

目录

1	数学	1	2.8.1	原根	14
1.1	FWT	1	2.8.2	BSGS	14
1.2	多项式	1	2.9	杂项	14
1.2.1	FFT	1	2.9.1	大数整除小数取模	14
1.2.2	NTT	2	2.9.2	立方根复杂度求 mobius 函数	14
1.2.3	拉格朗日插值	5	2.9.3	直接求 euler 函数	14
1.2.4	连通图计数	5	3	数据结构	16
1.3	线性代数	5	3.1	点分治	16
1.3.1	高斯消元	5	3.2	树链剖分	16
1.3.2	矩阵树定理	6	3.3	主席树	16
1.3.3	线性基	6	3.4	线段树合并	17
1.4	反演与容斥	6	3.5	zkw 线段树	17
1.4.1	二项式反演	6	3.6	矩形面积并	17
1.4.2	单位根反演	7	4	图论	18
1.4.3	Min-Max容斥	7	4.1	2-sat	18
1.5	组合数	7	4.2	Dinic 网络流	18
1.5.1	常用组合数	7	4.3	Dijkstra 费用流	18
1.5.2	斯特林数	7	4.4	二分图最大带权匹配	19
1.5.3	欧拉数	8	4.5	带花树	20
1.6	线性规划	8	4.6	Hall 定理	21
1.6.1	对偶原理	8	5	字符串	22
1.7	杂项	8	5.1	后缀自动机	22
1.7.1	约瑟夫环	8	5.2	广义后缀自动机	22
1.7.2	杨辉三角行区间和	9	5.2.1	对 Trie 建自动机	22
1.7.3	辛普森积分	9	5.2.2	对多模式串建自动机	22
1.7.4	纳什均衡	9	5.3	AC 自动机	22
1.7.5	康托展开	9	5.4	后缀数组	23
1.7.6	常用NTT素数及原根	9	5.4.1	倍增	23
2	数论	10	5.4.2	SAIS	23
2.1	常见预处理与快速幂	10	5.5	马拉车	24
2.2	因数分解与素性判定	10	6	动态规划	25
2.2.1	朴素因数分解	10	6.1	数位 DP	25
2.2.2	Miller-Rabin 与 Pollard-Rho	10	7	几何	26
2.3	筛法	11	8	杂项	35
2.3.1	线性筛	11	8.1	java 高精度	35
2.3.2	Min25 筛	11	8.2	重载 umap	35
2.4	扩展欧几里得	12	8.3	bitset	35
2.5	扩展欧拉定理	13	8.4	模拟退火	35
2.6	中国剩余定理	13	8.5	对拍	36
2.6.1	两个数的 crt	13	8.5.1	windows 对拍	36
2.6.2	exCRT	13	8.5.2	linux 对拍	36
2.7	卢卡斯定理	13	8.6	快读	36
2.7.1	模素数卢卡斯	13	8.7	随机	36
2.7.2	扩展卢卡斯	13	8.8	python	36
2.8	原根与离散对数	14			

1 数学

1.1 FWT

矩阵表示

or 形式 (子集卷积):

$$T_{ij} = [i|j = i] = [j \in i]$$

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

and 形式 (超集卷积):

$$T_{ij} = [i \& j = i] = [i \in j]$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

xor 形式 (T 与自身互为逆矩阵):

$$T_{ij} = (-1)^{\text{parity}(i \oplus j)}$$

$$\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

```
1 using ll = long long;
2
3 // or
4 void FWT(ll *a, int len, int inv) {
5     for (int h = 1; h < len; h <= 1) {
6         for (int i = 0; i < len; i += (h << 1)) {
7             for (int j = 0; j < h; ++j) {
8                 a[i + j + h] += a[i + j] * inv;
9             }
10        }
11    }
12 }
13 // and
14 void FWT(ll *a, int len, int inv) {
15     for (int h = 1; h < len; h <= 1) {
16         for (int i = 0; i < len; i += (h << 1)) {
17             for (int j = 0; j < h; ++j) {
18                 a[i + j] += a[i + j + h] * inv;
19             }
20        }
21    }
22 }
23 // xor
24 void FWT(ll *a, int len, int inv) {
25     for (int h = 1; h < len; h <= 1) {
26         for (int i = 0; i < len; i += (h << 1)) {
27             for (int j = 0; j < h; ++j) {
28                 ll x = a[i + j], y = a[i + j + h];
29                 a[i + j] = x + y, a[i + j + h] = x - y;
30                 if (inv == -1)
31                     a[i + j] /= 2, a[i + j + h] /= 2;
32             }
33        }
34    }
35 }
```

1.2 多项式

1.2.1 FFT

```
1 #include <bits/stdc++.h>
2 #include <vector>
3 using namespace std;
4
```

```
5 using db = double;
6 using ll = long long;
7
8 int mod;
9
10 namespace FFT {
11     const db pi = acos(-1.0);
12     struct Comp {
13         db x, y;
14         Comp() { x = 0.0, y = 0.0; }
15         Comp(db _x, db _y):x(_x),y(_y) {}
16         Comp operator + (const Comp&rhs) const {
17             return Comp(x+rhs.x, y+rhs.y);
18         }
19         Comp operator - (const Comp&rhs) const {
20             return Comp(x-rhs.x, y-rhs.y);
21         }
22         Comp operator * (const Comp&rhs) const {
23             return Comp(x*rhs.x-rhs.y*y, x*rhs.y+y*rhs.x);
24         }
25     };
26     Comp conj(const Comp&rhs) {
27         return Comp(rhs.x, -rhs.y);
28     }
29     Comp exp_i(const db &x) {
30         return Comp(cos(x), sin(x));
31     }
32
33     const int L = 21, N = 1 << L;
34     Comp roots[N];
35     int rev[N];
36     struct _init {
37         _init() {
38             for (int i = 0; i < N; i++) {
39                 rev[i] = (rev[i>>1]>>1)|((i&1)<<(L-1));
40             }
41             roots[1] = {1, 0};
42             for (int i = 1; i < L; i++) {
43                 db angle = 2*pi/(1<<i+1);
44                 for (int j = (1<<i-1); j < (1<<i); j++) {
45                     roots[j<<1] = roots[j];
46                     roots[j<<1|1] =
47                         ↪ exp_i((j*2+1-(1<<i))*angle);
48                 }
49             }
50         } init;
51         inline void trans(Comp &a, Comp &b, const Comp &c) {
52             Comp d = b * c;
53             b = a - d;
54             a = a + d;
55         }
56         void fft(vector<Comp> &a, int n) {
57             assert((n & (n - 1)) == 0);
58             int zeros = __builtin_ctz(n), shift = L - zeros;
59             for (int i = 0; i < n; i++) {
60                 if (i < (rev[i]>>shift)) {
61                     swap(a[i], a[rev[i]>>shift]);
62                 }
63             }
64             for (int i = 1; i < n; i <= 1)
65                 for (int j = 0; j < n; j += i * 2)
66                     for (int k = 0; k < i; k++)
67                         trans(a[j+k], a[i+j+k], roots[i+k]);
68         }
69
70         vector<Comp> fa, fb;
71         vector<int> multiply(const vector<int> &a, const
72             ↪ vector<int> &b) {
```

```

72 // 三次变两次
73 int la = a.size(), lb = b.size();
74 int need = la + lb - 1, n = 1 << (32 -
↳ __builtin_clz(need - 1));
75 if(n > fa.size()) fa.resize(n);
76 for(int i = 0; i < n; i++) {
77     fa[i] = Comp(i < la ? a[i] : 0, i < lb ? b[i] :
↳ 0);
78 }
79 fft(fa, n);
80 Comp r(0, -0.25/n);
81 for(int i = 0, j = 0; i <= (n>>1); i++, j = n - i)
↳ {
82     Comp x = fa[i] * fa[i], y = fa[j] * fa[j];
83     if(i != j) fa[j] = (x - conj(y)) * r;
84     fa[i] = (y - conj(x)) * r;
85 }
86 fft(fa, n);
87 vector<int> c(need);
88 for(int i = 0; i < need; i++) c[i]=fa[i].x+0.5;
89 return c;
90 }

91 vector<ll> multiply(const vector<ll> &a, const
↳ vector<ll> &b) {
92     // 朴素三次
93     int la = a.size(), lb = b.size();
94     int need = la + lb - 1, n = 1 << (32 -
↳ __builtin_clz(need - 1));
95     if(n > fa.size()) fa.resize(n), fb.resize(n);
96     for(int i = 0; i < n; i++) {
97         fa[i] = Comp(i < la ? a[i] : 0, 0);
98         fb[i] = Comp(i < lb ? b[i] : 0, 0);
99     }
100     fft(fa, n);
101     fft(fb, n);
102     for(int i = 0; i < n; i++) {
103         fa[i] = fa[i] * fb[i];
104     }
105     reverse(fa.begin() + 1, fa.begin() + n);
106     fft(fa, n);
107     Comp r(1.0 / n, 0);
108     for(int i = 0; i < n; i++) fa[i] = fa[i] * r;
109     vector<ll> c(need);
110     for(int i = 0; i < need; i++) c[i]=ll(fa[i].x+0.5);
111     return c;
112 }

113
114 const int M = (1 << 15) - 1;
115 vector<int> multiply_mod(const vector<int> &a, const
↳ vector<int> &b, bool eq = 0) {
116     // (a, b) * (c, d)
117     int la = a.size(), lb = b.size();
118     int need = la + lb - 1, n = need > 1 ? 1 << (32 -
↳ __builtin_clz(need - 1)) : 1;
119     if(fa.size() < n) fa.resize(n);
120     if(fb.size() < n) fb.resize(n);
121     for(int i = 0; i < n; i++) {
122         fa[i] = i < la ? Comp(a[i]>>15, a[i]&M) :
↳ Comp(0, 0);
123     }
124     fft(fa, n);
125     if(eq) copy(fa.begin(), fa.begin()+n, fb.begin());
126     else {
127         for(int i = 0; i < n; i++) {
128             fb[i] = i < lb ? Comp(b[i]>>15, b[i]&M) :
↳ Comp(0, 0);
129         }
130         fft(fb, n);
131     }
132 }

```

```

133     Comp r(0.5/n,0);
134     for(int i = 0, j = 0; i <= (n>>1); i++, j = n - i)
↳ {
135         Comp x = (fa[i]+conj(fa[j]))*fb[i]*r; // (a,
↳ 0)*(c, d)
136         Comp y = (fa[i]-conj(fa[j]))*conj(fb[j])*r; //
↳ (0, b)*(c, d)
137         if(i != j) {
138             Comp _x = (fa[j]+conj(fa[i]))*fb[j]*r;
139             Comp _y =
↳ (fa[j]-conj(fa[i]))*conj(fb[i])*r;
140             fa[i] = _x, fb[i] = _y;
141         }
142         fa[j] = x, fb[j] = y;
143     }
144     fft(fa, n);
145     fft(fb, n);
146     vector<int> c(need);
147     for(int i = 0; i < need; i++) {
148         ll x = ll(fa[i].x + 0.5) % mod, y = ll(fa[i].y
↳ + 0.5) % mod;
149         ll z = ll(fb[i].x + 0.5) % mod, w = ll(fb[i].y
↳ + 0.5) % mod;
150         c[i] = ((x<<30) + z + (y + w << 15)) % mod;
151     }
152     return c;
153 }
154 vector<int> sqr_mod(const vector<int> &a) {
155     return multiply_mod(a, a, 1);
156 }
157 }
158 using FFT::multiply;
159 using FFT::multiply_mod;
160 using FFT::sqr_mod;
161
162 int main() {
163     using namespace std;
164     ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
165     int n, m;
166     cin >> n >> m >> mod;
167     vector<int> f(n + 1), g(m + 1);
168     for(auto &x : f) cin >> x;
169     for(auto &x : g) cin >> x;
170     vector<int> h = multiply_mod(f, g);
171     for(auto &x : h) cout << x << " ";
172     cout << endl;
173 }
174 }

```

1.2.2 NTT

```

1 #include <bits/stdc++.h>
2
3 using ll = unsigned long long;
4 using Int = unsigned;
5
6 namespace Polynomial {
7
8     using Poly = std::vector<Int>;
9     constexpr Int P(998244353), G(3);
10     inline void inc(Int &x, Int y, Int mod = P) {
11         (x += y) >= mod ? x -= mod : 0;
12     }
13
14     inline Int fpow(Int x, Int k = P - 2, Int mod = P) {
15         Int r = 1;
16         for (; k >= 1, x = (ll) x * x % mod)
17             if (k & 1) r = (ll) r * x % mod;
18         return r;
19     }
20 }

```

```

19 }
20
21 const int MAXW = 1 << 22;
22 Int w[MAXW];
23 struct omegaGen {
24     omegaGen() {
25         Int x = fpow(G, (P - 1) / MAXW);
26         for(int i = MAXW >> 1; i; i >>= 1) {
27             w[i] = 1;
28             for(int j = 1; j < i; ++j)
29                 w[i + j] = (ll) w[i + j - 1] * x % P;
30             x = (ll) x * x % P;
31         }
32     }
33 } gen;
34
35 Poly &operator *= (Poly &a, Int b) {
36     for(auto &x : a)
37         x = (ll) x * b % P;
38     return a;
39 }
40 Poly operator * (Poly a, Int b) { return a *= b; }
41 Poly operator * (Int a, Poly b) { return b * a; }
42 Poly &operator /= (Poly &a, Int b) { return a *= fpow(b); }
43 Poly operator / (Poly a, Int b) { return a /= b; }
44 Poly &operator += (Poly &a, Poly b) {
45     a.resize(std::max(a.size(), b.size()));
46     for(size_t i = 0; i < b.size(); i++)
47         inc(a[i], b[i]);
48     return a;
49 }
50 Poly operator + (Poly a, Poly b) { return a += b; }
51 Poly &operator -= (Poly &a, Poly b) {
52     a.resize(std::max(a.size(), b.size()));
53     for(size_t i = 0; i < b.size(); i++)
54         inc(a[i], P - b[i]);
55     return a;
56 }
57 Poly operator - (Poly a, Poly b) { return a - b; }
58 Poly operator - (Poly a) {
59     for(auto &x : a)
60         if(x) x = P - x;
61     return a;
62 }
63 Poly &operator >>= (Poly &a, Int x) {
64     if (x >= (Int)a.size()) {
65         a.clear();
66     } else {
67         a.erase(a.begin(), a.begin() + x);
68     }
69     return a;
70 }
71 Poly &operator <<= (Poly &a, Int x) {
72     a.insert(a.begin(), x, 0);
73     return a;
74 }
75 Poly operator >> (Poly a, Int x) { return a >>= x; }
76 Poly operator << (Poly a, Int x) { return a <<= x; }
77
78 Poly &cdot(Poly &a, Poly b) {
79     assert(a.size() == b.size());
80     for (size_t i = 0; i < a.size(); i++)
81         a[i] = (ll) a[i] * b[i] % P;
82     return a;
83 }
84 Poly dot(Poly a, Poly b) { return cdot(a, b); }
85 void norm(Poly &a) {
86     if (!a.empty()) {
87         a.resize(1 << std::lg(a.size() * 2 - 1));
88     }
89 }
90

```

```

91 void dft(Int *a, int n) {
92     //assert((n & n - 1) == 0);
93     for(int k = n >> 1; k; k >>= 1) {
94         for(int i = 0; i < n; i += k << 1) {
95             for(int j = 0; j < k; j++) {
96                 Int x = a[i + j], y = a[i + j + k], ww =
97                     ↪ w[k + j];
98                 a[i + j] = x + y >= P ? x + y - P : x + y;
99                 a[i + j + k] = (ll) (x - y + P) * ww % P;
100             }
101         }
102     }
103 }
104 void idft(Int *a, int n) {
105     //assert((n & n - 1) == 0);
106     for(int k = 1; k < n; k <= 1) {
107         for(int i = 0; i < n; i += k << 1) {
108             for(int j = 0; j < k; j++) {
109                 Int x = a[i + j], y = (ll) a[i + j + k] *
110                     ↪ w[k + j] % P;
111                 if(x >= P) x -= P;
112                 a[i + j + k] = x - y + P;
113                 a[i + j] = x + y;
114             }
115         }
116     }
117     for (Int i = 0, inv = P - (P - 1) / n; i < n; i++)
118         a[i] = (ll) a[i] * inv % P;
119     std::reverse(a + 1, a + n);
120 }
121
122 void dft(Poly &a) { dft(a.data(), a.size()); }
123 void idft(Poly &a) { idft(a.data(), a.size()); }
124
125 Poly operator* (Poly a, Poly b) {
126     if(a.empty() || b.empty()) return {};
127     int len = a.size() + b.size() - 1;
128     if(a.size() <= 16 || b.size() <= 16) {
129         Poly c(len);
130         for(size_t i = 0; i < a.size(); i++)
131             for(size_t j = 0; j < b.size(); j++)
132                 c[i + j] = (c[i + j] + ll(a[i]) * b[j]) %
133                     ↪ P;
134         return c;
135     }
136     int n = 1 << std::lg(len - 1) + 1;
137     if (a != b) {
138         a.resize(n), b.resize(n);
139         dft(a), dft(b);
140         cdot(a, b);
141     } else {
142         a.resize(n), dft(a);
143         cdot(a, a);
144     }
145     idft(a);
146     a.resize(len);
147     return a;
148 }
149 // return: inv(a) mod x ^ n, n = a.size()
150 // require n = 2 ^ k
151 // resize: n' = 1 << lg(2 * n - 1) (n >= 1)
152 Poly inverse(Poly a) {
153     int n = a.size();
154     assert((n & n - 1) == 0);
155     if (n == 1) return {fpow(a[0])};
156     int m = n >> 1;
157     Poly b = inverse(Poly(a.begin(), a.begin() + m)), c =
158         ↪ b;
159     b.resize(n);

```

```

159     dft(a), dft(b), cdot(a, b), idft(a);
160     for (int i = 0; i < m; i++) a[i] = 0;
161     for (int i = m; i < n; i++) a[i] = P - a[i];
162     dft(a), cdot(a, b), idft(a);
163     for (int i = 0; i < m; i++) a[i] = c[i];
164     return a;
165 }
166
167 // return x s.t. x^2 = a mod x^n, n = a.size()
168 // not hold x is lexicographically smallest
169 // no need (n & (n - 1)) == 0
170 // require a[0] != 0
171 Poly polysqrt(Poly a) {
172     int n = a.size();
173     if (n == 1) return {Modroot::calc(2, a[0], P)};
174
175     int m = ((n + 1) >> 1);
176     Poly b = polysqrt(Poly(a.begin(), a.begin() + m));
177     b.resize(n);
178
179     Poly c = b * b;
180     c.resize(n);
181
182     b *= 2;
183     c += a;
184
185     norm(b);
186
187     c = c * inverse(b);
188     c.resize(n);
189
190     return c;
191 }
192
193 // return: q(Len = n - m + 1), a = b * q + r
194 Poly operator/ (Poly a, Poly b) {
195     int n = a.size(), m = b.size();
196     if (n < m) return {0};
197     int k = 1 << std::lg(n - m << 1 | 1);
198     std::reverse(a.begin(), a.end());
199     std::reverse(b.begin(), b.end());
200     a.resize(k), b.resize(k), b = inverse(b);
201     a = a * b;
202     a.resize(n - m + 1);
203     std::reverse(a.begin(), a.end());
204     return a;
205 }
206
207 // return: {q(Len = n - m + 1), r(Len = m - 1)}
208 // require: b.size() > 0
209 std::pair<Poly, Poly> operator% (Poly a, Poly b) {
210     int m = b.size();
211     Poly q = a / b;
212     b = b * q;
213     a.resize(m - 1);
214     for (int i = 0; i < m - 1; i++) inc(a[i], P - b[i]);
215     return {q, a};
216 }
217
218 Poly der(Poly a) {
219     int sz = a.size();
220     for (int i = 0; i + 1 < sz; i++)
221         a[i] = (1l) (i + 1) * a[i + 1] % P;
222     a.pop_back();
223     return a;
224 }
225
226 std::vector<Int> inv = {1, 1};
227 void updateInv(Int n) {
228     if (inv.size() <= n) {
229         Int p = inv.size();
230         inv.resize(n + 1);
231         for (Int i = p; i <= n; i++)
232             inv[i] = (1l) (P - P / i) * inv[P % i] % P;
233     }
234 }
235
236 Poly integ(Poly a, Int c = 0) {
237     int n = a.size();
238     updateInv(n);
239     a.resize(n + 1);
240     for (int i = n - 1; i >= 0; i--)
241         a[i + 1] = (1l) inv[i + 1] * a[i] % P;
242     a[0] = c;
243     return a;
244 }
245
246 // return: ln(a) mod x ^ n, n = a.size()
247 Poly ln(Poly a) {
248     int n = a.size();
249     norm(a);
250     assert(a[0] == 1);
251     a = inverse(a) * der(a);
252     a.resize(n - 1);
253     return integ(a);
254 }
255
256 // un-checked
257 Poly exp(Poly a) {
258     int n = a.size();
259     assert((n & n - 1) == 0);
260     assert(a[0] == 0);
261     if (n == 1) return {1};
262     int m = n >> 1;
263     Poly b = exp(Poly(a.begin(), a.begin() + m)), c;
264     b.resize(n), c = ln(b);
265     a.resize(n << 1), b.resize(n << 1), c.resize(n << 1);
266     dft(a), dft(b), dft(c);
267     for (int i = 0; i < n << 1; i++) a[i] = (1l)(1 + P +
        ↪ a[i] - c[i]) * b[i] % P;
268     idft(a);
269     a.resize(n);
270     return a;
271 }
272
273 // checked
274 Poly power(Poly a, Int k, Int kreal) {
275     int n = a.size();
276     long long d = 0;
277     while (d < n && !a[d]) d++;
278     if (d == n) return a;
279     a >>= d;
280     Int b = fpow(a[0]);
281     norm(a *= b);
282     a = exp(ln(a) * k) * fpow(b, P - 1 - k % (P - 1));
283     a.resize(n);
284     d *= kreal;
285     for (int i = n - 1; i >= d; i--) a[i] = a[i - d];
286     d = std::min(d, (long long) n);
287     for (int i = d - 1; i >= 0; --i) a[i] = 0;
288     return a;
289 }
290
291 // k1 = k % (P - 1), k2 = k % P
292 // kreal = min(k, P)
293 Poly power(Poly a, Int k1, Int k2, Int kreal) {
294     int n = a.size();
295     long long d = 0;
296     while (d < n && !a[d]) d++;
297     if (d == n) return a;
298     a >>= d;
299     Int b = fpow(a[0]);
300     norm(a *= b);
301     a = exp(ln(a) * k2) * fpow(b, P - 1 - k1 % (P - 1));

```

```

302     a.resize(n);
303     d *= kreal;
304     for (int i = n - 1; i >= d; i--) a[i] = a[i - d];
305     d = std::min(d, (long long) n);
306     for(int i = d - 1; i >= 0; --i) a[i] = 0;
307     return a;
308 }
309
310 // [x^n] f / g
311 Int divAt(Poly f, Poly g, ll n) {
312     int len = std::max(f.size(), g.size());
313     assert(len > 0);
314     int m = 1 << std::lg(2 * len - 1);
315     len = m << 1;
316     f.resize(len);
317     for (; n; n >>= 1) {
318         g.resize(len);
319         dft(f), dft(g);
320         for (int i = 0; i < len; ++i)
321             f[i] = (ll) f[i] * g[i < 1] % P;
322         for (int i = 0; i < m; ++i)
323             g[i] = (ll) g[i < 1] * g[i < 1 | 1] % P;
324         g.resize(m);
325         idft(f), idft(g);
326         for (int i = 0; i < m; ++i) f[i] = f[i * 2 + (n &
↪ 1)];
327         fill(f.begin() + m, f.end(), 0);
328     }
329     return (ll) f[0] * fpow(g[0]) % P;
330 }
331
332 } // namespace Polynomial
333
334 using namespace Polynomial;
335 using namespace std;
336
337 /*
338 int main() {
339     ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
340     int n;
341     cin >> n;
342     string s;
343     cin >> s;
344     Int k2 = 0, k1 = 0, kreal = 0;
345     for(int i = 0, w = s.size(); i < s.size(); i++) {
346         --w;
347         inc(k2, (ll) (s[i] - '0') * fpow(10, w) % P);
348         inc(k1, (ll) (s[i] - '0') * fpow(10, w, P - 1) % (P
↪ - 1), P - 1);
349         if(kreal * 10 + (s[i] - '0') <= P) kreal = kreal *
↪ 10 + (s[i] - '0');
350     }
351     Poly a(n);
352     for(int i = 0; i < n; ++i) cin >> a[i];
353     Poly b = power(a, k1, k2, kreal);
354     for(int i = 0; i < n; ++i) cout << b[i] << " \n"[i == n
↪ - 1];
355 } */
356
357 Int read() {
358     int x;
359     cin >> x;
360     x %= 998244353;
361     x += 998244353;
362     x %= 998244353;
363     return x;
364 }
365
366 int main() {
367     ios::sync_with_stdio(0), cin.tie(0), cout.tie(0);
368     ll n;
369     int k;

```

```

370     cin >> n >> k;
371     Poly f(k + 1), a(k);
372     f[0] = 1;
373     for(int i = 1; i <= k; ++i) {
374         f[i] = read();
375         if(f[i]) f[i] = P - f[i];
376     }
377     for(auto &x : a) {
378         x = read();
379     }
380     Poly b = a * f;
381     b.resize(k);
382     Int ans = divAt(b, f, n);
383     cout << ans << endl;
384 }

```

1.2.3 拉格朗日插值

$$f(x)=\sum_i f(x_i)\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}$$

1.2.4 连通图计数

设大小为 n 的满足一个限制 P 的简单无向图数量为 g_n , 满足限制 P 且连通的简单无向图数量为 f_n , 如果已知 $g_1\dots n$ 求 f_n , 可以得到递推式

$$f_n=g_n-\sum_{k=1}^{n-1}\binom{n-1}{k-1}f_k g_{n-k}$$

这个递推式的意义就是用任意图的数量减掉不连通的数量, 而不连通的数量可以通过枚举1号点所在连通块大小来计算.

注意, 由于 $f_0=0$, 因此递推式的枚举下界取0和1都是可以的. 推一推式子会发现得到一个多项式求逆, 再仔细看看, 其实就是一个多项式ln.

1.3 线性代数

1.3.1 高斯消元

辗转相除高斯消元

对于定义在环上而非域上的矩阵, 利用辗转相除进行消元. 下面例子中 mod 未必为素数, 通过辗转相除消元求行列式

```

1 using ll = long long;
2
3 int gauss(ll a[][N], int n) {
4     int ans = 1;
5     for (int j = 0; j < n; ++j) {
6         for (int i = j + 1; i < n; ++i) {
7             while (a[i][j]) {
8                 int t = a[j][j] / a[i][j];
9                 for (int k = j; k < n; ++k)
10                     a[j][k] = (a[j][k] + (mod - t) * a[i][
↪ k]) % mod;
11                 swap(a[i], a[j]);
12                 ans = mod - ans;
13             }
14         }
15     }
16     for (int i = 0; i < n; ++i)
17         ans = ans * a[i][i] % mod;
18     return ans;
19 }

```

求秩与解线性方程组

分别维护行列指针, 维护每一列所对应的有效行, 有效行的总数即为秩. 线性方程组有唯一解当且仅当 列满秩 且 系数矩阵的秩 等于 增广矩阵的秩

```

1 // 消增广矩阵, 注意第 m 行存放目标向量
2 // 若无解返回空 vector
3 // 若有无穷多解, 则非主元置 0
4
5 vector<ll> gauss(ll a[][N], int n, int m) {
6     vector<int> row(m, -1); // 每个变元所对应的有效行
7     vector<ll> ans(m, 0);
8
9     int r = 0;
10    for (int c = 0; c < m; ++c) { // 扫描每一列, 用 r 维护
11        int sig = -1;
12        for (int i = r; i < n; ++i)
13            if (a[i][c]) {
14                sig = i; break;
15            }
16        if (sig == -1) continue; // 无效列
17
18        row[c] = r;
19        if (sig != r)
20            swap(a[sig], a[r]);
21
22        ll inv = fpow(a[r][c], mod - 2);
23        for (int i = 0; i < n; ++i) {
24            if (i == r) continue;
25            ll del = inv * a[i][c] % mod;
26            for (int j = c; j <= m; ++j)
27                a[i][j] = (a[i][j] + (mod - del) * a[r][j]) % mod;
28        }
29        ++r;
30    }
31
32    if (r < n && a[r][m]) {
33        cerr << "no solution!" << endl;
34        return {};
35    }
36
37    for (int i = 0; i < m; ++i) { // ax = b
38        if (row[i] != -1) {
39            ans[i] = fpow(a[row[i]][i]) * a[row[i]][m] % mod;
40        } else {
41            ans[i] = 0; // 非主元置 0 即为合法解
42        }
43    }
44
45    return ans;
46 }

```

矩阵求逆

维护一个矩阵 B , 初始设为 n 阶单位矩阵, 在消元的同时对 B 进行一样的操作, 当把 A 消成单位矩阵时 B 就是逆矩阵。

1.3.2 矩阵树定理

无向图: 设图 G 的基尔霍夫矩阵 $L(G)$ 等于度数矩阵减去邻接矩阵, 则 G 的生成树个数等于 $L(G)$ 的任意一个代数余子式的值。

有向图: 类似地定义 $L_{in}(G)$ 等于入度矩阵减去邻接矩阵 (i 指向 j 有边, 则 $A_{i,j} = 1$), $L_{out}(G)$ 等于出度矩阵减去邻接矩阵。

则以 i 为根的内向树个数即为 L_{out} 的第 i 个主子式 (即关于第 i 行第 i 列的余子式), 外向树个数即为 L_{in} 的第 i 个主子式。

(可以看出, 只有无向图才满足 $L(G)$ 的所有代数余子式都相等。)

BEST定理 (有向图欧拉回路计数): 如果 G 是有向欧拉图, 则 G 的欧拉回路的个数等于以一个任意点为根的内/外向树个数乘以 $\prod_v (\deg(v) - 1)!$ 。

并且在欧拉图里, 无论以哪个结点为根, 也无论内向树还是外向树, 个数都是一样的。

另外无向图欧拉回路计数是 NP 问题。

1.3.3 线性基

```

1 using ll = long long;
2
3 const int N = 70;
4 const int Lg = 60;
5
6 ll bs[N];
7
8 // find x in S s.t. k ^ x >= Low and k ^ x is minimum,
9 // !!! return k ^ x, not x
10 // if not exist, return inf
11 ll lower(ll k, ll low) {
12
13     // ll x = k ^ low; // expected value in S
14     // ll res = 0; // value represented in S
15     ll x = 0;
16     ll ex = k ^ low;
17     int lb = -1;
18
19     // 在前缀可表示的范围内的寻找:
20     // 对于范围内 Low[i] = 0 的位,
21     // 考虑另一分支的可行性
22     for (int i = Lg - 1; i >= 0; i--) {
23         if (((low >> i) & 1ll) == 0) { // another
24             // branch
25             int d = ((ex ^ x) >> i) & 1ll; // cur branch
26             if (d || bs[i]) {
27                 lb = i;
28             }
29         }
30         if (((ex ^ x) >> i) & 1ll) {
31             if (bs[i]) {
32                 x ^= bs[i];
33             } else {
34                 break;
35             }
36         }
37     }
38
39     if ((ex ^ x) == 0) // 可表示
40         return k ^ x;
41
42     if (lb == -1) // 不可表示, 且不可更大
43         return inf;
44
45     if (((k ^ x ^ low) >> lb) & 1ll) == 0)
46         x ^= bs[lb];
47
48     for (int i = lb - 1; i >= 0; i--) {
49         if (((k ^ x) >> i) & 1ll) {
50             if (bs[i]) {
51                 x ^= bs[i];
52             }
53         }
54     }
55
56     return k ^ x;
57 }

```

1.4 反演与容斥

1.4.1 二项式反演

形式一:

$$f(n) = \sum_{i=m}^n \binom{n}{i} \iff g(i)g(n) = \sum_{i=m}^n (-1)^{n-i} \binom{n}{i} f(i)$$

形式二: (常用)

施罗德数

$$f(n) = \sum_{i=n}^m \binom{i}{n} g(i) \iff g(n) = \sum_{i=n}^m (-1)^{i-n} \binom{i}{n} f(i)$$

常见用法

钦定 (至少) k 个与恰好 k 个之间的转化

记号:

记 $f(n)$ 表示先钦定至少选 n 个, 再统计钦定情况如此的方案数之和, 其中会包含重复的方案数.

记 $g(n)$ 表示恰好选 n 个的方案数, 不会重复.

那么, 对于 $i \geq n$, $g(i)$ 在 $f(n)$ 中被重复计算了 $\binom{i}{n}$ 次, 故

$f(n) = \sum_{i=n}^m \binom{i}{n} g(i)$, 其中 m 为数目上限

通常, f 较易求, 再通过反演即可求 g

1.4.2 单位根反演

$$[n|k] = \frac{1}{n} \sum_{i=0}^{n-1} \omega_n^{ik}$$

$$\sum_{i \geq 0} [x^{ik}] f(x) = \frac{1}{k} \sum_{j=0}^{k-1} f(\omega_k^j)$$

1.4.3 Min-Max容斥

设有数集 S , 有:

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

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

第 k 大数的 Min-Max 容斥

$$\max_k(S) = \sum_{T \subseteq S, |T| \geq k} (-1)^{|T|-k} \times \binom{|T|-1}{k-1} \times \min(T)$$

1.5 组合数

1.5.1 常用组合数

卡特兰数

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n-1}$$

- n 个元素按顺序入栈, 出栈序列方案数
- 长为 $2n$ 的合法括号序列数
- $n+1$ 个叶子的满二叉树个数

递推式:

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-i-1}$$

$$C_n = C_{n-1} \frac{4n-2}{n+1}$$

普通生成函数:

$$C(x) = \frac{1 - \sqrt{1-4x}}{2x}$$

扩展: 如果有 n 个左括号和 m 个右括号, 方案数为

$$\binom{n+m}{n} - \binom{n+m}{m-1}$$

$$S_n = S_{n-1} + \sum_{i=0}^{n-1} S_i S_{n-i-1}$$

$$(n+1)s_n = (6n-3)s_{n-1} - (n-2)s_{n-2}$$

其中 S_n 是(大)施罗德数, s_n 是小施罗德数(也叫超级卡特兰数).

除了 $S_0 = s_0 = 1$ 以外, 都有 $S_i = 2s_i$.

施罗德数的组合意义:

- 从 $(0,0)$ 走到 (n,n) , 每次可以走右, 上, 或者右上一步, 并且不能超过 $y=x$ 这条线的方案数
- 长为 n 的括号序列, 每个位置也可以为空, 并且括号对数和空位置数加起来等于 n 的方案数
- 凸 n 边形的任意剖分方案数

(有些人会把大(而不是小)施罗德数叫做超级卡特兰数.)

默慈金数

$$M_{n+1} = M_n + \sum_{i=0}^{n-1} M_i M_{n-1-i} = \frac{(2n+3)M_n + 3nM_{n-1}}{n+3}$$

$$M_n = \sum_{i=0}^{\frac{n}{2}} \binom{n}{2i} C_i$$

在圆上的 n 个不同的点之间画任意条不相交(包括端点)的弦的方案数. 也等价于在网格图上, 每次可以走右上, 右下, 正右方一步, 且不能走到 $y < 0$ 的位置, 在此前提下从 $(0,0)$ 走到 $(n,0)$ 的方案数.

扩展: 默慈金数画的弦不可以共享端点. 如果可以共享端点的话是 A054726, 后面的表里可以查到.

1.5.2 斯特林数

1. 第一类斯特林数

$\left[\begin{smallmatrix} n \\ k \end{smallmatrix} \right]$ 表示 n 个元素划分成 k 个轮换的方案数.

递推式: $\left[\begin{smallmatrix} n \\ k \end{smallmatrix} \right] = \left[\begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right] + (n-1) \left[\begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right]$.

求同一行: 分治 FFT $O(n \log^2 n)$, 或者倍增 $O(n \log n)$ (每次都是 $f(x) = g(x)g(x+d)$ 的形式, 可以用 $g(x)$ 反转之后做一个卷积求出后者).

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

求同一列: 用一个轮换的指数生成函数做 k 次幂

$$\sum_{n=0}^{\infty} \left[\begin{smallmatrix} n \\ k \end{smallmatrix} \right] \frac{x^n}{n!} = \frac{(\ln(1-x))^k}{k!} = \frac{x^k}{k!} \left(\frac{\ln(1-x)}{x} \right)^k$$

2. 第二类斯特林数

$\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$ 表示 n 个元素划分成 k 个子集的方案数.

递推式: $\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} = \left\{ \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right\} + k \left\{ \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right\}$.

求一个: 容斥, 狗都会做

$$\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} = \frac{1}{k!} \sum_{i=0}^k (-1)^i \binom{k}{i} (k-i)^n = \sum_{i=0}^k \frac{(-1)^i}{i!} \frac{(k-i)^n}{(k-i)!}$$

求同一行: FFT, 狗都会做

求同一列: 指数生成函数

$$\sum_{n=0}^{\infty} \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} \frac{x^n}{n!} = \frac{(e^x - 1)^k}{k!} = \frac{x^k}{k!} \left(\frac{e^x - 1}{x} \right)^k$$

普通生成函数

$$\sum_{n=0}^{\infty} \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} x^n = x^k \left(\prod_{i=1}^k (1-ix) \right)^{-1}$$

3. 幂的转换

上升幂与普通幂的转换

$$x^{\overline{n}} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} x^k$$

$$x^n = \sum_k \left\{ \begin{matrix} n \\ k \end{matrix} \right\} (-1)^{n-k} x^{\overline{k}}$$

下降幂与普通幂的转换

$$x^n = \sum_k \left\{ \begin{matrix} n \\ k \end{matrix} \right\} x^{\overline{k}} = \sum_k \binom{x}{k} \left\{ \begin{matrix} n \\ k \end{matrix} \right\} k!$$

$$x^{\underline{n}} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} (-1)^{n-k} x^k$$

另外，多项式的点值表示的每项除以阶乘之后卷上 e^{-x} 乘上阶乘之后是牛顿插值表示，或者不乘阶乘就是下降幂系数表示。反过来的转换当然卷上 e^x 就行了。原理是每次差分等价于乘以 $(1-x)$ ，展开之后用一次卷积取代多次差分。

4. 斯特林多项式(斯特林数关于斜线的性质)

定义：

$$\sigma_n(x) = \frac{\begin{bmatrix} x \\ n \end{bmatrix}}{x(x-1)\dots(x-n)}$$

$\sigma_n(x)$ 的最高次数是 x^{n-1} 。（所以作为唯一的特例， $\sigma_0(x) = \frac{1}{x}$ 不是多项式。）

斯特林多项式实际上非常神奇，它与两类斯特林数都有关系。

$$\begin{bmatrix} n \\ n-k \end{bmatrix} = n^{\overline{k+1}} \sigma_k(n)$$

$$\left\{ \begin{matrix} n \\ n-k \end{matrix} \right\} = (-1)^{k+1} n^{\overline{k+1}} \sigma_k(-(n-k))$$

不过它并不好求。可以 $O(k^2)$ 直接计算前几个点值然后插值，或者如果要推式子的话可以用后面提到的二阶欧拉数。

1.5.3 欧拉数

1. 欧拉数

$\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle$: n 个数的排列，有 k 个上升的方案数。

$$\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle = (n-k) \left\langle \begin{matrix} n-1 \\ k-1 \end{matrix} \right\rangle + (k+1) \left\langle \begin{matrix} n-1 \\ k \end{matrix} \right\rangle$$

$$\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle = \sum_{i=0}^{k+1} (-1)^i \binom{n+1}{i} (k+1-i)^n$$

$$\sum_{k=0}^{n-1} \left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle = n!$$

$$x^n = \sum_{k=0}^{n-1} \left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle \binom{x+k}{n}$$

$$k! \left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \sum_{i=0}^{n-1} \left\langle \begin{matrix} n \\ i \end{matrix} \right\rangle \binom{i}{n-k}$$

2. 二阶欧拉数

$\left\langle\!\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle\!\right\rangle$: 每个数都出现两次的多重排列，并且每个数两次出现之间的数都比它要大。在此前提下有 k 个上升的方案数。

$$\left\langle\!\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle\!\right\rangle = (2n-k-1) \left\langle\!\left\langle \begin{matrix} n-1 \\ k-1 \end{matrix} \right\rangle\!\right\rangle + (k+1) \left\langle\!\left\langle \begin{matrix} n-1 \\ k \end{matrix} \right\rangle\!\right\rangle$$

$$\sum_{k=0}^{n-1} \left\langle\!\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle\!\right\rangle = (2n-1)!! = \frac{(2n)^{\underline{n}}}{2^n}$$

3. 二阶欧拉数与斯特林数的关系

$$\left\{ \begin{matrix} x \\ x-n \end{matrix} \right\} = \sum_{k=0}^{n-1} \left\langle\!\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle\!\right\rangle \binom{x+n-k-1}{2n}$$

$$\begin{bmatrix} x \\ x-n \end{bmatrix} = \sum_{k=0}^{n-1} \left\langle\!\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle\!\right\rangle \binom{x+k}{2n}$$

1.6 线性规划

1.6.1 对偶原理

给定一个原始线性规划：

$$\begin{aligned} &\text{Minimize} && \sum_{j=1}^n c_j x_j \\ &\text{Subject to} && \sum_{j=1}^n a_{ij} x_j \geq b_i, \\ &&& x_j \geq 0 \end{aligned}$$

定义它的对偶线性规划为：

$$\begin{aligned} &\text{Maximize} && \sum_{i=1}^m b_i y_i \\ &\text{Subject to} && \sum_{i=1}^m a_{ij} y_i \leq c_j, \\ &&& y_i \geq 0 \end{aligned}$$

用矩阵可以更形象地表示为：

$$\begin{aligned} &\text{Minimize} && \mathbf{c}^T \mathbf{x} \\ &\text{Subject to} && \mathbf{Ax} \geq \mathbf{b}, \\ &&& \mathbf{x} \geq 0 \end{aligned} \iff \begin{aligned} &\text{Maximize} && \mathbf{b}^T \mathbf{y} \\ &\text{Subject to} && \mathbf{A}^T \mathbf{y} \leq \mathbf{c}, \\ &&& \mathbf{y} \geq 0 \end{aligned}$$

1.7 杂项

1.7.1 约瑟夫环

```

1 typedef long long ll;
2
3 ll k;
4
5 // 约瑟夫环
6 int main() {
7     int T;
8     cin >> T;
9     ll n, m;
10    for (int cas = 1; cas <= T; ++cas) {
11        scanf("%lld %lld %lld", &n, &m, &k);
12        ll res = (k-1) % (n-m+1);
13        if (k == 1) res = (m-1) % n;
14        else for (ll i = n-m+1, j, t; i < n; i = j) {
15            if (i < k) {
16                j = i+1;
17                res = (res+k) % j;
18            } else {
19                j = i+i/(k-1);
20                if (j > n) j = n;
21                t = j-i;
22                res -= j-t*k;
23                if (res < 0) res += j;
24                else res += res/(k-1);
25            }
26        }
27        printf("Case %d: %lld\n", cas, res+1);

```

```
28 }
29 }
```

1.7.2 杨辉三角行区间和

```
1 using ll = long long;
2
3 // 边界从 (s, x) 移动到 (s + 1, nx)
4 ll move(int x, int nx, int s, ll sum) {
5     assert(x >= -1);
6     ll res = (2 * sum + mod - C(s, x)) % mod;
7     while (x + 1 <= nx) {
8         x++;
9         res = (res + C(s + 1, x)) % mod;
10    }
11    while (x > nx) {
12        res = (res + mod - C(s + 1, x)) % mod;
13        x--;
14    }
15    return res;
16 };
17
18 void proc(int k) {
19     // 第 s 行的 左右边界
20     auto le = [&](int s) -> int {
21         // ...
22     };
23     auto ri = [&](int s) -> int {
24         // ...
25     };
26     // 这里应该暴力计算首行和当首行为 0 时也可以这样写
27     ll lsum = move(-1, le(0), -1, 0);
28     ll rsum = move(-1, ri(0), -1, 0);
29
30     for (int s = 0; s <= k; s++) {
31         if (le(s) < ri(s)) {
32             // ... 第 s 行对答案的贡献
33         }
34         lsum = move(le(s), le(s + 1), s, lsum);
35         rsum = move(ri(s), ri(s + 1), s, rsum);
36     }
37 }
```

1.7.3 辛普森积分

```
1 // Adaptive Simpson's method : double simpson::solve
2   ↳ (double (*f)(double), double l, double r, double eps)
3   ↳ : integrates f over (L, r) with error eps.
4
5 double area (double (*f)(double), double l, double r) {
6     double m = l + (r - l) / 2;
7     return (f(l) + 4 * f(m) + f(r)) * (r - l) / 6;
8 }
9
10 double solve (double (*f)(double), double l, double r,
11   ↳ double eps, double a) {
12     double m = l + (r - l) / 2;
13     double left = area(f, l, m), right = area(f, m, r);
14     if (fabs(left + right - a) <= 15 * eps)
15         return left + right + (left + right - a) / 15.0;
16     return solve(f, l, m, eps / 2, left) + solve(f, m, r,
17   ↳ eps / 2, right);
18 }
19
20 double solve (double (*f)(double), double l, double r,
21   ↳ double eps) {
22     return solve(f, l, r, eps, area (f, l, r));
23 }
```

1.7.4 纳什均衡

首先定义纯策略和混合策略：纯策略是指你一定会选择某个选项，混合策略是指你对每个选项都有一个概率分布 p_i ，你会以相应的概率选择这个选项。

考虑这样的游戏：有几个人(当然也可以是两个)各自独立地做决定，然后同时公布每个人的决定，而每个人的收益和所有人的选择有关。

那么纳什均衡就是每个人都决定一个混合策略，使得在其他人都选择纯策略的情况下，这个人最坏情况下(也就是说其他人的纯策略最针对他的时候)的收益是最大的。也就是说，收益函数对这个人的混合策略求一个偏导，结果是0(因为是极大值)。

纳什均衡点可能存在多个，不过在一个双人零和游戏中，纳什均衡点一定唯一存在。

1.7.5 康托展开

求排列的排名：先对每个数都求出它后面有几个数比它小(可以用树状数组预处理)，记为 c_i ，则排列的排名就是

$$\sum_{i=1}^n c_i (n - i)!$$

已知排名构造排列：从前到后先分别求出 c_i ，有了 c_i 之后再用一个平衡树(需要维护排名)倒序处理即可。

1.7.6 常用NTT素数及原根

$p = r \times 2^k + 1$	r	k	最小原根
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
985661441	235	22	3
998244353	119	23	3
1004535809	479	21	3
1005060097*	1917	19	5
2013265921	15	27	31
2281701377	17	27	3
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

*注：1005060097有点危险，在变化长度大于 $524288 = 2^{19}$ 时不可用。

2 数论

2.1 常见预处理与快速幂

```

1 // 预处理组合数
2 const int N = 2e5 + 7;
3 const ll mod = 998244353;
4
5 ll fac[N], ifac[N];
6 void init() {
7     fac[0] = 1;
8     for (int i = 1; i < N; ++i) fac[i] = i * fac[i-1] %
    ↪ mod;
9     ifac[N - 1] = fpow(fac[N - 1], mod - 2);
10    for (int i = N - 1; i; --i) ifac[i - 1] = i * ifac[i] %
    ↪ mod;
11 }
12
13 ll C(int n, int k) {
14     if (k < 0 || k > n) return 0;
15     return (fac[n] * ifac[k] % mod) * ifac[n - k] % mod;
16 }
17
18 // 线性求逆元 注意有效的 i < mod
19 ll inv[maxn];
20 void init() {
21     inv[0] = 0, inv[1] = 1;
22     for (int i = 2; i < N; ++i)
23         inv[i] = inv[mod % i] * (mod - mod / i) % mod;
24 }
25
26 // 快速幂
27 ll fpow(ll a, ll k = mod - 2, ll p = mod) {
28     ll res = 1; a %= p;
29     for (; k >>= 1, a = a * a % p) {
30         if (k & 1)
31             res = res * a % p;
32     }
33     return res;
34 }

```

2.2 因数分解与素性判定

2.2.1 朴素因数分解

```

1 // 素因数分解
2 int p[maxn], l[maxn], cnt2 = 0;
3 void Fact(int x) {
4     cnt2 = 0;
5     for (int i = 2; 1ll * i * i <= x; i++) {
6         if (x % i == 0) {
7             p[++cnt2] = i; l[cnt2] = 0;
8             while (x % i == 0) {
9                 x /= i; ++l[cnt2];
10            }
11        }
12    }
13    if (x != 1) { // 则此时x一定是素数 且为原本x的大于根
    ↪ 号x的唯一素因子
14        p[++cnt2] = x; l[cnt2] = 1;
15    }
16 }
17
18 // vector ver. 无次数
19 void Fact(ll x, vector<int>& fact) {
20     for (ll i = 2; 1ll * i * i <= x; ++i) {
21         if (x % i == 0) {
22             fact.push_back(i);
23             while (x % i == 0) x /= i;
24         }
25     }
26 }

```

```

25     }
26     if (x != 1) fact.push_back(x);
27 }

```

2.2.2 Miller-Rabin 与 Pollard-Rho

```

1 typedef long long ll;
2
3 // 注意在 MR 里的 fpow 模数超过 int 范围 需要开 __int128
4
5 ll mul(ll a, ll b, ll p) {
6     return __int128(a) * b % p;
7 }
8
9 ll fpow(ll a, ll k, ll p) {
10    ll res = 1; a %= p;
11    for (; k >>= 1, a = mul(a, a, p)) {
12        if (k & 1)
13            res = mul(res, a, p);
14    }
15    return res;
16 }
17
18 ll randint(ll l, ll r) {
19     static mt19937 eng(time(0));
20     uniform_int_distribution<ll> dis(l, r);
21     return dis(eng);
22 }
23
24 bool is_prime(ll x) {
25     int s = 0; ll t = x - 1;
26     if (x == 2) return true;
27     if (x < 2 || !(x & 1)) return false;
28     while (!(t & 1)) { // 将x分解成(2^s)*t的样子
29         s++; t >>= 1;
30     }
31     ll lst[] = {2, 325, 9375, 28178, 450775, 9780504,
    ↪ 1795265022};
32     for (ll a : lst) { // 随便选一个素数进行测试
33         if (a >= x) break;
34         ll b = fpow(a, t, x); // 先算出 a^t
35         for (int j = 1; j <= s; ++j) { // 然后进行s次平方
36             ll k = mul(b, b, x); // 求b的平方
37             if (k == 1 && b != 1 && b != x - 1) // 用二次探
    ↪ 测判断
38                 return false;
39             b = k;
40         }
41         if (b != 1)
42             return false; // 用费马小定律判断
43     }
44     return true; // 如果进行多次测试都是对的 那么x就很有可
    ↪ 能是素数
45 }
46
47 ll gcd(ll a, ll b) { return b == 0 ? a : gcd(b, a % b); }
48
49 // @author: Pecco
50 ll Pollard_Rho(ll n) {
51     if (n == 4) return 2;
52     if (is_prime(n)) return n;
53     while (1) {
54         ll c = randint(1, n - 1); // 生成随机的c
55         auto f = [=](ll x) { return ((__int128)x * x + c) %
    ↪ n; }; // LLL表示__int128防溢出
56         ll t = f(0), r = f(f(0));
57         while (t != r) {
58             ll d = __gcd(abs(t - r), n);
59             if (d > 1)
60                 return d;
61         }
62     }
63 }

```

```

62     t = f(t), r = f(f(r));
63 }
64 }
65 }
66
67 // 优化掉一个log
68 ll Pollard_Rho(ll n) {
69     if (n == 4) return 2;
70     if (is_prime(n)) return n;
71     while (1) {
72         ll c = randint(1, n - 1);
73         auto f = [=](ll x) { return ((__int128)x * x + c) %
↪ n; };
74         ll t = 0, r = 0, p = 1, q;
75         do {
76             for (int i = 0; i < 128; ++i) { // 令固定距
↪ 离C=128
77                 t = f(t), r = f(f(r));
78                 if (t == r || (q = (ll1)p * abs(t - r) % n)
↪ == 0) // 如果发现环或者积即将为0退出
79                     break;
80                 p = q;
81             }
82             ll d = gcd(p, n);
83             if (d > 1)
84                 return d;
85         } while (t != r);
86     }
87 }
88 }
89
90 vector<ll> factors;
91
92 // 将 n 进行素因子分解factors: 存放 n 的所有不去重素因子
93 void getfactors(ll n) {
94     if (n == 1) return;
95     if (is_prime(n)) { factors.push_back(n); return; } //
↪ 如果是质因子
96     ll p = n;
97     while (p == n)
98         p = Pollard_Rho(n);
99     getfactors(n / p), getfactors(p); //递归处理
100 }

```

2.3 筛法

2.3.1 线性筛

```

1  const int maxn = 1000000 + 5;
2  bool isnt[maxn];
3  int prime[maxn];
4  int cnt = 0;
5
6  // 线性筛法 [1, n] 内素数
7  void Prime(int n) {
8      isnt[1] = true;
9      cnt = 0;
10     for (int i = 2; i <= n; i++) {
11         if (!isnt[i]) prime[++cnt] = i;
12         for (int j = 1; j <= cnt; j++) {
13             if (ll1 * i * prime[j] > n) break;
14             isnt[i * prime[j]] = 1;
15             if (i % prime[j] == 0) break;
16         }
17     }
18 }
19
20 // 线性筛求积性函数
21 int phi[maxn], mu[maxn], d[maxn], D[maxn], q[maxn];
22 void Sieve(int n) {
23     isnt[1] = true;

```

```

24     phi[1] = 1;
25     //mu[1] = 1;
26     cnt = 0;
27     for(int i = 2; i <= n; i++) {
28         if (!isnt[i]) {
29             prime[++cnt] = i;
30             phi[i] = i - 1;
31             //mu[i] = -1;
32             // d[i] = 2; q[i] = 1;
33             // D[i] = i + 1; q[i] = 1;
34         }
35         for (int j = 1; j <= cnt; j++) {
36             int x = i * prime[j];
37             if (x > n) break;
38             isnt[x] = 1;
39             if (i % prime[j] == 0) {
40                 phi[x] = phi[i] * prime[j];
41                 // mu[x] = 0;
42                 // d[x] = d[i] / (q[i] + 1) * (q[i] + 2),
↪ q[x] = q[i] + 1;
43                 // D[x] = D[i] / (prime[j] ^ (q[i] + 1) -
↪ 1) * (prime[j] ^ (q[i] + 2) - 1), q[x] = q[i] + 1;
44                 break;
45             } else {
46                 phi[x] = phi[i] * (prime[j] - 1); // mu[x]
↪ = -mu[i]
47                 // d[x] = 2 * d[i], q[x] = 1;
48                 // D[x] = (prime[j] + 1) * D[i], q[x] = 1;
49             }
50         }
51     }
52 }

```

2.3.2 Min25 筛

```

1  using ll = long long;
2  using i128 = __int128;
3  //using i128 = int64_t;
4
5  const ll mod = 998244353;
6
7  namespace min25 {
8      const int N = 1e6 + 10;
9      ll po[40][N];
10     inline ll fpow(ll e, ll k) {
11         return po[e][k];
12     }
13     void precalc() {
14         for(int e = 0; e < 40; ++e) {
15             po[e][0] = 1;
16             for(int i = 1; i < N; i++)
17                 po[e][i] = e * po[e][i - 1] % mod;
18         }
19     }
20     ll n;
21     int B;
22     int _id[N * 2];
23     inline int id(ll x) {
24         return x <= B ? x : n / x + B;
25     }
26     inline int Id(ll x) {
27         return _id[id(x)];
28     }
29     // f(p) = p - 1 = fh(p) - fg(p);
30     inline ll fg(ll x) {
31         // assert(x <= sqrt(n));
32         return 1;
33     }
34     inline ll fh(ll x) {
35         // assert(x <= sqrt(n));

```

```

36     return x;
37 }
38 // \sum_{i=2}^n fg(i)
39 inline ll sg(ll x) {
40     return (x - 1) % mod;
41 }
42 // \sum_{i=2}^n fh(i)
43 inline ll sh(ll x) {
44     return ((i128) x * (x + 1) / 2 + mod - 1) % mod;
45 }
46 // f(p^e)
47 inline ll f(ll p, ll e) {
48     //return (pe - pe / p) % mod;
49     return (p + (mod - 2) * fpow(e, p)) % mod;
50 }
51 bitset<N> np;
52 ll p[N>>2], pn;
53 ll pg[N>>2], ph[N>>2];
54 void sieve(ll sz) {
55     for(int i = 2; i <= sz; i++) {
56         if(!np[i]) {
57             p[++pn] = i;
58             pg[pn] = (pg[pn - 1] + fg(i)) % mod;
59             ph[pn] = (ph[pn - 1] + fh(i)) % mod;
60         }
61         for(int j = 1; j <= pn && i * p[j] <= sz; j++)
62             np[i * p[j]] = 1;
63         if(i % p[j] == 0) break;
64     }
65 }
66 }
67 }
68 ll m;
69 ll g[N * 2], h[N * 2];
70 ll w[N * 2];
71
72 void compress() {
73     for (int i = 1; i <= m; i++) {
74         g[i] = (h[i] + mod - g[i] + mod - g[i]) % mod;
75     }
76     for (int i = 1; i <= pn; i++) {
77         pg[i] = (ph[i] + mod - pg[i] + mod - pg[i]) %
78 mod;
79     }
80 }
81
82 ll dfs_F(int k, ll n) {
83     if (n < p[k] || n <= 1) return 0;
84     ll res = g[Id(n)] + mod - pg[k - 1], pw2;
85     for (int i = k; i <= pn && (pw2 = (ll) p[i] * p[i])
86 <= n; ++i) {
87         ll pw = p[i];
88         for (int c = 1; pw2 <= n; ++c, pw = pw2, pw2 *=
89 p[i])
90             res = (res + ((ll) f(p[i], c) * dfs_F(i +
91 1, n / pw) + f(p[i], c + 1))) % mod;
92     }
93     return res;
94 }
95
96 void init(ll _n) {
97     n = _n;
98     B = sqrt(n) + 100;
99     pn = 0;
100     sieve(B);
101     m = 0;
102     for(ll i = 1, j; i <= n; i = j + 1) {

```

```

100         j = n / (n / i);
101         ll t = n / i;
102         _id[id(t)] = ++m;
103         w[m] = t;
104         g[m] = sg(t);
105         h[m] = sh(t);
106         //printf("id: %lld, w: %lld, g: %lld, h:
107 <= %lld\n", m, t, g[m], h[m]);
108     }
109     for (int j = 1; j <= pn; j++) {
110         ll z = (ll) p[j] * p[j];
111         for(int i = 1; i <= m && z <= w[i]; i++) {
112             int k = Id(w[i] / p[j]);
113             g[i] = (g[i] + (ll) (mod - fg(p[j])) *
114 <= (g[k] - pg[j - 1] + mod)) % mod;
115             h[i] = (h[i] + (ll) (mod - fh(p[j])) *
116 <= (h[k] - ph[j - 1] + mod)) % mod;
117         }
118     }
119     compress();
120
121     /* 递推 min25
122     for(int j = pn; j > 0; j--) {
123         ll z = (ll) p[j] * p[j];
124         for(int i = 1; i <= m && z <= w[i]; i++) {
125             ll pe = p[j];
126             for(int e = 1; pe * p[j] <= w[i]; e++, pe
127 <= p[j]) {
128                 g[i] = (g[i] + (ll) f(p[j], e) *
129 <= (g[Id(w[i] / pe)] - pg[j] + mod) + f(p[j], e + 1)) %
130 mod;
131             }
132         }
133     } */
134 }
135 ll get(ll x) { // x == n / i
136     if(x < 1) return 0;
137     return (dfs_F(1, x) + 1) % mod;
138 }
139 ll get(ll l, ll r) {
140     return get(r) - get(l - 1);
141 }
142
143 void Solve() {
144     long long n;
145     scanf("%lld", &n);
146     min25::init(n);
147     long long res = min25::get(n);
148     printf("%lld\n", res);
149 }
150
151 int main() {
152     min25::precalc();
153     int T;
154     scanf("%d", &T);
155     while(T--) {
156         Solve();
157     }
158 }

```

2.4 扩展欧几里得

```

1 using i128 = __int128;
2
3 // ax + by = c
4 // 有解当且仅当 gcd(a, b) | c
5 // 要求 a, b 不全为 0
6 // 无合法性检查

```

```

7 void exgcd(i128 a, i128 b, i128 &x, i128 &y, i128 c = 1) {
8     if (b == 0) {
9         x = c / a;
10        y = 0;
11    } else {
12        exgcd(b, a % b, x, y, c);
13        i128 tmp = x;
14        x = y;
15        y = tmp - (a / b) * y;
16    }
17 }

```

2.5 扩展欧拉定理

$$a^b \equiv a^{b \bmod \phi(p)}, (a, b) = 1$$

$$a^b \equiv a^{b \bmod \phi(p) + \phi(p)}, (a, b) \neq 1$$

2.6 中国剩余定理

2.6.1 两个数的 crt

```

1 // mul 表示慢速乘
2 ll crt(ll a1, ll p, ll a2, ll q) {
3     ll ip = fpow(p, q-2, q);
4     ll iq = fpow(q, p-2, p);
5     ll n = p * q;
6     return (mul(mul(a1, q, n), iq, n) + mul(mul(a2, p, n),
7     ↪ ip, n)) % n;
8 }

```

2.6.2 excrt

```

1 using ll = long long;
2
3 ll gcd(ll a, ll b) {
4     return b == 0 ? a : gcd(b, a % b);
5 }
6
7 // x === a1 (mod b1), x === a2 (mod b2)
8 // 合法性检查: 返回 -1 则为无解
9 pair<ll, ll> excrt(ll a1, ll b1, ll a2, ll b2) {
10    ll g = gcd(b1, b2);
11    ll lcm = (b1 / g) * b2;
12
13    if ((a1 - a2) % g) return {-1, -1};
14
15    i128 x, y;
16    exgcd(b1, b2, x, y, a1 - a2);
17    ll res = (a1 - b1 * x) % lcm;
18    if (res < 0) res += lcm;
19    return {res, lcm};
20 }

```

2.7 卢卡斯定理

2.7.1 模素数卢卡斯

```

1 // 卢卡斯定理, 要求 p 为素数
2 ll lucas(ll n, ll m, ll p) {
3     if (!m) return 1;
4     return C(n % p, m % p, p) * lucas(n / p, m / p, p) % p;
5 }

```

2.7.2 扩展卢卡斯

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 // -p: 素因子
6 // -l: 次数
7 // return: 本质不同素因子个数
8 // 1-base
9 int Fact(int x, int *p, int *l);
10
11 ll gcd(ll a, ll b) {
12     return b == 0 ? a : gcd(b, a % b);
13 }
14
15 void exgcd(ll a, ll b, ll &x, ll &y, ll c = 1);
16
17 // x === a1 (mod b1), x === a2 (mod b2)
18 // 合法性检查: 返回 -1 则为无解
19 pair<ll, ll> excrt(ll a1, ll b1, ll a2, ll b2);
20
21 // -----
22
23 // 扩展卢卡斯定理
24
25 // 扩欧求逆元
26 ll INV(ll a, ll p) {
27     ll x, y;
28     exgcd(a, p, x, y);
29     return (x % p + p) % p;
30 }
31
32 // 递归求解(n! / px) mod pk
33 ll F(ll n, ll p, ll pk) {
34     if (n == 0) return 1;
35     ll rou = 1; // 循环节
36     ll rem = 1; // 余项
37     for (ll i = 1; i <= pk; ++i) {
38         if (i % p)
39             rou = rou * i % pk;
40     }
41     rou = fpow(rou, n / pk, pk);
42     for (ll i = pk * (n / pk); i <= n; ++i) {
43         if (i % p)
44             rem = rem * (i % pk) % pk; // 小心i炸int
45     }
46     return F(n / p, p, pk) * rou % pk * rem % pk;
47 }
48
49 // 素数p在n!中的次数
50 ll G(ll n, ll p) {
51     if (n < p) return 0;
52     return G(n / p, p) + (n / p);
53 }
54
55 ll C_pk(ll n, ll m, ll p, ll pk) {
56     ll fz = F(n, p, pk), fm1 = INV(F(m, p, pk), pk),
57         fm2 = INV(F(n - m, p, pk), pk);
58     ll mi = fpow(p, G(n, p) - G(m, p) - G(n - m, p), pk);
59     return fz * fm1 % pk * fm2 % pk * mi % pk;
60 }
61
62 const int N = 107;
63
64 int ps[N], l[N];
65
66 ll exlucas(ll n, ll m, ll P) {
67     int num = Fact(P, ps, l); // 素因子分解@见素因子分
68     ↪ 解.cpp

```

```

71 ll res = 0, mo = 1;
72 for (int i = 1; i <= num; ++i) {
73     ll pk = 1;
74     for (int j = 0; j < l[i]; ++j) {
75         pk *= ps[i];
76     }
77
78     ll x = C_pk(n, m, ps[i], pk);
79     pair<ll, ll> pr = excrt(x, pk, res, mo);
80
81     mo = pr.second;
82     res = pr.first;
83 }
84
85 return res % P;
86 }
87
88 int main() {
89     ll n, m, p; cin >> n >> m >> p;
90     ll ans = exlucas(n, m, p);
91
92     cout << ans << endl;
93 }

```

2.8 原根与离散对数

2.8.1 原根

```

1 // 得到 p 的原根
2 ll generator(ll p) {
3     static ll rec, ans;
4     if (p == rec)
5         return ans;
6     rec = p;
7     vector<ll> fact;
8     ll phi = p - 1, n = phi;
9     for (ll i = 2; 1ll * i * i <= n; ++i) {
10         if (n % i == 0) {
11             fact.push_back(i);
12             while (n % i == 0)
13                 n /= i;
14         }
15     }
16     if (n > 1)
17         fact.push_back(n);
18     for (ll res = 2; res <= p; ++res) {
19         bool ok = 1;
20         for (ll factor : fact) {
21             if (fpow(res, phi / factor, p) == 1) {
22                 ok = false;
23                 break;
24             }
25         }
26         if (ok)
27             return ans = res;
28     }
29     return ans = -1;
30 }

```

2.8.2 BSGS

```

1 // a ^ k == b mod p
2 ll BSGS(ll a, ll b, ll p) { // p <= 1e9
3     static ll rec;
4     static map<ll, ll> mp;
5
6     ll sq = (ll)ceil(sqrt(p));
7     if (rec != p) {
8         rec = p;
9         mp.clear();

```

```

10         ll le = 1, bs = fpow(a, sq, p);
11         for (ll i = 1; i <= sq; ++i) {
12             le = le * bs % p;
13             if (le < 0)
14                 le += p;
15             mp[le] = i * sq;
16         }
17     }
18
19     ll ri = (b % p);
20     if (ri < 0)
21         ri += p;
22     for (ll j = 0; j <= sq; ++j) {
23         if (mp.count(ri)) {
24             return mp[ri] - j;
25         }
26         ri = ri * a % p;
27         if (ri < 0)
28             ri += p;
29     }
30     return -1;
31 }
32
33 // x ^ a == b mod p
34 ll calc(ll a, ll b, ll p) {
35     ll g = generator(p); // 求原根-见原根.cpp
36     ll ga = fpow(g, a, p);
37     ll c = BSGS(ga, b, p);
38     ll res = fpow(g, c, p);
39     return res;
40 }

```

2.9 杂项

2.9.1 大数整除小数取模

计算 $\frac{a}{b} \bmod p$:

当 a 的本值太大无法表示时, 可以计算 a 对 b * p 取模的结果, 再除 b, 模 p

$$\frac{a}{b} \bmod p = \frac{a \bmod b * p}{b} \bmod p$$

2.9.2 立方根复杂度求 mobius 函数

```

1 int getmu(int x) {
2     int pr, cur = 0;
3     for (int i = 1; i <= cnt; ++i) {
4         cur = 0;
5         while (x % prime[i] == 0) {
6             ++cur; x /= prime[i];
7         }
8         if (cur > 1) return 0;
9     }
10    if (x == 1) return 1;
11    int sq = sqrt(x) + 0.5;
12    if (1ll * sq * sq == x) return 0;
13    return 1;
14 }

```

2.9.3 直接求 euler 函数

```

1 // primes 为预处理的素数表
2 ll getPhi(ll x) {
3     ll phi = x, num = x;
4     for (int p : primes) {
5         if (p > x / p) break;
6         if (x % p == 0)

```



```
7     phi = (phi / p) * (p - 1), x /= p;  
8     while (x % p == 0)  
9         x /= p;  
10    }  
11    if (x > 1)  
12        phi = phi / x * (x - 1);  
13    return phi;  
14 }
```

3 数据结构

3.1 点分治

```

1 vector<int> vec[N];
2 bool vis[N];
3
4 namespace DFZ {
5
6     int sz[N], maxx[N];
7
8     int rt, sum;
9     void calcsz(int x, int dad) {
10         //cerr << rt << " " << sum << endl;
11         maxx[x] = 0; sz[x] = 1;
12         for (int y : vec[x]) {
13             if(y == dad || vis[y]) continue;
14             calcsz(y, x);
15             sz[x] += sz[y];
16             maxx[x] = max(sz[y], maxx[x]);
17         }
18         assert(sum - sz[x] >= 0);
19         maxx[x] = max(maxx[x], sum - sz[x]);
20         if (maxx[x] < maxx[rt]) rt = x;
21     }
22
23     void dfz(int x) {
24         calcsz(x, 0); vis[rt] = 1;
25         TREE::init();
26         TREE::dfs(rt, 0, 0, 1);
27         ALL += TREE::gao();
28         // calc mx, dis, build seg treedp
29
30         int cen = rt, psum = sum;
31         for (int y : vec[cen]) {
32             if (vis[y]) continue;
33             rt = cen;
34             sum = (sz[y] > sz[cen] ? psum - sz[cen] :
35                 ↪ sz[y]);
36             dfz(y);
37         }
38     }
39
40     DFZ::rt = 0; DFZ::maxx[0] = inf; DFZ::sum = n;
41     DFZ::dfz(1);

```

3.2 树链剖分

```

1 vector<int> vec[N];
2
3 struct cutTree {
4     int dep[N], sz[N];
5     int nd[N], fn[N], gn[N], clc;
6     int top[N], son[N], fa[N];
7
8     void init(int n) {
9         clc = 0;
10         for(int x = 1; x <= n; x++) son[x] = 0;
11         for(int x = 1; x <= n; x++) fa[x] = 0;
12     }
13     // dfs1(rt, 1);
14     void dfs1(int x, int d) {
15         dep[x] = d, sz[x] = 1;
16         for(int y : vec[x]) {
17             if(y == fa[x]) continue;
18             fa[y] = x; dfs1(y, d + 1); sz[x] += sz[y];
19             if(!son[x] || sz[y] > sz[son[x]]) son[x] = y;
20         }
21     }

```

```

22     // dfs2(rt, rt);
23     void dfs2(int x, int an) {
24         top[x] = an;
25         nd[fn[x] = ++clc] = x;
26         if(son[x]) dfs2(son[x], an);
27         for(int y : vec[x]) {
28             if(y == fa[x] || y == son[x]) continue;
29             dfs2(y, y);
30         }
31         gn[x] = clc;
32     }
33     int lca(int x, int y) {
34         while(top[x] != top[y]) {
35             if(dep[top[x]] > dep[top[y]])
36                 swap(x, y);
37             y = fa[top[y]];
38         }
39         return dep[x] < dep[y] ? x : y;
40     }
41     // anc[x][0] = x, !! 默认根为 1, 若 rt 非 1, 需要传参
42     int up(int x, int k, int rt = 1) {
43         if(k < 0) return -1;
44         if(k >= dep[x]) return 0;
45         while(k) {
46             if(fn[x] - fn[top[x]] < k)
47                 ↪ k -= fn[x] - fn[top[x]] + 1, x =
48                 ↪ fa[top[x]];
49             else x = nd[fn[x] - k], k = 0;
50         }
51         return x;
52     }
53     int query(int x, int y) {
54         int res = 0;
55         while(top[x] != top[y]) {
56             if(dep[top[x]] > dep[top[y]]) swap(x, y);
57             int l = fn[top[y]], r = fn[y];
58             res += bit.query(r) - bit.query(l - 1);
59             y = fa[top[y]];
60         }
61         if(dep[x] > dep[y]) swap(x, y);
62         ↪ return res + bit.query(fn[y]) - bit.query(fn[x] -
63         ↪ 1);
64     }
65 } CT;

```

3.3 主席树

```

1 const int N = 100010;
2 const int M = N * 40;
3
4 struct node {
5     int l, r;
6     ll val;
7 } tr[M];
8
9 int rt[N], tot;
10
11 int clone(int k) {
12     ++tot;
13     tr[tot] = tr[k];
14     return tot;
15 }
16
17 void up(int k) {
18     tr[k].val = tr[tr[k].l].val ^ tr[tr[k].r].val;
19 }
20
21 void upd(int &k, int x, int l, int r, ll val) {
22     k = clone(k);
23     if(l == r) {
24         tr[k].val ^= val;

```

```

25     } else {
26         int m = ((l + r) >> 1);
27         if (x <= m) upd(tr[k].l, x, l, m, val);
28         else upd(tr[k].r, x, m + 1, r, val);
29         up(k);
30     }
31 }
32
33 /* 树形建主席树 */
34 void dfs(int x, int dad) {
35     rt[x] = rt[dad];
36     upd(rt[x], a[x], 1, n, val);
37     for (auto y : ch[x]) {
38         if (y == dad) continue;
39         dfs(y, x);
40     }
41 }

```

3.4 线段树合并

```

1 // merge v to u
2
3 void merge(int &u, int v, int l, int r) {
4     if (!v) return;
5     else if (!u) u = v;
6     else if (l == r) {
7         // merge at leaf
8     } else {
9         int mid = ((l + r) >> 1);
10        merge(tr[u].l, tr[v].l, l, mid);
11        merge(tr[u].r, tr[v].r, mid + 1, r);
12        up(u);
13    }
14 }

```

3.5 zkw 线段树

```

1 // 0-base
2
3 struct SegT {
4     typedef pair<int, int> T;
5     T ini = {0, 0};
6     T combine(T u, T v) {
7         return max(u, v);
8     }
9     int n; vector<T> arr;
10
11     // local
12     SegT(int sz): n{1} { while(n < sz) n <= 1;
13     ↪ arr.resize(n * 2, ini); };
14
15     // global
16     void init(int sz) {
17         arr.clear(); for (n = 1; n < sz; n <= 1);
18     ↪ arr.resize(n * 2, ini);
19     }
20
21     void update(int i, T v) {
22         for (arr[i += n] = v; i >>= 1; )
23             arr[i] = combine(arr[i<<1], arr[i<<1|1]);
24     }
25
26     T query(int l, int r) {
27         T resl = ini, resr = ini;
28         for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
29             if (l & 1) resl = combine(resl, arr[l++]);
30             if (r & 1) resr = combine(resr, arr[--r]);
31         }
32         return combine(resl, resr);
33     }
34 }

```

```

31 };

```

3.6 矩形面积并

```

1 // 矩形面积并，标记永久化，注意要开八倍空间
2
3 typedef long long ll;
4
5 const int maxn = 100050;
6
7 struct node {
8     int sum, lazy;
9     void clear() {
10         sum = 0; lazy = 0;
11     }
12 } tr[maxn << 3];
13
14 void up(int k, int l, int r) {
15     if (tr[k].lazy) {
16         tr[k].sum = r - l + 1;
17     } else {
18         tr[k].sum = tr[k<<1].sum + tr[k<<1|1].sum;
19     }
20 }
21
22 void build(int k, int l, int r) {
23     tr[k].clear();
24
25     if (l == r) {
26         tr[k<<1].clear();
27         tr[k<<1|1].clear();
28         return;
29     }
30
31     int mid = ((l + r) >> 1);
32     build(k<<1, l, mid);
33     build(k<<1|1, mid + 1, r);
34     up(k, l, r);
35 }
36
37 void upd(int k, int cl, int cr, int tag, int l, int r) {
38     if (cl <= l && r <= cr) {
39         tr[k].lazy += tag;
40         up(k, l, r);
41     } else {
42         int mid = ((l + r) >> 1);
43         if (cl <= mid) upd(k<<1, cl, cr, tag, l, mid);
44         if (cr > mid) upd(k<<1|1, cl, cr, tag, mid + 1, r);
45         up(k, l, r);
46     }
47 }
48
49 // 查询的时候记一下祖先结点是否有 lazy > 0

```

4 图论

4.1 2-sat

4.2 Dinic 网络流

```

1 using ll = long long;
2
3 // 前向链表多测注意清空 $\text{tot}$  初值要设为 1
4 struct edge {
5     int to, pre;
6     ll w;
7 } e[M];
8 int last[N], tot = 1;
9
10 void ine(int a, int b, ll w) {
11     e[++tot] = (edge){b, last[a], w};
12     last[a] = tot;
13 }
14
15 void add(int a, int b, ll w) {
16     ine(a, b, w);
17     ine(b, a, 0);
18 }
19
20 inline int sgn(ll v) {
21     if (v == 0) return 0;
22     return v > 0 ? 1 : -1;
23 }
24
25 namespace Dinic {
26
27     int n, s, t;
28     int lv[N], cur[M]; //  $lv$  层数  $cur$  当前弧
29
30     bool bfs() {
31         fill(lv + 1, lv + 1 + n, -1);
32         lv[s] = 0;
33         copy(last + 1, last + 1 + n, cur + 1);
34         queue<int> q;
35         q.push(s);
36         while(!q.empty()) {
37             int u = q.front(); q.pop();
38             for(int i = cur[u]; i; i = e[i].pre) {
39                 int to = e[i].to;
40                 ll vol = e[i].w;
41                 if(vol > 0 && lv[to] == -1)
42                     lv[to] = lv[u] + 1, q.push(to);
43             }
44         }
45         return lv[t] != -1; // 如果汇点未访问过则不可达
46     }
47
48     ll dfs(int u = s, ll f = inf) {
49         if(u == t) return f;
50         for(int &i = cur[u]; i; i = e[i].pre) {
51             int to = e[i].to;
52             ll vol = e[i].w;
53             if(vol > 0 && lv[to] == lv[u] + 1) {
54                 ll c = dfs(to, min(vol, f));
55                 if(sgn(c)) {
56                     e[i].w -= c;
57                     e[i ^ 1].w += c; // 反向边
58                     return c;
59                 }
60             }
61         }
62     }
63     return 0; // 输出流量大小
64 }
65

```

```

66     ll dinic(int _n, int _s, int _t) {
67         n = _n; s = _s; t = _t;
68         ll ans = 0;
69         while(bfs()) {
70             ll f;
71             while((f = dfs()) > 0)
72                 ans += f;
73         }
74         return ans;
75     }
76 } // namespace Dinic
77
78 using Dinic::dinic;

```

4.3 Dijkstra 费用流

```

1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/priority_queue.hpp>
3 using ll = long long;
4 using namespace std;
5 typedef pair<ll, ll> P;
6
7 const int N = 5e3 + 7, M = 1e6 + 7;
8 const ll inf = 0x3f3f3f3f3f3f3f3f;
9
10 struct edge {
11     int to, pre;
12     ll w, c; //  $w$ : 流量  $weight$   $c$ : 费用  $cost$ 
13 } e[M * 2];
14 int head[N], ecnt = 1; // 加入的第一条边编号为 2
15 void ine(int u, int v, ll w, ll c) {
16     e[++ecnt] = {v, head[u], w, c};
17     head[u] = ecnt;
18 }
19 void add(int u, int v, ll w, ll c) {
20     ine(u, v, w, c); ine(v, u, 0, -c);
21 }
22 void init(int n) {
23     fill(head + 1, head + 1 + n, 0);
24     ecnt = 1;
25 }
26
27 ll h[N], dis[N]; // bool vis[N];
28 int pe[N]; // 父边
29
30 // spfa 只用来预处理  $h$  不跑流也可以用来跑流见注释
31 //  $h$  在  $flow\_dij$  调用时清空
32 void spfa(int s, int n) {
33     static bool inq[N];
34
35     fill(dis + 1, dis + 1 + n, inf);
36     fill(inq + 1, inq + 1 + n, false);
37
38     queue<int> q;
39
40     q.push(s);
41     dis[s] = 0;
42
43     while (!q.empty()) {
44         int x = q.front();
45         q.pop();
46         inq[x] = false;
47
48         for (int i = head[x]; i; i = e[i].pre)
49             if (e[i].w > 0) {
50                 int y = e[i].to;
51
52                 if (dis[y] > dis[x] + e[i].c) {
53                     //  $pe[y] = i$ ; 若要跑流则加上这一行
54                     dis[y] = dis[x] + e[i].c;
55                     if (!inq[y]) {

```

```

56         q.push(y);
57         inq[y] = true;
58     }
59 }
60 }
61 }
62 }
63
64 void dij(int s, int n) {
65     fill(dis + 1, dis + 1 + n, inf);
66     dis[s] = 0;
67     // fill(vis + 1, vis + 1 + n, false);
68
69     using pq_t = __gnu_pbds::priority_queue<P, greater<P>,
70 ↪ __gnu_pbds::thin_heap_tag>;
71     pq_t q;
72     static pq_t::point_iterator it[N];
73
74     for (int i = 1; i <= n; i++)
75         it[i] = q.push({dis[i], i});
76
77     while(!q.empty()) {
78         auto tp = q.top();
79         int x = tp.second;
80         ll w = tp.first;
81         q.pop();
82
83         if(w != dis[x]) continue;
84         // if(vis[x]) continue;
85         // vis[x] = true;
86
87         for(int i = head[x]; i; i = e[i].pre) {
88             int y = e[i].to;
89             if(e[i].w > 0 && dis[y] > dis[x] + h[x] - h[y]
90 ↪ + e[i].c) {
91                 pe[y] = i;
92                 dis[y] = dis[x] + h[x] - h[y] + e[i].c;
93                 q.modify(it[y], {dis[y], y});
94             }
95         }
96     }
97
98 P flow_dij(int s, int t, int n) {
99     fill(h + 1, h + 1 + n, 0);
100     /*
101     spfa(s, n);
102     for(int i = 1; i <= n; i++)
103         h[i] = dis[i];
104     // 如果初始有负权就像这样跑一遍 SPFA 预处理 h 数组
105     */
106
107     ll cost = 0, flow = 0;
108
109     while (1) {
110         dij(s, n);
111         if(dis[t] >= inf) break;
112
113         for (int i = 1; i <= n; i++) // 先更新 h 数组
114             h[i] += dis[i];
115
116         ll nowflow = inf;
117         for (int x = t; x != s; x = e[pe[x] ^ 1].to)
118             nowflow = min(nowflow, e[pe[x]].w);
119
120         flow += nowflow;
121         cost += nowflow * h[t]; // 计算流量和费用
122
123         for (int x = t; x != s; x = e[pe[x] ^ 1].to) {
124             e[pe[x]].w -= nowflow;
125             e[pe[x] ^ 1].w += nowflow;
126         } // 更新边容量

```

```

126     }
127     return {flow, cost};
128 }
129
130 // https://ac.nowcoder.com/acm/problem/222408
131 int main() {
132     ios::sync_with_stdio(0), cin.tie(nullptr),
133     ↪ cout.tie(nullptr);
134     int n; cin >> n;
135     vector<int> z(n + 1), v(n + 1);
136     ll ans = 0;
137     for(int i = 1; i <= n; i++) {
138         int x, y; cin >> x >> y >> z[i] >> v[i];
139         ans += x * x;
140         ans += y * y;
141         ans += z[i] * z[i];
142     }
143     int num = n * 2 + 2, s = num - 1, t = num;
144     init(num);
145     for(int i = 1; i <= n; i++)
146         for(int j = 1; j <= n; j++)
147             add(i, j + n, 1, 1ll * v[i] * v[i] * (j - 1) *
148 ↪ (j - 1) + 2ll * v[i] * z[i] * (j - 1));
149     for(int i = 1; i <= n; i++)
150         add(s, i, 1, 0);
151     for(int i = 1; i <= n; i++)
152         add(n + i, t, 1, 0);
153     ans += flow_dij(s, t, num).second;
154     cout << ans << '\n';
155     return 0;

```

4.4 二分图最大带权匹配

```

1 // 1-base
2 // 主过程: 设置好 cost[N][N] 即可调用
3 // cost 可以为负数
4
5 using ll = long long;
6 #define prev prevv
7
8 const int N = 305;
9 const ll INF = 0x3f3f3f3f3f3f3f3f;
10
11 int n;
12 ll cost[N][N];
13 ll lx[N], ly[N];
14 int match[N];
15 ll slack[N];
16 int prev[N];
17 bool vy[N];
18
19 void augment(int root) {
20     fill(vy + 1, vy + n + 1, false);
21     fill(slack + 1, slack + n + 1, INF);
22     int py;
23     match[py = 0] = root;
24
25     do {
26         vy[py] = true;
27         int x = match[py];
28         ll delta = INF;
29         int yy;
30         for (int y = 1; y <= n; y++) {
31             if (!vy[y]) {
32                 if (lx[x] + ly[y] - cost[x][y] < slack[y])
33 ↪ {
34                     slack[y] = lx[x] + ly[y] - cost[x][y];
35                     prev[y] = py;
36                 }
37                 if (slack[y] < delta) {

```

```

37         delta = slack[y];
38         yy = y;
39     }
40 }
41 }
42 for (int y = 0; y <= n; y++) {
43     if (vy[y]) {
44         lx[match[y]] -= delta;
45         ly[y] += delta;
46     } else {
47         slack[y] -= delta;
48     }
49 }
50 py = yy;
51 } while (match[py] != -1);
52
53 do {
54     int pre = prev[py];
55     match[py] = match[pre];
56     py = pre;
57 } while (py);
58 }
59
60 ll KM() {
61     for (int i = 1; i <= n; i++) {
62         lx[i] = ly[i] = -INF;
63         match[i] = -1;
64         for (int j = 1; j <= n; j++) {
65             lx[i] = std::max(lx[i], cost[i][j]);
66         }
67     }
68     ll answer = 0;
69     for (int root = 1; root <= n; root++) {
70         augment(root);
71     }
72     for (int i = 1; i <= n; i++) {
73         answer += lx[i];
74         answer += ly[i];
75         // printf("%d %d\n", match[i], i);
76     }
77     return answer;
78 }

```

4.5 带花树

```

1 // 带花树：一般图最大匹配
2 const int maxn = 505;
3
4 struct Match {
5     int n, father[maxn], vst[maxn], match[maxn], pre[maxn],
6     ⇨ Type[maxn], times;
7     vector<int> edges[maxn];
8     queue<int> Q;
9
10    void ine(int x, int y) { edges[x].push_back(y); }
11    void ine2(int x, int y) {
12        ine(x, y);
13        ine(y, x);
14    }
15
16    void init(int num) {
17        times = 0;
18        n = num;
19        for (int i = 0; i <= n; ++i)
20            edges[i].clear(), vst[i] = 0, match[i] = 0,
21            ⇨ pre[i] = 0;
22    }
23
24    int LCA(int x, int y) {
25        times++;
26        x = father[x], y = father[y]; // 已知环位置

```

```

25    while (vst[x] != times) {
26        if (x) {
27            vst[x] = times;
28            x = father[pre[match[x]]];
29        }
30        swap(x, y);
31    }
32    return x;
33 }
34
35 void blossom(int x, int y, int lca) {
36     while (father[x] != lca) {
37         pre[x] = y;
38         y = match[x];
39         if (Type[y] == 1) {
40             Type[y] = 0;
41             Q.push(y);
42         }
43         father[x] = father[y] = father[lca];
44         x = pre[y];
45     }
46 }
47
48 int Augment(int s) {
49     for (int i = 0; i <= n; ++i)
50         father[i] = i, Type[i] = -1;
51     Q = queue<int>();
52     Type[s] = 0;
53     Q.push(s); // 仅入队o型点
54     while (!Q.empty()) {
55         int Now = Q.front();
56         Q.pop();
57         for (int Next : edges[Now]) {
58             if (Type[Next] == -1) {
59                 pre[Next] = Now;
60                 Type[Next] = 1; // 标记为i型点
61                 if (!match[Next]) {
62                     for (int to = Next, from = Now; to;
63 ⇨ from = pre[to]) {
64                         match[to] = from;
65                         swap(match[from], to);
66                     }
67                     return true;
68                 }
69                 Type[match[Next]] = 0;
70                 Q.push(match[Next]);
71             } else if (Type[Next] == 0 && father[Now] !=
72 ⇨ = father[Next]) {
73                 int lca = LCA(Now, Next);
74                 blossom(Now, Next, lca);
75                 blossom(Next, Now, lca);
76             }
77         }
78         return false;
79     }
80 }
81
82 void gao() {
83     int res = 0; // 最大匹配数
84     for (int i = n; i >= 1; --i)
85         if (!match[i])
86             res += Augment(i);
87     printf("%d\n", res);
88     for (int i = 1; i <= n; ++i)
89         printf("%d ", match[i]);
90     printf("\n");
91 }
92
93 int main() {
94     int n, m;
95     cin >> n >> m;

```

```
95     G.init(n);
96     for (int i = 1, x, y; i <= m; i++) {
97         scanf("%d %d", &x, &y);
98         G.ine2(x, y);
99     }
100     G.gao();
101     return 0;
102 }
```

4.6 Hall 定理

霍尔定理

二分图中, 左侧点集 X 存在最大匹配的充要条件是:

X 中的任意 k 个点至少与 Y 中 k 个点相邻.

推论: 正则二分图存在完美匹配 (正则图 指每个点度数相等的图)

5 字符串

5.1 后缀自动机

```

1 // 在字符集比较小的时候可以直接开go数组, 否则需要用map或者
  ↳ 哈希表替换
2 // 注意!!!结点数要开成串长的两倍
3
4 // 全局变量与数组定义
5 int last, len[maxn], fa[maxn], go[maxn][26], sam_cnt;
6 int c[maxn], q[maxn]; // 用来桶排序
7
8 // 在主函数开头加上这句初始化
9 last = sam_cnt = 1;
10
11 // 以下是按val进行桶排序的代码
12 for (int i = 1; i <= sam_cnt; i++)
13     c[len[i] + 1]++;
14 for (int i = 1; i <= n; i++)
15     c[i] += c[i - 1]; // 这里n是串长
16 for (int i = 1; i <= sam_cnt; i++)
17     q[++c[len[i]]] = i;
18
19 //加入一个字符 均摊O(1)
20 void extend(int c) {
21     int p = last, np = ++sam_cnt;
22     len[np] = len[p] + 1;
23
24     while (p && !go[p][c]) {
25         go[p][c] = np;
26         p = fa[p];
27     }
28
29     if (!p)
30         fa[np] = 1;
31     else {
32         int q = go[p][c];
33
34         if (len[q] == len[p] + 1)
35             fa[np] = q;
36         else {
37             int nq = ++sam_cnt;
38             len[nq] = len[p] + 1;
39             memcpy(go[nq], go[q], sizeof(go[q]));
40
41             fa[nq] = fa[q];
42             fa[np] = fa[q] = nq;
43
44             while (p && go[p][c] == q){
45                 go[p][c] = nq;
46                 p = fa[p];
47             }
48         }
49     }
50
51     last = np;
52 }

```

5.2 广义后缀自动机

5.2.1 对 Trie 建自动机

以 bfs 遍历 Trie 上的每一个结点, 以 父结点 在 SAM 中对应的结点为 last 调用 insert 即可。

5.2.2 对多模式串建自动机

```

1 // 多串创建 sam 的方法:
2 // 在插入每个串前, 设置 last = 1, 然后一路 last =
  ↳ extend(last, c)
3
4 int extend(int p, int c) {
5     int np = 0;
6
7     if (!go[p][c]) {
8         np = ++sam_cnt;
9         len[np] = len[p] + 1;
10        while (p && !go[p][c]) {
11            go[p][c] = np;
12            p = fa[p];
13        }
14    }
15
16    if (!p)
17        fa[np] = 1;
18    else {
19        int q = go[p][c];
20
21        if (len[q] == len[p] + 1) {
22            if (np)
23                fa[np] = q;
24            else
25                return q;
26        }
27        else {
28            int nq = ++sam_cnt;
29            len[nq] = len[p] + 1;
30            memcpy(go[nq], go[q], sizeof(go[q]));
31
32            fa[nq] = fa[q];
33            fa[q] = nq;
34            if (np)
35                fa[np] = nq;
36
37            while (p && go[p][c] == q){
38                go[p][c] = nq;
39                p = fa[p];
40            }
41
42            if (!np)
43                return nq;
44        }
45    }
46
47    return np;
48 }

```

5.3 AC 自动机

```

1 using namespace std;
2 const int maxn = 500000 + 5;
3
4 const int rt = 0; // 默认 trie 树根为 0
5
6 int tr[maxn][26], tot = rt; // 初始化时要设置 tot = rt
7 int e[maxn]; // 标记字符串结尾
8 int fail[maxn];
9
10 void insert(char *s) {
11     int p = rt; // from root
12     for (int i = 0; s[i]; i++) {
13         int k = s[i] - 'a';
14         if (!tr[p][k])
15             tr[p][k] = ++tot; // 根节点为0

```

```

16     p = tr[p][k];
17 }
18 e[p]++; // 尾部标记
19 }
20
21 void build() {
22     queue<int> q;
23     fill(fail, fail + 1 + tot, 0);
24
25     for (int i = 0; i < 26; i++)
26         if (tr[rt][i])
27             q.push(tr[rt][i]);
28
29     while (!q.empty()) {
30         int k = q.front(); q.pop();
31         for (int i = 0; i < 26; i++) {
32             if (tr[k][i]) {
33                 fail[tr[k][i]] = tr[fail[k]][i];
34                 q.push(tr[k][i]); // 入队
35             } else
36                 tr[k][i] = tr[fail[k]][i]; // trie图
37         }
38     }
39 }
40
41 int query(char *t) {
42     int p = rt, ans = 0;
43     for (int i = 0; t[i]; i++) {
44         p = tr[p][t[i] - 'a'];
45         for (int j = p; j && (e[j] != -1); j = fail[j]) {
46             ans += e[j];
47             e[j] = -1; // 防止重复遍历
48         }
49     }
50     return ans;
51 }

```

5.4 后缀数组

5.4.1 倍增

```

1 // h[i] = lcp(sa[i], sa[i - 1])
2 // sa[i] = 排名为 i 的后缀的下标
3 // rk[i] = 后缀 i 的排名
4 // 需要将 s[n] 设为一个比一切字符大的数才能正确地输出
5 // height
6 // 不需要清空
7
8 template<int MAXN> class SA {
9 public:
10     int n, sa[MAXN], rk[MAXN], h[MAXN];
11
12     void init() {
13         // 不需要 init
14     }
15
16     void compute(int *s, int n, int m) {
17         int i, p, w, j, k;
18         this->n = n;
19         if (n == 1) {
20             sa[0] = rk[0] = h[0] = 0; return;
21         }
22         memset(cnt, 0, m * sizeof(int));
23         for (i = 0; i < n; ++i) ++cnt[rk[i] = s[i]];
24         for (i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
25         for (i = n - 1; ~i; --i) sa[--cnt[rk[i]]] = i;
26         for (w = 1; w < n; w <= 1, m = p) {
27             for (p = 0, i = n - 1; i >= n - w; --i) id[p++]
28                 = i;
29             for (i = 0; i < n; ++i)
30                 if (sa[i] >= w) id[p++] = sa[i] - w;
31         }
32     }
33 }

```

```

29     memset(cnt, 0, m * sizeof(int));
30     for (i = 0; i < n; ++i) ++cnt[px[i] =
31         rk[id[i]]];
32     for (i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
33     for (i = n - 1; ~i; --i) sa[--cnt[px[i]]] =
34         id[i];
35     memcpy(old_rk, rk, n * sizeof(int));
36     for (i = p = 1, rk[sa[0]] = 0; i < n; ++i)
37         rk[sa[i]] = cmp(sa[i], sa[i-1], w) ? p - 1
38             : p++;
39     }
40     for (i = 0; i < n; ++i) rk[sa[i]] = i;
41     for (i = k = h[rk[0]] = 0; i < n; h[rk[i++]] = k)
42         if (rk[i])
43             for (k > 0 ? --k : 0, j = sa[rk[i] - 1];
44                 s[i + k] == s[j + k]; ++k) {}
45     }
46 private:
47     int old_rk[MAXN], id[MAXN], px[MAXN], cnt[MAXN];
48     bool cmp(int x, int y, int w) {
49         return old_rk[x] == old_rk[y] && old_rk[x + w] ==
50             old_rk[y + w];
51     }
52 }
53 };

```

5.4.2 SAIS

```

1 const int BUFFER_SIZE = 1u << 26 | 1;
2 char buf[BUFFER_SIZE], *buf_ptr = buf;
3
4 #define alloc(x, type, len) \
5     type *x = (type *)buf_ptr; \
6     buf_ptr += (len) * sizeof(type);
7
8 #define clear_buf() \
9     memset(buf, 0, buf_ptr - buf), buf_ptr = buf;
10
11 template<int MAXN> class SuffixArray {
12 #define ltype true
13 #define stype false
14
15 public:
16     int sa[MAXN], rk[MAXN], hei[MAXN];
17
18     void compute(int n, int m, int *s) {
19         sais(n, m, s, sa);
20         for (int i = 0; i < n; ++i) rk[sa[i]] = i;
21         for (int i = 0, h = 0; i < n; i++) {
22             if (rk[i]) {
23                 int j = sa[rk[i] - 1];
24                 while (s[i + h] == s[j + h]) ++h;
25                 hei[rk[i]] = h;
26             } else {
27                 h = 0;
28             }
29             if (h) --h;
30         }
31     }
32
33 private:
34     int lbuc[MAXN], sbuc[MAXN];
35
36     void induce(int n, int m, int *s, bool *type, int *sa,
37         int *buc,
38         int *lbuc, int *sbuc)
39     {
40         memcpy(lbuc + 1, buc, m * sizeof(int));
41         memcpy(sbuc + 1, buc + 1, m * sizeof(int));
42
43         sa[lbuc[s[n - 1]]++] = n - 1;
44     }
45 }

```

```

44     for (int i = 0; i < n; i++) {
45         int t = sa[i] - 1;
46         if (t >= 0 && type[t] == ltype)
47             sa[lbuc[s[t]]++] = t;
48     }
49
50     for (int i = n - 1; i >= 0; i--) {
51         int t = sa[i] - 1;
52         if (t >= 0 && type[t] == stype)
53             sa[--sbuc[s[t]]] = t;
54     }
55 }
56
57 void sais(int n, int m, int *s, int *sa) {
58     alloc(type, bool, n + 1);
59     alloc(buc, int, m + 1);
60
61     type[n] = false;
62
63     for (int i = n - 1; i >= 0; i--) {
64         ++buc[s[i]];
65         type[i] = s[i] > s[i + 1] || (s[i] == s[i + 1]
66     ↪ && type[i + 1] == ltype);
67     }
68     for (int i = 1; i <= m; i++) {
69         buc[i] += buc[i - 1];
70         sbuc[i] = buc[i];
71     }
72     memset(rk, -1, n * sizeof(int));
73
74     alloc(lms, int, n + 1);
75
76     int n1 = 0;
77     for (int i = 0; i < n; i++) {
78         if (!type[i] && (i == 0 || type[i - 1]))
79             lms[rk[i] = n1++] = i;
80     }
81     lms[n1] = n;
82     memset(sa, -1, n * sizeof(int));
83
84     for (int i = 0; i < n1; i++) sa[--sbuc[s[lms[i]]]]
85     ↪ = lms[i];
86     induce(n, m, s, type, sa, buc, lbuc, sbuc);
87
88     int m1 = 0;
89     alloc(s1, int, n + 1);
90
91     for (int i = 0, t = -1; i < n; i++) {
92         int r = rk[sa[i]];
93         if (r != -1) {
94             int len = lms[r + 1] - sa[i] + 1;
95             m1 += t == -1 || len != lms[rk[t] + 1] - t
96     ↪ + 1 ||
97             memcmp(s + t, s + sa[i], len *
98     ↪ sizeof(int)) != 0;
99             s1[r] = m1;
100             t = sa[i];
101         }
102     }
103     alloc(sa1, int, n + 1);
104
105     if (n1 == m1) {
106         for (int i = 0; i < n1; i++)
107             sa1[s1[i] - 1] = i;
108     } else {
109         sais(n1, m1, s1, sa1);
110     }
111
112     memset(sa, -1, n * sizeof(int));
113     memcpy(sbuc + 1, buc + 1, m * sizeof(int));

```

```

112     for (int i = n1 - 1; i >= 0; i--) {
113         int t = lms[sa1[i]];
114         sa[--sbuc[s[t]]] = t;
115     }
116     induce(n, m, s, type, sa, buc, lbuc, sbuc);
117 }
118 #undef stype
119 #undef rtype
120 };

```

5.5 马拉车

```

1  /* manacher */
2  const int maxn = 300007;
3  // s[0]: 特殊值 0s[1, 3 .. tot]: 分隔值
4  int s[maxn], p[maxn]; // p 为半径
5  //int L[maxn], R[maxn]; // 包含点i的回文串的回文中心最左最右
6  int n, tot;
7
8  void manacher() {
9      int right = 0, idx = 0;
10     for(int i = 1; i <= tot; i++) {
11         if(i < right)
12             p[i] = min(p[2 * idx - i], right - i);
13         else
14             p[i] = 1;
15
16         while(i + p[i] <= tot && s[i + p[i]] == s[i -
17     ↪ p[i]])
18             p[i]++;
19
20         p[i]--;
21
22         if(i + p[i] > right) {
23             right = i + p[i];
24             idx = i;
25         }
26     }

```

6 动态规划

```
63 }
64 }
```

6.1 数位 DP

```
1 // 记忆化搜索
2 struct cmp {
3     bool operator()(const state& a, const state& b) const {
4         // ...
5     }
6 };
7
8 struct _hash {
9     size_t operator()(const state& st) const {
10         size_t res = st.cnt[0];
11         for(int i = 1; i < 10; ++i) {
12             res *= 19260817;
13             res += st.cnt[i];
14         }
15         return res;
16     }
17 };
18 unordered_map <state, ll, _hash, cmp> dp[20][20];
19
20 // -pos: 搜到的位置
21 // -st: 当前状态
22 // -lead: 是否有前导 0
23 // -limit: 是否有最高位限制
24 ll dfs(int pos, state st, int lead, int limit){
25     // 边界情况
26     if(pos < 0 /* && ... */) return 0;
27     // 记忆化搜索
28     int wd = st.w[st.d];
29     if(!limit && !lead && dp[pos][wd].count(st)) return
    ↪ dp[pos][wd][st];
30
31     ll res = 0;
32     // 最高位最大值
33     int cur = limit ? a[pos] : 9;
34     for(int i = 0; i <= cur; ++i) {
35         // 有前导0且当前位也是0
36         if(!i && lead) res += dfs(pos-1, st, 1,
    ↪ limit&&(i==cur));
37         // 有前导0且当前位非0 (出现最高位)
38         else if(i && lead) res += dfs(pos-1, st.add(i), 0,
    ↪ limit&&(i==cur));
39         else res += dfs(pos-1, st.add(i), 0,
    ↪ limit&&(i==cur));
40     }
41     // 没有前导0和最高限制时可以直接记录当前dp值以便下次搜
    ↪ 到同样的情况可以直接使用
42     if(!limit&&!lead) dp[pos][wd][st] = res;
43     return res;
44 }
45
46 ll gao(ll x) {
47     memset(a, 0, sizeof(a));
48     int len=0;
49     while(x) a[len++]=x%10,x/=10;
50     // init st
51     return dfs(len-1, st, 1, 1);
52 }
53
54 int main()
55 {
56     int T;
57     ll l, r; int d;
58     cin >> T;
59     while(T--) {
60         scanf("%lld %lld %d", &l, &r, &d);
61         ll ans = gao(r, d) - gao(l - 1, d);
62         printf("%lld\n", ans);
63     }
```

7 几何

```

1 #include <bits/stdc++.h>
2 #include <vector>
3 #define mp make_pair
4 #define fi first
5 #define se second
6 #define pb push_back
7 using namespace std;
8 using db = double;
9
10 mt19937 eng(time(0));
11
12 const db eps = 1e-6;
13 const db pi = acos(-1);
14 int sgn(db k) {
15     if (k > eps)
16         return 1;
17     else if (k < -eps)
18         return -1;
19     return 0;
20 }
21 // -1: < | 0: == | 1: >
22 int cmp(db k1, db k2) { return sgn(k1 - k2); }
23 // k3 in [k1, k2]
24 int inmid(db k1, db k2, db k3) { return sgn(k1 - k3) *
    ↪ sgn(k2 - k3) <= 0; }
25 // 点 (x, y)
26 struct point {
27     db x, y;
28     point operator+(const point &k1) const {
29         return (point){k1.x + x, k1.y + y};
30     }
31     point operator-(const point &k1) const {
32         return (point){x - k1.x, y - k1.y};
33     }
34     point operator*(db k1) const { return (point){x * k1, y
    ↪ * k1}; }
35     point operator/(db k1) const { return (point){x / k1, y
    ↪ / k1}; }
36     int operator==(const point &k1) const {
37         return cmp(x, k1.x) == 0 && cmp(y, k1.y) == 0;
38     }
39     // 逆时针旋转 k1 弧度
40     point rotate(db k1) {
41         return (point){x * cos(k1) - y * sin(k1), x *
    ↪ sin(k1) + y * cos(k1)};
42     }
43     // 逆时针旋转 90 度
44     point rotleft() { return (point){-y, x}; }
45     // 优先比较 x 坐标
46     bool operator<(const point k1) const {
47         int a = cmp(x, k1.x);
48         if (a == -1)
49             return 1;
50         else if (a == 1)
51             return 0;
52         else
53             return cmp(y, k1.y) == -1;
54     }
55     // 模长
56     db abs() { return sqrt(x * x + y * y); }
57     // 模长的平方
58     db abs2() { return x * x + y * y; }
59     // 与点 k1 的距离
60     db dis(point k1) { return ((*this) - k1).abs(); }
61     // 化为单位向量, require: abs() > 0
62     point unit() {
63         db w = abs();
64         return (point){x / w, y / w};
65     }

```

```

66     // 读入
67     void scan() {
68         double k1, k2;
69         scanf("%lf%lf", &k1, &k2);
70         x = k1;
71         y = k2;
72     }
73     // 输出
74     void print() { printf("%.11lf %.11lf\n", x, y); }
75     // 方向角 atan2(y, x)
76     db getw() { return atan2(y, x); }
77     // 将向量对称到 (-pi, pi] 半平面中
78     point getdel() {
79         if (sgn(x) == -1 || (sgn(x) == 0 && sgn(y) == -1))
80             return (*this) * (-1);
81         else
82             return (*this);
83     }
84     // (-pi, 0] -> 0, (0, pi] -> 1
85     int getP() const { return sgn(y) == 1 || (sgn(y) == 0
    ↪ && sgn(x) == -1); }
86 };
87
88 /* 点与线段的位置关系及交点 */
89
90 // k3 在 矩形 [k1, k2] 中
91 int inmid(point k1, point k2, point k3) {
92     return inmid(k1.x, k2.x, k3.x) && inmid(k1.y, k2.y,
    ↪ k3.y);
93 }
94 db cross(point k1, point k2) { return k1.x * k2.y - k1.y *
    ↪ k2.x; }
95 db dot(point k1, point k2) { return k1.x * k2.x + k1.y *
    ↪ k2.y; }
96 // 从 k1 转到 k2 的方向角
97 db rad(point k1, point k2) { return atan2(cross(k1, k2),
    ↪ dot(k1, k2)); }
98 // k1 k2 k3 逆时针 1 顺时针 -1 否则 0
99 int clockwise(point k1, point k2, point k3) {
100     return sgn(cross(k2 - k1, k3 - k1));
101 }
102 // 按 (-pi, pi] 顺序进行极角排序
103 int cmpangle(point k1, point k2) {
104     return k1.getP() < k2.getP() ||
105         (k1.getP() == k2.getP() && sgn(cross(k1, k2)) >
    ↪ 0);
106 }
107 // 点 q 在线段 k1, k2 上
108 int onS(point k1, point k2, point q) {
109     return inmid(k1, k2, q) && sgn(cross(k1 - q, k2 - k1))
    ↪ == 0;
110 }
111 // q 到直线 k1, k2 的投影
112 point proj(point k1, point k2, point q) {
113     point k = k2 - k1;
114     return k1 + k * (dot(q - k1, k) / k.abs2());
115 }
116 // q 关于直线 k1, k2 的镜像
117 point reflect(point k1, point k2, point q) { return
    ↪ proj(k1, k2, q) * 2 - q; }
118 // 判断 直线 (k1, k2) 和 直线 (k3, k4) 是否相交
119 int checkLL(point k1, point k2, point k3, point k4) {
120     return cmp(cross(k3 - k1, k4 - k1), cross(k3 - k2, k4 -
    ↪ k2)) != 0;
121 }
122 // 求直线 (k1, k2) 和 直线 (k3, k4) 的交点
123 point getLL(point k1, point k2, point k3, point k4) {
124     db w1 = cross(k1 - k3, k4 - k3), w2 = cross(k4 - k3, k2
    ↪ - k3);
125     return (k1 * w2 + k2 * w1) / (w1 + w2);
126 }
127 int intersect(db l1, db r1, db l2, db r2) {

```

```

128     if (l1 > r1)
129         swap(l1, r1);
130     if (l2 > r2)
131         swap(l2, r2);
132     return cmp(r1, l2) != -1 && cmp(r2, l1) != -1;
133 }
134 // 线段与线段相交判断 (非严格相交)
135 int checkSS(point k1, point k2, point k3, point k4) {
136     return intersect(k1.x, k2.x, k3.x, k4.x) &&
137         intersect(k1.y, k2.y, k3.y, k4.y) &&
138         sgn(cross(k3 - k1, k4 - k1)) * sgn(cross(k3 -
139     ↪ k2, k4 - k2)) <= 0 &&
140         sgn(cross(k1 - k3, k2 - k3)) * sgn(cross(k1 -
141     ↪ k4, k2 - k4)) <= 0;
142 }
143 // 点 q 到 直线 (k1, k2) 的距离
144 db disLP(point k1, point k2, point q) {
145     return fabs(cross(k1 - q, k2 - q)) / k1.dis(k2);
146 }
147 // 点 q 到 线段 (k1, k2) 的距离
148 db disSP(point k1, point k2, point q) {
149     point k3 = proj(k1, k2, q);
150     if (inmid(k1, k2, k3))
151         return q.dis(k3);
152     else
153         return min(q.dis(k1), q.dis(k2));
154 }
155 // 线段 (k1, k2) 到 线段 (k3, k4) 的距离
156 db disSS(point k1, point k2, point k3, point k4) {
157     if (checkSS(k1, k2, k3, k4))
158         return 0;
159     else
160         return min(min(disSP(k1, k2, k3), disSP(k1, k2,
161     ↪ k4)),
162             min(disSP(k3, k4, k1), disSP(k3, k4,
163     ↪ k2)));
164 }
165 /* 直线与半平面交 */
166 // 直线 p[0] -> p[1]
167 struct line {
168     point p[2];
169     line(point k1, point k2) {
170         p[0] = k1;
171         p[1] = k2;
172     }
173     point &operator[(int) k] { return p[k]; }
174     // k 严格位于直线左侧 / 半平面 p[0] -> p[1]
175     int include(point k) { return sgn(cross(p[1] - p[0], k
176     ↪ - p[0])) > 0; }
177     // 方向向量
178     point dir() { return p[1] - p[0]; }
179     // 向左平移 d, 默认为 eps
180     line push(db d = eps) {
181         point delta = (p[1] - p[0]).rotleft().unit() * d;
182         return {p[0] + delta, p[1] + delta};
183     }
184 };
185 // 直线与直线交点
186 point getLL(line k1, line k2) { return getLL(k1[0], k1[1],
187     ↪ k2[0], k2[1]); }
188 // 两直线平行
189 int parallel(line k1, line k2) { return sgn(cross(k1.dir(),
190     ↪ k2.dir())) == 0; }
191 // 平行且同向
192 int sameDir(line k1, line k2) {
193     return parallel(k1, k2) && sgn(dot(k1.dir(), k2.dir()))
194     ↪ == 1;
195 }
196 // 同向则左侧优先, 否则按极角排序, 用于半平面交
197 int operator<(line k1, line k2) {

```

```

192     if (sameDir(k1, k2))
193         return k2.include(k1[0]);
194     return cmpangle(k1.dir(), k2.dir());
195 }
196 // k3 (半平面) 包含 k1, k2 的交点, 用于半平面交
197 int checkpos(line k1, line k2, line k3) { return
198     ↪ k3.include(getLL(k1, k2)); }
199 // 求半平面交, 半平面是逆时针方向, 输出按照逆时针
200 vector<line> getHL(vector<line> L) {
201     sort(L.begin(), L.end());
202     deque<line> q;
203     for (int i = 0; i < (int)L.size(); i++) {
204         if (i && sameDir(L[i], L[i - 1]))
205             continue;
206         while (q.size() > 1 &&
207             ↪ !checkpos(q[q.size() - 2], q[q.size() - 1],
208             ↪ L[i]))
209             q.pop_back();
210         while (q.size() > 1 && !checkpos(q[1], q[0], L[i]))
211             q.pop_front();
212         q.push_back(L[i]);
213     }
214     while (q.size() > 2 && !checkpos(q[q.size() - 2],
215     ↪ q[q.size() - 1], q[0]))
216         q.pop_back();
217     while (q.size() > 2 && !checkpos(q[1], q[0], q[q.size()
218     ↪ - 1]))
219         q.pop_front();
220     vector<line> ans;
221     for (int i = 0; i < q.size(); i++)
222         ans.push_back(q[i]);
223     return ans;
224 }
225 db closepoint(vector<point> &A, int l,
226     ↪ int r) { // 最近点对, 先要按照 x 坐标排序
227     if (r - l <= 5) {
228         db ans = 1e20;
229         for (int i = l; i <= r; i++)
230             for (int j = i + 1; j <= r; j++)
231                 ans = min(ans, A[i].dis(A[j]));
232         return ans;
233     }
234     int mid = l + r >> 1;
235     db ans = min(closepoint(A, l, mid), closepoint(A, mid +
236     ↪ 1, r));
237     vector<point> B;
238     for (int i = l; i <= r; i++)
239         if (abs(A[i].x - A[mid].x) <= ans)
240             B.push_back(A[i]);
241     sort(B.begin(), B.end(), [](point k1, point k2) {
242     ↪ return k1.y < k2.y; });
243     for (int i = 0; i < B.size(); i++)
244         for (int j = i + 1; j < B.size() && B[j].y - B[i].y
245     ↪ < ans; j++)
246             ans = min(ans, B[i].dis(B[j]));
247     return ans;
248 }
249 /* 圆基础操作 */
250 // 圆 (o, r)
251 struct circle {
252     point o;
253     db r;
254     void scan() {
255         o.scan();
256         scanf("%lf", &r);
257     }
258     int inside(point k) { return cmp(r, o.dis(k)); }
259 };
260 // 两圆位置关系 (两圆公切线数量)

```

```

257 int checkposCC(circle k1, circle k2) {
258     if (cmp(k1.r, k2.r) == -1)
259         swap(k1, k2);
260     db dis = k1.o.dis(k2.o);
261     int w1 = cmp(dis, k1.r + k2.r), w2 = cmp(dis, k1.r -
↪ k2.r);
262     if (w1 > 0)
263         return 4;
264     else if (w1 == 0)
265         return 3;
266     else if (w2 > 0)
267         return 2;
268     else if (w2 == 0)
269         return 1;
270     else
271         return 0;
272 }
273 // 直线与圆交点, 沿 k2->k3 方向给出, 相切给出两个
274 vector<point> getCL(circle k1, point k2, point k3) {
275     point k = proj(k2, k3, k1.o);
276     db d = k1.r * k1.r - (k - k1.o).abs2();
277     if (sgn(d) == -1)
278         return {};
279     point del = (k3 - k2).unit() * sqrt(max((db)0.0, d));
280     return {k - del, k + del};
281 }
282 // 两圆交点, 沿圆 k1 逆时针给出, 相切给出两个
283 vector<point> getCC(circle k1, circle k2) {
284     int pd = checkposCC(k1, k2);
285     if (pd == 0 || pd == 4)
286         return {};
287     db a = (k2.o - k1.o).abs2(), cosA = (k1.r * k1.r + a -
↪ k2.r * k2.r) /
288         (2 * k1.r *
↪ sqrt(max(a, (db)0.0)));
289     db b = k1.r * cosA, c = sqrt(max((db)0.0, k1.r * k1.r -
↪ b * b));
290     point k = (k2.o - k1.o).unit(), m = k1.o + k * b, del =
↪ k.rotleft() * c;
291     return {m - del, m + del};
292 }
293 // 点到圆的切点, 沿圆 k1 逆时针给出, 注意未判位置关系
294 vector<point> TangentCP(circle k1, point k2) {
295     db a = (k2 - k1.o).abs(), b = k1.r * k1.r / a,
296     c = sqrt(max((db)0.0, k1.r * k1.r - b * b));
297     point k = (k2 - k1.o).unit(), m = k1.o + k * b, del =
↪ k.rotleft() * c;
298     return {m - del, m + del};
299 }
300 // 外公切线
301 vector<line> TangentoutCC(circle k1, circle k2) {
302     int pd = checkposCC(k1, k2);
303     if (pd == 0)
304         return {};
305     if (pd == 1) {
306         point k = getCC(k1, k2)[0];
307         return {(line){k, k}};
308     }
309     if (cmp(k1.r, k2.r) == 0) {
310         point del = (k2.o -
↪ k1.o).unit().rotleft().getdel();
311         return {(line){k1.o - del * k1.r, k2.o - del *
↪ k2.r},
312             (line){k1.o + del * k1.r, k2.o + del *
↪ k2.r}};
313     } else {
314         point p = (k2.o * k1.r - k1.o * k2.r) / (k1.r -
↪ k2.r);
315         vector<point> A = TangentCP(k1, p), B =
↪ TangentCP(k2, p);
316         vector<line> ans;
317         for (int i = 0; i < A.size(); i++)
318             ans.push_back((line){A[i], B[i]});
319         return ans;
320     }
321 }
322 // 内公切线
323 vector<line> TangentinCC(circle k1, circle k2) {
324     int pd = checkposCC(k1, k2);
325     if (pd <= 2)
326         return {};
327     if (pd == 3) {
328         point k = getCC(k1, k2)[0];
329         return {(line){k, k}};
330     }
331     point p = (k2.o * k1.r + k1.o * k2.r) / (k1.r + k2.r);
332     vector<point> A = TangentCP(k1, p), B = TangentCP(k2,
↪ p);
333     vector<line> ans;
334     for (int i = 0; i < A.size(); i++)
335         ans.push_back((line){A[i], B[i]});
336     return ans;
337 }
338 // 所有公切线
339 vector<line> TangentCC(circle k1, circle k2) {
340     int flag = 0;
341     if (k1.r < k2.r)
342         swap(k1, k2), flag = 1;
343     vector<line> A = TangentoutCC(k1, k2), B =
↪ TangentinCC(k1, k2);
344     for (line k : B)
345         A.push_back(k);
346     if (flag)
347         for (line &k : A)
348             swap(k[0], k[1]);
349     return A;
350 }
351 // 圆 k1 与三角形 k2 k3 k1.o 的有向面积交
352 db getarea(circle k1, point k2, point k3) {
353     point k = k1.o;
354     k1.o = k1.o - k;
355     k2 = k2 - k;
356     k3 = k3 - k;
357     int pd1 = k1.inside(k2), pd2 = k1.inside(k3);
358     vector<point> A = getCL(k1, k2, k3);
359     if (pd1 >= 0) {
360         if (pd2 >= 0)
361             return cross(k2, k3) / 2;
362         return k1.r * k1.r * rad(A[1], k3) / 2 + cross(k2,
↪ A[1]) / 2;
363     } else if (pd2 >= 0) {
364         return k1.r * k1.r * rad(k2, A[0]) / 2 +
↪ cross(A[0], k3) / 2;
365     } else {
366         int pd = cmp(k1.r, disSP(k2, k3, k1.o));
367         if (pd <= 0)
368             return k1.r * k1.r * rad(k2, k3) / 2;
369         return cross(A[0], A[1]) / 2 +
370             k1.r * k1.r * (rad(k2, A[0]) + rad(A[1],
↪ k3)) / 2;
371     }
372 }
373 // 多边形与圆面积交
374 db getarea(vector<point> A, circle c) {
375     int n = A.size();
376     if (n <= 2)
377         return 0.0;
378     A.push_back(A[0]);
379     db res = 0.0;
380     for (int i = 0; i < n; i++) {
381         point k1 = A[i], k2 = A[i + 1];
382         res += getarea(c, k1, k2);
383     }
384     return fabs(res);

```



```

385 }
386 // 三角形外接圆
387 circle getcircle(point k1, point k2, point k3) {
388     db a1 = k2.x - k1.x, b1 = k2.y - k1.y, c1 = (a1 * a1 +
389 ↪ b1 * b1) / 2;
389     db a2 = k3.x - k1.x, b2 = k3.y - k1.y, c2 = (a2 * a2 +
390 ↪ b2 * b2) / 2;
390     db d = a1 * b2 - a2 * b1;
391     point o =
392     (point){k1.x + (c1 * b2 - c2 * b1) / d, k1.y + (a1
393 ↪ * c2 - a2 * c1) / d};
393     return (circle){o, k1.dis(o)};
394 }
395 // 最小圆覆盖
396 circle getScircle(vector<point> A) {
397     shuffle(A.begin(), A.end(), eng);
398     circle ans = (circle){A[0], 0};
399     for (int i = 1; i < A.size(); i++)
400         if (ans.inside(A[i]) == -1) {
401             ans = (circle){A[i], 0};
402             for (int j = 0; j < i; j++)
403                 if (ans.inside(A[j]) == -1) {
404                     ans.o = (A[i] + A[j]) / 2;
405                     ans.r = ans.o.dis(A[i]);
406                     for (int k = 0; k < j; k++)
407                         if (ans.inside(A[k]) == -1)
408                             ans = getcircle(A[i], A[j],
409 ↪ A[k]);
410                 }
411             return ans;
412         }
413 }
414 /* 多边形 */
415 // 多边形有向面积
416 db area(vector<point> A) {
417     db ans = 0;
418     for (int i = 0; i < A.size(); i++)
419         ans += cross(A[i], A[(i + 1) % A.size()]);
420     return ans / 2;
421 }
422 // 判断是否为逆时针凸包
423 int checkconvex(vector<point> A) {
424     int n = A.size();
425     A.push_back(A[0]);
426     A.push_back(A[1]);
427     for (int i = 0; i < n; i++)
428         if (sgn(cross(A[i + 1] - A[i], A[i + 2] - A[i])) ==
429 ↪ -1)
430             return 0;
431     return 1;
432 }
433 // 点与简单多边形位置关系: 2 内部 1 边界 0 外部
434 int contain(vector<point> A, point q) {
435     int pd = 0;
436     A.push_back(A[0]);
437     for (int i = 1; i < A.size(); i++) {
438         point u = A[i - 1], v = A[i];
439         if (onS(u, v, q))
440             return 1;
441         if (cmp(u.y, v.y) > 0)
442             swap(u, v);
443         if (cmp(u.y, q.y) >= 0 || cmp(v.y, q.y) < 0)
444             continue;
445         if (sgn(cross(u - v, q - v)) < 0)
446             pd ^= 1;
447     }
448     return pd << 1;
449 }
450 // flag=0 不严格 flag=1 严格
451 vector<point> ConvexHull(vector<point> A, int flag = 1) {
452     int n = A.size();
453     if (n == 1)
454         return A;
455     if (n == 2) {
456         if (A[0] == A[1])
457             return {A[0]};
458         else
459             return A;
460     }
461     vector<point> ans(n * 2);
462     sort(A.begin(), A.end());
463     int now = -1;
464     for (int i = 0; i < A.size(); i++) {
465         while (now > 0 &&
466 ↪ sgn(cross(ans[now] - ans[now - 1], A[i] -
467 ↪ ans[now - 1])) < flag)
468             now--;
469         ans[++now] = A[i];
470     }
471     int pre = now;
472     for (int i = n - 2; i >= 0; i--) {
473         while (now > pre &&
474 ↪ sgn(cross(ans[now] - ans[now - 1], A[i] -
475 ↪ ans[now - 1])) < flag)
476             now--;
477         ans[++now] = A[i];
478     }
479     ans.resize(now);
480     return ans;
481 }
482 // 凸包直径
483 db convexDiameter(vector<point> A) {
484     int now = 0, n = A.size();
485     db ans = 0;
486     for (int i = 0; i < A.size(); i++) {
487         now = max(now, i);
488         while (1) {
489             db k1 = A[i].dis(A[now % n]), k2 =
490 ↪ A[i].dis(A[(now + 1) % n]);
491             ans = max(ans, max(k1, k2));
492             if (k2 > k1)
493                 now++;
494             else
495                 break;
496         }
497     }
498     return ans;
499 }
500 // 直线切凸包, 保留 k1,k2,p 逆时针的所有点
501 vector<point> convexcut(vector<point> A, point k1, point
502 ↪ k2) {
503     int n = A.size();
504     A.push_back(A[0]);
505     vector<point> ans;
506     for (int i = 0; i < n; i++) {
507         int w1 = clockwise(k1, k2, A[i]), w2 =
508 ↪ clockwise(k1, k2, A[i + 1]);
509         if (w1 >= 0)
510             ans.push_back(A[i]);
511         if (w1 * w2 < 0)
512             ans.push_back(getLL(k1, k2, A[i], A[i + 1]));
513     }
514     return ans;
515 }
516 // 多边形 A 和 直线 (线段) k1->k2 严格相交, 注释部分为线段
517 int checkPoS(vector<point> A, point k1, point k2) {
518     struct ins {
519         point m, u, v;
520         int operator<(const ins &k) const { return m < k.m;
521 ↪ }
522     };
523     vector<ins> B;

```

```

518 // if (contain(A,k1)==2||contain(A,k2)==2) return 1;
519 vector<point> poly = A;
520 A.push_back(A[0]);
521 for (int i = 1; i < A.size(); i++)
522     if (checkLL(A[i - 1], A[i], k1, k2)) {
523         point m = getLL(A[i - 1], A[i], k1, k2);
524         if (inmid(A[i - 1], A[i], m) /
↪ *&&inmid(k1,k2,m)*/)
525             B.push_back((ins){m, A[i - 1], A[i]});
526     }
527 if (B.size() == 0)
528     return 0;
529 sort(B.begin(), B.end());
530 int now = 1;
531 while (now < B.size() && B[now].m == B[0].m)
532     now++;
533 if (now == B.size())
534     return 0;
535 int flag = contain(poly, (B[0].m + B[now].m) / 2);
536 if (flag == 2)
537     return 1;
538 point d = B[now].m - B[0].m;
539 for (int i = now; i < B.size(); i++) {
540     if (!(B[i].m == B[i - 1].m) && flag == 2)
541         return 1;
542     int tag = sgn(cross(B[i].v - B[i].u, B[i].m + d -
↪ B[i].u));
543     if (B[i].m == B[i].u || B[i].m == B[i].v)
544         flag += tag;
545     else
546         flag += tag * 2;
547 }
548 // return 0;
549 return flag == 2;
550 }
551 int checkin(point r, point l, point m) {
552     if (cmpangle(l, r)) {
553         return cmpangle(l, m) && cmpangle(m, r);
554     }
555     return cmpangle(l, m) || cmpangle(m, r);
556 }
557 // 快速检查线段是否和多边形严格相交
558 int checkPosFast(vector<point> A, point k1, point k2) {
559     if (contain(A, k1) == 2 || contain(A, k2) == 2)
560         return 1;
561     if (k1 == k2)
562         return 0;
563     A.push_back(A[0]);
564     A.push_back(A[1]);
565     for (int i = 1; i + 1 < A.size(); i++)
566         if (checkLL(A[i - 1], A[i], k1, k2)) {
567             point now = getLL(A[i - 1], A[i], k1, k2);
568             if (inmid(A[i - 1], A[i], now) == 0 ||
↪ inmid(k1, k2, now) == 0)
569                 continue;
570             if (now == A[i]) {
571                 if (A[i] == k2)
572                     continue;
573                 point pre = A[i - 1], ne = A[i + 1];
574                 if (checkin(pre - now, ne - now, k2 -
↪ now))
575                     return 1;
576             } else if (now == k1) {
577                 if (k1 == A[i - 1] || k1 == A[i])
578                     continue;
579                 if (checkin(A[i - 1] - k1, A[i] - k1, k2 -
↪ k1))
580                     return 1;
581             } else if (now == k2 || now == A[i - 1])
582                 continue;
583             else
584                 return 1;
585     }
586     return 0;
587 }
588 /* 普通凸包中的二分 */
589 // 求经过点 x 切凸包 A 的两个切点, 返回下标. 方向: A 上
↪ [fi, se] 为点 x
592 // 能看到的区域. 需要保证 x 严格在凸包 A 外侧, A 的点数 >=
↪ 3 需要保证 A
593 // 是严格凸包, 即无三点共线
594 pair<int, int> getTangentCoP(const vector<point> &A, point
↪ x) {
595     int sz = A.size();
596     assert(sz >= 3);
597     int res[2];
598     int flag = 1;
599     if (clockwise(A[sz - 1], A[0], x) == -1)
600         flag = -1;
601     int l = 0, r = sz - 1, ans = 0;
602     while (l < r) {
603         int mid = ((l + r) >> 1);
604         if (clockwise(A[mid], A[mid + 1], x) == flag &&
↪ clockwise(A[0], A[mid + 1], x) == flag)
605             ans = mid + 1, l = mid + 1;
606         else
607             r = mid;
608     }
609     res[0] = ans;
610     l = ans, r = sz - 1, ans = sz - 1;
611     while (l < r) {
612         int mid = ((l + r) >> 1);
613         if (clockwise(A[mid], A[mid + 1], x) == flag)
614             ans = mid, r = mid;
615         else
616             l = mid + 1;
617     }
618     res[1] = ans;
619     if (flag == -1)
620         swap(res[0], res[1]);
621     return {res[0], res[1]};
622 }
623 // 判断点是否在凸多边形 A 内部, flag = 1 严格, 0 不严格
624 bool containCoP(const vector<point> &A, point x, int flag =
↪ 1) {
625     int sz = A.size();
626     assert(sz >= 3);
627     if (!flag && (onS(A[0], A[1], x) || onS(A[sz - 1],
↪ A[0], x)))
628         return 1;
629     if (!(clockwise(A[0], A[1], x) == 1 && clockwise(A[sz -
↪ 1], A[0], x) == 1))
630         return 0;
631     int l = 1, r = sz - 1, ans = 1;
632     while (l < r) {
633         int mid = l + r >> 1;
634         if (clockwise(A[0], A[mid], x) == 1)
635             ans = mid, l = mid + 1;
636         else
637             r = mid;
638     }
639     return clockwise(A[ans], A[ans + 1], x) >= flag;
640 }
641 /* 上下凸包中的二分 */
642 // 拆分凸包成上下凸壳 凸包尽量都随机旋转一个角度来避免出现
↪ 相同横坐标
643 // 尽量特判只有一个点的情况 凸包逆时针
644 void getUDP(vector<point> A, vector<point> &U,
↪ vector<point> &D) {

```

```

649 db l = 1e100, r = -1e100;
650 for (int i = 0; i < A.size(); i++)
651     l = min(l, A[i].x), r = max(r, A[i].x);
652 int wherel, wherer;
653 for (int i = 0; i < A.size(); i++)
654     if (cmp(A[i].x, l) == 0)
655         wherel = i;
656 for (int i = A.size(); i; i--)
657     if (cmp(A[i - 1].x, r) == 0)
658         wherer = i - 1;
659 U.clear();
660 D.clear();
661 int now = wherel;
662 while (1) {
663     D.push_back(A[now]);
664     if (now == wherer)
665         break;
666     now++;
667     if (now >= A.size())
668         now = 0;
669 }
670 now = wherer;
671 while (1) {
672     U.push_back(A[now]);
673     if (now == wherel)
674         break;
675     now--;
676     if (now < 0)
677         now = A.size() - 1;
678 }
679 }
680 // 需要保证凸包点数大于等于 3, 2 内部, 1 边界, 0 外部
681 int containCoP(const vector<point> &U, const vector<point>
    ↪ &D, point k) {
682     db lx = U[0].x, rx = U[U.size() - 1].x;
683     if (k == U[0] || k == U[U.size() - 1])
684         return 1;
685     if (cmp(k.x, lx) == -1 || cmp(k.x, rx) == 1)
686         return 0;
687     int where1 =
688         lower_bound(U.begin(), U.end(), (point){k.x,
    ↪ -1e100}) - U.begin();
689     int where2 =
690         lower_bound(D.begin(), D.end(), (point){k.x,
    ↪ -1e100}) - D.begin();
691     int w1 = clockwise(U[where1 - 1], U[where1], k),
692         w2 = clockwise(D[where2 - 1], D[where2], k);
693     if (w1 == 1 || w2 == -1)
694         return 0;
695     else if (w1 == 0 || w2 == 0)
696         return 1;
697     return 2;
698 }
699 // d 是方向, 输出上方切点和下方切点
700 pair<point, point> getTangentCow(const vector<point> &U,
    ↪ const vector<point> &D,
701                                     point d) {
702     if (sgn(d.x) < 0 || (sgn(d.x) == 0 && sgn(d.y) < 0))
703         d = d * (-1);
704     point whereU, whereD;
705     if (sgn(d.x) == 0)
706         return mp(U[0], U[U.size() - 1]);
707     int l = 0, r = U.size() - 1, ans = 0;
708     while (l < r) {
709         int mid = l + r >> 1;
710         if (sgn(cross(U[mid + 1] - U[mid], d)) <= 0)
711             l = mid + 1, ans = mid + 1;
712         else
713             r = mid;
714     }
715     whereU = U[ans];
716     l = 0, r = D.size() - 1, ans = 0;

```

```

717     while (l < r) {
718         int mid = l + r >> 1;
719         if (sgn(cross(D[mid + 1] - D[mid], d)) >= 0)
720             l = mid + 1, ans = mid + 1;
721         else
722             r = mid;
723     }
724     whereD = D[ans];
725     return mp(whereU, whereD);
726 }
727 // 先检查 contain, 逆时针给出
728 pair<point, point> getTangentCoP(const vector<point> &U,
    ↪ const vector<point> &D,
729                                     point k) {
730     db lx = U[0].x, rx = U[U.size() - 1].x;
731     if (k.x < lx) {
732         int l = 0, r = U.size() - 1, ans = U.size() - 1;
733         while (l < r) {
734             int mid = l + r >> 1;
735             if (clockwise(k, U[mid], U[mid + 1]) == 1)
736                 l = mid + 1;
737             else
738                 ans = mid, r = mid;
739         }
740         point w1 = U[ans];
741         l = 0, r = D.size() - 1, ans = D.size() - 1;
742         while (l < r) {
743             int mid = l + r >> 1;
744             if (clockwise(k, D[mid], D[mid + 1]) == -1)
745                 l = mid + 1;
746             else
747                 ans = mid, r = mid;
748         }
749         point w2 = D[ans];
750         return mp(w1, w2);
751     } else if (k.x > rx) {
752         int l = 1, r = U.size(), ans = 0;
753         while (l < r) {
754             int mid = l + r >> 1;
755             if (clockwise(k, U[mid], U[mid - 1]) == -1)
756                 r = mid;
757             else
758                 ans = mid, l = mid + 1;
759         }
760         point w1 = U[ans];
761         l = 1, r = D.size(), ans = 0;
762         while (l < r) {
763             int mid = l + r >> 1;
764             if (clockwise(k, D[mid], D[mid - 1]) == 1)
765                 r = mid;
766             else
767                 ans = mid, l = mid + 1;
768         }
769         point w2 = D[ans];
770         return mp(w2, w1);
771     } else {
772         int where1 =
773             lower_bound(U.begin(), U.end(), (point){k.x,
    ↪ -1e100}) - U.begin();
774         int where2 =
775             lower_bound(D.begin(), D.end(), (point){k.x,
    ↪ -1e100}) - D.begin();
776         if ((k.x == lx && k.y > U[0].y) ||
777             (where1 && clockwise(U[where1 - 1], U[where1],
    ↪ k) == 1)) {
778             int l = 1, r = where1 + 1, ans = 0;
779             while (l < r) {
780                 int mid = l + r >> 1;
781                 if (clockwise(k, U[mid], U[mid - 1]) == 1)
782                     ans = mid, l = mid + 1;
783                 else
784                     r = mid;

```

```

785     }
786     point w1 = U[ans];
787     l = where1, r = U.size() - 1, ans = U.size() -
↪ 1;
788     while (l < r) {
789         int mid = l + r >> 1;
790         if (clockwise(k, U[mid], U[mid + 1]) == 1)
791             l = mid + 1;
792         else
793             ans = mid, r = mid;
794     }
795     point w2 = U[ans];
796     return mp(w2, w1);
797 } else {
798     int l = 1, r = where2 + 1, ans = 0;
799     while (l < r) {
800         int mid = l + r >> 1;
801         if (clockwise(k, D[mid], D[mid + 1]) == -1)
802             ans = mid, l = mid + 1;
803         else
804             r = mid;
805     }
806     point w1 = D[ans];
807     l = where2, r = D.size() - 1, ans = D.size() -
↪ 1;
808     while (l < r) {
809         int mid = l + r >> 1;
810         if (clockwise(k, D[mid], D[mid + 1]) == -1)
811             l = mid + 1;
812         else
813             ans = mid, r = mid;
814     }
815     point w2 = D[ans];
816     return mp(w1, w2);
817 }
818 }
819 }
820
821 // 三维计算几何
822
823 struct P3 {
824     db x, y, z;
825     P3 operator+(P3 k1) { return (P3){x + k1.x, y + k1.y, z
↪ + k1.z}; }
826     P3 operator-(P3 k1) { return (P3){x - k1.x, y - k1.y, z
↪ - k1.z}; }
827     P3 operator*(db k1) { return (P3){x * k1, y * k1, z *
↪ k1}; }
828     P3 operator/(db k1) { return (P3){x / k1, y / k1, z /
↪ k1}; }
829     db abs2() { return x * x + y * y + z * z; }
830     db abs() { return sqrt(x * x + y * y + z * z); }
831     P3 unit() { return (*this) / abs(); }
832     int operator<(const P3 k1) const {
833         if (cmp(x, k1.x) != 0)
834             return x < k1.x;
835         if (cmp(y, k1.y) != 0)
836             return y < k1.y;
837         return cmp(z, k1.z) == -1;
838     }
839     int operator==(const P3 k1) {
840         return cmp(x, k1.x) == 0 && cmp(y, k1.y) == 0 &&
↪ cmp(z, k1.z) == 0;
841     }
842     void scan() {
843         double k1, k2, k3;
844         scanf("%lf%lf%lf", &k1, &k2, &k3);
845         x = k1;
846         y = k2;
847         z = k3;
848     }
849 };
850 P3 cross(P3 k1, P3 k2) {
851     return (P3){k1.y * k2.z - k1.z * k2.y, k1.z * k2.x -
↪ k1.x * k2.z,
852                 k1.x * k2.y - k1.y * k2.x};
853 }
854 db dot(P3 k1, P3 k2) { return k1.x * k2.x + k1.y * k2.y +
↪ k1.z * k2.z; }
855 // p=(3,4,5),l=(13,19,21),theta=85 ans=(2.83,4.62,1.77)
856 P3 turn3D(db k1, P3 l, P3 p) {
857     l = l.unit();
858     P3 ans;
859     db c = cos(k1), s = sin(k1);
860     ans.x = p.x * (l.x * l.x * (1 - c) + c) +
861             p.y * (l.x * l.y * (1 - c) - l.z * s) +
862             p.z * (l.x * l.z * (1 - c) + l.y * s);
863     ans.y = p.x * (l.x * l.y * (1 - c) + l.z * s) +
864             p.y * (l.y * l.y * (1 - c) + c) +
865             p.z * (l.y * l.z * (1 - c) - l.x * s);
866     ans.z = p.x * (l.x * l.z * (1 - c) - l.y * s) +
867             p.y * (l.y * l.z * (1 - c) + l.x * s) +
868             p.z * (l.x * l.x * (1 - c) + c);
869     return ans;
870 }
871 typedef vector<P3> VP;
872 typedef vector<VP> VVP;
873 db Acos(db x) { return acos(max(-(db)1, min(x, (db)1))); }
874 // 球面距离, 圆心原点, 半径 1
875 db Odist(P3 a, P3 b) {
876     db r = Acos(dot(a, b));
877     return r;
878 }
879 db r;
880 P3 rnd;
881 vector<db> solve(db a, db b, db c) {
882     db r = sqrt(a * a + b * b), th = atan2(b, a);
883     if (cmp(c, -r) == -1)
884         return {0};
885     else if (cmp(r, c) <= 0)
886         return {1};
887     else {
888         db tr = pi - Acos(c / r);
889         return {th + pi - tr, th + pi + tr};
890     }
891 }
892 vector<db> jiao(P3 a, P3 b) {
893     // dot(rd+x*cos(t)+y*sin(t),b) >= cos(r)
894     if (cmp(Odist(a, b), 2 * r) > 0)
895         return {0};
896     P3 rd = a * cos(r), z = a.unit(), y = cross(z,
↪ rnd).unit(),
897         x = cross(y, z).unit();
898     vector<db> ret = solve(-(dot(x, b) * sin(r)), -(dot(y,
↪ b) * sin(r)),
899                         -(cos(r) - dot(rd, b)));
900     return ret;
901 }
902 db norm(db x, db l = 0, db r = 2 * pi) { // change x into
↪ [l,r)
903     while (cmp(x, l) == -1)
904         x += (r - l);
905     while (cmp(x, r) >= 0)
906         x -= (r - l);
907     return x;
908 }
909 db disLP(P3 k1, P3 k2, P3 q) {
910     return (cross(k2 - k1, q - k1)).abs() / (k2 -
↪ k1).abs();
911 }
912 db disLL(P3 k1, P3 k2, P3 k3, P3 k4) {
913     P3 dir = cross(k2 - k1, k4 - k3);
914     if (sgn(dir.abs()) == 0)
915         return disLP(k1, k2, k3);

```

```

916     return fabs(dot(dir.unit(), k1 - k2));
917 }
918 VP getFL(P3 p, P3 dir, P3 k1, P3 k2) {
919     db a = dot(k2 - p, dir), b = dot(k1 - p, dir), d = a -
    ↪ b;
920     if (sgn(fabs(d)) == 0)
921         return {};
922     return {(k1 * a - k2 * b) / d};
923 }
924 VP getFF(P3 p1, P3 dir1, P3 p2, P3 dir2) { // 返回一条线
925     P3 e = cross(dir1, dir2), v = cross(dir1, e);
926     db d = dot(dir2, v);
927     if (sgn(fabs(d)) == 0)
928         return {};
929     P3 q = p1 + v * dot(dir2, p2 - p1) / d;
930     return {q, q + e};
931 }
932 // 3D Convex Hull Template
933 db getV(P3 k1, P3 k2, P3 k3, P3 k4) { // get the Volume
934     return dot(cross(k2 - k1, k3 - k1), k4 - k1);
935 }
936 db rand_db() { return 1.0 * rand() / RAND_MAX; }
937 VP convexHull2D(VP A, P3 dir) {
938     P3 x = {(db)rand(), (db)rand(), (db)rand()};
939     x = x.unit();
940     x = cross(x, dir).unit();
941     P3 y = cross(x, dir).unit();
942     P3 vec = dir.unit() * dot(A[0], dir);
943     vector<point> B;
944     for (int i = 0; i < A.size(); i++)
945         B.push_back((point){dot(A[i], x), dot(A[i], y)});
946     B = ConvexHull(B);
947     A.clear();
948     for (int i = 0; i < B.size(); i++)
949         A.push_back(x * B[i].x + y * B[i].y + vec);
950     return A;
951 }
952 namespace CH3 {
953     VVP ret;
954     set<pair<int, int>> e;
955     int n;
956     VP p, q;
957     void wrap(int a, int b) {
958         if (e.find({a, b}) == e.end()) {
959             int c = -1;
960             for (int i = 0; i < n; i++)
961                 if (i != a && i != b) {
962                     if (c == -1 || sgn(getV(q[c], q[a], q[b],
    ↪ q[i])) > 0)
963                         c = i;
964                 }
965             if (c != -1) {
966                 ret.push_back({p[a], p[b], p[c]});
967                 e.insert({a, b});
968                 e.insert({b, c});
969                 e.insert({c, a});
970                 wrap(c, b);
971                 wrap(a, c);
972             }
973         }
974     }
975     VVP ConvexHull3D(VP _p) {
976         p = q = _p;
977         n = p.size();
978         ret.clear();
979         e.clear();
980         for (auto &i : q)
981             i = i + (P3){rand_db() * 1e-4, rand_db() * 1e-4,
    ↪ rand_db() * 1e-4};
982         for (int i = 1; i < n; i++)
983             if (q[i].x < q[0].x)
984                 swap(p[0], p[i]), swap(q[0], q[i]);
985         for (int i = 2; i < n; i++)
986             if ((q[i].x - q[0].x) * (q[1].y - q[0].y) >
987                 (q[i].y - q[0].y) * (q[1].x - q[0].x))
988                 swap(q[1], q[i]), swap(p[1], p[i]);
989         wrap(0, 1);
990         return ret;
991     }
992 } // namespace CH3
993 VVP reduceCH(VVP A) {
994     VVP ret;
995     map<P3, VP> M;
996     for (VP nowF : A) {
997         P3 dir = cross(nowF[1] - nowF[0], nowF[2] -
    ↪ nowF[0]).unit();
998         for (P3 k1 : nowF)
999             M[dir].pb(k1);
1000     }
1001     for (pair<P3, VP> nowF : M)
1002         ret.pb(convexHull2D(nowF.se, nowF.fi));
1003     return ret;
1004 }
1005 // 把一个面变成 ( 点 , 法向量 ) 的形式
1006 pair<P3, P3> getF(VP F) {
1007     return mp(F[0], cross(F[1] - F[0], F[2] -
    ↪ F[0]).unit());
1008 }
1009 // 3D Cut 保留 dot(dir,x-p)>=0 的部分
1010 VVP ConvexCut3D(VVP A, P3 p, P3 dir) {
1011     VVP ret;
1012     VP sec;
1013     for (VP nowF : A) {
1014         int n = nowF.size();
1015         VP ans;
1016         int dif = 0;
1017         for (int i = 0; i < n; i++) {
1018             int d1 = sgn(dot(dir, nowF[i] - p));
1019             int d2 = sgn(dot(dir, nowF[(i + 1) % n] - p));
1020             if (d1 >= 0)
1021                 ans.pb(nowF[i]);
1022             if (d1 * d2 < 0) {
1023                 P3 q = getFL(p, dir, nowF[i], nowF[(i + 1)
    ↪ % n])[0];
1024                 ans.push_back(q);
1025                 sec.push_back(q);
1026             }
1027             if (d1 == 0)
1028                 sec.push_back(nowF[i]);
1029             else
1030                 dif = 1;
1031             dif |= (sgn(dot(dir, cross(nowF[(i + 1) % n] -
    ↪ nowF[i],
1032                                     nowF[(i + 1) % n] -
    ↪ nowF[i]))) == -1);
1033         }
1034         if (ans.size() > 0 && dif)
1035             ret.push_back(ans);
1036     }
1037     if (sec.size() > 0)
1038         ret.push_back(convexHull2D(sec, dir));
1039     return ret;
1040 }
1041 db vol(VVP A) {
1042     if (A.size() == 0)
1043         return 0;
1044     P3 p = A[0][0];
1045     db ans = 0;
1046     for (VP nowF : A)
1047         for (int i = 2; i < nowF.size(); i++)
1048             ans += abs(getV(p, nowF[0], nowF[i - 1],
    ↪ nowF[i]));
1049     return ans / 6;
1050 }

```

```
1051 WVP init(db INF) {
1052     WVP pss(6, VP(4));
1053     pss[0][0] = pss[1][0] = pss[2][0] = {-INF, -INF, -INF};
1054     pss[0][3] = pss[1][1] = pss[5][2] = {-INF, -INF, INF};
1055     pss[0][1] = pss[2][3] = pss[4][2] = {-INF, INF, -INF};
1056     pss[0][2] = pss[5][3] = pss[4][1] = {-INF, INF, INF};
1057     pss[1][3] = pss[2][1] = pss[3][2] = {INF, -INF, -INF};
1058     pss[1][2] = pss[5][1] = pss[3][3] = {INF, -INF, INF};
1059     pss[2][2] = pss[4][3] = pss[3][1] = {INF, INF, -INF};
1060     pss[5][0] = pss[4][0] = pss[3][0] = {INF, INF, INF};
1061     return pss;
1062 }
```

8 杂项

8.1 java 高精度

```
import java.math.BigInteger;
import java.util.Scanner;

public class QHD {
    private static Scanner sc = new Scanner(System.in);
    public static void solve() {
        BigInteger n = sc.nextBigInteger();
        BigInteger m = sc.nextBigInteger();
        BigInteger one = BigInteger.valueOf(1);
        BigInteger ans = BigInteger.valueOf(0);
        BigInteger a1 = m.shiftLeft(1); // 左移
        BigInteger a2 = m.shiftRight(1); // 右移
        BigInteger a3 = m.multiply(n); // 乘
        BigInteger a4 = m.divide(n); // 除
        BigInteger a5 = m.subtract(n); // 减
        BigInteger a6 = m.add(n); // 加
        BigInteger a7 = m.mod(n); // 模

        String ns = n.toString();
        String s = "114514";
        s.substring(1, 3); // -> 145

        StringBuilder sb = new StringBuilder(ns);
        sb.append('1');
    }
    public static void main(String[] args) {
        int t = sc.nextInt();
        for(int cas = 1; cas <= t; cas++) {
            solve();
        }
    }
}
```

8.2 重载 umap

```
1 struct state {
2     int w[51];
3 } st;
4
5 struct cmp {
6     bool operator()(const state& a, const state& b) const {
7         for(int i = 1; i <= 50; ++i) {
8             if(a.w[i] != b.w[i]) return false;
9         }
10        return true;
11    }
12 };
13
14 struct _hash {
15     size_t operator()(const state& st) const {
16         size_t res = st.w[1];
17         for(int i = 2; i <= 50; ++i) {
18             res *= 19260817;
19             res += st.w[i];
20         }
21         return res;
22     }
23 };
24
25 unordered_map <state, ll, _hash, cmp> mp_st;
26
27 // mp.reserve(1e7);
```

8.3 bitset

```
1 using ull = unsigned long long;
2
3 const int Lg = 64;
4
5 // 寻找 bitset 中任意一个非 0 位
6 template<size_t MAXN>
7 int findAny(bitset<MAXN>& a) {
8     ull *p = (ull *)&a;
9
10    int len = MAXN / Lg;
11    for (int i = 0; i < len; i++) {
12        if (p[i]) {
13            for (int j = 0; j < Lg; j++) {
14                if ((p[i] >> j) & 1ll) {
15                    return (i * Lg + j);
16                }
17            }
18        }
19    }
20
21    assert(false);
22    return -1;
23 }
24
25 // bitset: 将第 i 位设置为 0/1
26 // s.set(i, val);
27
28 // bitset: 清空
29 // s.reset();
30
31 // bitset: 取第 i 位
32 // s.test(i);
33
34 // _Find_first(): 寻找第一位
35 int x = st[i]._Find_first();
36
37 // _Find_next(int i): 寻找下一位
38 int y = st[i]._Find_next(x);
```

8.4 模拟退火

```
1 /**
2  * cur: 起始位置
3  * initT: 初始温度
4  * c: 退火速率
5  */
6 db SA(point origin, db initT, db eps=1e-6, db c=0.9999, int
7     ↪ times=50000000) {
8     point cur = origin;
9
10    db t = initT;
11    db ans = calc(cur), fx = ans;
12    int steps = 0;
13    for (int cnt = 0; cnt < 10; cnt++) {
14        int i = 0;
15        cur = origin;
16        t = initT;
17        for (i = 1; i <= times && fabs(t) > eps; i++) {
18            point nex = cur.move(t);
19            db fy = calc(nex);
20            ans = min(ans, fy);
21            db delta = fy - fx;
22            if (exp(-delta / t) > Rand())
23                cur = nex, fx = fy;
24            t *= c;
25        }
26        steps = i;
27    }
```



```
28     cout << "times = " << steps << endl;
29     return ans;
30 }
```

8.5 对拍

8.5.1 windows 对拍

```
:loop
data.exe > in.txt
J-std.exe < in.txt > J-std.txt
J.exe < in.txt > J-ans.txt
fc /A J-std.txt J-ans.txt
if not errorlevel 1 goto loop
pause
:end
```

8.5.2 linux 对拍

```
#!/bin/bash

while true; do
    ./data > data.in
    ./C < data.in > ans.out
    ./C-baoli < data.in > baoli.out
    if diff ans.out baoli.out; then
        printf AC
    else
        echo WA
        exit 0
    fi
done
```

8.6 快读

```
1 // pku
2
3 char buf[1 << 20], *p1 = buf, *p2 = buf;
4
5 #ifdef ONLINE_JUDGE
6 char get() {
7     if (p1 == p2) {
8         p1 = buf;
9         p2 = buf + std::fread(buf, 1, 1 << 20, stdin);
10    }
11    if (p1 == p2) {
12        return EOF;
13    }
14    return *p1++;
15 }
16 #else
17 char get() { return std::getchar(); }
18 #endif
19
20 ll readLong() {
21     ll x = 0;
22     char c = get();
23     while (!std::isdigit(c)) {
24         c = get();
25     }
```

```
26     while (std::isdigit(c)) {
27         x = x * 10 + c - '0';
28         c = get();
29     }
30     return x;
31 }
32
33 int readInt() {
34     int x = 0;
35     char c = get();
36     while (!std::isdigit(c)) {
37         c = get();
38     }
39     while (std::isdigit(c)) {
40         x = x * 10 + c - '0';
41         c = get();
42     }
43     return x;
44 }
45
46 // pku
47 char gc() {
48     static char buf[1 << 20], *p1 = buf, *p2 = buf;
49     return p1 == p2 &&
50         (p2 = (p1 = buf) + fread(buf, 1, 1 << 20,
51     ↪ stdin), p1 == p2) ? EOF : *p1++;
52 }
53
54 inline ll read() {
55     ll x = 0;
56     char ch = gc();
57     bool positive = 1;
58     for (; !isdigit(ch); ch = gc())
59         if (ch == '-')
60             positive = 0;
61     for (; isdigit(ch); ch = gc())
62         x = x * 10 + ch - '0';
63     return positive ? x : -x;
64 }
```

8.7 随机

```
1 mt19937 eng(time(0));
2 int randInt(int a, int b) {
3     uniform_int_distribution<int> dis(a, b);
4     return dis(eng);
5 }
```

8.8 python

```
1 from functools import cmp_to_key
2
3 lst = [(1, 5), (2, 4), (3, 6), (9, 9), (5, 1)]
4
5 def cmp(a, b):
6     return a[1] * b[0] - a[0] * b[1]
7
8 sortedlst = sorted(lst, key=cmp_to_key(cmp))
9
10 print(lst)
11 print(sortedlst)
```