void dfs2(int x,int t){
17     top[x]=t;len[x]=maxd[x]-d[top[x]]+1;
18     dfn[x]=++dfn[0];
19     if(son[x])dfs2(son[x],t);
20     for(int i=hd[x];i;i=e[i].nx){
21         int y=e[i].to;
22         if(y!=f[x][0]&&y!=son[x])dfs2(y,y);
23     }
24 }
25 inline int find_kth_fa(int x,int k){
26     if(k>d[x])return 0;if(k==0)return x;
27     x=f[x][highbit[k]];k^=1<<highbit[k];
28     if(!k)return x;
29     if(d[x]-d[top[x]]==k)return top[x];
30     if(d[x]-d[top[x]]>k)return Down[top[x]][d[x]-d[top[x]]-k-1];
31     return Up[top[x]][k-d[x]+d[top[x]]-1];
32 }
33 int main(){
34     scanf("%d",&n);
35     for(int i=1;i<n;++i){
36         int x,y;scanf("%d%d",&x,&y);
37         add(x,y);add(y,x);
```

```

38     }
39     dfs1(1,0);dfs2(1,1);
40     for(int i=1;i<=n;++i){
41         if(i==top[i]){
42             int l=0,x=i;
43             while(l<len[i]&&x)f[x][0],++l,Up[i].push_back(x);
44             l=0,x=i;
45             while(l<len[i])x=son[x],++l,Down[i].push_back(x);
46         }
47     }
48     for(int i=1,mx=1;i<=n;++i){
49         if(i>>mx&1)++mx;
50         highbit[i]=mx-1;
51     }
52     find_kth_fa(x,k); // 调用x的k级祖先
53 }

```

5.7 虚树

```

1  int h,b[N],root;ll dp[N];
2  bool vis[N]; // 该点是询问点
3  inline bool cmp(int a,int b){
4      return dfn[a]<dfn[b];
5  }
6  int build_virtual_tree(int b[],int h){
7      static int sta[N],top;
8      tot1=0;
9      sort(b+1,b+1+h,cmp); // 按dfs序排序
10     sta[top=1]=b[1];
11     for(int i=2;i<=h;++i){
12         int ff=lca(b[i],sta[top]);
13         while(1){
14             if(d[ff]>=d[sta[top-1]]){
15                 if(ff!=sta[top]){
16                     add1(ff,sta[top]);
17                     if(ff!=sta[top-1])sta[top]=ff;
18                     else --top;
19                 }
20                 break;
21             }
22             else{

```

```

23         add1(sta[top-1],sta[top]);
24         --top;
25     }
26 }
27     sta[++top]=b[i];
28 }
29 while(--top)add1(sta[top],sta[top+1]);
30 return sta[1];//虚树的根节点(所有询问点的lca)
31 }
32 void dfs3(int x){//虚树上树形DP
33     ll sum=0;
34     for(int i=hd1[x];i;i=e1[i].nx){
35         int y=e1[i].to;dfs3(y);sum+=dp[y];
36     }
37     if(vis[x])dp[x]=mn[x];//询问点和非询问点分别算
38     else dp[x]=min(mn[x],sum);
39 }
40 void del_tree(int x){//清空虚树,记得在函数外tot1=0
41     for(int i=hd1[x];i;i=e1[i].nx)del_tree(e1[i].to);
42     hd1[x]=0;
43 }
44 int main(){
45     //询问b[1~h]这些结点
46     for(int i=1;i=h;++i)vis[b[i]]=1;
47     root=build_virtial_tree(b,h);
48     dfs3(root); ans=dp[root];
49     del_tree(root);
50     for(int i=1;i<=h;++i)vis[b[i]]=0;
51 }

```

6 动态规划

6.1 树形背包

1、具有树形先后关系的背包 (拿了根才能拿儿子)

```

1 void dfs(int x,int num){
2     for(auto y:e[x]){
3         for(int i=0;i<num;++i)dp[y][i]=dp[x][i]+a[y];
4         dfs(y,num-1);

```

```

5     for(int i=1;i<=num;++i)dp[x][i]=max(dp[x][i],dp[y][i-1]);
6 }
7 }

```

2、没有先修关系， $dp[x][y]$ =x 号子树中选了 y 个的价值

```

1 //假设预处理好子树size
2 void dfs(int x,int f){
3     dp[x][0]=dp[x][1]=0;
4     for(auto y:e[x]){
5         if(y==f)continue;//枚举到size,时间复杂度为O(nm)
6         for(int i=min(sz[x],k);i>=0;--i){//x子树选几个
7             for(int j=0;j<=i&&j<=sz[y];++j){//y子树选几个
8                 dp[x][i]=max(dp[x][i],dp[y][j]+dp[x][i-j]);
9             }
10        }
11    }
12 }

```

6.2 区间 DP

```

1 int main(){
2     for(int i=1;i<=n;++i)dp[i][i]=0;//设置初始状态
3     for(int len=2;len<=n;++len){
4         for(int i=1;i+len-1<=n;++i){
5             int j=i+len-1; //视情况而定,有两种可能的转移方式
6             //1、枚举分段点/最后一个加进去的,有时可优化枚举个数
7             for(int k=i;k<=j;++k)dp[i][j]+=dp[i][k]+dp[k+1][j]
8             //2、每次只从相邻位置转移
9             dp[i][j]=dp[i][j-1]+dp[i+1][j];
10        }
11    }
12 }

```

6.3 数位 DP

```

1 const int N=20;//数字长度
2 int h,a[N];
3 //dp第一维是搜索到第几位(从低到高),后面状态可以额外加
4 //时间复杂度是所有状态数,即dp数组的空间大小

```

```

5  ll dp[N][2][2][2]; //dp[pos][f4][f62][pre6]
6  ll dfs(int pos, bool f4, bool f62, bool pre6, bool lim){ //lim: 高位均相等
7      //lim是必要参数, 只有!lim才能剪枝, 否则后续状态受到高位限制
8      if(pos==0) return !f4&&!f62?1:0; //pos=0整个数字选好了, 如果满足就是搜到一个
9      if(!lim&&dp[pos][f4][f62][pre6]!=-1)
10         return dp[pos][f4][f62][pre6];
11         int mx=lim?a[pos]:9;
12         ll s=0;
13         for(int i=0; i<=mx; ++i)
14             s+=dfs(pos-1, f4||i==4, f62||(pre6&&i==2), i==6, lim&&i==mx);
15         if(!lim) dp[pos][f4][f62][pre6]=s;
16         return s;
17     }
18     ll solve(ll x){ //0~x的答案
19         h=0; memset(dp, -1, sizeof dp);
20         while(x){
21             a[++h]=x%10;
22             x/=10;
23         } //a[1]=个位, a[h]=最高位
24         return dfs(h, 0, 0, 0, 1);
25     }
26     int main(){ //没有62或4的数字
27         ll x, y;
28         while(scanf("%lld%lld", &x, &y)!=EOF&&x){
29             printf("%lld\n", solve(y)-solve(x-1));
30         }
31     }

```

6.4 最长上升子序列

```

1  int n, a[N];
2  int dp1[N], len1; //len1=最长上升子序列长度
3  int dp2[N], len2; //最少len2个上升子序列可以覆盖整个数组
4  int main(){
5      n=rd();
6      for(int i=1; i<=n; ++i) a[i]=rd();
7      dp1[len1=1]=a[1];
8      for(int i=2; i<=n; ++i){
9          if(a[i]>dp1[len1]) dp1[++len1]=a[i];
10         else{
11             /*lower_bound(dp1+1, dp1+1+len1, a[i])=a[i];

```



```

12         int l=1,r=len1;
13         while(l<=r){
14             int m=l+r>>1;
15             if(dp1[m]<=a[i])l=m+1;
16             else r=m-1;
17         }
18         // < < < < < > > > >
19         //           r l
20         dp1[l]=a[i];
21     }
22 }
23 printf("%d\n",len1);
24 dp2[len2=1]=a[1];
25 for(int i=2;i<=n;++i){
26     if(a[i]<dp2[len2])dp2[++len2]=a[i];
27     else{
28         int l=1,r=len2;
29         while(l<=r){
30             int m=l+r>>1;
31             if(dp2[m]>=a[i])l=m+1;
32             else r=m-1;
33         }
34         // > > > > > < < < < <
35         //           r l
36         dp2[l]=a[i];
37     }
38 }
39 printf("%d\n",len2);
40 }

```

6.5 所有字段和

```

1 for(int i=1;i<=n;++i){
2     dp[i]=dp[i-1]+i*a[i];
3     ans+=dp[i]; //dp[i]=所有以i结尾的字段和
4 }

```

6.6 公共子序列

最长公共子序列（若元素不重复，可以重编号后求最长上升子序列）

```

1  int n,m,a[N],b[N],dp[N][N];
2  int main(){
3      scanf("%d%d",&n,&m);
4      for(int i=1;i<=n;++i)scanf("%d",&a[i]);
5      for(int i=1;i<=m;++i)scanf("%d",&b[i]);
6      for(int i=1;i<=n;++i){
7          for(int j=1;j<=m;++j){
8              dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
9              if(a[i]==b[j])dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);
10         }
11     }
12     printf("%d\n",dp[n][m]);
13 }

```

所有公共子序列个数

```

1  int main(){
2      for(int i=1;i<=n;++i){
3          for(int j=1;j<=m;++j){
4              dp[i][j]=dp[i-1][j]+dp[i][j-1]-dp[i-1][j-1];
5              if(a[i]==b[j])dp[i][j]+=dp[i-1][j-1]+1;
6          }
7      }
8  }

```

6.7 错位排列数

```

1  dp[i]=长度为i的排列,所有i满足p[i]!=i的方案数
2  dp[1]=0;dp[2]=1;dp[i]=(i-1)*(dp[i-1]+dp[i-2])

```

6.8 基环树直径

```

1  struct edge{
2      int to,nx,v;
3  }e[N<<1];
4  int hd[N],tot;
5  inline void add(int x,int y,int z){
6      e[++tot]={y,hd[x],z};hd[x]=tot;
7  }
8  bool vis[N];
9  int sta[N],top;

```

```
10 int n,c[N],h;
11 ll b[N];
12 ll maxd[N];
13 ll treeD;
14 ll presum[N],sufsum[N];
15 ll preans[N],sufans[N];
16 ll premax[N],sufmax[N];
17 void find_circle(int x,int fa){
18     sta[++top]=x;vis[x]=1;
19     for(int i=hd[x];i;i=e[i].nx){
20         int y=e[i].to;
21         if(y==fa)continue;
22         if(h)return;
23         b[y]=e[i].v;
24         if(!vis[y])find_circle(y,x);
25         else{
26             while(sta[top]!=y){
27                 c[++h]=sta[top--];
28             }
29             c[++h]=y;
30             return;
31         }
32     }
33     --top;
34 }
35 void getmaxd(int x,int fa){
36     for(int i=hd[x];i;i=e[i].nx){
37         int y=e[i].to;
38         if(y==fa||vis[y])continue;
39         getmaxd(y,x);
40         treeD=max(treeD,maxd[x]+maxd[y]+e[i].v);
41         maxd[x]=max(maxd[x],maxd[y]+e[i].v);
42     }
43 }
44 int main(){
45     scanf("%d",&n);
46     for(int i=1;i<=n;++i){
47         int x,y,z;scanf("%d%d%d",&x,&y,&z);
48         add(x,y,z);add(y,x,z);
49     }
50     find_circle(1,0);
```

```

51     memset(vis+1,0,n);
52     for(int i=1;i<=h;++i)vis[c[i]]=1;
53     for(int i=1;i<=h;++i)getmaxd(c[i],0);
54     for(int i=1;i<=h;++i)presum[i]=presum[i-1]+b[c[i-1]];
55     for(int i=h-1;i--i)sufsum[i]=sufsum[i+1]+b[c[i]];
56     ll mx=0;
57     for(int i=1;i<=h;++i){
58         preans[i]=max(preans[i-1],presum[i]+maxd[c[i]]+mx);
59         mx=max(mx,maxd[c[i]]-presum[i]);
60     }
61     mx=0;
62     for(int i=h;i--i){
63         sufans[i]=max(sufans[i+1],sufsum[i]+maxd[c[i]]+mx);
64         mx=max(mx,maxd[c[i]]-sufsum[i]);
65     }
66     for(int i=1;i<=h;++i)premax[i]=max(premax[i-1],presum[i]+maxd[c[i]]);
67     for(int i=h;i--i)sufmax[i]=max(sufmax[i+1],sufsum[i]+maxd[c[i]]);
68     ll ans=1ll<<60;
69     for(int i=1;i<h;++i){
70         ll now=max(preans[i],sufans[i+1]);
71         now=max(now,premax[i]+sufmax[i+1]+b[c[h]]);
72         ans=min(ans,now);
73     }
74     ans=max(ans,treeD);
75     printf("%.11f\n",1.0*ans/2);
76 }

```

6.9 高维前缀和

SOS DP (sum of subset)

```

1 //a[i][j][k]=sum[1~i][1~j][1~k]
2 for(int i = 1; i <= n; i++)
3     for(int j = 1; j <= n; j++)
4         for(int k = 1; k <= n; k++)
5             a[i][j][k] += a[i - 1][j][k];
6 for(int i = 1; i <= n; i++)
7     for(int j = 1; j <= n; j++)
8         for(int k = 1; k <= n; k++)
9             a[i][j][k] += a[i][j - 1][k];
10 for(int i = 1; i <= n; i++)
11     for(int j = 1; j <= n; j++)

```

```

12         for(int k = 1; k <= n; k++)
13             a[i][j][k] += a[i][j][k - 1];
14 //类似地, 四维就再来一遍

```

若每一维只有 2 即 (0,1) 则可优化为 (也即是求子集和)

```

1  for(int j = 0; j < n; j++)
2      for(int i = 0; i < 1 << n; i++)
3          if(i >> j & 1) f[i] += f[i ^ (1 << j)];

```

超集和 (超集: 若 S2 的每一个元素都在 S1 且 S1 可能包含 S2 中没有的元素, 则 S1 为 S2 的一个超集合)(其实就是子集反一下)

```

1  for(int j = 0; j < n; j++)
2      for(int i = 0; i < 1 << n; i++)
3          if(!(i >> j & 1)) f[i] += f[i ^ (1 << j)];

```

7 字符串

7.1 KMP 算法

一个字符串 s 的 $border(s)$ 是 s 最长的既是 s 前缀又是 s 后缀的真子串

KMP 算法可以建立 $next$ 数组 (p 数组), 使得 $p[i] = border(prefix(i))$

字符串最小循环节为 $n - next[n]$ (下标从 1 开始)

下标从 1 开始

```

1  char s1[N], s2[N];
2  int n, m, p[N]; //p[i]=s2[1~i] 后缀=前缀的最大长度(不包括本身)
3  int main(){
4      scanf("%s%s", s1+1, s2+1);
5      n=strlen(s1+1); m=strlen(s2+1);
6      for(int i=2, j=0; i<=m; ++i){
7          while(j&& s2[i]!=s2[j+1]) j=p[j];
8          if(s2[i]==s2[j+1]) ++j;
9          p[i]=j;
10     }
11     for(int i=1, j=0; i<=n; ++i){
12         while(j&& s1[i]!=s2[j+1]) j=p[j];
13         if(s1[i]==s2[j+1]) ++j;
14         if(j==m){
15             printf("%d\n", i-m+1); //s1中出现s2的下标
16             j=p[j]; //不能重叠: j=0
17         }

```

```

18     }
19 }

```

下标从 0 开始

```

1  int n,m,p[N]; //p[i]=s2[0~i]的后缀=前缀的最大长度(不包括本身)
2  char s1[N],s2[N];
3  int main(){
4      scanf("%s%s",s1,s2);
5      n=strlen(s1);m=strlen(s2);
6      for(int i=1,j=0;i<m;++i){
7          while(j&& s2[i]!=s2[j])j=p[j-1];
8          if(s2[i]==s2[j])++j;
9          p[i]=j;
10     }
11     for(int i=0,j=0;i<n;++i){
12         while(j&& s1[i]!=s2[j])j=p[j-1];
13         if(s1[i]==s2[j])++j;
14         if(j==m){
15             j=p[j-1]; //不能重叠: j=0
16             printf("%d\n",i-m+1); //s1中出现s2的下标
17         }
18     }
19 }

```

7.2 失配树

建立 KMP 算法的 *next* 数组（下标从 1 开始），建立 *fail* 树， $fail[i] = p[i]$

0 到 n 是失配树从根到叶子的拓扑序

求前缀 $prefix(u)$ 和 $prefix(v)$ 的最大公共 *border*

法一、求出 *fail* 树上 $fa = lca(u, v)$ ，若 u 是 v 的祖先，则答案为 $next[fa]$ ，否则答案为 fa

```

1  int n,logn;char s[N]; //下标从1开始
2  int f[21][N],d[N]; //f[i][0]=s[1~i]后缀=前缀的最大长度(不包括本身)
3  inline int lca(int x,int y){
4      if(d[x]<d[y])swap(x,y);
5      for(int i=logn;i>=0;--i){
6          if(d[f[i][x]]>=d[y])x=f[i][x];
7      }
8      //若x==y,说明xy为祖先关系,最终答案为next[x],即f[0][x]
9      for(int i=logn;i>=0;--i){
10         if(f[i][x]!=f[i][y]){

```

```

11         x=f[i][x];y=f[i][y];
12     }
13 }
14 return f[0][x];
15 }
16 int main(){
17     scanf("%s",s+1);//ababaabab
18     n=strlen(s+1);logn=log2(n)+1;
19     d[1]=1;//root=0, fa[1]=0, dep[0]=0, dep[1]=1
20     for(int i=2,j=0;i<=n;++i){
21         while(j&& s[i]!=s[j+1])j=f[0][j];
22         if(s[i]==s[j+1])++j;
23         f[0][i]=j;d[i]=d[j]+1;
24     }
25     for(int i=1;i<=logn;++i){
26         for(int j=1;j<=n;++j){
27             f[i][j]=f[i-1][f[i-1][j]];
28         }
29     }
30     int q;scanf("%d",&q);
31     while(q--){
32         int x,y;scanf("%d%d",&x,&y);
33         printf("%d\n",lca(x,y));
34     }
35 }

```

法二、 $pre(u)$ 循环节长度为 $u - next[u]$ ，根据循环节长度是否大于一半来分类讨论，使 u 或 v 每次减半

```

1  int n,p[N];char s[N];//下标从1开始
2  int main(){
3      scanf("%s",s+1);n=strlen(s+1);//ababaabab
4      for(int i=2,j=0;i<=n;++i){
5          while(j&& s[i]!=s[j+1])j=p[j];
6          if(s[i]==s[j+1])++j;
7          p[i]=j;
8      }
9      int q;scanf("%d",&q);
10     while(q--){
11         int u,v;scanf("%d%d",&u,&v);
12         u=p[u];v=p[v];//一开始要先跳一次,保证非本身
13         while(u!=v){

```

```

14         if(u<v)swap(u,v);
15         int d=u-p[u];//最小循环节
16         if(d<u>>1){//小于一半
17             if((u-v)%d==0)u=v;//u和v恰好都是循环节的k倍,得到答案
18             else u=u%d+d;//循环节只保留一个abcabcabcab->abcab
19         }
20         else u=p[u];
21     }
22     printf("%d\n",u);
23 }
24 }

```

7.3 字符串哈希

如果自然溢出被卡,可以考虑自然溢出和 $1e9+7/998244353$ 的双模哈希

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef unsigned long long ull;
4  typedef long long ll;
5  const int N=1e5+5;
6  const ull k=13331;//k看心情随便换
7  int n;char s[N];
8  ull hash1[N],hash2[N],kpow[N];
9  bool check(int l,int r){//s[l,r]是回文串?
10     ull hs1=hash1[r]-hash1[l-1]*kpow[r-l+1];//不需要考虑负数
11     ull hs2=hash2[l]-hash2[r+1]*kpow[r-l+1];//自然溢出后是模 $2^{64}$ 意义下的
12     return hs1==hs2;
13 }
14 int main(){
15     scanf("%d%s",&n,s+1);//n=strlen(s+1);
16     kpow[0]=1;hash1[n+1]=0;
17     for(int i=1;i<N;++i)kpow[i]=kpow[i-1]*k;
18     for(int i=1;i<=n;++i)hash1[i]=hash1[i-1]*k+s[i];
19     for(int i=n;i>=1;--i)hash2[i]=hash2[i+1]*k+s[i];
20     int q;scanf("%d",&q);
21     while(q--){
22         int l,r;scanf("%d%d",&l,&r);
23         puts(check(l,r)?"YES":"NO");
24     }
25 }
26 /* hash("dcbe")='d'*k^3+'c'*k^2+'b'*k+'a'*1

```



```

27 hash1[i]=hash({s[1],s[2],...,s[i]})
28 hash2[i]=hash({s[n],s[n-1],...,s[i]})          */

```

7.4 Manacher

```

1  const int N=2.2e7+5;//strlen=2倍原字符串长度+1
2  int n,p[N];char s[N];
3  bool check(int l,int r){//[l,r]区间是否是回文串
4      return p[l+r]>=r-l+1;
5  }
6  int main(){
7      scanf("%s",s+1);n=strlen(s+1); //将字符串acbc变为 #a#c#b#c#
8      for(int i=n;i-->1)s[i<<1|1]='#',s[i<<1]=s[i];
9      s[1]='#';s[0]='@';//第0个位置赋为不存在的字符,控制循环中的while的边界
10     n=n<<1|1; int maxr=0,pos=0,ans=0;
11     for(int i=1;i<=n;++i){
12         if(i<maxr)p[i]=min(p[pos*2-i],maxr-i);
13         while(s[i-p[i]-1]==s[i+p[i]+1])++p[i];
14         if(i+p[i]>maxr)maxr=i+p[i],pos=i;
15     }
16     //p[i]=新串以i为中心的回文串单侧最大扩展长度
17     //举例 : str = #a#b#b#c(下标从1开始),p[5]=2,p[4]=1
18     //max(p[i])=原串最长回文串长度
19     //p[i]在原字符串中代表的回文区间是[(i+1)/2-p[i]/2,(i+1)/2+(p[i]+1)/2]
20     //原字符串以第i个字符为中心的回文串长度为p[i]*2
21     //原字符串以[i,i+1]两个字符为中心的回文串长度为p[i]*2+1
22 }

```

7.5 AC 自动机

```

1  const int N=2e5+10,M=N*30;//注意修改M的大小(文本串长度之和)
2  int t[M][26],fail[M],cnt[M],idx[M],tot;
3  string s[N];
4  vector<int>e[M];
5  int n;
6  void insert(const string &s,int id){
7      int p=0;
8      for(auto ch:s){
9          int c=ch-'a';
10         if(!t[p][c])t[p][c]=++tot;

```

```
11     p=t[p][c];
12 }
13 idx[id]=p;
14 }
15 void build_fail(){
16     queue<int>q;
17     for(int i=0;i<26;++i)if(t[0][i])q.push(t[0][i]);
18     for(;;!q.empty());{
19         int p=q.front();q.pop();
20         for(int i=0;i<26;++i){
21             if(t[p][i]){
22                 fail[t[p][i]]=t[fail[p]][i];
23                 q.push(t[p][i]);
24             }
25             else t[p][i]=t[fail[p]][i];
26         }
27     }
28 }
29 void query(const string &s){
30     int p=0;
31     for(auto c:s){
32         p=t[p][c-'a'];
33         ++cnt[p];
34     }
35 }
36 void build_failtree(){
37     for(int i=1;i<=tot;++i)e[fail[i]].push_back(i);
38 }
39 void dfs(int x){
40     for(auto y:e[x]){
41         dfs(y);
42         cnt[x]+=cnt[y];
43     }
44 }
45 int main(){
46     cin>>n;
47     for(int i=1;i<=n;++i)cin>>s[i];
48     for(int i=1;i<=n;++i)insert(s[i],i);
49     build_fail();    build_failtree();
50     string str; cin>>str;
51     query(str);    dfs(0);
```

```

52     for(int i=1;i<=n;++i)cout<<cnt[idx[i]]<<"\n";
53 }

```

7.6 后缀数组 SA

```

1  int n,m,s[N]; char str[N];
2  int sa[N]; //sa[i]=排名为i的后缀的下标(从1开始)
3  int rk[N]; //rk[i]=下标为i的后缀的排名
4  int height[N]; //height[i]=lcp(suf(sa[i]),suf(sa[i-1]))
5  //lcp{suf(sa[i]),suf(sa[j])}=min{height[i+1,...,j]}
6  int st[N][20]; //用于求longest common prefix
7  int c[N],x[N],y[N];
8  void get_sa(){ //也可以传入int数组
9      m=122; //ASCII最大值
10     for(int i=1;i<=n;++i)++c[x[i]=s[i]];
11     for(int i=2;i<=m;++i)c[i]+=c[i-1];
12     for(int i=1;i<=n;++i)sa[c[x[i]]--]=i;
13     for(int k=1;k<=n;k<=<1){
14         int num=0; memset(c+1,0,m<<2);
15         for(int i=n-k+1;i<=n;++i)y[++num]=i;
16         for(int i=1;i<=n;++i)if(sa[i]>k)y[++num]=sa[i]-k;
17         for(int i=1;i<=n;++i)++c[x[i]];
18         for(int i=2;i<=m;++i)c[i]+=c[i-1];
19         for(int i=n;i;--i)sa[c[x[y[i]]]--]=y[i],y[i]=0;
20         swap(x,y);
21         x[sa[1]]=1; num=1;
22         for(int i=2;i<=n;++i)
23             x[sa[i]]=y[sa[i]]==y[sa[i-1]]&&y[sa[i]+k]==y[sa[i-1]+k]?num:++num;
24         if (num==n) break;
25         m=num;
26     }
27 }
28 void get_height(){
29     for(int i=1;i<=n;++i)rk[sa[i]]=i;
30     int k=0; height[1]=0;
31     for(int i=1;i<=n;++i){
32         if(!rk[i])continue;
33         if(k)--k;
34         int j=sa[rk[i]-1];
35         while(i+k<n&&j+k<n&&s[i+k]==s[j+k])++k;
36         height[rk[i]]=k;

```

```

37     }
38 }
39 int Log2[N];
40 void build_st(){
41     for(int i=1;i<=n;++i)st[i][0]=height[i];
42     for(int j=1;1<=j<=n;++j){
43         for(int i=1;i+(1<=j)-1<=n;++i){
44             st[i][j]=min(st[i][j-1],st[i+(1<=j-1)][j-1]);
45         }
46     }
47 }
48 int lcp(int l,int r){
49     l=rk[l];r=rk[r];
50     if(l>r)swap(l, r);++l;
51     int k=Log2[r-l+1];
52     return min(st[l][k],st[r-(1<=k)+1][k]);
53 }
54 int main(){
55     for(int i=2;i<N;++i)Log2[i]=Log2[i>>1]+1;
56     scanf("%s",str+1);
57     n=strlen(str+1);
58     for(int i=1;i<=n;++i)s[i]=str[i];
59     get_sa();get_height();build_st();
60 }

```

7.7 后缀自动机 SAM

```

1  const int M=1e6+6;//M=size(SAM)=2*len
2  int ch[M][26];//当字符集较大可改为 map<int,int>ch[M];
3  int fa[M],sam_cnt,last,len[M],sz[M],sum[M];
4  void sam_init(){last=sam_cnt=1;}
5  void extend(int c){
6      int p=last,np=++sam_cnt;
7      last=np; len[np]=len[p]+1;
8      while(p&&!ch[p][c])ch[p][c]=np,p=fa[p];
9      if(!p)fa[np]=1;
10     else{
11         int q=ch[p][c];
12         if(len[q]==len[p]+1)fa[np]=q;
13         else{
14             int nq=++sam_cnt;

```

```

15         memcpy(ch[nq],ch[q],sizeof ch[nq]); //若ch是map,改为ch[nq]=ch[q]
16         len[nq]=len[p]+1; fa[nq]=fa[q]; fa[q]=fa[np]=nq;
17         while(p&&ch[p][c]==q)ch[p][c]=nq,p=fa[p];
18     }
19 }
20 }
21 //fa[i]=i点在parent树上的祖先
22 //len[i]=i点endpos类中的最大字符串长度
23 //sz[i]=i点endpos类的大小=i点字符串集合出现次数
24 //sum[i]=从起点开始经过i点的路径数量
25 void match_suffix(string &str){ //匹配后缀
26     int p=1,l=0;
27     for(int i=0;i<str.size();++i){
28         int c=str[i]-'a';
29         while(!ch[p][c]){
30             p=fa[p];l=len[p];
31         }
32         if(ch[p][c]){
33             p=t[p][c];++l; //l=str[0:i]能匹配上的最长后缀长度
34         }
35         //--l;if(p!=1&&len[fa[p]]>=1)p=fa[p]; //删除当前str最左边的一个字符
36     }
37 }
38 int temp[M],rk[M]; //rk[i]=len第i小的结点(树从根到叶子的拓扑序)
39 inline void calc(){
40     int Type=1; //Type=1:不同位置的相同字符串算不同的,0:相同字符串算一次
41     for(int i=1;i<=sam_cnt;++i)++temp[len[i]];
42     for(int i=1;i<=sam_cnt;++i)temp[i]+=temp[i-1];
43     for(int i=1;i<=sam_cnt;++i)rk[temp[len[i]]--]=i;
44     for(int i=sam_cnt;i--i)sz[fa[rk[i]]]+=sz[rk[i]];
45     for(int i=1;i<=sam_cnt;++i)Type?sum[i]=sz[i]:sum[i]=sz[i]=1;
46     sum[1]=sz[1]=0;
47     for(int i=sam_cnt;i--i){
48         for(int j=0;j<26;++j){
49             sum[rk[i]]+=sum[ch[rk[i]][j]];
50         }
51     }
52 }
53 void findkth(int x,int k){ //第k小字典序子串
54     if(k<=sz[x])return;
55     k-=sz[x];

```

```

56     for(int i=0;i<26;++i){
57         int y=ch[x][i];
58         if(sum[y]>=k){//空儿子的sum=0
59             putchar('a'+i);
60             findkth(y,k);
61             return;
62         }
63         else k-=sum[y];
64     }
65 }
66 int main(){
67     sam_init();
68     for(int i=1;i<=n;++i){
69         extend(s[i]-'a');
70         ++sz[last];
71     }
72 }

```

7.8 广义 SAM

建立一个 SAM，包含了多个串的所有子串

```

1  const int M=2e6+6;//M=2*sum(strlen)+1
2  int ch[M][26],len[M],fa[M],sz[M],last,sam_cnt;
3  void sam_init(){sam_cnt=last=1;}
4  int extend(int c,int last){//last=extend(c,last)
5      if(ch[last][c]){
6          int p=last,q=ch[p][c];
7          if(len[p]+1==len[q])return q;
8          else{
9              int np=++sam_cnt;len[np]=len[p]+1;
10             memcpy(ch[np],ch[q],sizeof ch[np]);
11             fa[np]=fa[q],fa[q]=np;
12             while(p&&ch[p][c]==q)ch[p][c]=np,p=fa[p];
13             return np;
14         }
15     }
16     int p=last,np=++sam_cnt;
17     last=np; len[np]=len[p]+1;
18     while(p&&!ch[p][c])ch[p][c]=np,p=fa[p];
19     if(!p)fa[np]=1;
20     else{

```

```

21     int q=ch[p][c];
22     if(len[q]==len[p]+1)fa[np]=q;
23     else{
24         int nq=++sam_cnt;
25         memcpy(ch[nq],ch[q],sizeof ch[nq]);
26         len[nq]=len[p]+1; fa[nq]=fa[q]; fa[q]=fa[np]=nq;
27         while(p&&ch[p][c]==q)ch[p][c]=nq,p=fa[p];
28     }
29 }
30 return np;
31 }
32 char s[M];
33 int main(){
34     int n;scanf("%d",&n);sam_init();
35     for(int i=1;i<=n;++i){//n个串在线建立广义SAM
36         scanf("%s",s);
37         int len=strlen(s);last=1;
38         for(int j=0;j<len;++j){
39             last=extend(s[j]-'a',last);
40             ++sz[last];//所有子串都存在
41         }
42         //++sz[last];//只有完整的串存在
43     }
44 }

```

按 Trie 树构造 (字符在边上)

```

1 //pos[x]=Trie树上的x在SAM中的编号
2 //trie.c[x] fa[x] t[x] 分别代表该点字符,父亲和出边
3 void build(){
4     queue<int>q;
5     for(int i=0;i<26;++i)if(trie.t[1][i])q.push({i});
6     while(q.size()){
7         int x=q.front();q.pop();
8         pos[x]=extend(trie.c[x],trie.fa[x]);
9         for(int i=0;i<26;++i)if(trie.t[x][i])q.push(trie.t[x][i]);
10    }
11 }

```

按树构造 (字符在点上)

```

1 void dfs(int x,int f,int last){
2     int t=extend(c[x],last);

```

```

3     for(auto y:e[x]){
4         if(y==f)continue;
5         dfs(y,x,t);
6     }
7 }

```

7.9 子序列自动机

建立一个 0 号起点，和所有位置相连，转移边输入该位置的字符，每个位置记录下一个字符最近的位置 (不包括自身位置)

这样，从 0 开始的所有路径都是原串的子序列

```

1 int ch[N][26],pos[26];
2 void build(char *s,int n){
3     memset(pos,0,sizeof pos);
4     for(int i=n;i>=0;--i){//注意i>=0,要保存0号点的出边
5         memcpy(ch[i],pos,sizeof ch[i]);
6         pos[s[i]-'a']=i;
7     }
8 }

```

值域较大时，可以用 vector/set 保存每个值出现过的位置，每个点右边最近的 x 需要在 vector 上二分

```

1 int n,q,a[N],b[N],h;
2 set<int>pos[N];//pos[i]=i出现的位置,vector不带修,set带修
3 bool match(int b[],int h){//b[1~h]是a[1~n]的子序列?
4     int x=0;
5     for(int i=1;i<=h;++i){
6         auto it=pos[b[i]].upper_bound(x);//二分出下一个位置
7         if(it==pos[b[i]].end())return 0;
8         x=*it;
9     }
10    return 1;
11 }
12 int main(){
13     scanf("%d%d",&n,&q);
14     for(int i=1;i<=n;++i){
15         scanf("%d",&a[i]);pos[a[i]].insert(i);
16     }
17     while(q--){
18         scanf("%d",&h);

```



```

19         for(int i=1;i<=h;++i)scanf("%d",&b[i]);
20         if(match(b,h))puts("Yes");
21         else puts("No");
22     }
23 }

```

7.10 回文自动机 PAM

```

1  const int N=3e5+10;//PAM结点数<=串长
2  char s[N];
3  int ch[N][26],pam_cnt,last,fail[N],len[N],sz[N],pos[N];
4  //0是偶回文串的根,1是奇回文串的根
5  //fail[i]=i结点代表的串的最长真回文后缀的结点
6  //len[i]=i结点代表的串的长度
7  //sz[i]=i结点包含的回文子串个数
8  //pos[i]=i结点代表的串的右端点在原串的下标
9  //cnt[i]=i结点代表的串出现次数,需要子树和
10 void pam_init(){
11     for(int i=0;i<=pam_cnt;++i){
12         memset(ch[i],0,sizeof ch[i]);
13         len[i]=fail[i]=sz[i]=cnt[i]=0;
14     }
15     pam_cnt=1; last=0; fail[0]=1; fail[1]=1; len[1]=-1;
16 }
17 int getfail(int x,int i){
18     while(i-len[x]-1<0||s[i-len[x]-1]!=s[i])x=fail[x];
19     return x;
20 }
21 void extend(int c,int i) {
22     int f=getfail(last,i);
23     if(!ch[f][c]){
24         int p=++pam_cnt;
25         pos[p]=i;
26         len[p]=len[f]+2;
27         int q=getfail(fail[f],i);
28         fail[p]=ch[q][c];
29         sz[p]=sz[fail[p]]+1;
30         ch[f][c]=p;
31     }
32     last=ch[f][c];
33     ++cnt[last];

```

```

34 }
35 int main(){
36     scanf("%s",s+1);
37     int n=strlen(s+1);
38     pam_init();
39     for(int i=1;i<=n;++i)extend(s[i]-'a',i);
40     for(int i=2;i<=pam_cnt;++i){//枚举所有本质不同回文串
41         int r=pos[i],l=r-len[i]+1;//s[l,r]
42     }
43 }
44 }

```

8 计算几何

8.1 实数运算

```

1  typedef long double db;//便于修改double和long double
2  const int N=2e5+5;
3  const db eps=1e-18,pi=acos(-1);
4  int sgn(db x){return x<-eps?-1:x>eps;}//负数:-1,零:0,正数:1
5  int cmp(db a,db b){return sgn(a-b);}//小于:-1,等于:0,大于:1
6  void wt(db x){printf("%.5Lf",abs(x)<5e-6?0:x);}//避免输出负0(-0.0000)
7  void wtb(db x){wt(x);putchar(' ');}
8  void wtl(db x){wt(x);putchar('\n');}
9  //ceil(x)向上取整,floor(x)向下取整,round(x)四舍五入
10 //解方程 ax^2+bx+c=0, 解为x1,x2(x1<=x2)
11 bool solve_equation(db a,db b,db c,db &x1,db &x2){
12     db delta=b*b-4*a*c;
13     if(sgn(delta)==-1)return 0;
14     db q,t=sqrt(delta);
15     if(sgn(b)==-1)q=-0.5*(b-t);
16     else q=-0.5*(b+t);
17     x1=q/a;x2=c/q;if(cmp(x1,x2)==1)swap(x1,x2);
18     return 1;
19 }

```

8.2 点/向量类

```

1  struct P{//点、向量类
2      db x,y;

```

```

3     P(db a=0,db b=0):x(a),y(b){}
4     P operator + (P p)const{return P(x+p.x,y+p.y);}
5     P operator - (P p)const{return P(x-p.x,y-p.y);}
6     P operator * (db a)const{return P(x*a,y*a);}
7     P operator / (db a)const{return P(x/a,y/a);}
8     db len(){return sqrt(x*x+y*y);}
9     db len2(){return x*x+y*y;}
10    P rotate90(){return P(-y,x);}
11    P rotate270(){return P(y,-x);}
12    P rotate(db a){db c=cos(a),s=sin(a);return P(c*x-s*y,s*x+c*y);}//绕原点逆时针
    旋转a
13    P stretch(db l){return *this*l/len();}
14    void rd(){scanf("%Lf%Lf",&x,&y);}
15    void show(){printf("%.2Lf %.2Lf\n",x,y);}
16    friend P operator * (db a,P p){return P(p.x*a,p.y*a);}
17    bool operator<(const P&p)const{int c=cmp(x,p.x);return c?c==-1:cmp(y,p.y);}
18    bool operator==(const P&p)const{return cmp(x,p.x)==0&&cmp(y,p.y)==0;}
19 };
20 db dot(P a,P b){return a.x*b.x+a.y*b.y;}//点积
21 db cross(P a,P b){return a.x*b.y-a.y*b.x;}//叉积
22 db dis(P a,P b){return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));}//距离
23 db sqdis(P a,P b){return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);}//距离的平方
24 P lerp(P a,P b,db t){return a*(1-t)+b*t;}//直线上截取某一点连线
25 db get_angle(P a,P b){return abs(atan2(abs(cross(a,b)),dot(a,b)));}//获得向量OA和
    OB的夹角[0,pi/2]

```

8.3 平面最近点对

乱搞法：随机旋转角度，按 x 排序/按 $x \times y$ 排序，每个点枚举附近 100 个

```

1  typedef double db;
2  struct P{db x,y;}a[N];
3  int n;db ans=1e100;
4  bool cmp(P a,P b){
5      return a.x<b.x||a.x==b.x&&a.y<b.y;
6  }
7  db dis(P a,P b){
8      return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
9  }
10 P b[N];int h;
11 void merge(int l,int r){
12     if(l>=r)return;

```

```

13     int mid=l+r>>1;
14     merge(l,mid);merge(mid+1,r);
15     h=0;
16     for(int i=mid;i>=l;--i){
17         if(a[mid].x-a[i].x>ans)break;
18         b[++h]=a[i];
19     }
20     for(int i=mid+1;i<=r;++i){
21         if(a[i].x-a[mid].x>ans)break;
22         b[++h]=a[i];
23     }
24     sort(b+1,b+1+h,[](P p,P q){return p.y<q.y;});
25     for(int i=1,k=1;i<h;++i){
26         while(k<h&&a[k].y-a[i].y<ans)++k;
27         for(int j=i+1;j<=k;++j){
28             ans=min(ans,dis(b[i],b[j]));
29         }
30     }
31 }
32 int main(){
33     scanf("%d",&n);
34     for(int i=1;i<=n;++i)scanf("%lf%lf",&a[i].x,&a[i].y);
35     sort(a+1,a+1+n,cmp);
36     merge(1,n);
37     printf("%.2lf\n",ans/2);
38 }

```

8.4 点线关系

```

1 bool point_on_line(P p,P s,P t){//p在点向式直线s+kt上
2     return !sgn(cross(p-s,t));
3 }
4 bool point_on_seg(P p,P a,P b){//p在线段a-b上
5     return sgn(cross(p-a,b-a))==0&&sgn(dot(p-a,p-b))<=0;
6 }
7 P point_proj_line(P p,P s,P t){//p投影到点向式直线s+kt
8     return s+t*(dot(p-s,t)/t.len2());
9 }

```

8.5 线线关系

```

1 //判断两直线平行,方向向量为t1,t2
2 bool parallel(P t1,P t2){return !sgn(cross(t1,t2));}
3 //直线求交点
4 P line_int_line(P a,P u,P b,P v){//点向式a+ku,b+kv,需要先判断不平行
5     db t=cross(b-a,v)/cross(u,v);
6     return a+u*t;
7 }
8 //判断直线s+kt与线段ab是否相交 0:no 1:yes -1:交于端点
9 int line_int_seg(P s,P t,P a,P b){
10     int d1=sgn(cross(t,a-s));
11     int d2=sgn(cross(t,b-s));
12     if((d1^d2)==-2)return 1;// 1 xor -1 = -2
13     if(d1==0||d2==0)return -1;
14     return 0;
15 }
16 //判断线段ab,pq间是否有交点,有交点的话可以用直线交点函数求出交点
17 int seg_int_seg(P a,P b,P p,P q) {
18     int d1 = sgn( cross(b-a,p-a) ),d2 = sgn( cross(b-a,q-a) );
19     int d3 = sgn( cross(q-p,a-p) ),d4 = sgn( cross(q-p,b-p) );
20     if( (d1^d2)==-2 && (d3^d4)==-2 )return 1; //有交点,且交点不在端点
21     if( ( d1 == 0 && point_on_seg(p,a,b) )||
22         ( d2 == 0 && point_on_seg(q,a,b) )||
23         ( d3 == 0 && point_on_seg(a,p,q) )||
24         ( d4 == 0 && point_on_seg(b,p,q) ) )
25         return -1; //重合或交点在端点上
26     return 0;
27 }
28 }

```

8.6 极角排序

```

1 //可以整数(long long)运算,以原点(0,0)为极点进行极角排序
2 bool cmp(P a,P b){//角度划分为[0,180)和[180,360),再判断叉积>0
3     int f1=a.y>0||a.y==0&&a.x>0;
4     int f2=b.y>0||b.y==0&&b.x>0;
5     if(f1!=f2)return f1>f2;
6     return cross(a,b)>0;//后半句可以不加,表示角度相同时离原点距离小的在前
7     // return cross(a,b)>0||cross(a,b)==0&&a.len2()<b.len2();
8 }
9 //按左下角的点极角排序

```

```

10 void polar_angle_sort(P p[],int n,P O=P(0,0)){//以O为极点,极角排序p[1~n]
11     for(int i=1;i<=n;++i){
12         if(cmp(p[i].y,O.y)==-1||cmp(p[i].y,O.y)==0&&cmp(p[i].x,O.x)==-1){
13             O=p[i];//选取y最小,x最小的点为极点
14         }
15     }
16     sort(p+1,p+1+n,[&](P a,P b){
17         db t=cross(a-O,b-O);
18         return sgn(t)?t>0:sqdis(a,O)<sqdis(b,O);
19     });
20 }

```

8.7 多边形、凸包

```

1 bool cmpxy(P a,P b){
2     int k=cmp(a.x,b.x);
3     return k==-1||k==0&&cmp(a.y,b.y)==-1;
4 }
5 int convex_hull(P a[],int n,P p[]){//a[1~n]凸包,返回p[1~top]
6     static int sta[N]; int top=0;
7     sort(a+1,a+1+n,cmpxy);
8     for(int i=1;i<=n;++i){
9         while(top>=2&&sgn(cross(a[i]-a[sta[top-1]],a[sta[top]]-a[sta[top-1]]))
10             >=0)--top;
11         sta[++top]=i;
12     }
13     int k=top;
14     for(int i=n-1;i-->0){
15         while(top>k&&sgn(cross(a[i]-a[sta[top-1]],a[sta[top]]-a[sta[top-1]]))>=0)
16             --top;
17         sta[++top]=i;
18     }
19     if(n>1)--top;//去除首尾相连点
20     for(int i=1;i<=top;++i)p[i]=a[sta[i]];
21     return top;
22 }
23 //求多边形面积
24 db getS(P a[],int n){
25     db s=0;
26     for(int i=1;i<=n;++i)s+=cross(a[i],a[i+1]);
27     return abs(s)/2;//s>0:点是逆时针顺序,s<0:点是顺时针顺序

```

```

26 }
27 //求多边形重心
28 P polycenter(P a[],int n){
29     P ans(0,0); a[0]=a[n];
30     for(int i=0;i<n;i++)ans=ans+(a[i]+a[i+1])*cross(a[i],a[i+1]);
31     return ans/getS(a,n)/6;
32 }
33 //判点是否在多边形内或边上
34 bool point_in_polygon(P p,P a[],int n){
35     int s=0;a[n+1]=a[1];
36     for(int i=1;i<=n;i++){
37         P u=a[i],v=a[i+1];
38         if(point_on_seg(p,u,v))return 1;
39         if(cmp(u.y,v.y)<=0)swap(u,v);
40         if(cmp(p.y,u.y)>0||cmp(p.y,v.y)<=0)continue;
41         if(sgn(cross(v-p,u-p))==1)++s;
42     }
43     return s&1;
44 }

```

8.8 旋转卡壳

```

1 db rotating_calipers(P a[],int n,P ans[]){//凸包a[1~n](逆时针)
2     //返回凸包直径,ans[0~3]是最小矩形覆盖的四个点(逆时针)
3     db d=0,S=1e100; a[0]=a[n];
4     if(n==2)return dis(a[0],a[1]);//只有两个点的时候不能构成多边形
5     int j=2,l=0,r=0;
6     for(int i=0;i<n;++i){
7         while(cmp(cross(a[i+1]-a[i],a[j]-a[i]),cross(a[i+1]-a[i],a[j+1]-a[i]))
8             ==-1)j=(j+1)%n;
9         while(cmp(dot(a[i+1]-a[i],a[r+1]-a[i]),dot(a[i+1]-a[i],a[r]-a[i]))>=0)r=(
10             r+1)%n;
11         if(i==0)l=r;
12         while(cmp(dot(a[i+1]-a[i],a[l+1]-a[i]),dot(a[i+1]-a[i],a[l]-a[i]))<=0)l=(
13             l+1)%n;
14         d=max({d,dis(a[i],a[j]),dis(a[i+1],a[j])}); //只求直径的话其他代码可以删除
15         P v=a[i+1]-a[i];
16         P A=point_proj_line(a[l],a[i],v);
17         P B=point_proj_line(a[r],a[i],v);
18         P C=point_proj_line(a[r],a[j],v);
19         P D=point_proj_line(a[l],a[j],v);

```

```

17         db now=dis(A,B)*dis(B,C);
18         if(cmp(now,S)==-1)S=now,ans[0]=A,ans[1]=B,ans[2]=C,ans[3]=D;
19     }
20     return d;//d:凸包直径 S:最小矩形覆盖的矩形面积
21 }

```

8.9 半平面交

```

1  typedef long double db;
2  const db eps=1e-20,inf=1e20;
3  const int N=1e4+10;
4  int sgn(db x){return x<-eps?-1:x>eps;}
5  struct P{
6      db x,y;
7      P(db a=0,db b=0):x(a),y(b){}
8      P operator + (P b){return P(x+b.x,y+b.y);}
9      P operator - (P b){return P(x-b.x,y-b.y);}
10     P operator * (db b){return P(x*b,y*b);}
11     int up(){return sgn(y)>0||sgn(y)==0&&sgn(x)>0;}//用于极角排序
12 }c[N],p[N];
13 db cross(P a,P b){return a.x*b.y-a.y*b.x;}
14 db dot(P a,P b){return a.x*b.x+a.y*b.y;}
15 db angle(P a,P b){return atan2(cross(a,b),dot(a,b));}
16 P line_int_line(P a,P u,P b,P v){
17     db t=cross(b-a,v)/cross(u,v);
18     return a+u*t;
19 }
20 struct line{
21     P a,v;//直线a+kv,半平面是向量v左边的区域
22     int id;//有时需要一个id
23 }a[N];
24 P line_int_line(line a,line b){
25     return line_int_line(a.a,a.v,b.a,b.v);
26 }
27 int n,h;
28 bool on_right(P p,line l){
29     //条件为<0:重合的点代表的边不算;条件为<=0:重合的点代表的边算上
30     return sgn(cross(l.v,p-l.a))<0;
31 }
32 bool cmp(line l1,line l2){
33     int f1=l1.v.up(),f2=l2.v.up();

```



```

34     if(f1!=f2)return f1>f2;
35     int c=sgn(cross(l1.v,l2.v));
36     if(c!=0)return c>0;
37     return sgn(cross(l1.v,l2.a-l1.a))>=0;
38 }
39 int half_plane_intersection(line a[],P p[],int n){
40     //输入a[1~n]是半平面,返回h=r-l+1半平面a[1~h],交点p[1~h]
41     //线a[i]与a[i+1<=h?i+1:1]的交点是p[i]
42     //通常需要考虑添加边界a[++n]=line(P(0,inf),P(-1,0))等
43     static line q[N];
44     int l=1,r=0;
45     sort(a+1,a+1+n,cmp);
46     for(int i=1;i<=n;++i){
47         while(i<n&&sgn(angle(a[i].v,a[i+1].v))==0)++i;
48         while(r-l>=1&&on_right(p[r-1],a[i]))--r;
49         while(r-l>=1&&on_right(p[l],a[i]))++l;
50         q[++r]=a[i];
51         if(r-l>=1)p[r-1]=line_int_line(q[r-1],q[r]);
52     }
53     while(r-l>=1&&on_right(p[r-1],q[l]))--r;
54     if(r-l<=1)return 0;
55     p[r]=line_int_line(q[l],q[r]);
56     for(int i=1;i<=r;++i)a[i-l+1]=q[i];
57     for(int i=1;i<=r;++i)p[i-l+1]=p[i];
58     return r-l+1;
59 }

```

8.10 圆

```

1 struct circle{
2     P c;db r;
3     circle(P p=P(0,0),db x=1):c(p),r(x){}
4     P point(db a){return P(c.x+r*cos(a),c.y+r*sin(a));}
5 };
6 //过点p做圆的切线
7 int circletan(P p,circle C,P v[]){
8     P u=C.c-p; db d=u.len();
9     if(sgn(d-C.r)==-1)return 0;
10    if(sgn(d-C.r)==0){v[0]=u.rotate90();return 1;}
11    db a=asin(C.r/d);
12    v[0]=u.rotate(-a);

```

```

13     v[1]=u.rotate(a);
14     return 2;
15 }
16 //两圆共切线,返回切线条数,无数条返回-1, v1[i],v2[i]第i条切线在圆A,B上的切点
17 int circle_tan(circle A,circle B,P *v1,P *v2){
18     int cnt=0;
19     if(cmp(A.r,B.r)==-1)swap(A,B),swap(v1,v2);
20     db d2=sqdis(A.c,B.c);
21     db rsub=A.r-B.r;
22     db rsum=A.r+B.r;
23     if(sgn(d2-rsub*rsub)==-1)return 0;//内含
24     db a=atan2(B.c.y-A.c.y,B.c.x-A.c.x);
25     if(!sgn(d2)&&!sgn(A.r-B.r))return -1;//重合,无数切线
26     if(!sgn(d2-rsub*rsub)){//内切,找到切点
27         v1[cnt]=A.point(a);
28         v2[cnt]=B.point(a);
29         ++cnt;return 1;
30     }
31     db b=acos((A.r-B.r)/sqrt(d2));
32     v1[cnt]=A.point(a+b);v2[cnt++]=B.point(a+b);
33     v1[cnt]=A.point(a-b);v2[cnt++]=B.point(a-b);
34     if(!sgn(d2-rsum*rsum)){
35         v1[cnt]=A.point(a);
36         v2[cnt++]=B.point(pi+a);
37     }else if(sgn(d2-rsum*rsum)==1){
38         b=acos((A.r+B.r)/sqrt(d2));
39         v1[cnt]=A.point(a+b);v2[cnt++]=B.point(pi+a+b);
40         v1[cnt]=A.point(a-b);v2[cnt++]=B.point(pi+a-b);
41     }
42     return cnt;
43 }
44 //圆与直线交点,lerp(a,b,x1),lerp(a,b,x2)
45 bool circle_int_line(circle c,P a,P b,db &x1,db &x2){
46     P d=b-a;
47     db A=dot(d,d),B=dot(d,(a-c.c))*2.0,C=dot(a-c.c,a-c.c)-c.r*c.r;
48     return solve_equation(A,B,C,x1,x2);
49 }
50 //圆圆相交的交点
51 bool circle_int_circle(circle a,circle b,P &p1,P &p2){
52     db d=(a.c-b.c).len();
53     if(cmp(d,a.r+b.r)==1||cmp(d,abs(a.r-b.r))==1)return 0;

```

```

54     db l=((a.c-b.c).len2()+a.r*a.r-b.r*b.r)/(2*d);
55     db h=sqrt(a.r*a.r-l*l);
56     P v1=(b.c-a.c).strech(l),vh=v1.rotate90().strech(h);
57     p1=a.c+v1+vh;p2=a.c+v1-vh;
58     return 1;
59 }
60 //圆和三角形abo交的面积，o是圆心
61 db circle_int_triangle(circle c,P a,P b) {
62     if(sgn(cross(a-c.c,b-c.c))==0)return 0;
63     P q[5];int cnt=0;
64     db t0,t1; q[cnt++]=a;
65     if( circle_int_line(c,a,b,t0,t1) ) {
66         if(0<=t0&&t0<=1)q[cnt++]=lerp(a,b,t0);
67         if(0<=t1&&t1<=1)q[cnt++]=lerp(a,b,t1);
68     }
69     q[cnt++]=b;db s=0;
70     for(int i=1;i<cnt;++i) {
71         P z=(q[i-1]+q[i])*0.5;
72         if((z-c.c).len2()<=c.r*c.r)
73             s+=abs(cross(q[i-1]-c.c,q[i]-c.c))*0.5;
74         else
75             s+=c.r*c.r*get_angle(q[i-1]-c.c,q[i]-c.c)*0.5;
76     }
77     return s;
78 }
79 //圆与多边形相交的面积
80 db circle_int_polygon(circle C,P p[],int n) {
81     db s = 0;p[n+1]=p[1];
82     for(int i=1;i<=n;++i)
83         s+=circle_int_triangle(C,p[i],p[i+1])*sgn(cross(p[i]-C.c,p[i+1]-C.c));
84     return abs(s);
85 }
86 db circle_cover(vector<P>v,P &o){//最小圆覆盖,不得有重复点,时间复杂度O(N)
87     random_shuffle(v.begin(),v.end());
88     db r2=0;
89     for(int i=0;i<v.size();++i){
90         if((v[i]-o).len2()>r2){
91             o=v[i];r2=0;
92             for(int j=0;j<i;++j){
93                 if((v[j]-o).len2()>r2){
94                     o=(v[i]+v[j])*0.5; r2=(v[j]-o).len2();

```

```

95         for(int k=0;k<j;++k){
96             if((v[k]-o).len2(>r2){
97                 o=line_int_line((v[i]+v[j])*0.5,(v[i]-v[j]).rotate90
98                     (),(v[i]+v[k])*0.5,(v[i]-v[k]).rotate90());
99                 r2=(v[k]-o).len2();
100             }
101         }
102     }
103 }
104 }
105 return sqrt(r2);
106 }

```

8.11 公式

海伦公式：三角形三边为 a, b, c ，令 $p = \frac{a+b+c}{2}$, $S = \sqrt{p(p-a)(p-b)(p-c)}$

球缺：半径为 r 的球被切除高位 h 的球缺，剩下体积 $V = \frac{h^2(3r-h)\pi}{3}$ ，若球缺高为 h ，截面直径为 d ，则 $V = \frac{h(3d^2+4h^2)\pi}{24}$

平面图欧拉公式：对于连通的平面图，有区域数 $F = \text{点数} E - \text{边数} V + 1$ 。

9 博弈

9.1 必败态

必胜态：存在一个后继状态是必败态

必败态：不存在一个后继状态是必败态

复杂状态：当前状态根据后继状态的最坏情况来判断。比如有输、赢、平局，后继最差是输，当前就是赢，后继最差是平局，当前就是平局，后继最差是赢，当前就是输

9.2 SG 函数

必败状态的 $SG = 0$

一个状态的 SG 函数值等于其后继状态 SG 值的 mex ，即 $SG(x) = mex_{x \rightarrow y}(SG(y))$

多个组合博弈游戏的和的 SG 值等于其所有子游戏 SG 值的异或和

```

1 int mex(vector<int>&v){
2     sort(v.begin(),v.end());
3     int ans=0;
4     for(int x:v){
5         if(x==ans)++ans;

```

```

6     else if(x>ans)return ans;
7 }
8 return ans;
9 }

```

9.3 反 SG 游戏

最先不能操作的人获胜

必胜： $SG \neq 0$ 且至少有一个子游戏的 $SG > 1$ ，或 $SG = 0$ 且每一个子游戏的 $SG \leq 1$

否则必败

9.4 巴什博弈

n 个石头，每次可以拿 1 到 m 个，不能拿的失败， $SG = n\%(m+1)$

9.5 Nim 游戏

n 堆石子，第 i 堆有 a_i 个，两人轮流操作，每次选一堆石头并拿走任意个，先不能拿的失败相当于 n 个子游戏，第 i 堆石子的 SG 值是 a_i ，总游戏的 $SG = \text{xor}_{i=1}^n a_i$

$SG \neq 0$ 时先手拿最多石头的方案： $O(n)$ 枚举 i ， a_i 应该变为 $\text{xorsum} \oplus a_i$

必败态只拿一个，并让对手也只能只拿一个的方案： $O(\log C)$ 枚举所有 $\text{lowbit} = lb$ ，拿一个之后 xorsum 会变为 $\text{xorsum} \oplus lb \oplus (lb-1)$ ，并 $O(n)$ 判断之后状态对手是否最多拿一个

9.6 阶梯 Nim

每次选某一堆，拿走任意个并放到前一堆，第 0 堆不能拿，即 $a[i-1] += x, a[i] -= x$

后手可以让先手的偶数堆无效：当 A 把 $2k$ 堆拿出 x 个，B 就可以从 $2k-1$ 堆拿出 x 个相当于只看奇数堆的 nim 游戏，设先手是 A

当 A 的奇数堆 nim 游戏必败时，B 会使 A 的偶数堆无效

当 A 的奇数堆 nim 游戏必胜时，A 会先把奇数堆 nim 游戏变为必败，再使 B 的偶数堆无效

9.7 Nim-k

每次选 k 堆，在这 k 堆拿走分别任意个（可以不同），先不能拿的失败

若二进制每一位的出现次数都是 $k+1$ 的倍数则必败，否则必胜

9.8 威佐夫博弈

有两堆物品，两人轮流操作，每次操作从一堆中取任意个，或从两堆中取相同多个，先不能操作的人失败

必败态为: $(1, 2), (3, 5), (4, 7), (6, 10) \dots (\lfloor \frac{k(1+\sqrt{5})}{2} \rfloor, \lfloor \frac{k(1+\sqrt{5})}{2} \rfloor + k)$

每个 (a_i, b_i) 中 $a_i = \text{mex}(\{0, a_1, b_1, a_2, b_2, \dots, a_{i-1}, b_{i-1}\})$

9.9 树上删边游戏

给定一棵有根树，两人轮流操作，每次操作选择一条边，删掉该边以及与根不连通的所有点和边（整棵子树），不能操作的人时失败，则叶子的 $SG = 0$ ，非叶子的 $SG(x) = \text{xor}_{y \in \text{son}(x)} [SG(y) + 1]$

多棵竹子的删边游戏等价于 nim 游戏，等价于这些竹子的根合并为一个点后的树上删边游戏

克朗原理: 一个结点的所有儿子的子树都是竹子，则该节点子树可以等价于一根竹子，长度为各分支长度的异或和

9.10 无向图删边游戏

给定一张无向图，定义一些点在地上，每次选择一条边，删掉该边以及与所有地板上的点不连通的点和边，不能操作的人失败

费森原理: 偶环可以直接缩成一个点（类似边双连通分量缩点），奇环缩成一个点并额外挂着一一条长为 1 的竹子

10 基础算法

10.1 二分查找

构造单调 *bool* 函数 $check(x)$, 返回值为小的全 1, 大的全 0, 形如 11...1100...00

可以用二分查找算出最大的 x 使得 $check(x) = 1$, 反过来也同理

```

1 //第k小: <=x的个数>=k的最小的x
2 //第k大: >=x的个数>=k的最大的x
3 while(l<=r){
4     int mid=l+r>>1;// l+(r-l>>1)
5     if(check(mid))ans=mid,l=mid+1;
6     else r=mid-1;
7 }
8
9 vector<int>v;//求vector中[l,r]区间有几个数字
10 int cnt=upper_bound(v.begin(),v.end(),r)-lower_bound(v.begin(),v.end(),l);
    
```

10.2 去重, 离散化

```

1 int main(){
2     int a[N];//数组离散化
    
```

```

3    sort(a+1,a+1+h);//a[1...h]中包括有所有可能的数字
4    h=unique(a+1,a+1+h)-a-1;
5    x=lower_bound(a+1,a+1+h,x)-a;//将x离散化后的值
6    vector<int>v;//vector离散化
7    sort(v.begin(),v.end());
8    v.erase(unique(v.begin(),v.end()),v.end());
9    x=lower_bound(v.begin(),v.end(),x)-v.begin()+1;
10 }

```

10.3 取出第 k 小/中位数

```

1  int a[N];
2  nth_element(a+1,a+k,a+1+n);//O(N)
3  //未排序,但保证a[1]~a[k-1]比a[k]小,a[k+1]~a[n]比a[k]大

```

10.4 bitset

```

1  int main(){    bitset<100>a;
2      a.set();/*全设为1*/ a.reset();/*全设为0*/
3      a[2]=1;/*访问下标*/ cout<<a<<endl;//打印bitset,a[0]在最右边
4      a.count();/*1的个数*/a.any();/*有1?1:0*/a.flip();//全部取反
5  }

```

10.5 builtin 函数

```

1  __builtin_popcount(x);//二进制1的个数
2  __builtin_popcountll(x);//ll表示参数为long long,后面几个函数也可以加ll
3  __builtin_clz(x);//二进制前导零个数
4  __builtin_ctz(x);//二进制后导零个数
5  __builtin_parity(x);//二进制1的个数是奇数?1:0
6  __builtin_ffs(x);//二进制最后一个1在第几位,从1开始,ffs(1)=1,ffs(0)=0
7  __builtin_sqrt(x);//x位double,开根号,比sqrt(x)快

```

10.6 归并排序求逆序对

```

1  int a[N],temp[N];ll ans;
2  void merge(int l,int r){
3      if(l>=r)return;
4      int mid=l+r>>1;

```

```

5     merge(l,mid);merge(mid+1,r);
6     int p=l,q=mid+1;
7     for(int i=l;i<=r;++i){
8         if(p>mid||q<=r&& a[q]<a[p]){//左边用完了or右边没用完and右边小
9             ans+=mid-p+1;//逆序对个数
10            temp[i]=a[q++];
11        }
12        else temp[i]=a[p++];
13    }
14    for(int i=l;i<=r;++i)a[i]=temp[i];
15 }

```

10.7 计数排序

值域较小时适用，时间复杂度为 $O(N + C)$

```

1  int n,a[N],cnt[N],C;//值域大小为[1,C]
2  void count_sort(int a[],int n){//计数排序a[1~n]
3      for(int i=1;i<=n;++i)++cnt[a[i]];
4      int h=0;
5      for(int i=1;i<=C;++i){
6          while(cnt[i--])a[++h]=i;
7      }
8  }
9  int rk[N];
10 void count_sort_rank(int a[],int n){//求出rk[i]=a[i]是第几小
11     for(int i=1;i<=n;++i)++cnt[a[i]];
12     for(int i=1;i<=C;++i)cnt[i]+=cnt[i-1];
13     for(int i=1;i<=n;++i)rk[i]=cnt[a[i]]--;
14 }

```

10.8 基数排序

```

1  const int N=1e5+5,base=(1<<15)-1;
2  int n,a[N],cnt[base+1],id[N],t[N];//id[i]=第i小的位置
3  void radix_sort(int a[],int n){//int数组a[1~n],值域[0,2^30-1]
4      for(int i=1;i<=n;++i)++cnt[a[i]&base];
5      for(int i=1;i<=base;++i)cnt[i]+=cnt[i-1];
6      for(int i=n;i-->0)id[cnt[a[i]&base]--]=i;
7      memset(cnt,0,sizeof cnt);
8      for(int i=1;i<=n;++i)++cnt[a[i]>>15],t[i]=id[i];

```



```

9     for(int i=1;i<=base;++i)cnt[i]+=cnt[i-1];
10    for(int i=n;i-->0)id[cnt[a[t[i]]>>15]--]=t[i];
11    for(int i=1;i<=n;++i)t[i]=a[id[i]];
12    for(int i=1;i<=n;++i)a[i]=t[i];
13 }

```

10.9 维护区间并集

```

1  map<int,int>mp;//对于区间[l,r] : mp[l]=r
2  // [1,2]和[3,4]不能合并版本
3  void insert(ll l,ll r){
4      auto it=mp.upper_bound(l);
5      if(it!=mp.begin()){
6          --it;
7          if(it->second>=l){
8              l=it->first;
9              r=max(r,it->second);
10             mp.erase(it);
11         }
12     }
13     for(it=mp.lower_bound(l);it!=mp.end()&&it->first<=r;it=mp.erase(it))
14         r=max(r,it->second);
15     mp[l]=r;
16 }
17 // [1,2]和[3,4]能合并为[1,4]版本
18 void insert(ll l,ll r){
19     auto it=mp.upper_bound(l);
20     if(it!=mp.begin()){
21         --it;
22         if(it->second>=l-1){
23             l=it->first;
24             r=max(r,it->second);
25             mp.erase(it);
26         }
27     }
28     for(it=mp.lower_bound(l);it!=mp.end()&&it->first<=r+1;it=mp.erase(it))
29         r=max(r,it->second);
30     mp[l]=r;
31 }

```

10.10 计算天数

```

1  int isleap(int x){return x%4==0&&x%100!=0||x%400==0;}
2  int month[]={0,31,28,31,30,31,30,31,31,30,31,30,31};
3  int calcdays(int y1,int m1,int d1,int y2,int m2,int d2){//从y1/m1/d1到y2/m2/d2过了
    几天
4      int sum=0;
5      if(y1==y2){
6          if(m1==m2)return d2-d1;
7          month[2]=isleap(y1)?29:28;
8          sum=month[m1]-d1;
9          for(int i=m1+1;i<m2;++i)sum+=month[i];
10         sum+=d2;
11     }
12     else{
13         month[2]=isleap(y1)?29:28;
14         sum=month[m1]-d1;
15         for(int i=m1+1;i<=12;++i)sum+=month[i];
16         for(int i=y1+1;i<y2;++i)sum+=isleap(i)?366:365;
17         month[2]=isleap(y2)?29:28;
18         for(int i=1;i<m2;++i)sum+=month[i];
19         sum+=d2;
20     }
21     return sum;
22 }

```

蔡勒公式:

$c = \lfloor \frac{year}{100} \rfloor$ = 年份前两位, $y = year \% 100$ = 年份后两位, m = 月, d = 日

$w = (\lfloor \frac{c}{4} \rfloor - 2c + y + \lfloor \frac{y}{4} \rfloor + \lfloor \frac{13(m+1)}{5} \rfloor + d - 1) \bmod 7$

$year$ 年 m 月 d 日是星期 w , ($w = 0$ 是星期日)

10.11 表达式求值

```

1  inline ll fun(ll x,ll y,char c){
2      switch(c){
3          case '+':return x+y;
4          case '-':return x-y;
5          case '*':return x*y;
6          case '/':{
7              if(y==0)throw -1;//div 0
8              if(x%y!=0)throw 1;//不能整除
9              return x/y;

```

```

10         }
11         case '%':{
12             if(y==0)throw -1;// div 0
13             return x%y;
14         }
15     }
16 }
17 inline ll calc(string s){// "1+2+-3*4+-(3+4*1/2+4+5)"
18     s="("+s+")";
19     vector<ll>num,flag({1});
20     vector<char>op;
21     for(unsigned i=0;i<s.size();++i){
22         if(isdigit(s[i])){
23             ll x=s[i]-'0';
24             while(i+1<s.length()&&isdigit(s[i+1]))x=x*10+s[i+1]-'0',++i;
25             if(flag.back()=='-')x=-x;
26             flag.pop_back();
27             num.push_back(x);
28         }
29         else{
30             if(s[i]=='+'||s[i]=='-'){
31                 if(num.size()!=op.size()){
32                     while(!op.empty()&&op.back()!='('){
33                         ll x=num.back();num.pop_back();
34                         char c=op.back();op.pop_back();
35                         try{
36                             num.back()=fun(num.back(),x,c);
37                         }
38                         catch(int x){
39                             throw x;
40                         }
41                     }
42                     op.push_back(s[i]);
43                     flag.push_back(1);
44                 }
45                 else if(s[i]=='-')flag.back()=-flag.back();
46             }
47             else if(s[i]=='*'||s[i]=='/'||s[i]=='%'){
48                 if(!op.empty()&&op.back()!='('){
49                     if(op.back()=='+'||op.back()=='-'){
50                         op.push_back(s[i]);

```

```
51         }
52         else{// * / %
53             int x=num.back();num.pop_back();
54             char c=op.back();op.pop_back();
55             try{
56                 num.back()=fun(num.back(),x,c);
57             }
58             catch(int x){
59                 throw x;
60             }
61             op.push_back(s[i]);
62         }
63     }
64     else op.push_back(s[i]);
65     flag.push_back(1);
66 }
67 else if(s[i]=='('){
68     num.push_back(0);op.push_back('(');
69     flag.push_back(1);
70 }
71 else if(s[i]==')'){
72     while(!op.empty()&&op.back()!='('){
73         ll y=num.back();num.pop_back();
74         char c=op.back();op.pop_back();
75         try{
76             num.back()=fun(num.back(),y,c);
77         }
78         catch(int x){
79             throw x;
80         }
81     }
82     ll x=num.back();num.pop_back();
83     if(flag.back()==-1)x=-x;
84     flag.pop_back();
85     num.back()=x;op.pop_back();
86 }
87 else{
88     throw 2;//不合法运算符
89 }
90 }
91 }
```

```

92     return num.front();
93 }

```

11 模拟游戏

11.1 算 24 点

```

1  int a[5],op[4];//options: 1+ 2- 3* 4/
2  const int goal=24;
3  double f(double a,double b,int op){
4      switch(op){
5          case 0:return a+b;
6          case 1:return a-b;
7          case 2:return a*b;
8          default:
9              if(abs(b)<1e-6)throw -1;
10             return a/b;
11     }
12 }
13 int g(int a,int b,int op){
14     switch(op){
15         case 0:return a+b;
16         case 1:return a-b;
17         case 2:return a*b;
18         default:
19             if(b==0||a%b!=0)throw -1;
20             return a/b;
21     }
22 }
23 const char *ch="+-*/";
24 bool find_double_one(){
25     sort(a+1,a+1+4);
26 do{
27     for(int i=0;i<4;++i)
28     for(int j=0;j<4;++j)
29     for(int k=0;k<4;++k){
30         double s;
31         try{s=f(f(f(a[1],a[2],i),a[3],j),a[4],k);}
32         catch(...){s=-1;}
33         if(abs(s-goal)<1e-6){
34             printf("( (%d%c%d)%c%d)%c%d\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);

```

```

35         return 1;
36     }
37
38     try{s=f(f(a[1],a[2],i),f(a[3],a[4],k),j);}
39     catch(...){s=-1;}
40     if(abs(s-goal)<1e-6){
41         printf("(%d%c%d)%c(%d%c%d)\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
42         return 1;
43     }
44
45     try{s=f(f(a[1],f(a[2],a[3],j),i),a[4],k);}
46     catch(...){s=-1;}
47     if(abs(s-goal)<1e-6){
48         printf("(%d%c(%d%c%d))%c%d\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
49         return 1;
50     }
51
52     try{s=f(a[1],f(a[2],f(a[3],a[4],k),j),i);}
53     catch(...){s=-1;}
54     if(abs(s-goal)<1e-6){
55         printf("(%d%c(%d%c(%d%c%d))\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
56         return 1;
57     }
58
59     try{s=f(a[1],f(f(a[2],a[3],j),a[4],k),i);}
60     catch(...){s=-1;}
61     if(abs(s-goal)<1e-6){
62         printf("(%d%c((%d%c%d)%c%d)\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
63         return 1;
64     }
65 }
66 }while(next_permutation(a+1,a+5));
67     return 0;
68 }
69 bool find_int_one(){
70     sort(a+1,a+1+4);
71     do{
72         for(int i=0;i<4;++i)
73         for(int j=0;j<4;++j)
74         for(int k=0;k<4;++k){
75             int s=g(g(g(a[1],a[2],i),a[3],j),a[4],k);

```

```

76     if(s==goal){
77         printf("( (%d%c%d)%c%d)%c%d\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
78         return 1;
79     }
80     s=g(g(a[1],a[2],i),g(a[3],a[4],k),j);
81     if(s==goal){
82         printf("( (%d%c%d)%c(%d%c%d)\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
83         return 1;
84     }
85     s=g(g(a[1],g(a[2],a[3],j),i),a[4],k);
86     if(s==goal){
87         printf("( (%d%c(%d%c%d))%c%d\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
88         return 1;
89     }
90     s=g(a[1],g(a[2],g(a[3],a[4],k),j),i);
91     if(s==goal){
92         printf("( (%d%c(%d%c(%d%c%d)))\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
93         return 1;
94     }
95     s=g(a[1],g(g(a[2],a[3],j),a[4],k),i);
96     if(s==goal){
97         printf("( (%d%c((%d%c%d)%c%d)\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
98         return 1;
99     }
100 }
101 }while(next_permutation(a+1,a+5));
102     return 0;
103 }

```

11.2 德州扑克

判断牌型

```

1  struct poker{
2      int a,b;//a=牌的数值(必须为2~14),b=牌的花色(1~4),也可以是任意四个不同的数
3  };
4  int hand_type(poker v[]){//判断v[0~4]这五张卡的牌型
5      static int a[5],s;
6      for(int i=0;i<5;++i)a[i]=v[i].a;
7      sort(a,a+5);
8      bool straight=1,flush=1;//是顺子 / 是同花
9      for(int i=1;i<5;++i){

```

```

10     if(a[i]-a[0]!=i)straight=0;
11     if(v[i].b!=v[0].b)flush=0;
12 }
13 if(a[0]==2&&a[1]==3&&a[2]==4&&a[3]==5&&a[4]==14)straight=1;//特判A2345
14 if(straight&&flush){
15     if(a[4]==14&&a[3]==13)s=10;//royal flush 皇家同花顺(10JQKA)
16     else s=9;//straight flush 同花顺
17 }
18 else if(straight)s=5;//straight 顺子
19 else if(flush)s=6;//flush 同花
20 else{
21     int cnt[15]={0};
22     int mx=0;//出现次数最多的牌的
23     for(int i=0;i<5;++i)++cnt[a[i]],mx=max(mx,cnt[a[i]]);
24     if(mx==4)s=8;//four of a kind 四条(炸弹)
25     else if(mx==3){
26         int f2=0;
27         for(int i=0;i<=4;++i){
28             if(cnt[a[i]]==2)f2=1;
29         }
30         if(f2)s=7;//full house 葫芦(三带二)
31         else s=4;//three of a kind 三条(三带一)
32     }
33     else if(mx==2){
34         int c=0;
35         for(int i=0;i<=4;++i){
36             if(cnt[a[i]]==2)++c;
37         }
38         if(c==4)s=3;//two pairs 两对
39         else s=2;//pair 一对
40     }
41     else s=1;//high card 高牌(散牌)
42 }
43 return s;
44 }

```

相同牌型比大小

```

1 int cmp_same_type(poker c[],poker d[],int id){//双方牌型均为id,比大小
2     static int a[5],b[5];
3     for(int i=0;i<5;++i)a[i]=c[i].a,b[i]=d[i].a;
4     switch(id){

```



```

5      case 5://straight 顺子
6      case 9://straight flush 同花顺
7      {
8          sort(a,a+5);sort(b,b+5);
9          int u=a[4],v=b[4];
10         if(a[0]==2&&a[4]==14)u=5;
11         if(b[0]==2&&b[4]==14)v=5;
12         if(u!=v)return u>v?1:-1;
13         return 0;
14     }
15     case 1://highcard 高牌
16     case 6://flush 同花
17     {
18         sort(a,a+5);sort(b,b+5);
19         for(int i=4;i>=0;--i){
20             if(a[i]!=b[i])return a[i]>b[i]?1:-1;
21         }
22     }
23     case 10://royal flush 皇家同花顺
24         return 0;
25     default:
26         //      case 2://pair 一对
27         //      case 3://two pairs 两对
28         //      case 4://three of a kind 三条
29         //      case 7://full house 葫芦
30         //      case 8://four of a kind 四条
31         {
32             int cnt1[15]={0},cnt2[15]={0};
33             static int t1[5],t2[5];
34             int h1=0,h2=0,p1=4,p2=4,mx=0;
35             for(int i=0;i<5;++i){
36                 ++cnt1[a[i]];++cnt2[b[i]];
37                 mx=max(mx,cnt1[a[i]]);
38             }
39             for(int i=0;i<5;++i){
40                 if(cnt1[a[i]]==mx)t1[p1--]=a[i];
41                 else t1[h1++]=a[i];
42                 if(cnt2[b[i]]==mx)t2[p2--]=b[i];
43                 else t2[h2++]=b[i];
44             }
45             if(id==2||id==3)sort(t1,t1+h1),sort(t2,t2+h2);

```

```

46         if(id==3)sort(t1+h1,t1+5),sort(t2+h2,t2+5);
47         for(int i=4;i>=0;--i){
48             if(t1[i]!=t2[i])return t1[i]>t2[i]?1:-1;
49         }
50         return 0;
51     }
52 }
53 }

```

全手牌比大小

```

1  int cmp_hand(poker a[],poker b[]){//五张手牌a[0~4],b[0~4]比大小,返回 -1:< 0:=
    1:>
2      int u=hand_type(a),v=hand_type(b);
3      if(u!=v)return u>v?1:-1;//先比牌型
4      return cmp_same_type(a,b,u);//同牌型再比牌大小
5  }

```

结算筹码

```

1  void calc_money(int n,int rank[],long long c[],long long ans[]){
2      //数组下标 1~n,rank[i]=第i个人的排名是第几大,rank[i]=1是最大的,可能存在并列
3      //c[i]=第i个人出的筹码      ans[i]=结算后第i个人获得的筹码
4      static int id[N];
5      for(int i=1;i<=n;++i)id[i]=i;
6      sort(id+1,id+1+n,[&](int i,int j){return c[i]<c[j];});
7      map<int,int>cnt;//cnt[i]=排名为i的人数
8      for(int i=1;i<=n;++i)++cnt[rank[i]];
9      int rk1=cnt.begin()->first;
10     long long now=0,sub=0;
11     for(int i=1;i<=n;++i){
12         int t=id[i];
13         long long x=c[t]-sub;
14         now+=x*(n+1-i)/cnt.begin()->second;
15         ans[t]=rk1==rank[t]?now:0;
16         if(--cnt[rank[t]]){
17             cnt.erase(rank[t]);
18             if(rk1==rank[t]&&cnt.size()){
19                 rk1=cnt.begin()->first;
20                 now=0;
21             }
22         }
23         sub+=x;

```

```

24     }
25 }

```

12 其他技巧

12.1 C++ 编译命令

```

1 DEV C++使用C++11 工具->编译选项
2 编译时加入以下命令(√):-std=c++11
3 debug:工具->编译选项->代码生成/优化->产生调试信息YES
4 无限栈空间:#pragma comment(linker, "/STACK:102400000,102400000")
5 或在编译命令中加入 -Wl,--stack=102400000

```

12.2 快读快写

```

1 ll rd(){
2     char c=getchar();ll x=0;bool f=1;
3     for(;c<'0' || c>'9';c=getchar())f|=c=='-';
4     for(;c>='0'&&c<='9';c=getchar())x=(x<<3)+(x<<1)+(c^48);
5     return f?-x:x;
6 }
7 template<typename T>bool read(T &s){
8     ll x=0;bool f=0;char c=getchar();
9     for(;c<'0' || c>'9';c=getchar()){
10         if(c==EOF)return 0;f|=c=='-';
11     }
12     for(;c>='0'&&c<='9';c=getchar())x=(x<<3)+(x<<1)+(c^48);
13     if(c!='.')s=f?-x:x;
14     else{
15         double y=0,k=0.1;
16         for(c=getchar();c>='0'&&c<='9';c=getchar())y+=(c^48)*k,k*=0.1;
17         s=f?-x-y:x+y;
18     }
19     return 1;
20 }
21 void wt(ll x){
22     static char a[44];
23     short h=0;
24     if(x<0)x=-x,putchar('-');
25     do{

```

```

26     a[++h]='0'+x%10;x/=10;
27 }while(x);
28 for(int i=h;i-->0)putchar(a[i]);
29 }
30 inline void wt1(ll x){wt(x);puts("");}
31 inline void wt64(ll x){wt(x);putchar(' ');}

```

fread/fwrite

```

1 struct FastIO{//不可与scanf,printf,getchar,putchar,gets,puts,cin,cout混用
2     private:
3         static const int BUFSIZE=1e5;//BUFSIZE不需要改
4         char buf[BUFSIZE];int pos,len;//读入buffer(缓冲器)以及读入指针
5         int wpos;char wbuf[BUFSIZE];//输出指针以及输出buffer
6         #define gc() (pos==len&&(len=(pos=0)+fread(buf,1,BUFSIZE,stdin),!len)?EOF:buf
          [pos++])
7         #define pc(c) (wpos==BUFSIZE?fwrite(wbuf,1,BUFSIZE,stdout),wpos=0,wbuf[wpos
          ++]=c:wbuf[wpos++]=c)
8     public:
9         FastIO():wpos(0),pos(0),len(0){}
10        ~FastIO(){if(wpos)fwrite(wbuf,1,wpos,stdout),wpos=0;}
11        char getc(){return gc();};//读取char,直接=gc()也可以
12        void putc(char c){pc(c);};//输出字符,直接pc(c)也可以
13        long long rd(){//读取long long
14            long long x=0;char c=gc();bool f=0;
15            for(;c<'0' || c>'9';c=gc())f|=c=='-';
16            for(;c>='0'&& c<='9';c=gc())x=(x<<3)+(x<<1)+(c^48);
17            return f?~x+1:x;
18        }
19        template<typename T>bool read(T &x){//多测读整数while(io.read(n))work();本地
          测试请输入两次ctrl Z
20            x=0;char c=gc();bool f=0;
21            for(;c<'0' || c>'9';c=gc()){if(c==EOF)return 0;f|=c=='-';}
22            for(;c>='0'&& c<='9';c=gc())x=(x<<3)+(x<<1)+(c^48);
23            if(f)x=-x; return 1;
24        }
25        template<typename T>void wt(T x){//输出整数
26            static char a[22];
27            if(x<0)pc('-'),x=-x;short h=0;
28            for(a[++h]='0'+x%10,x/=10;x/=10)a[++h]='0'+x%10;
29            while(h)pc(a[h--]);
30        }

```

```

31  template<typename T>void wtl(T x){wt(x);pc('\n');};//write line输出整数并换行
32  template<typename T>void wtb(T x){wt(x);pc(' ');};//write blank输出整数并空格
33  int gets(char *s){int l=0;char c=gc();for(;c<=' ';c=gc());for(;c>' ';c=gc())s
    [l++]=c;s[l]=0;return l;}
34  void puts(const char *s){const char *p=s;while(*p)pc(*p++);};//输出字符串 (不
    带换行)
35  template<typename T>FastIO & operator >> (T &a){return read(a),*this;};//io>>a
    >>b;只能输入整数
36  template<typename T>FastIO & operator << (T a){return wtb(a),*this;};//io<<a<<
    b;输出整数并带有空格
37 }io;//本地测试出入结束后请输入一次ctrl Z

```

12.3 对拍

```

1  查看是否配置g++编译环境:
2  右键此电脑,属性,高级系统设置,环境变量,下面的系统变量,找到PATH,选中,编辑
3  新建一条:.....\Dev-Cpp\MinGW64\bin
4  新建.txt文件,后缀名改为.bat,也可以右键.bat文件->编辑
5      g++ -std=c++11 rand.cpp -o rand.exe
6      g++ -std=c++11 std.cpp -o std.exe
7      g++ -std=c++11 tested.cpp -o tested.exe
8      :loop
9      rand.exe >in.txt
10     std.exe <in.txt >out.txt
11     tested.exe <in.txt >tested.txt
12     fc out.txt tested.txt
13     if not errorlevel 1 goto loop
14     pause
15     goto loop

```