

Algorithm Library

stlvdv

2021 年 11 月 2 日

目录

1 多项式

1.1 FFT - tourist

```

1  /* copy from tourist */
2  namespace FFT {
3      typedef double dbl;
4
5      struct num {
6          dbl x, y;
7          num() { x = y = 0; }
8          num(dbl x, dbl y) : x(x), y(y) { }
9      };
10
11     inline num operator+(num a, num b) { return num(a.x + b.x, a.y + b.y); }
12     inline num operator-(num a, num b) { return num(a.x - b.x, a.y - b.y); }
13     inline num operator*(num a, num b) { return num(a.x * b.x - a.y * b.y, a
        .x * b.y + a.y * b.x); }
14     inline num conj(num a) { return num(a.x, -a.y); }
15
16     int base = 1;
17     vector<num> roots = { {0, 0}, {1, 0} };
18     vector<int> rev = { 0, 1 };
19
20     const dbl PI = acos(-1.0);
21
22     void ensure_base(int nbase) {
23         if (nbase <= base) {
24             return;
25         }
26         rev.resize(1 << nbase);
27         for (int i = 0; i < (1 << nbase); i++) {
28             rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
29         }
30         roots.resize(1 << nbase);
31         while (base < nbase) {
32             dbl angle = 2 * PI / (1 << (base + 1));
33             for (int i = 1 << (base - 1); i < (1 << base); i++) {
34                 roots[i << 1] = roots[i];
35                 dbl angle_i = angle * (2 * i + 1 - (1 << base));
36                 roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
37             }
38             base++;
39         }

```

```

40     }
41
42     void fft(vector<num>& a, int n = -1) {
43         if (n == -1) {
44             n = a.size();
45         }
46         assert((n & (n - 1)) == 0);
47         int zeros = __builtin_ctz(n);
48         ensure_base(zeros);
49         int shift = base - zeros;
50         for (int i = 0; i < n; i++) {
51             if (i < (rev[i] >> shift)) {
52                 swap(a[i], a[rev[i] >> shift]);
53             }
54         }
55         for (int k = 1; k < n; k <= 1) {
56             for (int i = 0; i < n; i += 2 * k) {
57                 for (int j = 0; j < k; j++) {
58                     num z = a[i + j + k] * roots[j + k];
59                     a[i + j + k] = a[i + j] - z;
60                     a[i + j] = a[i + j] + z;
61                 }
62             }
63         }
64     }
65
66     vector<num> fa, fb;
67
68     vector<long long> multiply(vector<int>& a, vector<int>& b) {
69         int need = a.size() + b.size() - 1;
70         int nbase = 1;
71         while ((1 << nbase) < need) nbase++;
72         ensure_base(nbase);
73         int sz = 1 << nbase;
74         if (sz > (int)fa.size()) {
75             fa.resize(sz);
76         }
77         for (int i = 0; i < sz; i++) {
78             int x = (i < (int)a.size() ? a[i] : 0);
79             int y = (i < (int)b.size() ? b[i] : 0);
80             fa[i] = num(x, y);
81         }
82         fft(fa, sz);

```

```

83     num r(0, -0.25 / (sz >> 1));
84     for (int i = 0; i <= (sz >> 1); i++) {
85         int j = (sz - i) & (sz - 1);
86         num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
87         if (i != j) {
88             fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
89         }
90         fa[i] = z;
91     }
92     for (int i = 0; i < (sz >> 1); i++) {
93         num A0 = (fa[i] + fa[i + (sz >> 1)]) * num(0.5, 0);
94         num A1 = (fa[i] - fa[i + (sz >> 1)]) * num(0.5, 0) * roots[(sz
95             >> 1) + i];
96         fa[i] = A0 + A1 * num(0, 1);
97     }
98     fft(fa, sz >> 1);
99     vector<long long> res(need);
100    for (int i = 0; i < need; i++) {
101        if (i % 2 == 0) {
102            res[i] = fa[i >> 1].x + 0.5;
103        } else {
104            res[i] = fa[i >> 1].y + 0.5;
105        }
106    }
107    return res;
108 }
109
110 vector<long long> square(const vector<int>& a) {
111     int need = a.size() + a.size() - 1;
112     int nbase = 1;
113     while ((1 << nbase) < need) nbase++;
114     ensure_base(nbase);
115     int sz = 1 << nbase;
116     if ((sz >> 1) > (int)a.size()) {
117         fa.resize(sz >> 1);
118     }
119     for (int i = 0; i < (sz >> 1); i++) {
120         int x = (2 * i < (int)a.size() ? a[2 * i] : 0);
121         int y = (2 * i + 1 < (int)a.size() ? a[2 * i + 1] : 0);
122         fa[i] = num(x, y);
123     }
124     fft(fa, sz >> 1);
125     num r(1.0 / (sz >> 1), 0.0);

```

```

125     for (int i = 0; i <= (sz >> 2); i++) {
126         int j = ((sz >> 1) - i) & ((sz >> 1) - 1);
127         num fe = (fa[i] + conj(fa[j])) * num(0.5, 0);
128         num fo = (fa[i] - conj(fa[j])) * num(0, -0.5);
129         num aux = fe * fe + fo * fo * roots[(sz >> 1) + i] * roots[(sz
            >> 1) + i];
130         num tmp = fe * fo;
131         fa[i] = r * (conj(aux) + num(0, 2) * conj(tmp));
132         fa[j] = r * (aux + num(0, 2) * tmp);
133     }
134     fft(fa, sz >> 1);
135     vector<long long> res(need);
136     for (int i = 0; i < need; i++) {
137         if (i % 2 == 0) {
138             res[i] = fa[i >> 1].x + 0.5;
139         } else {
140             res[i] = fa[i >> 1].y + 0.5;
141         }
142     }
143     return res;
144 }
145
146 vector<int> multiply_mod(vector<int>& a, vector<int>& b, int m, int eq =
    0) {
147     int need = a.size() + b.size() - 1;
148     int nbase = 0;
149     while ((1 << nbase) < need) nbase++;
150     ensure_base(nbase);
151     int sz = 1 << nbase;
152     if (sz > (int)fa.size()) {
153         fa.resize(sz);
154     }
155     for (int i = 0; i < (int)a.size(); i++) {
156         int x = (a[i] % m + m) % m;
157         fa[i] = num(x & ((1 << 15) - 1), x >> 15);
158     }
159     fill(fa.begin() + a.size(), fa.begin() + sz, num{ 0, 0 });
160     fft(fa, sz);
161     if (sz > (int)fb.size()) {
162         fb.resize(sz);
163     }
164     if (eq) {
165         copy(fa.begin(), fa.begin() + sz, fb.begin());

```

```

166     } else {
167         for (int i = 0; i < (int)b.size(); i++) {
168             int x = (b[i] % m + m) % m;
169             fb[i] = num(x & ((1 << 15) - 1), x >> 15);
170         }
171         fill(fb.begin() + b.size(), fb.begin() + sz, num{ 0, 0 });
172         fft(fb, sz);
173     }
174     dbl ratio = 0.25 / sz;
175     num r2(0, -1);
176     num r3(ratio, 0);
177     num r4(0, -ratio);
178     num r5(0, 1);
179     for (int i = 0; i <= (sz >> 1); i++) {
180         int j = (sz - i) & (sz - 1);
181         num a1 = (fa[i] + conj(fa[j]));
182         num a2 = (fa[i] - conj(fa[j])) * r2;
183         num b1 = (fb[i] + conj(fb[j])) * r3;
184         num b2 = (fb[i] - conj(fb[j])) * r4;
185         if (i != j) {
186             num c1 = (fa[j] + conj(fa[i]));
187             num c2 = (fa[j] - conj(fa[i])) * r2;
188             num d1 = (fb[j] + conj(fb[i])) * r3;
189             num d2 = (fb[j] - conj(fb[i])) * r4;
190             fa[i] = c1 * d1 + c2 * d2 * r5;
191             fb[i] = c1 * d2 + c2 * d1;
192         }
193         fa[j] = a1 * b1 + a2 * b2 * r5;
194         fb[j] = a1 * b2 + a2 * b1;
195     }
196     fft(fa, sz);
197     fft(fb, sz);
198     vector<int> res(need);
199     for (int i = 0; i < need; i++) {
200         long long aa = fa[i].x + 0.5;
201         long long bb = fb[i].x + 0.5;
202         long long cc = fa[i].y + 0.5;
203         res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
204     }
205     return res;
206 }
207
208 vector<int> square_mod(vector<int>& a, int m) {

```

```

209         return multiply_mod(a, a, m, 1);
210     }
211 };

```

1.2 形式幂级数

```

1  #define db double
2  #ifndef ONLINE_JUDGE // 这三个函数是给MSVC用的，G++不需要
3  inline int __builtin_clz(int v) { // 返回前导0的个数
4      return __lzcnt(v);
5  }
6  inline int __builtin_ctz(int v) { // 返回末尾0的个数
7      if (v == 0) {
8          return 0;
9      }
10     __asm {
11         bsf eax, dword ptr[v];
12     }
13 }
14 inline int __builtin_popcount(int v) { // 返回二进制中1的个数
15     return __popcnt(v);
16 }
17 #endif
18 struct Complex {
19     db real, imag;
20     Complex(db x = 0, db y = 0) : real(x), imag(y) {}
21     Complex& operator+=(const Complex& rhs) {
22         real += rhs.real; imag += rhs.imag;
23         return *this;
24     }
25     Complex& operator-=(const Complex& rhs) {
26         real -= rhs.real; imag -= rhs.imag;
27         return *this;
28     }
29     Complex& operator*=(const Complex& rhs) {
30         db t_real = real * rhs.real - imag * rhs.imag;
31         imag = real * rhs.imag + imag * rhs.real;
32         real = t_real;
33         return *this;
34     }
35     Complex& operator/=(double x) {
36         real /= x, imag /= x;
37         return *this;

```



```

38     }
39     friend Complex operator + (const Complex& a, const Complex& b) { return
        Complex(a) += b; }
40     friend Complex operator - (const Complex& a, const Complex& b) { return
        Complex(a) -= b; }
41     friend Complex operator * (const Complex& a, const Complex& b) { return
        Complex(a) *= b; }
42     friend Complex operator / (const Complex& a, const db& b) { return
        Complex(a) /= b; }
43     Complex power(long long p) const {
44         assert(p >= 0);
45         Complex a = *this, res = { 1, 0 };
46         while (p > 0) {
47             if (p & 1) res = res * a;
48             a = a * a;
49             p >>= 1;
50         }
51         return res;
52     }
53     static long long val(double x) { return x < 0 ? x - 0.5 : x + 0.5; }
54     inline long long Real() const { return val(real); }
55     inline long long Imag() const { return val(imag); }
56     Complex conj() const { return Complex(real, -imag); }
57     explicit operator int() const { return Real(); }
58     friend ostream& operator<<(ostream& stream, const Complex& m) {
59         return stream << complex<db>(m.real, m.imag);
60     }
61 };
62 constexpr int MOD = 998244353;
63 constexpr int Phi_MOD = 998244352;
64 inline int exgcd(int a, int md = MOD) {
65     a %= md;
66     if (a < 0) a += md;
67     int b = md, u = 0, v = 1;
68     while (a) {
69         int t = b / a;
70         b -= t * a; swap(a, b);
71         u -= t * v; swap(u, v);
72     }
73     assert(b == 1);
74     if (u < 0) u += md;
75     return u;
76 }

```

```

77 inline int add(int a, int b) { return a + b >= MOD ? a + b - MOD : a + b; }
78 inline int sub(int a, int b) { return a - b < 0 ? a - b + MOD : a - b; }
79 inline int mul(int a, int b) { return 1LL * a * b % MOD; }
80 inline int powmod(int a, long long b) {
81     int res = 1;
82     while (b > 0) {
83         if (b & 1) res = mul(res, a);
84         a = mul(a, a);
85         b >>= 1;
86     }
87     return res;
88 }
89
90 vector<int> inv, fac, ifac;
91 void prepare_factorials(int maximum) {
92     inv.assign(maximum + 1, 1);
93     // Make sure MOD is prime, which is necessary for the inverse algorithm
94     // below.
95     for (int p = 2; p * p <= MOD; p++)
96         assert(MOD % p != 0);
97     for (int i = 2; i <= maximum; i++)
98         inv[i] = mul(inv[MOD % i], (MOD - MOD / i));
99
100     fac.resize(maximum + 1);
101     ifac.resize(maximum + 1);
102     fac[0] = ifac[0] = 1;
103
104     for (int i = 1; i <= maximum; i++) {
105         fac[i] = mul(i, fac[i - 1]);
106         ifac[i] = mul(inv[i], ifac[i - 1]);
107     }
108 }
109 namespace FFT {
110     vector<Complex> roots = { Complex(0, 0), Complex(1, 0) };
111     vector<int> bit_reverse;
112     int max_size = 1 << 20;
113     const long double pi = acosl(-1.0l);
114     constexpr int FFT_CUTOFF = 150;
115     inline bool is_power_of_two(int n) { return (n & (n - 1)) == 0; }
116     inline int round_up_power_two(int n) {
117         assert(n > 0);
118         while (n & (n - 1)) {
119             n = (n | (n - 1)) + 1;

```

```

119     }
120     return n;
121 }
122 // Given n (a power of two), finds k such that  $n = 1 \ll k$ .
123 inline int get_length(int n) {
124     assert(is_power_of_two(n));
125     return __builtin_ctz(n);
126 }
127 // Rearranges the indices to be sorted by lowest bit first, then second
128 // lowest, etc., rather than highest bit first.
129 // This makes even-odd div-conquer much easier.
130 void bit_reorder(int n, vector<Complex>& values) {
131     if ((int)bit_reverse.size() != n) {
132         bit_reverse.assign(n, 0);
133         int length = get_length(n);
134         for (int i = 0; i < n; i++) {
135             bit_reverse[i] = (bit_reverse[i >> 1] >> 1) + ((i & 1) << (
136                 length - 1));
137         }
138     }
139     for (int i = 0; i < n; i++) {
140         if (i < bit_reverse[i]) {
141             swap(values[i], values[bit_reverse[i]]);
142         }
143     }
144 }
145 void prepare_roots(int n) {
146     assert(n <= max_size);
147     if ((int)roots.size() >= n)
148         return;
149     int length = get_length(roots.size());
150     roots.resize(n);
151     // The roots array is set up such that for a given power of two  $n \geq$ 
152     // 2, roots[n / 2] through roots[n - 1] are
153     // the first half of the n-th primitive roots of MOD.
154     while (1 << length < n) {
155         for (int i = 1 << (length - 1); i < 1 << length; i++) {
156             roots[2 * i] = roots[i];
157             long double angle = pi * (2 * i + 1) / (1 << length);
158             roots[2 * i + 1] = Complex(-cos(angle), -sin(angle));
159         }
160         length++;
161     }

```

```

159     }
160     void fft_iterative(int N, vector<Complex>& values) {
161         assert(is_power_of_two(N));
162         prepare_roots(N);
163         bit_reorder(N, values);
164         for (int n = 1; n < N; n *= 2) {
165             for (int start = 0; start < N; start += 2 * n) {
166                 for (int i = 0; i < n; i++) {
167                     Complex& even = values[start + i];
168                     Complex odd = values[start + n + i] * roots[n + i];
169                     values[start + n + i] = even - odd;
170                     values[start + i] = even + odd;
171                 }
172             }
173         }
174     }
175     vector<long long> multiply(vector<int> a, vector<int> b) { // 普通FFT
176         int n = a.size();
177         int m = b.size();
178         if (min(n, m) < FFT_CUTOFF) {
179             vector<long long> res(n + m - 1);
180             for (int i = 0; i < n; i++) {
181                 for (int j = 0; j < m; j++) {
182                     res[i + j] += 1LL * a[i] * b[j];
183                 }
184             }
185             return res;
186         }
187         int N = round_up_power_two(n + m - 1);
188         vector<Complex> tmp(N);
189         for (int i = 0; i < a.size(); i++) tmp[i].real = a[i];
190         for (int i = 0; i < b.size(); i++) tmp[i].imag = b[i];
191         fft_iterative(N, tmp);
192         for (int i = 0; i < N; i++) tmp[i] = tmp[i] * tmp[i];
193         reverse(tmp.begin() + 1, tmp.end());
194         fft_iterative(N, tmp);
195         vector<long long> res(n + m - 1);
196         for (int i = 0; i < res.size(); i++) {
197             res[i] = tmp[i].imag / 2 / N + 0.5;
198         }
199         return res;
200     }
201     vector<int> mod_multiply(vector<int> a, vector<int> b, int lim =

```

```

max_size) { // 任意模数FFT
202     int n = a.size();
203     int m = b.size();
204     if (min(n, m) < FFT_CUTOFF) {
205         vector<int> res(n + m - 1);
206         for (int i = 0; i < n; i++) {
207             for (int j = 0; j < m; j++) {
208                 res[i + j] += 1LL * a[i] * b[j] % MOD;
209                 res[i + j] %= MOD;
210             }
211         }
212         return res;
213     }
214     int N = round_up_power_two(n + m - 1);
215     N = min(N, lim);
216     vector<Complex> P(N);
217     vector<Complex> Q(N);
218     for (int i = 0; i < n; i++) {
219         P[i] = Complex(a[i] >> 15, a[i] & 0x7fff);
220     }
221     for (int i = 0; i < m; i++) {
222         Q[i] = Complex(b[i] >> 15, b[i] & 0x7fff);
223     }
224     fft_iterative(N, P);
225     fft_iterative(N, Q);
226     vector<Complex> A(N), B(N), C(N), D(N);
227     for (int i = 0; i < N; i++) {
228         Complex P2 = P[(N - i) & (N - 1)].conj();
229         A[i] = (P2 + P[i]) * Complex(0.5, 0),
230         B[i] = (P2 - P[i]) * Complex(0, 0.5);
231         Complex Q2 = Q[(N - i) & (N - 1)].conj();
232         C[i] = (Q2 + Q[i]) * Complex(0.5, 0),
233         D[i] = (Q2 - Q[i]) * Complex(0, 0.5);
234     }
235     for (int i = 0; i < N; i++) {
236         P[i] = (A[i] * C[i]) + (B[i] * D[i]) * Complex(0, 1),
237         Q[i] = (A[i] * D[i]) + (B[i] * C[i]) * Complex(0, 1);
238     }
239     reverse(P.begin() + 1, P.end());
240     reverse(Q.begin() + 1, Q.end());
241     fft_iterative(N, P);
242     fft_iterative(N, Q);
243     for (int i = 0; i < N; i++) {

```

```

244         P[i] /= N, Q[i] /= N;
245     }
246     int size = min(n + m - 1, lim);
247     vector<int> res(size);
248     for (int i = 0; i < size; i++) {
249         long long ac = P[i].Real() % MOD, bd = P[i].Imag() % MOD,
250             ad = Q[i].Real() % MOD, bc = Q[i].Imag() % MOD;
251         res[i] = ((ac << 30) + bd + ((ad + bc) << 15)) % MOD;
252     }
253     return res.resize(n + m - 1), res;
254 }
255 vector<int> mod_inv(vector<int> a) { // 多项式逆
256     int n = a.size();
257     int N = round_up_power_two(a.size());
258     a.resize(N * 2);
259     vector<int> res(1);
260     res[0] = exgcd(a[0]);
261     for (int i = 2; i <= N; i <= 1) {
262         vector<int> tmp(a.begin(), a.begin() + i);
263         int n = (i < 1);
264         tmp = mod_multiply(tmp, mod_multiply(res, res, n), n);
265         res.resize(i);
266         for (int j = 0; j < i; j++) {
267             res[j] = add(res[j], sub(res[j], tmp[j]));
268         }
269     }
270     res.resize(n);
271     return res;
272 }
273 vector<int> integral(vector<int> a) { // 多项式积分
274     assert(a.size() <= inv.size());
275     a.push_back(0);
276     for (int i = (int)a.size() - 1; i >= 1; i--) {
277         a[i] = mul(a[i - 1], inv[i]);
278     }
279     return a;
280 }
281 vector<int> differential(vector<int> a) { // 多项式求导
282     for (int i = 0; i < (int)a.size() - 1; i++) {
283         a[i] = mul(i + 1, a[i + 1]);
284     }
285     a.pop_back();
286     return a;

```

```

287     }
288     vector<int> ln(vector<int> a) { // 多项式对数函数
289         assert((int)a[0] == 1);
290         auto b = mod_multiply(differential(a), mod_inv(a));
291         b = integral(b);
292         b[0] = 0;
293         return b;
294     }
295     vector<int> exp(vector<int> a) { // 多项式指数函数
296         int N = round_up_power_two(a.size());
297         int n = a.size();
298         a.resize(N * 2);
299         vector<int> res{ 1 };
300         for (int i = 2; i <= N; i <= 1) {
301             auto tmp = res;
302             tmp.resize(i);
303             tmp = ln(tmp);
304             for (int j = 0; j < i; j++) {
305                 tmp[j] = sub(a[j], tmp[j]);
306             }
307             tmp[0] = add(tmp[0], 1);
308             res.resize(i);
309             res = mod_multiply(res, tmp, i < 1);
310             fill(res.begin() + i, res.end(), 0);
311         }
312         res.resize(n);
313         return res;
314     }
315     // Multiplies many polynomials whose total degree is n in  $O(n \log^2 n)$ .
316     vector<int> mod_multiply_all(const vector<vector<int>>& polynomials) {
317         if (polynomials.empty())
318             return { 1 };
319         struct compare_size {
320             bool operator()(const vector<int>& x, const vector<int>& y) {
321                 return x.size() > y.size();
322             }
323         };
324         priority_queue<vector<int>, vector<vector<int>>, compare_size> pq(
            polynomials.begin(), polynomials.end());
325         while (pq.size() > 1) {
326             vector<int> a = pq.top(); pq.pop();
327             vector<int> b = pq.top(); pq.pop();
328             pq.push(mod_multiply(a, b));

```

```

329     }
330     return pq.top();
331 }
332 tuple<int, int, bool> power_reduction(string s, int n) { // 多项式快速幂
    预处理
333     int p = 0, q = 0; bool zero = false;
334     for (int i = 0; i < s.length(); i++) {
335         p = mul(p, 10);
336         p = add(p, s[i] - '0');
337         q = 1LL * q * 10 % Phi_MOD; // Phi_MOD 是MOD的欧拉函数值
338         q = (q + s[i] - '0');
339         if (q >= Phi_MOD) q -= Phi_MOD;
340         if (q >= (int)n) zero = true;
341     }
342     return { p, q, zero };
343 }
344 vector<int> power(vector<int> a, string s) { // 多项式快速幂 a^s O(nlogn)
    345     int n = a.size();
    346     auto [p, q, zero] = power_reduction(s, (int)a.size()); // 不需要降幂
    的话可以省去这部分
    347     if (a[0] == 1) {
    348         auto res = ln(a);
    349         while ((int)res.size() > n) res.pop_back();
    350         for (auto& i : res) {
    351             i = mul(p, i);
    352         }
    353         res = exp(res);
    354         return res;
    355     } else {
    356         int mn = -1;
    357         vector<int> copy_a;
    358         for (int i = 0; i < (int)a.size(); i++) {
    359             if (a[i]) {
    360                 mn = i;
    361                 break;
    362             }
    363         }
    364         if ((mn == -1) || (mn && (zero || (1LL * mn * p > n)))) { // a中
    所有元素都是0 或 偏移过大
    365             return vector<int>(n, 0);
    366         }
    367         int inverse_amin = exgcd(a[mn]);

```



```

368         for (int i = mn; i < n; i++) {
369             copy_a.emplace_back(mul(a[i], inverse_amin));
370         }
371         copy_a = ln(copy_a);
372         while ((int)copy_a.size() > n) copy_a.pop_back();
373         for (auto& i : copy_a) {
374             i = mul(i, p);
375         }
376         copy_a = exp(copy_a);
377         vector<int> res(n, 0);
378         // shift是偏移量 power_k 是a_min^q(q是扩展欧拉定理降出来的幂次)
379         int shift = mn * p, power_k = powmod(a[mn], q);
380         for (int i = 0; i + shift < n; i++) {
381             res[i + shift] = mul(copy_a[i], power_k);
382         }
383         return res;
384     }
385 }
386 vector<long long> sub_convolution(vector<int> a, vector<int> b) { // 减
    法卷积 只保留非负次项
387     int n = b.size();
388     reverse(b.begin(), b.end());
389     auto res = multiply(a, b);
390     return vector<long long>(res.begin() + n - 1, res.end());
391 }
392 int bostan_mori(vector<int> p, vector<int> q, long long n) { // [x^n]p(x)
    )/q(x) O(2/3 dlog(d) log(n+1)) d是多项式度数
393     int i;
394     for (; n; n >>= 1) {
395         auto r = q;
396         for (i = 1; i < r.size(); i += 2) {
397             r[i] = MOD - r[i];
398         }
399         p = mod_multiply(p, r);
400         q = mod_multiply(q, r);
401         for (i = (n & 1); i < p.size(); i += 2) {
402             p[i / 2] = p[i];
403         }
404         p.resize(i / 2);
405         for (i = 0; i < q.size(); i += 2) {
406             q[i / 2] = q[i];
407         }
408         q.resize(i / 2);

```

```

409     }
410     return p[0];
411 }
412 };

```

2 数论

2.1 简单的防爆模板

2.1.1 类型 1

```

1 namespace SimpleMod {
2     constexpr int MOD = (int)1e9 + 7;
3     inline int norm(long long a) { return (a % MOD + MOD) % MOD; }
4     inline int add(int a, int b) { return a + b >= MOD ? a + b - MOD : a + b
5         ; }
6     inline int sub(int a, int b) { return a - b < 0 ? a - b + MOD : a - b; }
7     inline int mul(int a, int b) { return (int)((long long)a * b % MOD); }
8     inline int powmod(int a, long long b) {
9         int res = 1;
10        while (b > 0) {
11            if (b & 1) res = mul(res, a);
12            a = mul(a, a);
13            b >>= 1;
14        }
15        return res;
16    }
17    inline int inv(int a) {
18        a %= MOD;
19        if (a < 0) a += MOD;
20        int b = MOD, u = 0, v = 1;
21        while (a) {
22            int t = b / a;
23            b -= t * a; swap(a, b);
24            u -= t * v; swap(u, v);
25        }
26        assert(b == 1);
27        if (u < 0) u += MOD;
28        return u;
29    }
30 }

```

2.1.2 类型 2

```

1  // copy from jiangly
2  constexpr int P = 1e9 + 7;
3  // assume  $-P \leq x < 2P$ 
4  int norm(int x) {
5      if (x < 0) {
6          x += P;
7      }
8      if (x >= P) {
9          x -= P;
10     }
11     return x;
12 }
13 template<class T>
14 T power(T a, int b) {
15     T res = 1;
16     for (; b; b /= 2, a *= a) {
17         if (b % 2) {
18             res *= a;
19         }
20     }
21     return res;
22 }
23 struct Z {
24     int x;
25     Z(int x = 0) : x(norm(x)) {}
26     int val() const {
27         return x;
28     }
29     Z operator-() const {
30         return Z(norm(P - x));
31     }
32     Z inv() const {
33         assert(x != 0);
34         return power(*this, P - 2);
35     }
36     Z& operator*=(const Z& rhs) {
37         x = 1LL * x * rhs.x % P;
38         return *this;
39     }
40     Z& operator+=(const Z& rhs) {
41         x = norm(x + rhs.x);
42         return *this;
43     }

```

```

44     Z& operator--(const Z& rhs) {
45         x = norm(x - rhs.x);
46         return *this;
47     }
48     Z& operator/=(const Z& rhs) {
49         return *this *= rhs.inv();
50     }
51     friend Z operator*(const Z& lhs, const Z& rhs) {
52         Z res = lhs;
53         res *= rhs;
54         return res;
55     }
56     friend Z operator+(const Z& lhs, const Z& rhs) {
57         Z res = lhs;
58         res += rhs;
59         return res;
60     }
61     friend Z operator-(const Z& lhs, const Z& rhs) {
62         Z res = lhs;
63         res -= rhs;
64         return res;
65     }
66     friend Z operator/(const Z& lhs, const Z& rhs) {
67         Z res = lhs;
68         res /= rhs;
69         return res;
70     }
71 };

```

2.2 筛法

2.2.1 线性素数筛

```

1  vector<bool> isPrime; // true 表示非素数 false 表示是素数
2  vector<int> prime; // 保存素数
3  int sieve(int n) {
4      isPrime.resize(n + 1, false);
5      isPrime[0] = isPrime[1] = true;
6      for (int i = 2; i <= n; i++) {
7          if (!isPrime[i]) prime.emplace_back(i);
8          for (int j = 0; j < (int)prime.size() && prime[j] * i <= n; j++) {
9              isPrime[prime[j] * i] = true;
10             if (!(i % prime[j])) break;
11         }

```

```

12     }
13     return (int)prime.size();
14 }

```

2.2.2 线性欧拉函数筛

```

1 bool is_prime[SIZE];
2 int prime[SIZE], phi[SIZE]; // phi[i] 表示 i 的欧拉函数值
3 int Phi(int n) { // 线性筛素数的同时线性求欧拉函数
4     phi[1] = 1; is_prime[1] = true;
5     int p = 0;
6     for (int i = 2; i <= n; i++) {
7         if (!is_prime[i]) prime[p++] = i, phi[i] = i - 1;
8         for (int j = 0; j < p && prime[j] * i <= n; j++) {
9             is_prime[prime[j] * i] = true;
10            if (!(i % prime[j])) {
11                phi[i * prime[j]] = phi[i] * prime[j];
12                break;
13            }
14            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
15        }
16    }
17    return p;
18 }

```

2.2.3 线性约数个数函数筛

```

1 bool is_prime[SIZE];
2 int prime[SIZE], d[SIZE], num[SIZE]; // d[i] 表示 i 的因子数 num[i] 表示 i
   // 的最小质因子出现次数
3 int getFactors(int n) { // 线性筛因子数
4     d[1] = 1; is_prime[1] = true;
5     int p = 0;
6     for (int i = 2; i <= n; i++) {
7         if (!is_prime[i]) prime[p++] = i, d[i] = 2, num[i] = 1;
8         for (int j = 0; j < p && prime[j] * i <= n; j++) {
9             is_prime[prime[j] * i] = true;
10            if (!(i % prime[j])) {
11                num[i * prime[j]] = num[i] + 1;
12                d[i * prime[j]] = d[i] / num[i * prime[j]] * (num[i * prime[
13                j]] + 1);
14                break;
15            }
16        }
17    }
18 }

```

```

15         num[i * prime[j]] = 1;
16         d[i * prime[j]] = d[i] + d[i];
17     }
18 }
19 return p;
20 }

```

2.2.4 线性素因子个数函数筛

```

1 bool is_prime[SIZE];
2 int prime[SIZE], num[SIZE]; // num[i] 表示 i 的质因子数
3 int getPrimeFactors(int n) { // 线性筛质因子数
4     is_prime[1] = true;
5     int p = 0;
6     for (int i = 2; i <= n; i++) {
7         if (!is_prime[i]) prime[p++] = i, num[i] = 1;
8         for (int j = 0; j < p && prime[j] * i <= n; j++) {
9             is_prime[prime[j] * i] = true;
10            if (!(i % prime[j])) {
11                num[i * prime[j]] = num[i];
12                break;
13            }
14            num[i * prime[j]] = num[i] + 1;
15        }
16    }
17    return p;
18 }

```

2.2.5 线性约数和函数筛

```

1 bool is_prime[SIZE];
2 int prime[SIZE], f[SIZE], g[SIZE]; // f[i] 表示 i 的约数和
3 int getSigma(int n) {
4     g[1] = f[1] = 1; is_prime[1] = true;
5     int p = 0;
6     for (int i = 2; i <= n; i++) {
7         if (!is_prime[i]) prime[p++] = i, f[i] = g[i] = i + 1;
8         for (int j = 0; j < p && prime[j] * i <= n; j++) {
9             is_prime[prime[j] * i] = true;
10            if (!(i % prime[j])) {
11                g[i * prime[j]] = g[i] * prime[j] + 1;
12                f[i * prime[j]] = f[i] / g[i] * g[i * prime[j]];
13                break;

```

```

14         }
15         f[i * prime[j]] = f[i] * f[prime[j]];
16         g[i * prime[j]] = 1 + prime[j];
17     }
18 }
19 return p;
20 }

```

2.2.6 线性莫比乌斯函数筛

```

1 bool is_prime[SIZE];
2 int prime[SIZE], mu[SIZE]; // mu[i] 表示 i 的莫比乌斯函数值
3 int getMu(int n) { // 线性筛莫比乌斯函数
4     mu[1] = 1; is_prime[1] = true;
5     int p = 0;
6     for (int i = 2; i <= n; i++) {
7         if (!is_prime[i]) prime[p++] = i, mu[i] = -1;
8         for (int j = 0; j < p && prime[j] * i <= n; j++) {
9             is_prime[prime[j] * i] = true;
10            if (!(i % prime[j])) {
11                mu[i * prime[j]] = 0;
12                break;
13            }
14            mu[i * prime[j]] = -mu[i];
15        }
16    }
17    return p;
18 }

```

2.3 Pollard-Rho

```

1 namespace Pollard_Rho {
2     typedef long long ll;
3     vector<ll> ans; // 存储质因子的数组
4     inline ll gcd(ll a, ll b) { ll c; while (b) c = a % b, a = b, b = c;
5         return a; }
6     inline ll mulmod(ll x, ll y, const ll z) {
7         return (x * y - (ll)((long double)x * y + 0.5) / (long double)z) *
8             z + z) % z;
9     }
10    inline ll powmod(ll a, ll b, const ll mo) {
11        ll s = 1;

```

```

10     for (; b; b >>= 1, a = mulmod(a, a, mo)) if (b & 1) s = mulmod(s, a,
11         mo);
12     return s;
13 }
14 bool isPrime(ll p) { // Miller-Rabin  $O(k \log^3(n))$  k为素性测试轮数
15     const int lena = 10, a[lena] = { 2,3,5,7,11,13,17,19,23,29 };
16     if (p == 2) return true;
17     if (p == 1 || !(p & 1) || (p == 4685624825598111)) return false;
18     ll D = p - 1;
19     while (!(D & 1)) D >>= 1;
20     for (int i = 0; i < lena && a[i] < p; i++) {
21         ll d = D, t = powmod(a[i], d, p);
22         if (t == 1) continue;
23         for (; d != p - 1 && t != p - 1; d <<= 1) t = mulmod(t, t, p);
24         if (d == p - 1) return false;
25     }
26     return true;
27 }
28 void reportFactor(ll n) { // 得到一个素因子
29     ans.emplace_back(n); // 存储素因子
30 }
31 ll ran() { return rand(); } // 随机数
32 void getFactor(ll n) { // Pollard-Rho  $O(n^{1/4})$ 
33     if (n == 1) return;
34     if (isPrime(n)) { reportFactor(n); return; }
35     while (true) {
36         ll c = ran() % n, i = 1, x = ran() % n, y = x, k = 2;
37         do {
38             ll d = gcd(n + y - x, n);
39             if (d != 1 && d != n) { getFactor(d); getFactor(n / d);
40                 return; }
41             if (++i == k) y = x, k <<= 1;
42             x = (mulmod(x, x, n) + c) % n;
43         } while (y != x);
44     }
45 }
46 using namespace Pollard_Rho;

```

2.4 扩展欧几里得

2.4.1 线性同余方程最小非负整数解

exgcd 求 $ax + by = c$ 的最小非负整数解详解:

1. 求出 a, b 的最大公约数 $g = \gcd(a, b)$ ，根据裴蜀定理检查是否满足 $c \% g = 0$ ，不满足则无解；
2. 调整系数 a, b, c 为 $a' = \frac{a}{g}, b' = \frac{b}{g}, c' = \frac{c}{g}$ ，这是因为 $ax + by = c$ 和 $a'x + b'y = c'$ 是完全等价的；
3. 实际上 `exgcd` 求解的方程是 $a'x + b'y = 1$ ，求解前需要注意让系数 $a', b' \geq 0$ （举个例子，如果系数 b' 原本 < 0 ，我们可以翻转 b' 的符号然后令解 (x, y) 为 $(x, -y)$ ，但是求解的时候要把 y 翻回来）；
4. 我们通过 `exgcd` 求出一组解 (x_0, y_0) ，这组解满足 $a'x_0 + b'y_0 = 1$ ，为了使解合法我们需要令 $x_0 = c'x_0, y_0 = c'y_0$ ，于是有 $a'(c'x_0) + b'(c'y_0) = c'$ ；
5. 考虑到 $a'x_0 + b'y_0 = 1$ 等价于同余方程 $a'x_0 \equiv 1 \pmod{b'}$ ，因此为了求出最小非负整数解，我们最后还需要对 b' 取模；
6. 最后注意特判 $c' = 0$ 的情况，如果要求解 y 且系数 b 发生了翻转，将其翻转回来。

```

1 long long exgcd(long long a, long long b, long long& x, long long& y) {
2     if (!b) {
3         x = 1, y = 0;
4         return a;
5     }
6     long long g = exgcd(b, a % b, y, x);
7     y -= (a / b) * x;
8     return g;
9 }
10
11 ll x, y; // 最小非负整数解
12 bool solve(ll a, ll b, ll c) { // ax+by=c
13     ll g = gcd(a, b);
14     if (c % g) return false;
15     a /= g, b /= g, c /= g;
16     bool flag = false;
17     if (b < 0) b = -b, flag = true;
18     exgcd(a, b, x, y);
19     x = (x * c % b + b) % b;
20     if (flag) b = -b;
21     y = (c - a * x) / b;
22     if (!c) x = y = 0; // ax+by=0
23     return true;
24 }

```

2.4.2 一定范围内线性方程整数解数

`exgcd` 通解：假设我们通过上方的 `exgcd` 流程获得了一组解 (x_0, y_0) （没有乘 c ），那么 $a'x + b'y = 1$ 的通解就是 $(x_0 + b't, y_0 - a't)$ ，因此 $a'x + b'y = c'$ 的通解是 $(c'(x_0 + b't), c'(y_0 - a't))$ 。

```

1 /*
2 * 求解 ax+by+c=0 模板CF710D

```

```

3  * 返回 [xl, xr] x [yl, yr] 的解数
4  * 若至少有一组解 则x是一个合法解
5  * 可以根据x推出y 但要注意b=0/a=0等特殊情况
6  * 特别注意!!! 设置边界时的取整问题!!!
7  * x/y 向上取整时(x+y-1)/y 向下取整时floor(1.0*x/y)
8  * */
9  ll a, b, c, xl, xr, yl, yr;
10 ll x, y, d;
11 ll exgcd(ll a, ll b, ll& x, ll& y) {
12     if (!b) return x = 1, y = 0, a;
13     ll d = exgcd(b, a % b, x, y), t = x;
14     return x = y, y = t - a / b * y, d;
15 }
16 ll solve(ll a, ll b, ll c, ll xl, ll xr, ll yl, ll yr) {
17     if (xl > xr) return 0;
18     if (yl > yr) return 0;
19     if (!a && !b) {
20         if (c) return 0;
21         return (xr - xl + 1) * (yr - yl + 1);
22     }
23     if (!b) {
24         swap(a, b);
25         swap(xl, yl);
26         swap(xr, yr);
27     }
28     if (!a) {
29         if (c % b) return 0;
30         ll y = -c / b;
31         if (y < yl || y > yr) return 0;
32         return xr - xl + 1;
33     }
34     d = exgcd((a % abs(b) + abs(b)) % abs(b), abs(b), x, y);
35     if (c % d) return 0;
36     x = (x % abs(b) + abs(b)) % abs(b) * (((-c) % abs(b)) + abs(b)) % abs(b)
        / d % abs(b / d);
37     d = abs(b / d);
38     ll kl = (xl - x) / d - 3, kr = (xr - x) / d + 3;
39     while (x + kl * d < xl) kl++;
40     while (x + kr * d > xr) kr--;
41     ll A = (-yl * b - a * x - c) / (a * d), B = (-yr * b - a * x - c) / (a *
        d);
42     if (A > B) swap(A, B);
43     kl = max(kl, A - 3);

```

```

44     kr = min(kr, B + 3);
45     while (kl <= kr) {
46         ll y = (-c - a * x - a * d * kl) / b;
47         if (y1 <= y && y <= yr) break;
48         kl++;
49     }
50     while (kl <= kr) {
51         ll y = (-c - a * x - a * d * kr) / b;
52         if (y1 <= y && y <= yr) break;
53         kr--;
54     }
55     if (kl > kr) return 0;
56     return kr - kl + 1;
57 }

```

2.5 类欧几里得

2.5.1 naive

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

原理：

$$\begin{aligned}
 f(a, b, c, n) &= \sum_{i=0}^n \left\lfloor \frac{ai + b}{c} \right\rfloor \\
 &= \sum_{i=0}^n \left\lfloor \frac{(\lfloor \frac{a}{c} \rfloor c + a \bmod c) i + (\lfloor \frac{b}{c} \rfloor c + b \bmod c)}{c} \right\rfloor \\
 &= \frac{n(n+1)}{2} \left\lfloor \frac{a}{c} \right\rfloor + (n+1) \left\lfloor \frac{b}{c} \right\rfloor + \sum_{i=0}^n \left\lfloor \frac{(a \bmod c) i + (b \bmod c)}{c} \right\rfloor \\
 &= \frac{n(n+1)}{2} \left\lfloor \frac{a}{c} \right\rfloor + (n+1) \left\lfloor \frac{b}{c} \right\rfloor + f(a \bmod c, b \bmod c, c, n)
 \end{aligned}$$

```

1  ll f(ll a, ll b, ll c, ll n) { // O(log n)
2      if (a == 0)
3          return (n + 1) * (b / c);
4      if (a >= c || b >= c)
5          return (f(a % c, b % c, c, n) + (a / c) * n * (n + 1) / 2 + (b / c)
6              * (n + 1));
7      ll m = (a * n + b) / c;
8      return (n * m - f(c, c - b - 1, a, m - 1));
9  }

```

2.5.2 general

$$f(a, b, c, n, k1, k2) = \sum_{i=0}^n i^{k1} \left\lfloor \frac{ai+b}{c} \right\rfloor^{k2}$$

```

1  // n,a,b,c>=1e9 && k1+k2<=10
2  // O(k^4log(MAXN))
3  using ll = long long;
4  const int K = 10;
5  const int mod = 1e9 + 7;
6  const ll lmod = ll(mod) << 32;
7  const int signs[2] = { 1, mod - 1 };
8  int invs[K + 2], binom[K + 2][K + 2], B[K + 1], polys[K + 1][K + 2];
9
10 int add(int a, int b) { return (a += b - mod) < 0 ? a + mod : a; }
11 ll add64(ll a, ll b) { return (a += b - lmod) < 0 ? a + lmod : a; }
12 int mul(int a, int b) { return 1LL * a * b % mod; }
13 int power_sum(int e, int x) {
14     int ret = 0;
15     for (int i = 0; i < e + 2; ++i)
16         ret = add(mul(ret, x), polys[e][i]);
17     return mul(ret, invs[e + 1]);
18 }
19
20 void init() {
21     invs[0] = invs[1] = 1; B[0] = 1;
22     for (int i = 2; i <= K + 1; ++i)
23         invs[i] = mul(invs[mod % i], mod - mod / i);
24     for (int i = 0; i <= K + 1; ++i) {
25         binom[i][0] = 1;
26         for (int j = 1; j <= i; ++j)
27             binom[i][j] = add(binom[i - 1][j - 1], binom[i - 1][j]);
28     }
29     for (int i = 1; i <= K; ++i) {
30         int s = 0;
31         for (int j = 0; j < i; ++j)
32             s = add(s, mul(binom[i + 1][j], B[j]));
33         B[i] = mul(mul(s, invs[i + 1]), signs[1]);
34     }
35     for (int i = 0; i <= K; ++i) {
36         for (int j = 0; j <= i; ++j)
37             polys[i][j] = mul(mul(binom[i + 1][j], B[j]), signs[j & 1]);
38         polys[i][i + 1] = 0;
39     }

```

```

40     polys[0][1] = 1;
41 }
42
43 int euclidLike(int N, int a, int b, int c, int k1, int k2) {
44     assert(N >= 0); assert(a >= 0); assert(b >= 0); assert(c >= 1); assert(
        k1 + k2 <= K);
45     using T = tuple<int, int, int, int>;
46     stack<T> stac;
47     while (1) {
48         stac.emplace(N, a, b, c);
49         if (N < 0 || a == 0)
50             break;
51         if (a >= c) {
52             a %= c;
53         } else if (b >= c) {
54             b %= c;
55         } else {
56             N = (ll(a) * N + b) / c - 1;
57             b = c - 1 - b;
58             swap(a, c);
59         }
60     }
61
62     const int S = k1 + k2;
63     static int curr[K + 1][K + 1] = {}, next[K + 1][K + 1] = {};
64     while (!stac.empty()) {
65         tie(N, a, b, c) = stac.top();
66         stac.pop();
67         if (N < 0) {
68             ;
69         } else if (a == 0) {
70             int q = b / c;
71             for (int e1 = 0; e1 <= S; ++e1) {
72                 int s = power_sum(e1, N);
73                 for (int e2 = 0; e2 <= S - e1; ++e2)
74                     next[e1][e2] = s, s = mul(s, q);
75             }
76         } else if (a >= c || b >= c) {
77             int q = (a >= c) ? a / c : b / c;
78             int d = (a >= c) ? 1 : 0;
79             for (int e1 = 0; e1 <= S; ++e1) {
80                 for (int e2 = 0; e2 <= S - e1; ++e2) {
81                     ll s = 0;

```

```

82         int p = 1;
83         for (int i2 = 0; i2 <= e2; ++i2) {
84             s = add64(s, ll(p) * mul(binom[e2][i2], curr[e1 + i2
85                                     * d][e2 - i2]));
86             p = mul(p, q);
87         }
88         next[e1][e2] = s % mod;
89     }
90 } else {
91     static int cumu[K + 1][K + 1];
92     for (int e2 = 0; e2 <= S - 1; ++e2) {
93         for (int e1 = 0; e1 <= S - e2 - 1; ++e1) {
94             ll s = 0;
95             for (int j = 0; j <= e1 + 1; ++j) {
96                 s = add64(s, ll(polys[e1][e1 + 1 - j]) * curr[e2][j
97                                     ]));
98             }
99             cumu[e1][e2] = mul(s % mod, invs[e1 + 1]);
100         }
101     }
102     const int M = (ll(a) * N + b) / c;
103     for (int e1 = 0; e1 <= S; ++e1) {
104         int p = power_sum(e1, N);
105         for (int e2 = 0; e2 <= S - e1; ++e2) {
106             ll t = 0;
107             for (int i2 = 0; i2 < e2; ++i2) {
108                 t = add64(t, ll(cumu[e1][i2]) * binom[e2][i2]);
109             }
110             next[e1][e2] = add(p, mod - t % mod);
111             p = mul(p, M);
112         }
113     }
114     swap(curr, next);
115 }
116 return curr[k1][k2];
117 }
```

2.6 欧拉定理

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

2.7 欧拉函数

2.7.1 暴力单点查询

```

1 long long phi(long long n) { // O(sqrt(N))
2     int m = int(sqrt(n + 0.5));
3     long long ans = n;
4     for (int i = 2; i <= m; i++) {
5         if (n % i == 0) {
6             ans = ans / i * (i - 1);
7             while (n % i == 0) n /= i;
8         }
9     }
10    if (n > 1) ans = ans / n * (n - 1);
11    return ans;
12 }
```

2.7.2 预处理单点查询

```

1 vector<int> prime; // 求 n 的欧拉函数需要先把 <= ceil(sqrt(n)) 的素数筛出
2 long long phi(long long n) { // O(sqrt(N)/log(N))
3     long long res = n;
4     for (int i = 0; i < (int)prime.size(); i++) {
5         long long p = prime[i];
6         if (p * p > n) break;
7         if (n % p == 0) {
8             res = res / p * (p - 1);
9             while (n % p == 0) n /= p;
10        }
11    }
12    if (n > 1) res = res / n * (n - 1);
13    return res;
14 }
```

2.8 中国剩余定理

2.8.1 CRT

```

1 // 求解形如  $x = c_i \pmod{m_i}$  的线性方程组 ( $m_i, m_j$ ) 必须两两互质
2 long long CRT(vector<long long>& c, vector<long long>& m) {
3     long long M = m[0], ans = 0;
4     for (int i = 1; i < (int)m.size(); ++i) M *= m[i];
5     for (int i = 0; i < (int)m.size(); ++i) { //  $M_i * t_i * c_i$ 
6         long long mi = M / m[i];
7         long long ti = inv(mi, m[i]); //  $m_i$  模  $m[i]$  的逆元
8         ans = (ans + mi * ti % M * c[i] % M) % M;
9     }
10    ans = (ans + M) % M; // 返回模 M 意义下的唯一解
11    return ans;
12 }

```

2.8.2 EXCRT

```

1 long long exgcd(long long a, long long b, long long& x, long long& y) {
2     if (!b) {
3         x = 1, y = 0;
4         return a;
5     }
6     long long g = exgcd(b, a % b, y, x);
7     y -= (a / b) * x;
8     return g;
9 }
10
11 long long mulmod(long long x, long long y, const long long z) { //  $x * y \% z$ 
12     防爆
13     return (x * y - (long long)(((long double)x * y + 0.5) / (long double)z)
14         * z + z) % z;
15 }
16
17 // 求解形如  $x = c_i \pmod{m_i}$  的线性方程组
18 long long EXCRT(vector<long long>& c, vector<long long>& m) {
19     long long M = m[0], ans = c[0];
20     for (int i = 1; i < (int)m.size(); ++i) { //  $M * x - m_i * y = c_i - C$ 
21         long long x, y, C = ((c[i] - ans) % m[i] + m[i]) % m[i]; //  $c_i - C$ 
22         long long G = exgcd(M, m[i], x, y);
23         if (C % G) return -1; // 无解
24         long long P = m[i] / G;
25         x = mulmod(C / G, x, P); // 防爆求最小正整数解 x
26         ans += x * M;
27         M *= P; // LCM(M,  $m_i$ )
28         ans = (ans % M + M) % M;
29     }
30 }

```



```

27     }
28     return ans;
29 }

```

2.9 BSGS

2.10 迪利克雷卷积

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

2.11 杜教筛

$$(f * g)(n) = \sum_{d|n} f(d)g(\frac{n}{d}) = \sum_{xy=n} f(x)g(y)$$

2.12 Berlekamp Massey

```

1 namespace Berlekamp_Massey {
2     typedef long long ll;
3     constexpr ll MOD = 1e9 + 7;
4     constexpr int N = 10010;
5     ll res[N], base[N], _c[N], _md[N];
6     vector<int> Md;
7     ll powmod(ll a, ll b) {
8         ll res = 1;
9         while (b > 0) {
10             if (b & 1) res = res * a % MOD;
11             a = a * a % MOD;
12             b >>= 1;
13         }
14         return res;
15     }
16     void mul(ll* a, ll* b, int k) {
17         for (int i = 0; i < k + k; i++)
18             _c[i] = 0;
19         for (int i = 0; i < k; i++) {
20             if (!a[i]) continue;
21             for (int j = 0; j < k; j++) {
22                 _c[i + j] = (_c[i + j] + a[i] * b[j]) % MOD;
23             }
24         }
25         for (int i = k + k - 1; i >= k; i--) {
26             if (!_c[i]) continue;

```

```

27         for (int j = 0; j < Md.size(); j++) {
28             __c[i - k + Md[j]] = (__c[i - k + Md[j]] - __c[i] * __md[Md[j]])
                % MOD;
29         }
30     }
31     for (int i = 0; i < k; i++)
32         a[i] = __c[i];
33 }
34 int solve(ll n, vector<int> a, vector<int> b) { //a系数 b初值 b[n+1]=a
    [0]*b[n]+...
35     //printf("%d\n", (int)b.size());
36     //for (int i = 0; i < b.size(); i++)
37     //    printf("b[%d] = %d\n", i, b[i]);
38     //printf("%d\n", (int)a.size());
39     //printf("b[n]");
40     //for (int i = 0; i < a.size(); i++) {
41     //    if (!i) putchar('='); else putchar('+');
42     //    printf("%d*b[%d]", a[i], i + 1);
43     //}
44     //puts("");
45     ll ans = 0, pnt = 0;
46     int k = a.size();
47     for (int i = 0; i < k; i++) {
48         __md[k - 1 - i] = -a[i];
49     }
50     __md[k] = 1;
51     Md.clear();
52     for (int i = 0; i < k; i++) {
53         if (__md[i]) {
54             Md.push_back(i);
55         }
56         res[i] = base[i] = 0;
57     }
58     res[0] = 1;
59     while ((1LL << pnt) <= n) pnt++;
60     for (int p = pnt; p >= 0; p--) {
61         mul(res, res, k);
62         if ((n >> p) & 1) {
63             for (int i = k - 1; i >= 0; i--)
64                 res[i + 1] = res[i];
65             res[0] = 0;
66             for (int j = 0; j < Md.size(); j++) {
67                 res[Md[j]] = (res[Md[j]] - res[k] * __md[Md[j]]) % MOD;

```

```

68         }
69     }
70 }
71 for (int i = 0; i < k; i++)
72     ans = (ans + res[i] * b[i]) % MOD;
73 return (ans < 0 ? ans + MOD : ans);
74 }
75 vector<int> BM(vector<int> s) { // O(n^2)
76     vector<int> C(1, 1), B(1, 1);
77     int L = 0, m = 1, b = 1;
78     for (int n = 0; n < (int)s.size(); n++) {
79         ll d = 0;
80         for (int i = 0; i <= L; i++)
81             d = (d + (ll)C[i] * s[n - i]) % MOD;
82         if (!d) {
83             ++m;
84         } else if (2 * L <= n) {
85             auto T = C;
86             ll c = MOD - d * powmod(b, MOD - 2) % MOD;
87             while (C.size() < B.size() + m)
88                 C.push_back(0);
89             for (int i = 0; i < B.size(); i++)
90                 C[i + m] = (C[i + m] + c * B[i]) % MOD;
91             L = n + 1 - L; B = T; b = d; m = 1;
92         } else {
93             ll c = MOD - d * powmod(b, MOD - 2) % MOD;
94             while (C.size() < B.size() + m) C.push_back(0);
95             for (int i = 0; i < B.size(); i++) {
96                 C[i + m] = (C[i + m] + c * B[i]) % MOD;
97             }
98             ++m;
99         }
100     }
101     return C;
102 }
103 int work(vector<int> a, ll n) { // 这里的n不是数组大小 是求数列第n项的值
104     vector<int> c = BM(a); // 求第n项的复杂度为 O(k^2 log n) k是递推
    数列大小
105     c.erase(c.begin());
106     for (int i = 0; i < c.size(); i++)
107         c[i] = (MOD - c[i]) % MOD;
108     return solve(n, c, vector<int>(a.begin(), a.begin() + (int)c.size()))
    );

```

```

109     }
110 }

```

3 线性代数

3.1 矩阵

3.2 高斯-约旦消元法

```

1  /*
2  * 高斯-约旦消元法
3  * 可以修改为解异或方程组 修改策略为
4  * a+b -> a^b
5  * a-b -> a^b
6  * a*b -> a&b
7  * a/b -> a*(b==1)
8  * */
9  constexpr double eps = 1e-7;
10 double a[SIZE][SIZE], ans[SIZE];
11 void gauss(int n) {
12     vector<bool> vis(n, false);
13     for (int i = 0; i < n; i++) {
14         for (int j = 0; j < n; j++) {
15             if (vis[j]) continue;
16             if (fabs(a[j][i]) > eps) {
17                 vis[i] = true;
18                 for (int k = 0; k <= n; k++) swap(a[i][k], a[j][k]);
19                 break;
20             }
21         }
22         if (fabs(a[i][i]) < eps) continue;
23         for (int j = 0; j <= n; j++) {
24             if (i != j && fabs(a[j][i]) > eps) {
25                 double res = a[j][i] / a[i][i];
26                 for (int k = 0; k <= n; k++) a[j][k] -= a[i][k] * res;
27             }
28         }
29     }
30 }
31
32 int check(int n) { // 解系检测
33     int status = 1;
34     for (int i = n - 1; i >= 0; i--) {
35         if (fabs(a[i][i]) < eps && fabs(a[i][n]) > eps) return -1; // 无解

```

```

36         if (fabs(a[i][i]) < eps && fabs(a[i][n]) < eps) status = 0; // 无穷
           解
37         ans[i] = a[i][n] / a[i][i];
38         if (fabs(ans[i]) < eps) ans[i] = 0;
39     }
40     return status; // 唯一解或无穷解
41 }

```

3.3 高斯消元法-bitset

```

1  constexpr int SIZE = 1001;
2  bitset<SIZE> a[SIZE];
3  int ans[SIZE];
4  void gauss(int n) { // bitset版高斯消元 用于求解异或线性方程组
5      bitset<SIZE> vis;
6      for (int i = 0; i < n; i++) {
7          for (int j = 0; j < n; j++) {
8              if (vis[j]) continue;
9              if (a[j][i]) {
10                 vis.set(i);
11                 swap(a[i], a[j]);
12                 break;
13             }
14         }
15         if (!a[i][i]) continue;
16         for (int j = 0; j <= n; j++) {
17             if (i != j && (a[j][i] & a[i][i])) {
18                 a[j] ^= a[i];
19             }
20         }
21     }
22 }

```

3.4 线性基

```

1  struct linearBasis {
2      /* 线性基性质:
3       * 1.若a[i]!=0 (即主元i存在)
4       *    则线性基中只有a[i]的第i位是1 (只存在一个主元)
5       *    且此时a[i]的最高位就是第i位
6       * 2.将数组a插入线性基 假设有|B|个元素成功插入
7       *    则数组a中每个不同的子集异或和都出现 2^(n-|B|) 次
8       * */

```

```

9      static const int MAXL = 60;
10     long long a[MAXL + 1];
11     int id[MAXL + 1];
12     int zero;
13     /* 0的标志位 =1则表示0可以被线性基表示出来
14      * 求第k大元素时 需要注意题意中线性基为空时是否可以表示0
15      * 默认不可以表示 有必要时进行修改即可
16      * */
17     linearBasis() {
18         zero = 0;
19         fill(a, a + MAXL + 1, 0);
20     }
21     long long& operator[] (int k) { return a[k]; }
22     bool insert(long long x) {
23         for (int j = MAXL; ~j; j--) {
24             if (!(x & (1LL << j))) { // 如果 x 的第 j 位为 0, 则跳过
25                 continue;
26             }
27             if (a[j]) { // 如果 a[j] != 0, 则用 a[j] 消去 x 的第 j 位上的 1
28                 x ^= a[j];
29             } else { // 找到插入位置
30                 for (int k = 0; k < j; k++) {
31                     if (x & (1LL << k)) { // 如果x存在某个低位线性基的主元k
32                         // 则消去
33                         x ^= a[k];
34                     }
35                 }
36                 for (int k = j + 1; k <= MAXL; k++) {
37                     if (a[k] & (1LL << j)) { // 如果某个高位线性基存在主元j
38                         // 则消去
39                         a[k] ^= x;
40                     }
41                 }
42                 a[j] = x;
43                 return true;
44             }
45         }
46         zero = 1;
47         return false;
48     }
49     long long query_max() { // 最大值
50         long long res = 0;
51         for (int i = MAXL; ~i; i--) {

```

```

50         res ^= a[i];
51     }
52     return res;
53 }
54 long long query_max(long long x) { // 线性基异或x的最大值
55     for (int i = MAXL; ~i; i--) {
56         if ((x ^ a[i]) > x) {
57             x ^= a[i];
58         }
59     }
60     return x;
61 }
62 long long query_min() { // 最小值
63     for (int i = 0; i < MAXL; i++) {
64         if (a[i]) {
65             return a[i];
66         }
67     }
68     return -1; // 线性基为空
69 }
70 long long query_min(long long x) { // 线性基异或x的最小值
71     for (int i = MAXL; ~i; i--) {
72         if ((x ^ a[i]) < x) {
73             x ^= a[i];
74         }
75     }
76     return x;
77 }
78 int count(long long x) { // 元素 x 能否被线性基内元素表示
79     int res = 0;
80     vector<long long> b(MAXL + 1);
81     for (int i = 0; i <= MAXL; i++) {
82         b[i] = a[i];
83     }
84     res = this->insert(x);
85     for (int i = 0; i <= MAXL; i++) {
86         a[i] = b[i];
87     }
88     return !res; // 成功插入则无法表示 失败则可以表示
89 }
90 int size() { // 线性基有效元素数量
91     int res = 0;
92     for (int i = 0; i <= MAXL; i++) {

```

```

93         if (a[i]) {
94             res++;
95         }
96     }
97     return res;
98 }
99 long long kth_element(long long k) { // 第k大元素
100     vector<long long> b;
101     for (int i = 0; i <= MAXL; i++) {
102         if (a[i]) {
103             b.push_back(a[i]);
104         }
105     }
106     if (zero) {
107         if (--k == 0) {
108             return 0;
109         }
110     }
111     if (k >= (1LL << this->size())) { // k超过了线性基可以表示的最大数量
112         return -1;
113     }
114     long long res = 0;
115     for (int i = 0; i <= MAXL; i++) {
116         if (k & (1LL << i)) {
117             res ^= b[i];
118         }
119     }
120     return res;
121 }
122 long long rank(long long x) { // 元素x在线性基内的排名（默认不考虑0）
123     vector<long long> b;
124     for (int i = 0; i <= MAXL; i++) {
125         if (a[i]) {
126             b.push_back(1LL << i);
127         }
128     }
129     long long res = 0;
130     for (int i = 0; i < (int)b.size(); i++) {
131         if (x & b[i]) {
132             res |= (1LL << i);
133         }
134     }
135     return res;

```



```

136     }
137     void clear() {
138         zero = 0;
139         fill(a, a + MAXL + 1, 0);
140     }
141 };

```

3.5 矩阵树定理

```

1  /*
2  * 矩阵树定理
3  * 有向图：若  $u \rightarrow v$  有一条权值为  $w$  的边 基尔霍夫矩阵  $a[v][v] += w$ ,  $a[v][u] -= w$ 
4  * 生成树数量为除去 根所在行和列 后的 $n-1$ 阶行列式的值
5  * 无向图：若  $u \rightarrow v$  有一条权值为  $w$  的边 基尔霍夫矩阵  $a[v][v] += w$ ,  $a[v][u] -= w$ ,  $a[u][u] += w$ ,  $a[u][v] -= w$ 
6  * 生成树数量为除去 任意一行和列 后的 $n-1$ 阶行列式的值
7  * 无权图则边权默认为1
8  * */
9  typedef long long ll;
10 typedef unsigned long long u64;
11 int a[SIZE][SIZE];
12 int gauss(int a[][SIZE], int n) { // 任意模数求行列式  $O(n^2(n + \log(\text{mod})))$ 
13     int ans = 1;
14     for (int i = 1; i <= n; i++) {
15         int* x = 0, * y = 0;
16         for (int j = i; j <= n; j++) {
17             if (a[j][i] && (x == NULL || a[j][i] < x[i])) {
18                 x = a[j];
19             }
20         }
21         if (x == 0) {
22             return 0;
23         }
24         for (int j = i; j <= n; j++) {
25             if (a[j] != x && a[j][i]) {
26                 y = a[j];
27                 for (;) {
28                     int v = md - y[i] / x[i], k = i;
29                     for (; k + 3 <= n; k += 4) {
30                         y[k + 0] = (y[k + 0] + u64(x[k + 0]) * v) % md;
31                         y[k + 1] = (y[k + 1] + u64(x[k + 1]) * v) % md;
32                         y[k + 2] = (y[k + 2] + u64(x[k + 2]) * v) % md;

```

```

33         y[k + 3] = (y[k + 3] + u64(x[k + 3]) * v) % md;
34     }
35     for (; k <= n; ++k) {
36         y[k] = (y[k] + u64(x[k]) * v) % md;
37     }
38     if (!y[i]) break;
39     swap(x, y);
40 }
41 }
42 }
43 if (x != a[i]) {
44     for (int k = i; k <= n; k++) {
45         swap(x[k], a[i][k]);
46     }
47     ans = md - ans;
48 }
49 ans = 1LL * ans * a[i][i] % md;
50 }
51 return ans;
52 }

```

3.6 LGV 引理

一般用于有向无环图不相交路径计数（常见于网格图）。

$$M = \begin{bmatrix} e(A_1, B_1) & e(A_1, B_2) & \cdots & e(A_1, B_n) \\ e(A_2, B_1) & e(A_2, B_2) & \cdots & e(A_2, B_n) \\ \vdots & \vdots & \ddots & \vdots \\ e(A_n, B_1) & e(A_n, B_2) & \cdots & e(A_n, B_n) \end{bmatrix} \det(M) = \sum_{S:A \rightarrow B} (-1)^{N(\sigma(S))} \prod_{i=1}^n \omega(S_i)$$

4 组合数学

4.1 组合数预处理

```

1 namespace BinomialCoefficient {
2     vector<int> fac, ifac, iv;
3     // 组合数预处理 option=1则还会预处理线性逆元
4     void prepareFactorials(int maximum = 1000000, int option = 0) {
5         fac.assign(maximum + 1, 0);
6         ifac.assign(maximum + 1, 0);
7         fac[0] = ifac[0] = 1;
8         if (option) { // O(3n)
9             iv.assign(maximum + 1, 1);
10            for (int p = 2; p * p <= MOD; p++)

```

```

11         assert(MOD % p != 0);
12         for (int i = 2; i <= maximum; i++)
13             iv[i] = mul(iv[MOD % i], (MOD - MOD / i));
14         for (int i = 1; i <= maximum; i++) {
15             fac[i] = mul(i, fac[i - 1]);
16             ifac[i] = mul(iv[i], ifac[i - 1]);
17         }
18     } else { // O(2n + log(MOD))
19         for (int i = 1; i <= maximum; i++)
20             fac[i] = mul(fac[i - 1], i);
21         ifac[maximum] = inv(fac[maximum]);
22         for (int i = maximum; i; i--)
23             ifac[i - 1] = mul(ifac[i], i);
24     }
25 }
26 inline int binom(int n, int m) {
27     if (n < m || n < 0 || m < 0) return 0;
28     return mul(fac[n], mul(ifac[m], ifac[n - m]));
29 }
30 }

```

4.2 卢卡斯定理

对于质数 p ，有：

$$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$$

```

1 long long p; // C(n, m) % p
2 long long lucas(long long n, long long m) { // O(p + log(n))
3     if (m == 0) return 1;
4     return (binom(n % p, m % p) * lucas(n / p, m / p)) % p; // binom(x, y)
5     // 在线性预处理组合数之后是 O(1) 的
6 }

```

4.3 小球盒子模型

设有 n 个球， k 个盒子：

1. 球之间互不相同，盒子之间互不相同，可以空盒

根据乘法原理，答案就是 k^n 。

2. 球之间互不相同，盒子之间互不相同，每个盒子至多装一个球

相当于每个球找一个没有被选过的盒子放进去，答案是 k^n ，即 $k(k-1)\cdots(k-n+1)$ 。

3. 球之间互不相同，盒子之间互不相同，每个盒子至少装一个球

可以先把盒子视为相同： n 个球放进 k 个相同盒子、不能空盒，这就是第二类斯特林数 S_n^k 的定义。最后由于盒子不同，再乘上一个排列数，因此答案就是 $k!S_n^k$ 。

4. 球之间互不相同，盒子全部相同，可以空盒

枚举非空盒子数量，相当于第二类斯特林数一行求和： $\sum_{i=1}^k S_n^i$ 。

5. 球之间互不相同，盒子全部相同，每个盒子至多装一个球

因为盒子相同，不论怎么放都是一样的，答案是 $[n \leq k]$ （这是一个布尔运算式，若 $n \leq k$ 成立则取 1，否则 0）。

6. 球之间互不相同，盒子全部相同，每个盒子至少装一个球

就是第二类斯特林数 S_n^k 。

7. 球全部相同，盒子之间互不相同，可以空盒

隔板法经典应用， $n + k - 1$ 个球选 $k - 1$ 个板，因此答案是 $\binom{n+k-1}{k-1}$ 。

8. 球全部相同，盒子之间互不相同，每个盒子至多装一个球

盒子不同，相当于要选出 n 个盒子装球，因此答案是 $\binom{n}{k}$ 。

9. 球全部相同，盒子之间互不相同，每个盒子至少装一个球

隔板法经典应用， $n - 1$ 个空隙选 $k - 1$ 个插板（可以看作是情况 7 时每个盒子里都预先加入一个球），因此答案是 $\binom{n-1}{k-1}$ 。

10. 球全部相同，盒子全部相同，可以空盒

定义划分数 $p_{n,k}$ 表示将自然数 n 拆成 k 份的方案数，那么本例的结论就是 $p_{n,k}$ 。

这个问题有一个经典递推式： $p(n, k) = p(n, k - 1) + p(n - k, k)$ 。意义是将 j 个自然数 +1 或者加入一个 0。下面给出一个代码实现：

```

1  p[0][0] = 1;
2  for (int i = 1; i <= n; i++) {
3      p[0][i] = 1;
4      for (int j = 1; j <= m; j++) {
5          if (i >= j) {
6              p[i][j] = add(p[i][j - 1], p[i - j][j]);
7          } else {
8              p[i][j] = p[i][j - 1];
9          }
10     }
11 }
```

11. 球全部相同，盒子全部相同，每个盒子至多装一个球

和情况 5 一致，就是 $[n \leq k]$ 。

12. 球全部相同，盒子全部相同，每个盒子至少装一个球

显然也是一个划分数： $p_{n-k,k}$ 。

4.4 斯特林数

4.4.1 第一类斯特林数

第一类斯特林数 $\begin{bmatrix} n \\ k \end{bmatrix}$ 表示将 n 个不同元素划分入 k 个非空圆排列的方案数。

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

边界是 $\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1$ 。

第一类斯特林数三角形，从 $s(1, 1)$ 开始：

1									
1	1								
2	3	1							
6	11	6	1						
24	50	35	10	1					
120	274	225	85	15	1				
720	1764	1624	735	175	21	1			
5040	13068	13132	6769	1960	322	28	1		
40320	109584	118124	67284	22449	4536	546	36	1	
362880	1026576	1172700	723680	269325	63273	9450	870	45	1

4.4.2 第二类斯特林数

第二类斯特林数 $\begin{Bmatrix} n \\ k \end{Bmatrix}$ 表示将 n 个不同元素划分为 k 个非空子集的方案数。

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k \begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

边界是 $\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = 1$ 。

基于容斥原理的递推方法：

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

第二类斯特林数三角形，从 $S(1, 1)$ 开始：

1									
1	1								
1	3	1							
1	7	6	1						
1	15	25	10	1					
1	31	90	65	15	1				
1	63	301	350	140	21	1			
1	127	966	1701	1050	266	28	1		
1	255	3025	7770	6951	2646	462	36	1	
1	511	9330	34105	42525	22827	5880	750	45	1

5 博弈论

5.1 SG 定理

一个状态的 SG 函数值是它所有后继状态的 MEX，当 MEX 为 0 时该状态后手必胜，反之先手必胜。当某个游戏由多个独立的子游戏组成时，所有子游戏的 SG 函数异或和为 0 时后手必胜，否则先手必胜。

在 Nim 博弈中的简单应用：显然第 i 堆石子的 SG 函数值就是它的石子数量 a_i ，每一堆石子都是一个独立的子游戏，因此 $a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1} = 0$ 时后手必胜，否则先手必胜。

5.2 Bash 博弈

一共 N 个石子，先后手轮流取石子，每次最多取 M 个石子，先取完者获胜。

先手必胜： $N \pmod{M+1} \neq 0$ ；必败： $N \pmod{M+1} = 0$ 。

5.3 Nim-K 博弈

有 N 堆石子，先后手轮流取石子，每次最多可以选择 K 堆石子，被选中的每堆石子都可以取任意个，先取完者获胜。

把 N 堆石子的石子数量都用二进制表示，对于二进制意义下的每一位，如果 1 的数量在模 $K+1$ 意义下全部为 0，则先手必败。

5.4 Anti-Nim 博弈

规则和 Nim 博弈一致，但是获胜条件改为：不能取石子的一方获胜。

满足以下任意条件，则先手必胜：

1. 所有堆的石子数量 ≤ 1 并且异或和 $= 0$ 。
2. 至少存在一堆石子个数 ≥ 2 并且异或和 $\neq 0$ 。

5.5 Anti-SG 博弈

SG 博弈中最先不能行动的一方获胜。满足以下任意条件，则先手必胜：

1. SG 为 0 并且每一个游戏的 SG 都不超过 1。
2. SG 不为 0 并且至少有一个游戏的 SG 大于 1。

5.6 阶梯博弈

有 N 个阶梯（下标从 0 开始），每个阶梯上有一定数量的石子，先后手轮流行动，每次可以从一个阶梯上拿走任意个石子放到下一层阶梯上，最先不能操作者失败。

SG 函数为奇数阶梯上的石子的异或和，如果移动偶数层的石子到奇数层，对手一定可以继续移动这些石子到偶数层使得 SG 不变。

5.7 Wythoff 博弈

有两堆石子，石子数量分别为 A, B ，每次可以从一堆或者两堆里拿走**相同数量**的石子，最先不能取石子的人输。

必败态为： $A_k = \lfloor \frac{k(1+\sqrt{5})}{2} \rfloor, B_k = a_k + k$ ，假设 $A_k < B_k$ 。

5.8 树上删边博弈

给定一棵 N 个点的有根树，两人轮流操作，每次删除树上的一条边，然后删除所有不与根节点联通的部分，最先不能进行删除操作的人失败。

所有叶子节点的 SG 函数值为 0，非叶子节点的 SG 值为其所有子节点（SG 值 +1）的异或和。

5.9 无向图删边博弈

和树上删边博弈的规则一样，不过给出的是一个无向图。

可以将图中的任意一个偶环缩成一个新点，任意一个奇环缩成一个新点加一个新边；所有连到原先环上的边全部改为与新点相连。

5.10 二分图博弈

给出一张二分图和起点 S ，先后手轮流操作，每次只能从当前点（最开始是点 S ）移动到一个**相邻**的点，且每个点只能被**遍历一次**，无法继续移动的人输。

先手必胜：二分图的最大匹配**一定**包含起点 S ；先手必败：二分图的最大匹配**不一定**包含起点 S 。

6 图论

6.1 并查集

```

1 struct dsu {
2 private:
3     // number of nodes
4     int n;
5     // root node: -1 * component size
6     // otherwise: parent
7     std::vector<int> pa;
8 public:
9     dsu(int n_ = 0) : n(n_), pa(n_, -1) {}
10    // find node x's parent
11    int find(int x) {

```

```

12     return pa[x] < 0 ? x : pa[x] = find(pa[x]);
13 }
14 // merge node x and node y
15 // if x and y had already in the same component, return false, otherwise
    return true
16 // Implement (union by size) + (path compression)
17 bool unite(int x, int y) {
18     int xr = find(x), yr = find(y);
19     if (xr != yr) {
20         if (-pa[xr] < -pa[yr]) std::swap(xr, yr);
21         pa[xr] += pa[yr];
22         pa[yr] = xr; // y -> x
23         return true;
24     }
25     return false;
26 }
27 // size of the connected component that contains the vertex x
28 int size(int x) {
29     return -pa[find(x)];
30 }
31 };

```

6.2 最短路

```

1 namespace Dijkstra {
2 #define ll long long
3     static constexpr ll INF = 1e18;
4     int n, m; // 点数 边数
5     struct edge {
6         int to; // 点
7         ll val; // 边权
8         edge(int to_ = 0, ll val_ = 0) : to(to_), val(val_) {}
9         bool operator < (const edge& k) const { return val > k.val; }
10 };
11     vector<vector<edge>> g;
12     void init() { // 建图操作需要根据题意修改
13         cin >> n >> m;
14         g.resize(n);
15         for (int i = 0; i < m; i++) {
16             int u, v, w;
17             cin >> u >> v >> w;
18             —u, —v;
19             g[u].push_back(edge(v, w));

```



```

20     }
21 }
22 ll dijkstra(int s, int t) { // 最短路
23     vector<ll> dis(n, INF);
24     vector<bool> vis(n, false);
25     dis[s] = 0;
26     priority_queue<edge> pq;
27     pq.push(edge(s, 0));
28     while (!pq.empty()) {
29         auto top = pq.top();
30         pq.pop();
31         if (!vis[top.to]) {
32             vis[top.to] = true;
33             for (auto nxt : g[top.to]) {
34                 if (!vis[nxt.to] && dis[nxt.to] > nxt.val + dis[top.to])
35                     {
36                         dis[nxt.to] = nxt.val + dis[top.to];
37                         pq.push(edge(nxt.to, dis[nxt.to]));
38                     }
39             }
40         }
41         return dis[t];
42     }
43 #undef ll
44 }

```

6.3 最小树形图

```

1 namespace ZL {
2     // a 尽量开大，之后的边都塞在这个里面
3     const int N = 100010, M = 100010, inf = 1e9;
4     struct edge {
5         int u, v, w, use, id;
6         edge(int u_ = 0, int v_ = 0, int w_ = 0, int use_ = 0, int id_ = 0)
7             : u(u_), v(v_), w(w_), use(use_), id(id_) {}
8     } b[M], a[2000100];
9     int n, m, ans, pre[N], id[N], vis[N], root, In[N], h[N], len, way[M];
10    // 从root 出发能到达每一个点的最小树形图
11    // 先调用init 然后把边add 进去，需要方案就getway, way[i] 为1 表示使用
12    // 最小值保存在ans
13    void init(int _n, int _root) { // 点数 根节点
14        n = _n; m = 0; b[0].w = inf; root = _root;

```

```

15     }
16     void add(int u, int v, int w) {
17         m++;
18         b[m] = edge(u, v, w, 0, m);
19         a[m] = b[m];
20     }
21     int work() {
22         len = m;
23         for (;;) {
24             for (int i = 1; i <= n; i++) { pre[i] = 0; In[i] = inf; id[i] =
25                 0; vis[i] = 0; h[i] = 0; }
26             for (int i = 1; i <= m; i++) {
27                 if (b[i].u != b[i].v && b[i].w < In[b[i].v]) {
28                     pre[b[i].v] = b[i].u; In[b[i].v] = b[i].w; h[b[i].v] = b
29                         [i].id;
30                 }
31             }
32             for (int i = 1; i <= n; i++) if (pre[i] == 0 && i != root)
33                 return 0;
34             int cnt = 0; In[root] = 0;
35             for (int i = 1; i <= n; i++) {
36                 if (i != root) a[h[i]].use++; int now = i; ans += In[i];
37                 while (vis[now] == 0 && now != root) { vis[now] = i; now =
38                     pre[now]; }
39                 if (now != root && vis[now] == i) {
40                     cnt++; int kk = now;
41                     while (1) {
42                         id[now] = cnt; now = pre[now];
43                         if (now == kk) break;
44                     }
45                 }
46             }
47             if (cnt == 0) return 1;
48             for (int i = 1; i <= n; i++) if (id[i] == 0) id[i] = ++cnt;
49             // 缩环，每一条接入的边都会茶包原来接入的那条边，所以要调整边权
50             // 新加的边是u，茶包的边是v
51             for (int i = 1; i <= m; i++) {
52                 int k1 = In[b[i].v], k2 = b[i].v;
53                 b[i].u = id[b[i].u];
54                 b[i].v = id[b[i].v];
55                 if (b[i].u != b[i].v) {
56                     b[i].w -= k1; a[++len].u = b[i].id; a[len].v = h[k2]; b[
57                         i].id = len;

```

```

53         }
54     }
55     n = cnt; root = id[root];
56 }
57 return 1;
58 }
59 void getway() {
60     for (int i = 1; i <= m; i++) way[i] = 0;
61     for (int i = len; i > m; i--) { a[a[i].u].use += a[i].use; a[a[i].v
        ].use -= a[i].use; }
62     for (int i = 1; i <= m; i++) way[i] = a[i].use;
63 }
64 }

```

6.4 最近公共祖先

```

1  constexpr int SIZE = 200010;
2  constexpr int DEPTH = 21; // 最大深度  $2^{\text{DEPTH}} - 1$ 
3  int pa[SIZE][DEPTH], dep[SIZE];
4  vector<int> g[SIZE]; // 邻接表
5  void dfs(int rt, int fin) { // 预处理深度和祖先
6      pa[rt][0] = fin;
7      dep[rt] = dep[pa[rt][0]] + 1; // 深度
8      for (int i = 1; i < DEPTH; i++) { // rt 的  $2^i$  祖先等价于 rt 的  $2^{(i-1)}$ 
          祖先的  $2^{(i-1)}$  祖先
9          pa[rt][i] = pa[pa[rt][i-1]][i-1];
10     }
11     int sz = g[rt].size();
12     for (int i = 0; i < sz; ++i) {
13         if (g[rt][i] == fin) continue;
14         dfs(g[rt][i], rt);
15     }
16 }
17
18 int LCA(int x, int y) {
19     if (dep[x] > dep[y]) swap(x, y);
20     int dif = dep[y] - dep[x];
21     for (int j = 0; dif; ++j, dif >>= 1) {
22         if (dif & 1) {
23             y = pa[y][j]; // 先跳到同一高度
24         }
25     }
26     if (y == x) return x;

```

```

27     for (int j = DEPTH - 1; j >= 0 && y != x; j--) { //从底往上跳
28         if (pa[x][j] != pa[y][j]) { //如果当前祖先不相等 我们就需要更新
29             x = pa[x][j];
30             y = pa[y][j];
31         }
32     }
33     return pa[x][0];
34 }

```

6.5 欧拉回路

```

1 // UOJ117 返回欧拉回路的边集（负数代表走了反向边）
2 #include <bits/stdc++.h>
3 using namespace std;
4 // 有向图欧拉回路 任意点的入度=出度
5 vector<int> directed_euler_circuit(int n, int m, const vector<vector<pair<
    int, int>>>& g) {
6     vector<int> d(n);
7     for (const auto& A : g) {
8         for (auto p : A) {
9             d[p.first]++;
10        }
11    }
12    for (int i = 0; i < n; i++) {
13        if (g[i].size() != d[i]) {
14            return {};
15        }
16    }
17    vector<vector<pair<int, int>>::const_iterator> it(n);
18    for (int i = 0; i < n; i++) it[i] = g[i].begin();
19    vector<int> vis(m + 1), p;
20    function<void(int)> dfs = [&](int u) {
21        for (auto& nxt = it[u]; nxt != g[u].end(); ) {
22            if (!vis[nxt->second]) {
23                vis[nxt->second] = 1;
24                int v = nxt->second;
25                dfs(nxt->first);
26                p.push_back(v);
27            } else {
28                nxt = next(nxt);
29            }
30        }
31    };

```

```

32     for (int i = 0; i < n; i++) {
33         if (!g[i].empty()) {
34             dfs(i);
35             break;
36         }
37     }
38     if (p.size() < m) return {};
39     reverse(p.begin(), p.end());
40     return p;
41 }
42 // 无向图欧拉回路 任意点的度数为偶数
43 vector<int> undirected_euler_circuit(int n, int m, const vector<vector<pair<
    int, int>>>& g) {
44     for (const auto& A : g) {
45         if (A.size() & 1) {
46             return {};
47         }
48     }
49     vector<vector<pair<int, int>>::const_iterator> it(n);
50     for (int i = 0; i < n; i++) it[i] = g[i].begin();
51     vector<int> vis(m + 1), p;
52     function<void(int)> dfs = [&](int u) {
53         for (auto& nxt = it[u]; nxt != g[u].end(); ) {
54             if (!vis[abs(nxt->second)]) {
55                 vis[abs(nxt->second)] = 1;
56                 int v = nxt->second;
57                 dfs(nxt->first);
58                 p.push_back(v);
59             } else {
60                 nxt = next(nxt);
61             }
62         }
63     };
64     for (int i = 0; i < n; i++) {
65         if (!g[i].empty()) {
66             dfs(i);
67             break;
68         }
69     }
70     if (p.size() < m) return {};
71     reverse(p.begin(), p.end());
72     return p;
73 }

```

```

74
75 int main() {
76     ios::sync_with_stdio(false);
77     cin.tie(nullptr);
78     cout.tie(nullptr);
79     int t, n, m;
80     cin >> t >> n >> m; // t=1是无向图 t=-1是有向图
81     vector<vector<pair<int, int>>> G(n + 1);
82     for (int i = 1, u, v; i <= m; i++) {
83         cin >> u >> v;
84         if (t == 1) G[v].push_back({ u, -i });
85         G[u].push_back({ v, i });
86     }
87     auto p = t == 1 ? undirected_euler_circuit(n + 1, m, G) :
        directed_euler_circuit(n + 1, m, G);
88     if (p.size() == m) {
89         cout << "YES\n";
90         for (int x : p)
91             cout << x << " \n"[x == p.back()];
92     } else {
93         cout << "NO\n";
94     }
95     return 0;
96 }

```

6.6 强连通分量

```

1 namespace SCC {
2     // Compressed Sparse Row
3     template <class E> struct csr {
4         std::vector<int> start;
5         std::vector<E> elist;
6         explicit csr(int n, const std::vector<std::pair<int, E>>& edges)
7             : start(n + 1), elist(edges.size()) {
8             for (auto e : edges) {
9                 start[e.first + 1]++;
10            }
11            for (int i = 1; i <= n; i++) {
12                start[i] += start[i - 1];
13            }
14            auto counter = start;
15            for (auto e : edges) {
16                elist[counter[e.first]++] = e.second;

```

```

17     }
18 }
19 };
20
21 struct scc_graph {
22 public:
23     explicit scc_graph(int n) : _n(n) {}
24
25     int num_vertices() { return _n; }
26
27     void add_edge(int from, int to) { edges.push_back({ from, {to} }); }
28
29     // @return pair of (# of scc, scc id)
30     std::pair<int, std::vector<int>> scc_ids() {
31         auto g = csr<edge>(_n, edges);
32         int now_ord = 0, group_num = 0;
33         std::vector<int> visited, low(_n), ord(_n, -1), ids(_n);
34         visited.reserve(_n);
35         auto dfs = [&](auto self, int v) -> void {
36             low[v] = ord[v] = now_ord++;
37             visited.push_back(v);
38             for (int i = g.start[v]; i < g.start[v + 1]; i++) {
39                 auto to = g.elist[i].to;
40                 if (ord[to] == -1) {
41                     self(self, to);
42                     low[v] = std::min(low[v], low[to]);
43                 } else {
44                     low[v] = std::min(low[v], ord[to]);
45                 }
46             }
47             if (low[v] == ord[v]) {
48                 while (true) {
49                     int u = visited.back();
50                     visited.pop_back();
51                     ord[u] = _n;
52                     ids[u] = group_num;
53                     if (u == v) break;
54                 }
55                 group_num++;
56             }
57         };
58         for (int i = 0; i < _n; i++) {
59             if (ord[i] == -1) dfs(dfs, i);

```

```

60     }
61     for (auto& x : ids) {
62         x = group_num - 1 - x;
63     }
64     return { group_num, ids };
65 }
66
67 // O(N + M)
68 // It returns the list of the SCC in topological order.
69 std::vector<std::vector<int>>> scc() {
70     auto ids = scc_ids();
71     int group_num = ids.first;
72     std::vector<int> counts(group_num);
73     for (auto x : ids.second) counts[x]++;
74     std::vector<std::vector<int>>> groups(ids.first);
75     for (int i = 0; i < group_num; i++) {
76         groups[i].reserve(counts[i]);
77     }
78     for (int i = 0; i < _n; i++) {
79         groups[ids.second[i]].push_back(i);
80     }
81     return groups;
82 }
83
84 private:
85     int _n;
86     struct edge {
87         int to;
88     };
89     std::vector<std::pair<int, edge>>> edges;
90 };
91 }

```

6.7 2-sat

```

1 struct two_sat {
2 public:
3     two_sat() : _n(0), scc(0) {}
4     explicit two_sat(int n) : _n(n), _answer(n), scc(2 * n) {}
5
6     // 加入一个限制: (i=f) or (j=g)
7     void add_clause(int i, bool f, int j, bool g) {
8         assert(0 <= i && i < _n);

```



```

9      assert(0 <= j && j < _n);
10     scc.add_edge(2 * i + (f ? 0 : 1), 2 * j + (g ? 1 : 0));
11     scc.add_edge(2 * j + (g ? 0 : 1), 2 * i + (f ? 1 : 0));
12 }
13 // 加入一个限制: i=f => j=g
14 void derive(int i, bool f, int j, bool g) {
15     add_clause(i, !f, j, g);
16 }
17 // O(N + M) 如果返回true, 则一个方案会保存在answer里
18 bool satisfiable() {
19     auto id = scc.scc_ids().second;
20     for (int i = 0; i < _n; i++) {
21         if (id[2 * i] == id[2 * i + 1]) return false;
22         _answer[i] = id[2 * i] < id[2 * i + 1];
23     }
24     return true;
25 }
26 std::vector<bool> answer() { return _answer; }
27
28 private:
29     int _n;
30     std::vector<bool> _answer;
31     SCC::scc_graph scc;
32 };

```

6.8 最大流

```

1  template <class T> struct simple_queue {
2      std::vector<T> payload;
3      int pos = 0;
4      void reserve(int n) { payload.reserve(n); }
5      int size() const { return int(payload.size()) - pos; }
6      bool empty() const { return pos == int(payload.size()); }
7      void push(const T& t) { payload.push_back(t); }
8      T& front() { return payload[pos]; }
9      void clear() {
10         payload.clear();
11         pos = 0;
12     }
13     void pop() { pos++; }
14 };
15
16 template <class Cap> struct mf_graph {

```

```

17 public:
18     mf_graph() : _n(0) {}
19     mf_graph(int n) : _n(n), g(n) {}
20
21     // returns an integer k such that this is the k-th edge that is added.
22     int add_edge(int from, int to, Cap cap) {
23         assert(0 <= from && from < _n);
24         assert(0 <= to && to < _n);
25         assert(0 <= cap);
26         int m = int(pos.size());
27         pos.push_back({ from, int(g[from].size()) });
28         int from_id = int(g[from].size());
29         int to_id = int(g[to].size());
30         if (from == to) to_id++;
31         g[from].push_back(_edge{ to, to_id, cap });
32         g[to].push_back(_edge{ from, from_id, 0 });
33         return m;
34     }
35
36     struct edge {
37         int from, to;
38         Cap cap, flow;
39     };
40
41     edge get_edge(int i) {
42         int m = int(pos.size());
43         assert(0 <= i && i < m);
44         auto _e = g[pos[i].first][pos[i].second];
45         auto _re = g[_e.to][_e.rev];
46         return edge{ pos[i].first, _e.to, _e.cap + _re.cap, _re.cap };
47     }
48     std::vector<edge> edges() {
49         int m = int(pos.size());
50         std::vector<edge> result;
51         for (int i = 0; i < m; i++) {
52             result.push_back(get_edge(i));
53         }
54         return result;
55     }
56     void change_edge(int i, Cap new_cap, Cap new_flow) {
57         int m = int(pos.size());
58         assert(0 <= i && i < m);
59         assert(0 <= new_flow && new_flow <= new_cap);

```

```

60     auto& _e = g[pos[i].first][pos[i].second];
61     auto& _re = g[_e.to][_e.rev];
62     _e.cap = new_cap - new_flow;
63     _re.cap = new_flow;
64 }
65
66 // max flow from s to t
67 // O(M*N^2) general
68 // O(min(M*N^2/3, M^3/2)) if capacities of edges are 1
69 Cap flow(int s, int t) {
70     return flow(s, t, std::numeric_limits<Cap>::max());
71 }
72 Cap flow(int s, int t, Cap flow_limit) {
73     assert(0 <= s && s < _n);
74     assert(0 <= t && t < _n);
75     assert(s != t);
76
77     std::vector<int> level(_n), iter(_n);
78     simple_queue<int> que;
79
80     auto bfs = [&]() {
81         std::fill(level.begin(), level.end(), -1);
82         level[s] = 0;
83         que.clear();
84         que.push(s);
85         while (!que.empty()) {
86             int v = que.front();
87             que.pop();
88             for (auto e : g[v]) {
89                 if (e.cap == 0 || level[e.to] >= 0) continue;
90                 level[e.to] = level[v] + 1;
91                 if (e.to == t) return;
92                 que.push(e.to);
93             }
94         }
95     };
96     auto dfs = [&](auto self, int v, Cap up) {
97         if (v == s) return up;
98         Cap res = 0;
99         int level_v = level[v];
100         for (int& i = iter[v]; i < int(g[v].size()); i++) {
101             _edge& e = g[v][i];
102             if (level_v <= level[e.to] || g[e.to][e.rev].cap == 0)

```

```

103         continue;
104         Cap d =
105             self(self, e.to, std::min(up - res, g[e.to][e.rev].cap))
106             ;
107         if (d <= 0) continue;
108         g[v][i].cap += d;
109         g[e.to][e.rev].cap -= d;
110         res += d;
111         if (res == up) break;
112     }
113     return res;
114 };
115
116 Cap flow = 0;
117 while (flow < flow_limit) {
118     bfs();
119     if (level[t] == -1) break;
120     std::fill(iter.begin(), iter.end(), 0);
121     while (flow < flow_limit) {
122         Cap f = dfs(dfs, t, flow_limit - flow);
123         if (!f) break;
124         flow += f;
125     }
126 }
127
128 std::vector<bool> min_cut(int s) {
129     std::vector<bool> visited(_n);
130     simple_queue<int> que;
131     que.push(s);
132     while (!que.empty()) {
133         int p = que.front();
134         que.pop();
135         visited[p] = true;
136         for (auto e : g[p]) {
137             if (e.cap && !visited[e.to]) {
138                 visited[e.to] = true;
139                 que.push(e.to);
140             }
141         }
142     }
143     return visited;

```

```

144     }
145
146 private:
147     int __n;
148     struct __edge {
149         int to, rev;
150         Cap cap;
151     };
152     std::vector<std::pair<int, int>> pos;
153     std::vector<std::vector<__edge>> g;
154 };

```

6.9 最小费用最大流

```

1  /*
2   * 费用流Cost常用类型的上限: int 范围内 0 <= nx <= 2e9 + 1000, long long 范围
   * 内: 0 <= nx <= 8e18 + 1000
3   *
4   * min_cost_slope() 函数返回的是一个分段函数F(x) (其中x代表流量上界, F(x)代
   * 表当前最大流量的最小费用)
5   * 返回的vector是所有F(x)改变的点
6   * 时间复杂度 O(f(N + M))log(N + M) f(N + M) 代表图的流量总和
7   * */
8  namespace MCMF {
9      template <class T> struct simple_queue {
10         std::vector<T> payload;
11         int pos = 0;
12         void reserve(int n) { payload.reserve(n); }
13         int size() const { return int(payload.size()) - pos; }
14         bool empty() const { return pos == int(payload.size()); }
15         void push(const T& t) { payload.push_back(t); }
16         T& front() { return payload[pos]; }
17         void clear() {
18             payload.clear();
19             pos = 0;
20         }
21         void pop() { pos++; }
22     };
23
24     template <class E> struct csr {
25         std::vector<int> start;
26         std::vector<E> elist;
27         explicit csr(int n, const std::vector<std::pair<int, E>>& edges)

```

```

28         : start(n + 1), elist(edges.size()) {
29     for (auto e : edges) {
30         start[e.first + 1]++;
31     }
32     for (int i = 1; i <= n; i++) {
33         start[i] += start[i - 1];
34     }
35     auto counter = start;
36     for (auto e : edges) {
37         elist[counter[e.first]++] = e.second;
38     }
39 }
40 };
41
42 template <class Cap, class Cost> struct mcf_graph {
43 public:
44     mcf_graph() {}
45     explicit mcf_graph(int n) : _n(n) {}
46
47     int add_edge(int from, int to, Cap cap, Cost cost) {
48         assert(0 <= from && from < _n);
49         assert(0 <= to && to < _n);
50         assert(0 <= cap);
51         assert(0 <= cost);
52         int m = int(_edges.size());
53         _edges.push_back({ from, to, cap, 0, cost });
54         return m;
55     }
56
57     struct edge {
58         int from, to;
59         Cap cap, flow;
60         Cost cost;
61     };
62
63     edge get_edge(int i) {
64         int m = int(_edges.size());
65         assert(0 <= i && i < m);
66         return _edges[i];
67     }
68     std::vector<edge> edges() { return _edges; }
69
70     std::pair<Cap, Cost> flow(int s, int t) {

```

```

71         return flow(s, t, std::numeric_limits<Cap>::max());
72     }
73     std::pair<Cap, Cost> flow(int s, int t, Cap flow_limit) {
74         return slope(s, t, flow_limit).back();
75     }
76     std::vector<std::pair<Cap, Cost>> slope(int s, int t) {
77         return slope(s, t, std::numeric_limits<Cap>::max());
78     }
79     std::vector<std::pair<Cap, Cost>> slope(int s, int t, Cap flow_limit
80     ) {
81         assert(0 <= s && s < _n);
82         assert(0 <= t && t < _n);
83         assert(s != t);
84
85         int m = int(_edges.size());
86         std::vector<int> edge_idx(m);
87
88         auto g = [&]() {
89             std::vector<int> degree(_n), redge_idx(m);
90             std::vector<std::pair<int, _edge>> elist;
91             elist.reserve(2 * m);
92             for (int i = 0; i < m; i++) {
93                 auto e = _edges[i];
94                 edge_idx[i] = degree[e.from]++;
95                 redge_idx[i] = degree[e.to]++;
96                 elist.push_back({ e.from, {e.to, -1, e.cap - e.flow, e.
97                     cost} });
98                 elist.push_back({ e.to, {e.from, -1, e.flow, -e.cost} })
99                     ;
100             }
101             auto _g = csr<_edge>(_n, elist);
102             for (int i = 0; i < m; i++) {
103                 auto e = _edges[i];
104                 edge_idx[i] += _g.start[e.from];
105                 redge_idx[i] += _g.start[e.to];
106                 _g.elist[edge_idx[i]].rev = redge_idx[i];
107                 _g.elist[redge_idx[i]].rev = edge_idx[i];
108             }
109             return _g;
110         }();
111
112         auto result = slope(g, s, t, flow_limit);

```

```

111         for (int i = 0; i < m; i++) {
112             auto e = g.elist[edge_idx[i]];
113             _edges[i].flow = _edges[i].cap - e.cap;
114         }
115
116         return result;
117     }
118
119 private:
120     int _n;
121     std::vector<edge> _edges;
122
123     // inside edge
124     struct _edge {
125         int to, rev;
126         Cap cap;
127         Cost cost;
128     };
129
130     std::vector<std::pair<Cap, Cost>> slope(csr<_edge>& g,
131         int s,
132         int t,
133         Cap flow_limit) {
134         // variants (C = maxcost):
135         //  $-(n-1)C \leq \text{dual}[s] \leq \text{dual}[i] \leq \text{dual}[t] = 0$ 
136         // reduced cost  $(= e.\text{cost} + \text{dual}[e.\text{from}] - \text{dual}[e.\text{to}]) \geq 0$  for
            all edge
137
138         // dual_dist[i] = (dual[i], dist[i])
139         std::vector<std::pair<Cost, Cost>> dual_dist(_n);
140         std::vector<int> prev_e(_n);
141         std::vector<bool> vis(_n);
142         struct Q {
143             Cost key;
144             int to;
145             bool operator<(Q r) const { return key > r.key; }
146         };
147         std::vector<int> que_min;
148         std::vector<Q> que;
149         auto dual_ref = [&]() {
150             for (int i = 0; i < _n; i++) {
151                 dual_dist[i].second = std::numeric_limits<Cost>::max();
152             }

```



```

153         std::fill(vis.begin(), vis.end(), false);
154         que_min.clear();
155         que.clear();
156
157         // que[0..heap_r) was heapified
158         size_t heap_r = 0;
159
160         dual_dist[s].second = 0;
161         que_min.push_back(s);
162         while (!que_min.empty() || !que.empty()) {
163             int v;
164             if (!que_min.empty()) {
165                 v = que_min.back();
166                 que_min.pop_back();
167             } else {
168                 while (heap_r < que.size()) {
169                     heap_r++;
170                     std::push_heap(que.begin(), que.begin() + heap_r
171                                     );
172                 }
173                 v = que.front().to;
174                 std::pop_heap(que.begin(), que.end());
175                 que.pop_back();
176                 heap_r--;
177             }
178             if (vis[v]) continue;
179             vis[v] = true;
180             if (v == t) break;
181             // dist[v] = shortest(s, v) + dual[s] - dual[v]
182             // dist[v] >= 0 (all reduced cost are positive)
183             // dist[v] <= (n-1)C
184             Cost dual_v = dual_dist[v].first, dist_v = dual_dist[v].
185                 second;
186             for (int i = g.start[v]; i < g.start[v + 1]; i++) {
187                 auto e = g.elist[i];
188                 if (!e.cap) continue;
189                 // |-dual[e.to] + dual[v]| <= (n-1)C
190                 // cost <= C - -(n-1)C + 0 = nC
191                 Cost cost = e.cost - dual_dist[e.to].first + dual_v;
192                 if (dual_dist[e.to].second - dist_v > cost) {
193                     Cost dist_to = dist_v + cost;
194                     dual_dist[e.to].second = dist_to;
195                     prev_e[e.to] = e.rev;

```

```

194         if (dist_to == dist_v) {
195             que_min.push_back(e.to);
196         } else {
197             que.push_back(Q{ dist_to, e.to });
198         }
199     }
200 }
201 }
202 if (!vis[t]) {
203     return false;
204 }
205
206 for (int v = 0; v < _n; v++) {
207     if (!vis[v]) continue;
208     // dual[v] = dual[v] - dist[t] + dist[v]
209     //          = dual[v] - (shortest(s, t) + dual[s] - dual[
210         //          (shortest(s, v) + dual[s] - dual[v]) = -
211         //          t]) +
212         //          shortest(s,
213         //          t) + dual[t] + shortest(s, v) = shortest(s, v
214         //          ) -
215         //          shortest(s, t) >= 0 - (n-1)C
216     dual_dist[v].first -= dual_dist[t].second - dual_dist[v
217         ].second;
218 }
219 return true;
220 };
221 Cap flow = 0;
222 Cost cost = 0, prev_cost_per_flow = -1;
223 std::vector<std::pair<Cap, Cost>> result = { {Cap(0), Cost(0)}
224 };
225 while (flow < flow_limit) {
226     if (!dual_ref()) break;
227     Cap c = flow_limit - flow;
228     for (int v = t; v != s; v = g.elist[prev_e[v]].to) {
229         c = std::min(c, g.elist[g.elist[prev_e[v]].rev].cap);
230     }
231     for (int v = t; v != s; v = g.elist[prev_e[v]].to) {
232         auto& e = g.elist[prev_e[v]];
233         e.cap += c;
234         g.elist[e.rev].cap -= c;
235     }
236     Cost d = -dual_dist[s].first;

```

```

232         flow += c;
233         cost += c * d;
234         if (prev_cost_per_flow == d) {
235             result.pop_back();
236         }
237         result.push_back({ flow, cost });
238         prev_cost_per_flow = d;
239     }
240     return result;
241 }
242 };
243 }

```

6.10 全局最小割

```

1  constexpr int N = 601;
2  constexpr int inf = 0x3f3f3f3f;
3  int edge[N][N]; // 边权存这里
4  int dis[N], vis[N], bin[N];
5  int n, m;
6  int contract(int& s, int& t) { // Find s,t
7      memset(dis, 0, sizeof(dis));
8      memset(vis, false, sizeof(vis));
9      int i, j, k, mincut, maxc;
10     for (i = 1; i <= n; i++) {
11         k = -1;
12         maxc = -1;
13         for (j = 1; j <= n; j++) {
14             if (!bin[j] && !vis[j] && dis[j] > maxc) {
15                 k = j;
16                 maxc = dis[j];
17             }
18         }
19         if (k == -1) return mincut;
20         s = t; t = k;
21         mincut = maxc;
22         vis[k] = true;
23         for (j = 1; j <= n; j++) {
24             if (!bin[j] && !vis[j]) {
25                 dis[j] += edge[k][j];
26             }
27         }
28     }

```

```

29     return mincut;
30 }
31
32 int stoerWagner() { // O(NM + N^2logN) <=> O(N^3)
33     int mincut, i, j, s, t, ans;
34     for (mincut = inf, i = 1; i < n; i++) {
35         ans = contract(s, t);
36         bin[t] = true;
37         if (mincut > ans) mincut = ans;
38         if (mincut == 0) return 0;
39         for (j = 1; j <= n; j++) {
40             if (!bin[j]) {
41                 edge[s][j] = (edge[j][s] += edge[j][t]);
42             }
43         }
44     }
45     return mincut;
46 }

```

6.11 二分图最大权匹配

```

1 namespace KM {
2     typedef long long ll;
3     const int maxn = 510;
4     const int inf = 1e9;
5     int vx[maxn], vy[maxn], lx[maxn], ly[maxn], slack[maxn];
6     int w[maxn][maxn]; // 以上为权值类型
7     int pre[maxn], left[maxn], right[maxn], NL, NR, N;
8     void match(int& u) {
9         for (; u; std::swap(u, right[pre[u]]))
10             left[u] = pre[u];
11     }
12     void bfs(int u) {
13         static int q[maxn], front, rear;
14         front = 0; vx[q[rear = 1] = u] = true;
15         while (true) {
16             while (front < rear) {
17                 int u = q[++front];
18                 for (int v = 1; v <= N; ++v) {
19                     int tmp;
20                     if (vy[v] || (tmp = lx[u] + ly[v] - w[u][v]) > slack[v])
21                         continue;
22                     pre[v] = u;

```

```

23         if (!tmp) {
24             if (!left[v]) return match(v);
25             vy[v] = vx[q[++rear] = left[v]] = true;
26         } else slack[v] = tmp;
27     }
28 }
29 int a = inf;
30 for (int i = 1; i <= N; ++i)
31     if (!vy[i] && a > slack[i]) a = slack[u = i];
32 for (int i = 1; i <= N; ++i) {
33     if (vx[i]) lx[i] -= a;
34     if (vy[i]) ly[i] += a;
35     else slack[i] -= a;
36 }
37 if (!left[u]) return match(u);
38 vy[u] = vx[q[++rear] = left[u]] = true;
39
40 }
41
42 }
43 void exec() {
44     for (int i = 1; i <= N; ++i) {
45         for (int j = 1; j <= N; ++j) {
46             slack[j] = inf;
47             vx[j] = vy[j] = false;
48         }
49         bfs(i);
50     }
51 }
52 ll work(int nl, int nr) { // NL, NR 为左右点数, 返回最大权匹配的权值和
53     NL = nl; NR = nr;
54     N = std::max(NL, NR);
55     for (int u = 1; u <= N; ++u)
56         for (int v = 1; v <= N; ++v)
57             lx[u] = std::max(lx[u], w[u][v]);
58     exec();
59     ll ans = 0;
60     for (int i = 1; i <= N; ++i)
61         ans += lx[i] + ly[i];
62     return ans;
63 }
64 void output() { // 输出左边点与右边哪个点匹配, 没有匹配输出0
65     for (int i = 1; i <= NL; ++i)

```

```

66         printf("%d ", (w[i][right[i]] ? right[i] : 0));
67     printf("\n");
68 }
69 }

```

6.12 一般图最大匹配

```

1  // UOJ79 copy from jiangly
2  #include <bits/stdc++.h>
3  struct Graph {
4      int n;
5      std::vector<std::vector<int>>> e;
6      Graph(int n) : n(n), e(n) {}
7      void addEdge(int u, int v) {
8          e[u].push_back(v);
9          e[v].push_back(u);
10     }
11     std::vector<int> findMatching() {
12         std::vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);
13         // disjoint set union
14         auto find = [&](int u) {
15             while (f[u] != u)
16                 u = f[u] = f[f[u]];
17             return u;
18         };
19         auto lca = [&](int u, int v) {
20             u = find(u);
21             v = find(v);
22             while (u != v) {
23                 if (dep[u] < dep[v])
24                     std::swap(u, v);
25                 u = find(link[match[u]]);
26             }
27             return u;
28         };
29
30         std::queue<int> que;
31         auto blossom = [&](int u, int v, int p) {
32             while (find(u) != p) {
33                 link[u] = v;
34                 v = match[u];
35                 if (vis[v] == 0) {
36                     vis[v] = 1;

```

```

37         que.push(v);
38     }
39     f[u] = f[v] = p;
40     u = link[v];
41 }
42 };
43
44 // find an augmenting path starting from u and augment (if exist)
45 auto augment = [&](int u) {
46     while (!que.empty())
47         que.pop();
48     std::iota(f.begin(), f.end(), 0);
49     // vis = 0 corresponds to inner vertices, vis = 1 corresponds to
50     // outer vertices
51     std::fill(vis.begin(), vis.end(), -1);
52     que.push(u);
53     vis[u] = 1, dep[u] = 0;
54     while (!que.empty()) {
55         int u = que.front();
56         que.pop();
57         for (auto v : e[u]) {
58             if (vis[v] == -1) {
59                 vis[v] = 0;
60                 link[v] = u;
61                 dep[v] = dep[u] + 1;
62                 // found an augmenting path
63                 if (match[v] == -1) {
64                     for (int x = v, y = u, temp; y != -1; x = temp,
65                         y = x == -1 ? -1 : link[x]) {
66                         temp = match[y];
67                         match[x] = y;
68                         match[y] = x;
69                     }
70                     return;
71                 }
72                 vis[match[v]] = 1;
73                 dep[match[v]] = dep[u] + 2;
74                 que.push(match[v]);
75             } else if (vis[v] == 1 && find(v) != find(u)) {
76                 // found a blossom
77                 int p = lca(u, v);
78                 blossom(u, v, p);
79                 blossom(v, u, p);

```

```

78         }
79     }
80 }
81 };
82
83 // find a maximal matching greedily (decrease constant)
84 auto greedy = [&]() {
85     for (int u = 0; u < n; ++u) {
86         if (match[u] != -1)
87             continue;
88         for (auto v : e[u]) {
89             if (match[v] == -1) {
90                 match[u] = v;
91                 match[v] = u;
92                 break;
93             }
94         }
95     }
96 };
97 greedy();
98 for (int u = 0; u < n; ++u)
99     if (match[u] == -1)
100         augment(u);
101 return match;
102 }
103 };
104
105 int main() {
106     std::ios::sync_with_stdio(false);
107     std::cin.tie(nullptr);
108     int n, m;
109     std::cin >> n >> m;
110     Graph g(n);
111     for (int i = 0; i < m; ++i) {
112         int u, v;
113         std::cin >> u >> v;
114         —u, —v;
115         g.addEdge(u, v);
116     }
117     auto match = g.findMatching();
118     int ans = 0;
119     for (int u = 0; u < n; ++u)
120         if (match[u] != -1)

```



```

121         ++ans;
122     std::cout << ans / 2 << "\n";
123     for (int u = 0; u < n; ++u) // 输出每个人匹配的对象，如果没有则输出0
124         std::cout << match[u] + 1 << " \n"[u == n - 1];
125     return 0;
126 }

```

6.13 最大团

```

1  /*
2  * 最大团 Bron-Kerbosch algorithm
3  * 最劣复杂度  $O(3^{(n/3)})$ 
4  * 采用位运算形式实现
5  * */
6  namespace Max_clique {
7  #define ll long long
8  #define TWOL(x) (1ll << (x))
9      const int N = 60;
10     int n, m; // 点数 边数
11     int r = 0; // 最大团大小
12     ll G[N]; // 以二进制形式存图
13     ll clique = 0; // 最大团 以二进制形式存储
14     void BronK(int S, ll P, ll X, ll R) { // 调用时参数这样设置: 0, TWOL(n)
15         -1, 0, 0
16         if (P == 0 && X == 0) {
17             if (r < S) {
18                 r = S;
19                 clique = R;
20             }
21         }
22         if (P == 0) return;
23         int u = __builtin_ctzll(P | X);
24         ll c = P & ~G[u];
25         while (c) {
26             int v = __builtin_ctzll(c);
27             ll pv = TWOL(v);
28             BronK(S + 1, P & G[v], X & G[v], R | pv);
29             P ^= pv; X |= pv; c ^= pv;
30         }
31     }
32     void init() {
33         cin >> n >> m;
34         for (int i = 0; i < m; i++) {

```

```

34         int u, v;
35         cin >> u >> v;
36         —u, —v;
37         G[u] |= TWOL(v);
38         G[v] |= TWOL(u);
39     }
40     BronK(0, TWOL(n) - 1, 0, 0);
41     cout << r << ' ' << clique << '\n';
42 }
43 }

```

7 数据结构

7.1 树状数组

```

1  template<typename T> struct fenwickTree {
2      int n, hbit;
3      vector<T> tree;
4      fenwickTree(int n_ = 0) : n(n_), tree(n_ + 1), hbit(log2(n_) + 1) {}
5      int lowbit(int x) { return x & (-x); }
6      int size() { return n; }
7      void add(int pos, int x) { // pos位置加上x
8          for (; pos <= n; pos += lowbit(pos)) {
9              tree[pos] += x;
10         }
11     }
12     T query(int pos) { // 查询pos位置的前缀和 即a[1] + a[2] + ... + a[pos]
13         T res = 0;
14         for (; pos > 0; pos -= lowbit(pos)) {
15             res += tree[pos];
16         }
17         return res;
18     }
19     T sum(int l, int r) { // [l, r]区间查询
20         return query(r) - query(l - 1);
21     }
22     int kth(int k) { // 第k大元素
23         int ans = 0, cnt = 0;
24         for (int i = hbit; i >= 0; i--) {
25             ans += (1 << i);
26             if (ans > n || cnt + tree[ans] >= k) ans -= (1 << i);
27             else cnt += tree[ans];
28         }

```

```

29         return ++ans;
30     }
31 };

```

7.2 线段树

8 字符串

8.1 KMP

```

1  namespace KMP {
2      vector<int> getPrefixTable(string s) { // 求前缀表
3          int n = s.length();
4          vector<int> nxt(n, 0);
5          for (int i = 1; i < n; i++) {
6              int j = nxt[i - 1];
7              while (j > 0 && s[i] != s[j]) {
8                  j = nxt[j - 1];
9              }
10             if (s[i] == s[j]) j++;
11             nxt[i] = j;
12         }
13         return nxt;
14     }
15
16     vector<int> kmp(string s, string t) { // 返回所有匹配位置的集合
17         int n = s.length(), m = t.length();
18         vector<int> res;
19         vector<int> nxt = getPrefixTable(t);
20         for (int i = 0, j = 0; i < n; i++) {
21             while (j > 0 && j < m && s[i] != t[j]) {
22                 j = nxt[j - 1];
23             }
24             if (s[i] == t[j]) j++;
25             if (j == m) {
26                 res.push_back(i + 1 - m);
27                 j = nxt[m - 1];
28             }
29         }
30         return res;
31     }
32 }

```

8.2 Z-Function

```

1 // O(N) 查询字符串s每一位开始的LCP
2 vector<int> z_function(string s) {
3     int n = (int)s.length();
4     vector<int> z(n);
5     for (int i = 1, l = 0, r = 0; i < n; ++i) {
6         if (i <= r && z[i - 1] < r - i + 1) {
7             z[i] = z[i - 1];
8         } else {
9             z[i] = max(0, r - i + 1);
10            while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
11        }
12        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
13    }
14    return z;
15 }

```

8.3 Manacher

```

1 namespace Manacher {
2     static constexpr int SIZE = 1e5 + 5; // 预设原串长度
3     int len = 1; // manacher 预处理后字符串的长度
4     char stk[SIZE << 1]; // manacher 预处理字符串 需要2倍空间+1
5     void init(string s) { // 初始化stk
6         stk[0] = '*'; len = 1;
7         for (int i = 0; i < s.length(); ++i) {
8             stk[len++] = s[i];
9             stk[len++] = '*';
10        }
11    }
12    int manacher() { // 返回最长回文子串长度
13        vector<int> rad(len << 1); // 存储每个点作为对称中心可拓展的最大半径
14        int md = 0; // 最远回文串对称中心下标
15        for (int i = 1; i < len; ++i) {
16            int& r = rad[i] = 0;
17            if (i <= md + rad[md]) {
18                r = min(rad[2 * md - i], md + rad[md] - i);
19            }
20            while (i - r - 1 >= 0 && i + r + 1 < len &&
21                stk[i - r - 1] == stk[i + r + 1]) ++r;
22            if (i + r >= md + rad[md]) md = i;
23        }
24        int res = 0;

```

```

25     for (int i = 0; i < len; ++i) {
26         if (rad[i] > res) {
27             res = rad[i];
28         }
29     }
30     return res;
31 }
32 }

```

8.4 Trie

```

1 struct trie {
2     int cnt;
3     vector<vector<int>> nxt;
4     vector<bool> vis;
5     /* 初始化的时候size需要设置为字符串总长之和 26是字符集大小 */
6     trie(int size_ = 0) : cnt(0), vis(size_, false), nxt(size_, vector<int>
7         >(26, 0)) {}
8     void insert(string s) { // 插入字符串
9         int p = 0;
10        for (int i = 0; i < (int)s.length(); i++) {
11            int c = s[i] - 'a';
12            if (!nxt[p][c]) nxt[p][c] = ++cnt;
13            p = nxt[p][c];
14        }
15        vis[p] = true;
16    }
17    bool find(string s) { // 查找字符串
18        int p = 0;
19        for (int i = 0; i < (int)s.length(); i++) {
20            int c = s[i] - 'a';
21            if (!nxt[p][c]) return false;
22            p = nxt[p][c];
23        }
24        return vis[p];
25    }
26 };

```

8.5 01-Trie

```

1 template<typename T> struct xorTrie {
2     int HIGHBIT, cnt;
3     vector<vector<int>> nxt;

```

```

4     vector<bool> vis;
5     xorTrie(int n_ = 0, int highbit_ = 30) : HIGHBIT(highbit_), cnt(0) {
6         int size_ = upperBoundEstimate(n_);
7         nxt.resize(size_, vector<int>(2, 0));
8         vis.resize(size_, false);
9     }
10    int upperBoundEstimate(int n) { // 求内存上界
11        int hbit = log2(n);
12        return n * (HIGHBIT - hbit + 1) + (1 << (hbit + 1)) - 1;
13    }
14    void insert(T x) { // 插入
15        int p = 0;
16        for (int i = HIGHBIT; ~i; i--) {
17            int s = ((x >> i) & 1);
18            if (!nxt[p][s]) nxt[p][s] = ++cnt;
19            p = nxt[p][s];
20        }
21        vis[p] = true;
22    }
23    bool find(T x) { // 查询
24        int p = 0;
25        for (int i = HIGHBIT; ~i; i--) {
26            int s = ((x >> i) & 1);
27            if (!nxt[p][s]) return false;
28            p = nxt[p][s];
29        }
30        return vis[p];
31    }
32 };

```

9 计算几何

```

1 namespace Geometry {
2 #define db long double
3 #define pi acos(-1.0)
4     constexpr db eps = 1e-7;
5     int sign(db k) {
6         if (k > eps) return 1;
7         else if (k < -eps) return -1;
8         return 0;
9     }
10    int cmp(db k1, db k2) { // k1 < k2 : -1, k1 == k2 : 0, k1 > k2 : 1
11        return sign(k1 - k2);
12    }
13 }

```

```

12     }
13     int inmid(db k1, db k2, db k3) { // k3 在 [k1, k2] 内
14         return sign(k1 - k3) * sign(k2 - k3) <= 0;
15     }
16
17     struct point { // 点类
18         db x, y;
19         point() {}
20         point(db x_, db y_) :x(x_), y(y_) {}
21         point operator + (const point& k) const { return point(k.x + x, k.y
22             + y); }
23         point operator - (const point& k) const { return point(x - k.x, y -
24             k.y); }
25         point operator * (db k) const { return point(x * k, y * k); }
26         point operator / (db k1) const { return point(x / k1, y / k1); }
27         point turn(db k1) { return point(x * cos(k1) - y * sin(k1), x * sin(
28             k1) + y * cos(k1)); } // 逆时针旋转
29         point turn90() { return point(-y, x); } // 逆时针方向旋转 90 度
30         db len() { return sqrt(x * x + y * y); } // 向量长度
31         db len2() { return x * x + y * y; } // 向量长度的平方
32         db getPolarAngle() { return atan2(y, x); } // 向量极角
33         db dis(point k) { return ((*this) - k).len(); } // 到点k的距离
34         point unit() { db d = len(); return point(x / d, y / d); } // 单位向
35             量
36         point getdel() { // 将向量的方向调整为指向第一/四象限 包括y轴正方向
37             if (sign(x) == -1 || (sign(x) == 0 && sign(y) == -1))
38                 return (*this) * (-1);
39             else return (*this);
40         }
41         bool operator < (const point& k) const { // 水平序排序 x坐标为第一关
42             键字,y坐标第二关键字
43             return x == k.x ? y < k.y : x < k.x;
44         }
45         bool operator == (const point& k) const { return cmp(x, k.x) == 0 &&
46             cmp(y, k.y) == 0; }
47         bool getP() const { // 判断点是否在上半平面 含x负半轴 不含x正半轴及
48             零点
49             return sign(y) == 1 || (sign(y) == 0 && sign(x) == -1);
50         }
51         void input() { cin >> x >> y; }
52     };
53     db cross(point k1, point k2) { return k1.x * k2.y - k1.y * k2.x; } // 向
54         量 k1,k2 的 叉积

```

```

47 db dot(point k1, point k2) { return k1.x * k2.x + k1.y * k2.y; } // 向
    量 k1,k2 的点积
48 db rad(point k1, point k2) { // 向量 k1,k2 之间的有向夹角
49     return atan2(cross(k1, k2), dot(k1, k2));
50 }
51 int inmid(point k1, point k2, point k3) { // k1 k2 k3共线时 判断点 k3 是
    否在线段 k1k2 上
52     return inmid(k1.x, k2.x, k3.x) && inmid(k1.y, k2.y, k3.y);
53 }
54 int compareAngle(point k1, point k2) { // 比较向量 k1,k2 的角度大小 角度
    按照atan2()函数定义
55     // k1 < k2 返回 1, k1 >= k2 返回 0
56     return k1.getP() < k2.getP() || (k1.getP() == k2.getP() && sign(
        cross(k1, k2)) > 0);
57 }
58 point proj(point k1, point k2, point q) { // q 到直线 k1,k2 的投影
59     point k = k2 - k1; return k1 + k * (dot(q - k1, k) / k.len2());
60 }
61 point reflect(point k1, point k2, point q) { return proj(k1, k2, q) * 2
    - q; } // q 关于直线 k1,k2 的对称点
62 int counterclockwise(point k1, point k2, point k3) { // k1 k2 k3 逆时针1
    顺时针-1 否则0
63     return sign(cross(k2 - k1, k3 - k1));
64 }
65 int checkLL(point k1, point k2, point k3, point k4) { // 判断直线 k1k2
    和直线k3k4 是否相交
66     // 即判断直线 k1k2 和 k3k4 是否平行 平行返回0 不平行返回1
67     return sign(cross(k2 - k1, k4 - k3)) != 0;
68 }
69 point getLL(point k1, point k2, point k3, point k4) { // 求 k1k2 k3k4 两
    直线交点
70     db w1 = cross(k1 - k3, k4 - k3), w2 = cross(k4 - k3, k2 - k3);
71     return (k1 * w2 + k2 * w1) / (w1 + w2);
72 }
73 int intersect(db l1, db r1, db l2, db r2) { // 判断 [l1,r1] 和 [l2, r2]
    是否相交
74     if (l1 > r1) swap(l1, r1);
75     if (l2 > r2) swap(l2, r2);
76     return cmp(r1, l2) != -1 && cmp(r2, l1) != -1;
77 }
78 int checkSS(point k1, point k2, point k3, point k4) { // 判断线段 k1k2
    和线段 k3k4 是否相交
79     return intersect(k1.x, k2.x, k3.x, k4.x) && intersect(k1.y, k2.y, k3

```



```

        .y, k4.y) &&
80         sign(cross(k3 - k1, k4 - k1)) * sign(cross(k3 - k2, k4 - k2)) <=
            0 &&
81         sign(cross(k1 - k3, k2 - k3)) * sign(cross(k1 - k4, k2 - k4)) <=
            0;
82     }
83     db disSP(point k1, point k2, point q) { // 点 q 到线段 k1k2 的最短距离
84         point k3 = proj(k1, k2, q);
85         if (inmid(k1, k2, k3)) return q.dis(k3);
86         else return min(q.dis(k1), q.dis(k2));
87     }
88     db disLP(point k1, point k2, point q) { // 点 q 到直线 k1k2 的最短距离
89         point k3 = proj(k1, k2, q);
90         return q.dis(k3);
91     }
92     db disSS(point k1, point k2, point k3, point k4) { // 线段 k1k2 和线段
        k3k4 的最短距离
93         if (checkSS(k1, k2, k3, k4)) return 0;
94         else return min(min(disSP(k1, k2, k3), disSP(k1, k2, k4)),
95             min(disSP(k3, k4, k1), disSP(k3, k4, k2)));
96     }
97     bool onLine(point k1, point k2, point q) { // 判断点 q 是否在直线 k1k2
        上
98         return sign(cross(k1 - q, k2 - q)) == 0;
99     }
100    bool onSegment(point k1, point k2, point q) { // 判断点 q 是否在线段
        k1k2 上
101        if (!onLine(k1, k2, q)) return false; // 如果确定共线 要删除这个特判
102        return inmid(k1, k2, q);
103    }
104    void polarAngleSort(vector<point>& p, point t) { // p为待排序点集 t为极
        角排序中心
105        sort(p.begin(), p.end(), [&](const point& k1, const point& k2) {
106            return compareAngle(k1 - t, k2 - t);
107        });
108    }
109
110    struct line { // 直线 / 线段类
111        point p[2];
112        line() {}
113        line(point k1, point k2) { p[0] = k1, p[1] = k2; }
114        point& operator [] (int k) { return p[k]; }
115        point dir() { return p[1] - p[0]; } // 向量 p[0] -> p[1]

```

```

116     bool include(point k) { // 判断点是否在直线上
117         return sign(cross(p[1] - p[0], k - p[0])) > 0;
118     }
119     bool includeS(point k) { // 判断点是否在线段上
120         return onSegment(p[0], p[1], k);
121     }
122     line push(db len) { // 向外（左手边）平移 len 个单位
123         point delta = (p[1] - p[0]).turn90().unit() * len;
124         return line(p[0] - delta, p[1] - delta);
125     }
126 };
127 bool parallel(line k1, line k2) { // 判断是否平行
128     return sign(cross(k1.dir(), k2.dir())) == 0;
129 }
130 bool sameLine(line k1, line k2) { // 判断是否共线
131     return parallel(k1, k2) && parallel(k1, line(k2.p[0], k1.p[0]));
132 }
133 bool sameDir(line k1, line k2) { // 判断向量 k1 k2 是否同向
134     return parallel(k1, k2) && sign(dot(k1.dir(), k2.dir())) == 1;
135 }
136 bool operator < (line k1, line k2) {
137     if (sameDir(k1, k2)) return k2.include(k1[0]);
138     return compareAngle(k1.dir(), k2.dir());
139 }
140 bool checkLL(line k1, line k2) {
141     return checkLL(k1[0], k1[1], k2[0], k2[1]);
142 }
143 point getLL(line k1, line k2) { // 求 k1 k2 两直线交点 不要忘了判平行!
144     return getLL(k1[0], k1[1], k2[0], k2[1]);
145 }
146 bool checkpos(line k1, line k2, line k3) { // 判断是否三线共点
147     return k3.include(getLL(k1, k2));
148 }
149
150 struct circle { // 圆类
151     point o;
152     double r;
153     circle() {}
154     circle(point o_, double r_) : o(o_), r(r_) {}
155     int inside(point k) { // 判断点 k 和圆的位置关系
156         return cmp(r, o.dis(k)); // 圆外:-1, 圆上:0, 圆内:1
157     }
158 };

```

```

159     int checkposCC(circle k1, circle k2) { // 返回两个圆的公切线数量
160         if (cmp(k1.r, k2.r) == -1) swap(k1, k2);
161         db dis = k1.o.dis(k2.o);
162         int w1 = cmp(dis, k1.r + k2.r), w2 = cmp(dis, k1.r - k2.r);
163         if (w1 > 0) return 4; // 外离
164         else if (w1 == 0) return 3; // 外切
165         else if (w2 > 0) return 2; // 相交
166         else if (w2 == 0) return 1; // 内切
167         else return 0; // 内离(包含)
168     }
169     vector<point> getCL(circle k1, point k2, point k3) { // 求直线 k2k3 和圆
        k1 的交点
        // 沿着 k2->k3 方向给出 相切给出两个
170         point k = proj(k2, k3, k1.o);
171         db d = k1.r * k1.r - (k - k1.o).len2();
172         if (sign(d) == -1) return {};
173         point del = (k3 - k2).unit() * sqrt(max((db)0.0, d));
174         return { k - del, k + del };
175     }
176 }
177     vector<point> getCC(circle k1, circle k2) { // 求圆 k1 和圆 k2 的交点
178         // 沿圆 k1 逆时针给出, 相切给出两个
179         int pd = checkposCC(k1, k2); if (pd == 0 || pd == 4) return {};
180         db a = (k2.o - k1.o).len2(), cosA = (k1.r * k1.r + a -
181             k2.r * k2.r) / (2 * k1.r * sqrt(max(a, (db)0.0)));
182         db b = k1.r * cosA, c = sqrt(max((db)0.0, k1.r * k1.r - b * b));
183         point k = (k2.o - k1.o).unit(), m = k1.o + k * b, del = k.turn90() *
            c;
184         return { m - del, m + del };
185     }
186     vector<point> tangentCP(circle k1, point k2) { // 点 k2 到圆 k1 的切点
        沿圆 k1 逆时针给出
187         db a = (k2 - k1.o).len(), b = k1.r * k1.r / a, c = sqrt(max((db)0.0,
            k1.r * k1.r - b * b));
188         point k = (k2 - k1.o).unit(), m = k1.o + k * b, del = k.turn90() * c
            ;
189         return { m - del, m + del };
190     }
191     vector<line> tangentOutCC(circle k1, circle k2) {
192         int pd = checkposCC(k1, k2);
193         if (pd == 0) return {};
194         if (pd == 1) {
195             point k = getCC(k1, k2)[0];
196             return { line(k, k) };

```

```

197     }
198     if (cmp(k1.r, k2.r) == 0) {
199         point del = (k2.o - k1.o).unit().turn90().getdel();
200         return { line(k1.o - del * k1.r, k2.o - del * k2.r),
201                 line(k1.o + del * k1.r, k2.o + del * k2.r) };
202     } else {
203         point p = (k2.o * k1.r - k1.o * k2.r) / (k1.r - k2.r);
204         vector<point> A = tangentCP(k1, p), B = tangentCP(k2, p);
205         vector<line> ans; for (int i = 0; i < A.size(); i++)
206             ans.push_back(line(A[i], B[i]));
207         return ans;
208     }
209 }
210 vector<line> tangentInCC(circle k1, circle k2) {
211     int pd = checkposCC(k1, k2);
212     if (pd <= 2) return {};
213     if (pd == 3) {
214         point k = getCC(k1, k2)[0];
215         return { line(k, k) };
216     }
217     point p = (k2.o * k1.r + k1.o * k2.r) / (k1.r + k2.r);
218     vector<point> A = tangentCP(k1, p), B = tangentCP(k2, p);
219     vector<line> ans;
220     for (int i = 0; i < (int)A.size(); i++) ans.push_back(line(A[i], B[i]
221     ));
222     return ans;
223 }
224 vector<line> tangentCC(circle k1, circle k2) { // 求两圆公切线
225     int flag = 0;
226     if (k1.r < k2.r) swap(k1, k2), flag = 1;
227     vector<line> A = tangentOutCC(k1, k2), B = tangentInCC(k1, k2);
228     for (line k : B) A.push_back(k);
229     if (flag) for (line& k : A) swap(k[0], k[1]);
230     return A;
231 }
232 db getAreaCT(circle k1, point k2, point k3) { // 圆 k1 与三角形 k2k3k1.o
233     的有向面积交
234     point k = k1.o; k1.o = k1.o - k; k2 = k2 - k; k3 = k3 - k;
235     int pd1 = k1.inside(k2), pd2 = k1.inside(k3);
236     vector<point> A = getCL(k1, k2, k3);
237     if (pd1 >= 0) {
238         if (pd2 >= 0) return cross(k2, k3) / 2;
239         return k1.r * k1.r * rad(A[1], k3) / 2 + cross(k2, A[1]) / 2;

```

```

238     } else if (pd2 >= 0) {
239         return k1.r * k1.r * rad(k2, A[0]) / 2 + cross(A[0], k3) / 2;
240     } else {
241         int pd = cmp(k1.r, disSP(k2, k3, k1.o));
242         if (pd <= 0) return k1.r * k1.r * rad(k2, k3) / 2;
243         return cross(A[0], A[1]) / 2 + k1.r * k1.r * (rad(k2, A[0]) +
            rad(A[1], k3)) / 2;
244     }
245 }
246 db getAreaCC(circle k1, circle k2) { // 两圆面积交
247     db d = k1.o.dis(k2.o);
248     if (cmp(d, k1.r + k2.r) >= 0) return 0; // 两圆相离
249     if (cmp(k1.r, k2.r) == -1) swap(k1, k2);
250     if (cmp(k1.r - k2.r, d) >= 0) return pi * k2.r * k2.r; // 圆k1包含k2
251     db g1 = acos((k1.r * k1.r + d * d - k2.r * k2.r) / (2 * k1.r * d));
252     db g2 = acos((k2.r * k2.r + d * d - k1.r * k1.r) / (2 * k2.r * d));
253     return g1 * k1.r * k1.r + g2 * k2.r * k2.r - k1.r * d * sin(g1);
254 }
255 circle getCircleOut(point k1, point k2, point k3) { // 三角形外接圆
256     db a1 = k2.x - k1.x, b1 = k2.y - k1.y, c1 = (a1 * a1 + b1 * b1) / 2;
257     db a2 = k3.x - k1.x, b2 = k3.y - k1.y, c2 = (a2 * a2 + b2 * b2) / 2;
258     db d = a1 * b2 - a2 * b1;
259     point o((k1.x + (c1 * b2 - c2 * b1) / d, k1.y + (a1 * c2 - a2 * c1) /
        d);
260     return circle(o, k1.dis(o));
261 }
262 circle getCircleIn(point k1, point k2, point k3) { // 三角形内切圆
263     db a = k1.dis(k2), b = k2.dis(k3), c = k3.dis(k1);
264     db len = a + b + c;
265     db r = abs(cross(k1 - k2, k1 - k3)) / len;
266     point o((k1.x * b + k2.x * c + k3.x * a) / len, (k1.y * b + k2.y * c
        + k3.y * a) / len);
267     return circle(o, r);
268 }
269 circle minCircleCovering(vector<point> A) { // 最小圆覆盖 O(n)随机增量法
270     // random_shuffle(A.begin(), A.end()); // <= C++14
271     auto seed = chrono::steady_clock::now().time_since_epoch().count();
272     default_random_engine e(seed);
273     shuffle(A.begin(), A.end(), e); // >= C++11
274     circle ans = circle(A[0], 0);
275     for (int i = 1; i < A.size(); i++) {
276         if (ans.inside(A[i]) == -1) {
277             ans = circle(A[i], 0);

```

```

278         for (int j = 0; j < i; j++) {
279             if (ans.inside(A[j]) == -1) {
280                 ans.o = (A[i] + A[j]) / 2;
281                 ans.r = ans.o.dis(A[i]);
282                 for (int k = 0; k < j; k++) {
283                     if (ans.inside(A[k]) == -1)
284                         ans = getCircleOut(A[i], A[j], A[k]);
285                 }
286             }
287         }
288     }
289 }
290 return ans;
291 }
292
293 typedef vector<point> polygon;
294 db area(polygon p) { // 多边形有向面积
295     if (p.size() < 3) return 0;
296     db ans = 0;
297     for (int i = 1; i < p.size() - 1; i++)
298         ans += cross(p[i] - p[0], p[i + 1] - p[0]);
299     return 0.5L * ans;
300 }
301
302 int checkConvexP(polygon p, point a) { // O(logn)判断点是否在凸包内 2内部 1边界 0外部
303     // 必须保证凸多边形是一个水平序凸包且不能退化
304     // 退化情况 比如凸包退化成线段 可使用 onSegment() 函数特判
305     auto check = [&](int x) {
306         int ccw1 = counterclockwise(p[0], a, p[x]),
307             ccw2 = counterclockwise(p[0], a, p[x + 1]);
308         if (ccw1 == -1 && ccw2 == -1) return 2;
309         else if (ccw1 == 1 && ccw2 == 1) return 0;
310         else if (ccw1 == -1 && ccw2 == 1) return 1;
311         else return 1;
312     };
313     if (counterclockwise(p[0], a, p[1]) <= 0 && counterclockwise(p[0], a,
314         p.back()) >= 0) {
315         int l = 1, r = p.size() - 2, mid;
316         while (l <= r) {
317             mid = (l + r) >> 1;
318             int chk = check(mid);
319             if (chk == 1) l = mid + 1;

```

```

319         else if (chk == -1) r = mid;
320         else break;
321     }
322     int res = counterclockwise(p[mid], a, p[mid + 1]);
323     if (res < 0) return 2;
324     else if (res == 0) return 1;
325     else return 0;
326 } else {
327     return 0;
328 }
329 }
330 int checkPolyP(vector<point> p, point q) { // O(n)判断点是否在一般多边形
    内
331     // 必须保证简单多边形的点按逆时针给出 返回 2 内部 1 边界 0 外部
332     int pd = 0, n = p.size();
333     for (int i = 0; i < n; i++) {
334         point u = p[i], v = p[(i + 1) % n];
335         if (onSegment(u, v, q)) return 1;
336         if (cmp(u.y, v.y) > 0) swap(u, v);
337         if (cmp(u.y, q.y) >= 0 || cmp(v.y, q.y) < 0) continue;
338         if (sign(cross(u - v, q - v)) < 0) pd ^= 1;
339     }
340     return pd << 1;
341 }
342 db convexDiameter(polygon p) { // 0(n)旋转卡壳求凸包直径 / 平面最远点对
    的平方
343     int n = p.size(); // 请保证多边形是凸包
344     db ans = 0;
345     for (int i = 0, j = n < 2 ? 0 : 1; i < j; i++) {
346         for (; j = (j + 1) % n) {
347             ans = max(ans, (p[i] - p[j]).len2());
348             if (sign(cross(p[i + 1] - p[i], p[(j + 1) % n] - p[j])) <=
                0) break;
349         }
350     }
351     return ans;
352 }
353 polygon convexHull(polygon A, int flag = 1) { // 凸包 flag=0 不严格 flag
    =1 严格
354     int n = A.size(); polygon ans(n + n);
355     sort(A.begin(), A.end()); int now = -1;
356     for (int i = 0; i < A.size(); i++) {
357         while (now > 0 && sign(cross(ans[now] - ans[now - 1], A[i] - ans

```

```

        [now - 1])) < flag)
358         now--;
359         ans[++now] = A[i];
360     }
361     int pre = now;
362     for (int i = n - 2; i >= 0; i--) {
363         while (now > pre && sign(cross(ans[now] - ans[now - 1], A[i] -
            ans[now - 1])) < flag)
364             now--;
365         ans[++now] = A[i];
366     }
367     ans.resize(now);
368     return ans;
369 }
370 bool checkConvexHull(polygon p) { // 检测多边形是否是凸包 (可以有三点共
    线)
371     int sgn, n = p.size(), i = 0; // 如果三点共线不算凸包 去掉ccw=0的情
    况
372     for (; i++) { // 这一步是为了防止第一步遇到共线的三个点
373         sgn = counterclockwise(p[i], p[(i + 1) % n], p[(i + 2) % n]);
374         if (sgn) break;
375     }
376     for (; i < n; i++) {
377         int ccw = counterclockwise(p[i], p[(i + 1) % n], p[(i + 2) % n])
            ;
378         if (ccw && ccw != sgn) {
379             return false;
380         }
381     }
382     return true;
383 }
384 polygon convexCut(polygon A, point k1, point k2) { // 半平面 k1k2 切凸包
    A
385     int n = A.size(); // 保留所有满足 k1 -> p -> k2 为逆时针方向的点
386     A.push_back(A[0]); // 保留的点可能有重点
387     polygon ans;
388     line cut(k1, k2);
389     for (int i = 0; i < n; i++) {
390         int ccw1 = counterclockwise(k1, k2, A[i]);
391         int ccw2 = counterclockwise(k1, k2, A[i + 1]);
392         if (ccw1 >= 0) ans.push_back(A[i]);
393         if (ccw1 * ccw2 <= 0) {
394             if (sameLine(cut, line(A[i], A[i + 1]))) { // 半平面恰好切到

```



```

        凸包上某条边
395         ans.push_back(A[i]);
396         ans.push_back(A[i + 1]);
397     } else {
398         ans.push_back(getLL(k1, k2, A[i], A[i + 1]));
399     }
400 }
401 }
402 return ans;
403 }
404
405 vector<line> getHL(vector<line>& L) { // 求半平面交 逆时针方向存储
406     sort(L.begin(), L.end());
407     deque<line> q;
408     for (int i = 0; i < (int)L.size(); ++i) {
409         if (i && sameDir(L[i], L[i - 1])) continue;
410         while (q.size() > 1 && !checkpos(q[q.size() - 2], q[q.size() - 1], L[i])) q.pop_back();
411         while (q.size() > 1 && !checkpos(q[1], q[0], L[i])) q.pop_front();
412         q.push_back(L[i]);
413     }
414     while (q.size() > 2 && !checkpos(q[q.size() - 2], q[q.size() - 1], q[0])) q.pop_back();
415     while (q.size() > 2 && !checkpos(q[1], q[0], q[q.size() - 1])) q.pop_front();
416     vector<line> ans;
417     for (int i = 0; i < q.size(); ++i) ans.push_back(q[i]);
418     return ans;
419 }
420
421 db closestPoint(vector<point>& A, int l, int r) { // 最近点对，先要按照
422     x 坐标排序
423     if (r - l <= 5) {
424         db ans = 1e20;
425         for (int i = l; i <= r; ++i)
426             for (int j = i + 1; j <= r; j++)
427                 ans = min(ans, A[i].dis(A[j]));
428         return ans;
429     }
430     int mid = l + r >> 1;
431     db ans = min(closestPoint(A, l, mid), closestPoint(A, mid + 1, r));
    vector<point> B;

```

```

432     for (int i = l; i <= r; i++)
433         if (abs(A[i].x - A[mid].x) <= ans)
434             B.push_back(A[i]);
435     sort(B.begin(), B.end(), [&](const point& k1, const point& k2) {
436         return k1.y < k2.y;
437     });
438     for (int i = 0; i < B.size(); i++)
439         for (int j = i + 1; j < B.size() && B[j].y - B[i].y < ans; j++)
440             ans = min(ans, B[i].dis(B[j]));
441     return ans;
442 }
443 }
444 using namespace Geometry;

```

10 杂项

10.1 快速 IO

```

1  // fast IO by yosupo
2  struct Scanner {
3      FILE* fp = nullptr;
4      char line[(1 << 15) + 1];
5      size_t st = 0, ed = 0;
6      void reread() {
7          memmove(line, line + st, ed - st);
8          ed -= st;
9          st = 0;
10         ed += fread(line + ed, 1, (1 << 15) - ed, fp);
11         line[ed] = '\0';
12     }
13     bool succ() {
14         while (true) {
15             if (st == ed) {
16                 reread();
17                 if (st == ed) return false;
18             }
19             while (st != ed && isspace(line[st])) st++;
20             if (st != ed) break;
21         }
22         if (ed - st <= 50) reread();
23         return true;
24     }
25     template <class T, enable_if_t<is_same<T, string>::value, int> = 0>

```

```

26     bool read_single(T& ref) {
27         if (!succ()) return false;
28         while (true) {
29             size_t sz = 0;
30             while (st + sz < ed && !isspace(line[st + sz])) sz++;
31             ref.append(line + st, sz);
32             st += sz;
33             if (!sz || st != ed) break;
34             reread();
35         }
36         return true;
37     }
38     template <class T, enable_if_t<is_integral<T>::value, int> = 0>
39     bool read_single(T& ref) {
40         if (!succ()) return false;
41         bool neg = false;
42         if (line[st] == '-') {
43             neg = true;
44             st++;
45         }
46         ref = T(0);
47         while (isdigit(line[st])) {
48             ref = 10 * ref + (line[st++] - '0');
49         }
50         if (neg) ref = -ref;
51         return true;
52     }
53     template <class T> bool read_single(vector<T>& ref) {
54         for (auto& d : ref) {
55             if (!read_single(d)) return false;
56         }
57         return true;
58     }
59     void read() {}
60     template <class H, class... T> void read(H& h, T&... t) {
61         bool f = read_single(h);
62         assert(f);
63         read(t...);
64     }
65     Scanner(FILE* _fp) : fp(_fp) {}
66 };
67
68 struct Printer {

```

```

69 public:
70     template <bool F = false> void write() {}
71     template <bool F = false, class H, class... T>
72     void write(const H& h, const T&... t) {
73         if (F) write_single(' ');
74         write_single(h);
75         write<true>(t...);
76     }
77     template <class... T> void writeln(const T&... t) {
78         write(t...);
79         write_single('\n');
80     }
81
82     Printer(FILE* _fp) : fp(_fp) {}
83     ~Printer() { flush(); }
84
85 private:
86     static constexpr size_t SIZE = 1 << 15;
87     FILE* fp;
88     char line[SIZE], small[50];
89     size_t pos = 0;
90     void flush() {
91         fwrite(line, 1, pos, fp);
92         pos = 0;
93     }
94     void write_single(const char& val) {
95         if (pos == SIZE) flush();
96         line[pos++] = val;
97     }
98     template <class T, enable_if_t<is_integral<T>::value, int> = 0>
99     void write_single(T val) {
100         if (pos > (1 << 15) - 50) flush();
101         if (val == 0) {
102             write_single('0');
103             return;
104         }
105         if (val < 0) {
106             write_single('-');
107             val = -val; // todo min
108         }
109         size_t len = 0;
110         while (val) {
111             small[len++] = char('0' + (val % 10));

```

```

112         val /= 10;
113     }
114     for (size_t i = 0; i < len; i++) {
115         line[pos + i] = small[len - 1 - i];
116     }
117     pos += len;
118 }
119 void write_single(const string& s) {
120     for (char c : s) write_single(c);
121 }
122 void write_single(const char* s) {
123     size_t len = strlen(s);
124     for (size_t i = 0; i < len; i++) write_single(s[i]);
125 }
126 template <class T> void write_single(const vector<T>& val) {
127     auto n = val.size();
128     for (size_t i = 0; i < n; i++) {
129         if (i) write_single(' ');
130         write_single(val[i]);
131     }
132 }
133 void write_single(long double d) {
134     {
135         long long v = d;
136         write_single(v);
137         d -= v;
138     }
139     write_single('.');
140     for (int _ = 0; _ < 8; _++) {
141         d *= 10;
142         long long v = d;
143         write_single(v);
144         d -= v;
145     }
146 }
147 };
148
149 Scanner sc(stdin);
150 Printer pr(stdout);

```

10.2 蔡勒公式

```

1 int zeller(int y, int m, int d) { // 蔡勒公式 返回星期几

```

```

2      if (m <= 2) y--, m += 12;
3      int c = y / 100; y %= 100;
4      int w = ((c >> 2) - (c << 1) + y + (y >> 2) +
5              (13 * (m + 1) / 5) + d - 1) % 7;
6      if (w < 0) w += 7;
7      return (w);
8  }
9  int getId(int y, int m, int d) { // 返回到公元1年1月1日的天数
10     if (m < 3) { y--; m += 12; }
11     return 365 * y + y / 4 - y / 100 + y / 400 +
12           (153 * (m - 3) + 2) / 5 + d - 307;
13 }

```

10.3 枚举子集

10.3.1 暴力遍历

```

1  /* O(3^n) 遍历子集 */
2  for (int i = 0; i < (1 << n); i++) { // i是当前需要遍历的全集
3      for (int l = i;; l = i & (l - 1)) { // l是i的子集
4          int r = i - l; // l+r=i
5          if (!l) break;
6      }
7  }

```

10.3.2 遍历大小为 k 的子集

```

1  int n, k;
2  int s = (1 << k) - 1;
3  while (s < (1 << n)) { // O(binom(n, k))
4      // 每次取出s就是一个大小为k的子集
5      int x = s & -s, y = s + x;
6      s = (((s & ~y) / x) >> 1) | y;
7  }

```

10.4 高维前缀和/SoSDP

```

1  /* 高维前缀和/子集前缀和 */
2  for (int i = 0; i < n; i++) { // O(n2^n)
3      for (int j = 0; j < (1 << n); j++) {
4          if (j & (1 << i)) {
5              pre[j] += pre[j ^ (1 << i)];
6          }
7      }
8  }

```

```

7     }
8 }

```

10.5 压位 BFS

给定一个 n 个点的有向图，当所有边权均为 1 时， $O(\frac{n^3}{w})$ 求任意两点之间的最短路。

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int SIZE = 1001;
4 bitset<SIZE> g[SIZE], vis, now;
5 int dis[SIZE][SIZE];
6
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(nullptr);
10    cout.tie(nullptr);
11    memset(dis, -1, sizeof(dis));
12    int n;
13    cin >> n;
14    for (int i = 1; i <= n; i++) {
15        for (int j = 1; j <= n; j++) {
16            int x;
17            cin >> x;
18            if (x) g[i].set(j);
19        }
20    }
21    for (int i = 1; i <= n; i++) {
22        vis.reset(); // 清空已遍历的数组
23        vis.set(i);
24        queue<int> q;
25        q.push(i);
26        dis[i][i] = 0;
27        while (!q.empty()) {
28            auto top = q.front();
29            q.pop();
30            now = g[top] ^ (g[top] & vis); // 去掉已经遍历到的节点
31            // 本方法的关键: O(n/w) 遍历 bitset
32            for (int to = now._Find_first(); to != now.size(); to = now._Find_next(to)) {
33                dis[i][to] = dis[i][top] + 1;
34                q.push(to);
35            }
36            vis |= now; // 更新已遍历的节点

```

```

37     }
38 }
39 for (int i = 1; i <= n; i++) {
40     for (int j = 1; j <= n; j++) {
41         cout << dis[i][j] << " \n"[j == n];
42     }
43 }
44 return 0;
45 }

```

10.6 随机数生成

```

1 int rd(int l, int r) {
2     mt19937_64 gen(chrono::steady_clock::now().time_since_epoch().count());
3     int p = uniform_int_distribution<int>(l, r)(gen);
4     return p;
5 }

```

10.7 简单对拍

```

1  /*
2   * gen.exe是数据生成器
3   * a.exe 和 std.exe 是对拍程序和标程
4   */
5  while (1) {
6      system("gen.exe > in.txt");
7      system("a.exe < in.txt > a.out");
8      system("std.exe < in.txt > std.out");
9      if (system("fc a.out std.out")) {
10         break;
11     }
12 }

```