

## Pollard

## Team Reference

```
1  // pollard begins
2
3  const int max_step = 4e5;
4
5  unsigned long long gcd(unsigned long long a,
6  ↪ unsigned long long b){
7      if (!a) return 1;
8      while (a) swap(a, b%=a);
9      return b;
10 }
11
12 unsigned long long get(unsigned long long a,
13 ↪ unsigned long long b){
14     if (a > b)
15         return a-b;
16     else
17         return b-a;
18 }
19
20 unsigned long long pollard(unsigned long long n){
21     unsigned long long x = (rand() + 1) % n, y =
22     ↪ 1, g;
23     int stage = 2, i = 0;
24     g = gcd(get(x, y), n);
25     while (g == 1) {
26         if (i == max_step)
27             break;
28         if (i == stage) {
29             y = x;
30             stage <<= 1;
31         }
32         x = (x * (__int128)x + 1) % n;
33         i++;
34         g = gcd(get(x, y), n);
35     }
36     return g;
37 }
38
39 // pollard ends
```

**pragma**

```
#pragma GCC optimize('O3,no-stack-protector')
#pragma GCC target('sse,sse2,sse4,ssse3,popcnt,abm,mmx,avx,tune=native')
```

- prime:  $13631489 = 13 \cdot 2^{20} + 1; w : 3(w^{2^{20}} = 1)$

**Алгебра Pick**

$B + \Gamma / 2 - 1 = \text{AREA}$ ,  
где  $B$  — количество целочисленных точек внутри многоугольника, а  $\Gamma$  — количество целочисленных точек на границе многоугольника.

- prime:  $23068673 = 11 \cdot 2^{21} + 1; w : 38(w^{2^{21}} = 1)$

**Newton**

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

- prime:  $69206017 = 33 \cdot 2^{21} + 1; w : 45(w^{2^{21}} = 1)$

**Catalan**

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

$$C_i = \frac{1}{n+1} \binom{2n}{n}$$

- prime:  $81788929 = 39 \cdot 2^{21} + 1; w : 94(w^{2^{21}} = 1)$

**Кол-во графов**

$$G_N := 2^{n(n-1)/2}$$

Количество связанных помеченных графов

$$\text{Conn}_N = G_N - \frac{1}{N} \sum_{K=1}^{N-1} K \binom{N}{K} \text{Conn}_K G_{N-K}$$

Количество помеченных графов с  $K$  компонентами связности

$$D[N][K] = \sum_{S=1}^N \binom{N-1}{S-1} \text{Conn}_S D[N-S][K-1]$$

- prime:  $104857601 = 25 \cdot 2^{22} + 1; w : 21(w^{2^{22}} = 1)$

- prime:  $113246209 = 27 \cdot 2^{22} + 1; w : 66(w^{2^{22}} = 1)$

**Miller-Rabbin**

```
a=a^t
FOR i = 1...s
  if a^2=1 && |a|!=1
    NOT PRIME
  a=a^2
return a==1 ? PRIME : NOT PRIME
```

- prime:  $138412033 = 33 \cdot 2^{22} + 1; w : 30(w^{2^{22}} = 1)$

- prime:  $167772161 = 5 \cdot 2^{25} + 1; w : 17(w^{2^{25}} = 1)$

**Интегрирование по формуле Симпсона**

$$\int_a^b f(x) dx?$$

$$x_i := a + ih, i = 0 \dots 2n$$

$$h = \frac{b-a}{2n}$$

$$\int = (f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \dots + 4f(x_{2n-1}) + f(x_{2n})) \frac{h}{3}$$

$$O(n^4).$$

- prime:  $469762049 = 7 \cdot 2^{26} + 1; w : 30(w^{2^{26}} = 1)$

**Простые числа**

1009,1013;10007,10009;100003,100019  
1000003,1000033;10000019,10000079  
100000007,100000037  
10000000019,10000000033  
100000000039,1000000000061  
10000000000031,10000000000067  
1000000000000061,1000000000000069  
10000000000000003,10000000000000009

- prime:  $998244353 = 7 \cdot 17 \cdot 2^{23} + 1; w : 3^{7 \cdot 17}$ .

**Числа для Фурье**

- prime:  $7340033 = 7 \cdot 2^{20} + 1; w : 5(w^{2^{20}} = 1)$

**Erdős–Gallai theorem**

A sequence of non-negative integers  $d_1 \geq \dots \geq d_n$  can be represented as the degree sequence of a finite simple graph on  $n$  vertices if and only if  $d_1 + \dots + d_n$  is even and

$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$  holds for every  $k$  in  $1 \leq k \leq n$ .

```

1  // sk fast allocation begins
2  const int MAX_MEM = 5e8;
3  int mpos = 0;
4  char mem[MAX_MEM];
5  inline void * operator new ( size_t n ) {
6      assert((mpos += n) <= MAX_MEM);
7      return (void *) (mem + mpos - n);
8  }
9  inline void operator delete ( void * ) noexcept {
10     ↪ } // must have!
11
12 // sk fast allocation ends
13
14
15 // sk fast read-write begins
16
17 inline int readChar();
18 template <class T = int> inline T readInt();
19 template <class T> inline void writeInt( T x,
20     ↪ char end = 0 );
21 inline void writeChar( int x );
22 inline void writeWord( const char *s );
23
24 /** Read */
25
26 static const int buf_size = 2048;
27
28 inline int getChar() {
29     static char buf[buf_size];
30     static int len = 0, pos = 0;
31     if (pos == len)
32         pos = 0, len = fread(buf, 1, buf_size,
33             ↪ stdin);
34     if (pos == len)
35         return -1;
36     return buf[pos++];
37 }
38
39 inline int readWord(char * buffer) {
40     int c = getChar();
41     while (c <= 32) {
42         c = getChar();
43     }
44
45     int len = 0;
46     while (c > 32) {
47         *buffer = (char) c;
48         c = getChar();
49         buffer++;
50         len++;
51     }
52     return len;
53 }
54
55 inline int readChar() {
56     int c = getChar();
57     while (c <= 32)
58         c = getChar();
59     return c;
60
61 template <class T>
62 inline T readInt() {
63     int s = 1, c = readChar();
64     T x = 0;
65     if (c == '-')
66         s = -1, c = getChar();
67     while ('0' <= c && c <= '9')
68         x = x * 10 + c - '0', c = getChar();
69     return s == 1 ? x : -x;
70 }
71
72 /** Write */
73
74 static int write_pos = 0;
75 static char write_buf[buf_size];
76
77 inline void writeChar( int x ) {
78     if (write_pos == buf_size)
79         fwrite(write_buf, 1, buf_size, stdout),
80         ↪ write_pos = 0;
81     write_buf[write_pos++] = x;
82 }
83
84 template <class T>
85 inline void writeInt( T x, char end ) {
86     if (x < 0)
87         writeChar('-'), x = -x;
88
89     char s[24];
90     int n = 0;
91     while (x || !n)
92         s[n++] = '0' + x % 10, x /= 10;
93     while (n--)
94         writeChar(s[n]);
95     if (end)
96         writeChar(end);
97 }
98
99 inline void writeWord( const char *s ) {
100     while (*s)
101         writeChar(*s++);
102 }
103
104 struct Flusher {
105     ~Flusher() {
106         if (write_pos)
107             fwrite(write_buf, 1, write_pos,
108                 ↪ stdout), write_pos = 0;
109     }
110 } flusher;
111
112 // sk fast read-write ends
113
114 // extended euclid begins
115
116 int gcd (int a, int b, int & x, int & y) {
117     if (a == 0) {
118         x = 0; y = 1;
119         return b;
120     }

```

```

7     }
8     int x1, y1;
9     int d = gcd (b%a, a, x1, y1);
10    x = y1 - (b / a) * x1;
11    y = x1;
12    return d;
13 }
14
15 // extended euclid ends
16
17 // FFT begins
18
19 const int LOG = 19;
20 const int N = (1 << LOG);
21
22 typedef std::complex<double> cd;
23
24 int rev[N];
25 cd W[N];
26
27 void precalc() {
28     const double pi = std::acos(-1);
29     for (int i = 0; i != N; ++i)
30         W[i] = cd(std::cos(2 * pi * i / N),
31                 ↪ std::sin(2 * pi * i / N));
32
33     int last = 0;
34     for (int i = 0; i != N; ++i) {
35         if (i == (2 << last))
36             ++last;
37
38         rev[i] = rev[i ^ (1 << last)] | (1 <<
39             ↪ (LOG - 1 - last));
40     }
41 }
42
43 void fft(vector<cd>& a) {
44     for (int i = 0; i != N; ++i)
45         if (i < rev[i])
46             std::swap(a[i], a[rev[i]]);
47
48     for (int lvl = 0; lvl != LOG; ++lvl)
49         for (int start = 0; start != N; start +=
50             ↪ (2 << lvl))
51             for (int pos = 0; pos != (1 << lvl);
52                 ↪ ++pos) {
53                 cd x = a[start + pos];
54                 cd y = a[start + pos + (1 <<
55                     ↪ lvl)];
56
57                 y *= W[pos << (LOG - 1 - lvl)];
58
59                 a[start + pos] = x + y;
60                 a[start + pos + (1 << lvl)] = x -
61                     ↪ y;
62             }
63 }
64
65 void inv_fft(vector<cd>& a) {
66     fft(a);
67     std::reverse(a.begin() + 1, a.end());

```

```

47     for (cd& elem: a)
48         elem /= N;
49 }
50
51 // FFT ends
52
53 // fast gauss begins
54
55 using elem_t = int;
56 // a[i][rows[i][j].first]=rows[i][j].second;
57 ↪ b[i]=a[i][n]
58 bool gauss(vector<vector<pair<int, elem_t>>>
59     ↪ rows, vector<elem_t> &res) {
60     int n = rows.size();
61
62     res.resize(n + 1, 0);
63     vector<int> p(n + 1);
64     iota(p.begin(), p.end(), 0);
65     vector<int> toZero(n + 1, -1);
66     vector<int> zro(n + 1);
67     vector<elem_t> a(n + 1);
68
69     // optional: sort rows
70
71     sort(p.begin(), p.begin() + n, [&rows](int i,
72     ↪ int j) { return rows[i].size() <
73     ↪ rows[j].size(); });
74     vector<int> invP(n + 1);
75     vector<vector<pair<int, elem_t>>> rs(n);
76     for (int i = 0; i < n; i++) {
77         invP[p[i]] = i;
78         rs[i] = rows[p[i]];
79     }
80     for (int i = 0; i < n; i++) {
81         rows[i] = rs[i];
82         for (auto& el: rows[i]) {
83             if (el.first < n) {
84                 el.first = invP[el.first];
85             }
86         }
87     }
88
89     for (int i = 0; i < n; i++) {
90         for (auto& el: rows[i]) {
91             a[el.first] = el.second;
92         }
93         while (true) {
94             int k = -1;
95             for (auto& el: rows[i]) {
96                 if (!isZero(a[el.first]) &&
97                     ↪ toZero[el.first] != -1 &&
98                     (k == -1 || toZero[el.first]
99                     ↪ < toZero[k])) {
100                     k = el.first;
101                 }
102             }
103             if (k == -1)
104                 break;
105
106             int j = toZero[k];
107             elem_t c = a[k];

```

```

51                                     2
52     for (auto el: rows[j]) {       3     struct Point {
53         if (isZero(a[el.first]))    4         double x, y;
54                                     5         Point operator+(const Point& p) const {
55                                     ↪     return {x + p.x, y + p.y}; }
56                                     ↪     0);
57                                     6         Point operator-(const Point& p) const {
58         a[el.first] = sub(a[el.first], ↪     return {x - p.x, y - p.y}; }
59                                     ↪     mult(c, el.second));
60                                     7         Point operator*(const double d) const {
61                                     ↪     return {x * d, y * d}; }
62                                     8         Point rotate() const { return {y, -x}; }
63                                     9         double operator*(const Point& p) const {
64                                     ↪     return x * p.x + y * p.y; }
65                                     10        double operator^(const Point& p) const {
66                                     ↪     return x * p.y - y * p.x; }
67                                     11        double dist() const { return sqrt(x * x + y *
68                                     ↪     rows[i].erase(std::remove_if(rows[i].begin(y)); }
69                                     ↪     rows[i].end(), cond),
70                                     ↪     rows[i].end());
71                                     12    };
72                                     13
73                                     14    struct Line {
74                                     15        double a, b, c;
75                                     16        Line(const Point& p1, const Point& p2) {
76                                     17            a = p1.y - p2.y;
77                                     18            b = p2.x - p1.x;
78                                     19            c = - a * p1.x - b * p1.y;
79                                     20
79                                     21            double d = sqrt(sqr(a) + sqr(b));
80                                     ↪     a /= d, b /= d, c /= d;
81                                     22        }
82                                     23        bool operator||(const Line& l) const { return
83                                     ↪     fabs(a * l.b - l.a * b) < EPS; }
84                                     24        double dist(const Point& p) const { return
85                                     ↪     fabs(a * p.x + b * p.y + c); }
86                                     25        Point operator^(const Line& l) const {
87                                     ↪     return {(l.c * b - c * l.b) / (a * l.b -
88                                     ↪     l.a * b),
89                                     ↪     (l.c * a - c * l.a) / (l.a * b -
90                                     ↪     a * l.b)};
91                                     26        }
92                                     27        Point projection(const Point& p) const {
93                                     ↪     return p - Point{a, b} * (a * p.x + b *
94                                     ↪     p.y + c);
95                                     28        }
96                                     29    };
97                                     30
98                                     31    struct Circle {
99                                     32        Point c;
100                                    33        double r;
101                                    34        Circle(const Point& c, double r) : c(c), r(r)
102                                    ↪     {}
103                                    35        Circle(const Point& a, const Point& b, const
104                                    ↪     Point& c) {
105                                    36            Point p1 = (a + b) * 0.5, p2 = (a + c) *
106                                    ↪     0.5;
107                                    37            Point q1 = p1 + (b - a).rotate(), q2 = p2
108                                    ↪     + (c - a).rotate();
109                                    38            this->c = Line(p1, q1) ^ Line(p2, q2);
110                                    39            r = (a - this->c).dist();
111                                    40        }
112                                    41    };
113                                    42
114                                    43    };
115                                    44
116                                    45    };
117                                    46
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500

```

```

47 inline bool on_segment(const Point& p1, const      8      bool operator<(const Point& p) const { return
   ↪ Point& p2, const Point& x, bool strictly) {      ↪ x != p.x ? x < p.x : y < p.y; }
48     if (fabs((p1 - x) ^ (p2 - x)) > EPS)           9 };
49         return false;                             10
50     return (p1 - x) * (p2 - x) < (strictly ? -      11 // all point on convex hull are included
   ↪ EPS : EPS);                                     12 vector<Point> convex_hull(vector<Point> pt) {
51 }                                                  13     int n = pt.size();
52                                                    14     Point p0 = *std::min_element(pt.begin(),
53 // in case intersection is not a segment          ↪ pt.end());
54 inline bool intersect_segments(const Point& p1,    15     std::sort(pt.begin(), pt.end(), [&p0](const
   ↪ const Point& p2, const Point& q1, const          ↪ Point& a, const Point& b) {
   ↪ Point& q2, Point& x) {                            16         int64_t cp = (a - p0) ^ (b - p0);
55     Line l1(p1, p2), l2(q1, q2);                 17         return cp != 0 ? cp > 0 : (a - p0).dist()
56     if (l1 || l2) return false;                   ↪ < (b - p0).dist();
57     x = l1 ^ l2;                                   18     });
58     return on_segment(p1, p2, x, false);          19
59 }                                                  20     int i = n - 1;
60                                                    21     for (; i > 0 && ((pt[i] - p0) ^ (pt[i - 1] -
61 // in case circles are not equal                  ↪ p0)) == 0; i--);
62 inline bool intersect_circles(const Circle& c1,    22     std::reverse(pt.begin() + i, pt.end());
   ↪ const Circle& c2, Point& p1, Point& p2) {        23
63     double d = (c2.c - c1.c).dist();             24     vector<Point> ch;
64     if (d > c1.r + c2.r + EPS || d < fabs(c1.r -   25     for (auto& p : pt) {
   ↪ c2.r) - EPS)                                     26         while (ch.size() > 1) {
65         return false;                             27             auto& p1 = ch[(int) ch.size() - 1];
66     double cosa = (sqr(d) + sqr(c1.r) -           28             auto& p2 = ch[(int) ch.size() - 2];
   ↪ sqr(c2.r)) / (2 * c1.r * d);                     29             int64_t cp = (p1 - p2) ^ (p - p1);
67     double l = c1.r * cosa, h = sqrt(sqr(c1.r) - 30             if (cp >= 0) break;
   ↪ sqr(l));                                           31             ch.pop_back();
68     Point v = (c2.c - c1.c) * (1 / d), p = c1.c + 32         }
   ↪ v * l;                                           33         ch.push_back(p);
69     p1 = p + v.rotate() * h, p2 = p - v.rotate() 34     }
   ↪ * h;                                           35
70     return true;                                   36     return ch;
71 }                                                  37 }
72                                                    38
73 inline bool intersect_circle_and_line(const         39 // convex hull ends
   ↪ Circle& c, const Line& l, Point& p1, Point&      1 // convex hull trick begins
   ↪ p2) {                                             2
74     double d = l.dist(c.c);                       3     typedef long long ftype;
75     if (d > c.r + EPS)                             4     typedef complex<ftype> point;
76         return false;                             5     #define x real
77     Point p = l.projection(c.c);                   6     #define y imag
78     Point n{l.b, -l.a};
79     double h = sqrt(sqr(c.r) - sqr(l.dist(c.c)));  7     ftype dot(point const& a, point const& b) {
80     p1 = p + n * h, p2 = p - n * h;               8         return (conj(a) * b).x();
81     return true;                                   9     }
82 }                                                  10
83                                                    11
84 // simple geometry ends                          12     ftype f(point const& a, int x) {
1 // convex hull begins                            13         return dot(a, {compressed[x], 1});
2                                                    14         //return dot(a, {x, 1});
3     struct Point {                                15     }
4         int x, y;
5         Point operator-(const Point& p) const {    16
   ↪ return {x - p.x, y - p.y}; }                    17     int pos = 0;
6         int64_t operator^(const Point& p) const {  18
   ↪ return x * 1ll * p.y - y * 1ll * p.x; }          19 // (x, y) -> (k, b) -> kb + x
7         int64_t dist() const { return x * 1ll * x + 20     struct li_chao { // for min
   ↪ * 1ll * y; }                                     21         vector<point> line;
                                                    22
                                                    23         li_chao(int maxn) {
                                                    24             line.resize(4 * maxn, {0, inf});

```

```

25     }
26
27     void add_line(int v, int l, int r, int a, int
    ↪ b, point nw) {
28         if (r <= a || b <= l) return; // remove
    ↪ if no [a, b) query
29
30         int m = (l + r) >> 1;
31
32         if (!(a <= l && r <= b)) { // remove if
    ↪ no [a, b) query
33             add_line(v + v + 1, l, m, a, b, nw);
34             add_line(v + v + 2, m, r, a, b, nw);
35             return;
36         }
37
38         bool lef = f(nw, l) < f(line[v], l);
39         bool mid = f(nw, m) < f(line[v], m);
40
41         if (mid) swap(line[v], nw);
42
43         if (l == r - 1)
44             return;
45
46         if (lef != mid)
47             add_line(v + v + 1, l, m, a, b, nw);
48         else
49             add_line(v + v + 2, m, r, a, b, nw);
50     }
51
52     ftype get(int v, int l, int r, int x) {
53         if (l == r - 1)
54             return f(line[v], x);
55         int m = (l + r) / 2;
56         if (x < m) {
57             return min(f(line[v], x), get(v + v
    ↪ 1, l, m, x));
58         } else {
59             return min(f(line[v], x), get(v + v
    ↪ 2, m, r, x));
60         }
61     }
62
63     } cdt(maxn);
64
65     // convex hull with stack
66
67     ftype cross(point a, point b) {
68         return (conj(a) * b).y();
69     }
70
71     vector<point> hull, vecs;
72
73     void add_line(ftype k, ftype b) {
74         point nw = {k, b};
75         while(!vecs.empty() && dot(vecs.back(), nw -
    ↪ hull.back()) < 0) {
76             hull.pop_back();
77             vecs.pop_back();
78         }
79         if(!hull.empty()) {
80             vecs.push_back(1i * (nw - hull.back()));
81         }
82         hull.push_back(nw);
83     }
84
85     int get(ftype x) {
86         point query = {x, 1};
87         auto it = lower_bound(vecs.begin(),
    ↪ vecs.end(), query, [](point a, point b) {
88             return cross(a, b) > 0;
89         });
90         return dot(query, hull[it - vecs.begin()]);
91     }
92
93     // convex hull trick ends
94
95     // heavy-light begins
96
97     int sz[maxn];
98
99     void dfs_sz(int v, int par = -1) {
100         sz[v] = 1;
101         for (int x : gr[v])
102             if (x != par) {
103                 dfs_sz(x, v);
104                 sz[v] += sz[x];
105             }
106         for (int i = 0; i < gr[v].size(); i++)
107             if (gr[v][i] != par)
108                 if (sz[gr[v][i]] * 2 >= sz[v]) {
109                     swap(gr[v][i], gr[v][0]);
110                     break;
111                 }
112     }
113
114     int rev[maxn];
115     int t_in[maxn];
116     int upper[maxn];
117     int par[maxn];
118     int dep[maxn];
119
120     int T = 0;
121
122     void dfs_build(int v, int uppr, int pr = -1) {
123         rev[T] = v;
124         t_in[v] = T++;
125         dep[v] = pr == -1 ? 0 : dep[pr] + 1;
126         par[v] = pr;
127         upper[v] = uppr;
128
129         bool first = true;
130
131         for (int x : gr[v])
132             if (x != pr) {
133                 dfs_build(x, first ? upper[v] : x,
    ↪ v);
134                 first = false;
135             }
136     }
137
138     struct interval {
139         int l;

```

```

46     int r;
47     bool inv; // should direction be reversed
48 };
49
50 // node-weighted hld
51 vector<interval> get_path(int a, int b) {
52     vector<interval> front;
53     vector<interval> back;
54
55     while (upper[a] != upper[b]) {
56         if (dep[upper[a]] > dep[upper[b]]) {
57             front.push_back({t_in[upper[a]],
58                             ↪ t_in[a], true});
59             a = par[upper[a]];
60         } else {
61             back.push_back({t_in[upper[b]],
62                             ↪ t_in[b], false});
63             b = par[upper[b]];
64         }
65     }
66
67     front.push_back({min(t_in[a], t_in[b]),
68                     ↪ max(t_in[a], t_in[b]), t_in[a] >
69                     ↪ t_in[b]});
70     // for edge-weighted hld add:
71     ↪ "front.back().l++;";
72     front.insert(front.end(), back.rbegin(),
73                 ↪ back.rend());
74
75     return front;
76 }
77
78 // heavy-light ends
79
80 // max flow begins
81
82 struct edge{
83     int from, to;
84     int c, f, num;
85     edge(int from, int to, int c, int
86         ↪ num):from(from), to(to), c(c), f(0),
87         ↪ num(num){}
88     edge(){}
89 };
90
91 const int max_n = 600;
92
93 edge eds[150000];
94 int num = 0;
95 int it[max_n];
96 vector<int> gr[max_n];
97 int s, t;
98 vector<int> d(max_n);
99
100 bool bfs(int k) {
101     queue<int> q;
102     q.push(s);
103     fill(d.begin(), d.end(), -1);
104     d[s] = 0;
105     while (!q.empty()) {
106         int v = q.front();
107         q.pop();

```

```

27     for (int x : gr[v]) {
28         int to = eds[x].to;
29         if (d[to] == -1 && eds[x].c -
30             ↪ eds[x].f >= (1 << k)){
31             d[to] = d[v] + 1;
32             q.push(to);
33         }
34     }
35
36     return (d[t] != -1);
37 }
38
39 int dfs(int v, int flow, int k) {
40     if (flow < (1 << k))
41         return 0;
42     if (v == t)
43         return flow;
44     for (; it[v] < gr[v].size(); it[v]++) {
45         int num = gr[v][it[v]];
46         if (d[v] + 1 != d[num].to)
47             continue;
48         int res = dfs(eds[num].to, min(flow,
49             ↪ eds[num].c - eds[num].f), k);
50         if (res){
51             eds[num].f += res;
52             eds[num ^ 1].f -= res;
53             return res;
54         }
55     }
56     return 0;
57 }
58
59 void add(int fr, int to, int c, int nm) {
60     gr[fr].push_back(num);
61     eds[num++] = edge(fr, to, c, nm);
62     gr[to].push_back(num);
63     eds[num++] = edge(to, fr, 0, nm); //corrected
64     ↪ c
65 }
66
67 int ans = 0;
68 for (int k = 30; k >= 0; k--)
69     while (bfs(k)) {
70         memset(it, 0, sizeof(it));
71         while (int res = dfs(s, 1e9 + 500,
72             ↪ k))
73             ans += res;
74     }
75
76 // decomposition
77
78 int path_num = 0;
79 vector<int> paths[max_n];
80 int flows[max_n];
81
82 int decomp(int v, int flow) {
83     if (flow < 1)
84         return 0;
85     if (v == t) {

```



```

84     path_num++;
85     flows[path_num - 1] = flow;
86     return flow;
87 }
88 for (int i = 0; i < gr[v].size(); i++) {
89     int num = gr[v][i];
90     int res = decomp(eds[num].to, min(flow,
91         ↪ eds[num].f));
92     if (res) {
93         eds[num].f -= res;
94         paths[path_num -
95             ↪ 1].push_back(eds[num].num);
96         return res;
97     }
98 }
99 return 0;
100 while (decomp(s, 1e9 + 5));
101
102 // max flow ends
103
104 // min-cost flow begins
105
106 long long ans = 0;
107 int mx = 2 * n + 2;
108
109 memset(upd, 0, sizeof(upd));
110 for (int i = 0; i < mx; i++)
111     dist[i] = inf;
112 dist[st] = 0;
113 queue<int> q;
114 q.push(st);
115 upd[st] = 1;
116 while (!q.empty()){
117     int v = q.front();
118     q.pop();
119     if (upd[v]){
120         for (int x : gr[v]) {
121             edge &e = edges[x];
122             if (e.c - e.f > 0 && dist[v] != inf
123                 ↪ && dist[e.to] > dist[v] + e.w) {
124                 dist[e.to] = dist[v] + e.w;
125                 if (!upd[e.to])
126                     q.push(e.to);
127                 upd[e.to] = true;
128                 p[e.to] = x;
129             }
130         }
131         upd[v] = false;
132     }
133 }
134
135 for (int i = 0; i < k; i++){
136     for (int i = 0; i < mx; i++)
137         d[i] = inf;
138     d[st] = 0;
139     memset(used, false, sizeof(used));
140     set<pair<int, int>> s;
141     s.insert(make_pair(0, st));
142     for (int i = 0; i < mx; i++){
143         int x;
144         while (!s.empty() && used[(s.begin() ->
145             ↪ second)]){
146             s.erase(s.begin());
147         }
148         if (s.empty())
149             break;
150         x = s.begin() -> second;
151         used[x] = true;
152         s.erase(s.begin());
153         for (int i = 0; i < gr[x].size(); i++){
154             edge &e = edges[gr[x][i]];
155             if (!used[e.to] && e.c - e.f > 0){
156                 if (d[e.to] > d[x] + (e.c - e.f)
157                     ↪ * e.w + dist[x] -
158                     ↪ dist[e.to]){
159                     d[e.to] = d[x] + (e.c - e.f)
160                         ↪ * e.w + dist[x] -
161                         ↪ dist[e.to];
162                     p[e.to] = gr[x][i];
163                     s.insert(make_pair(d[e.to],
164                         ↪ e.to));
165                 }
166             }
167         }
168         dist[x] += d[x];
169     }
170 }
171
172 // min-cost flow ends
173
174 // bad hungarian begins
175
176 fill(par, par + 301, -1);
177 fill(par2, par2 + 301, -1);
178
179 int ans = 0;
180 for (int v = 0; v < n; v++){
181     memset(useda, false, sizeof(useda));
182     memset(usedb, false, sizeof(usedb));
183     useda[v] = true;
184     for (int i = 0; i < n; i++)
185         w[i] = make_pair(a[v][i] + row[v] +
186             ↪ col[i], v);
187     memset(prev, 0, sizeof(prev));
188     int pos;
189     while (true){
190         pair<pair<int, int>, int> p =
191             ↪ make_pair(make_pair(1e9, 1e9), 1e9);
192         for (int i = 0; i < n; i++)
193             if (!usedb[i])
194                 p = min(p, make_pair(w[i], i));
195         for (int i = 0; i < n; i++)
196             if (!useda[i])
197                 row[i] += p.first.first;
198         for (int i = 0; i < n; i++)

```

```

24         if (!usedb[i]){
25             col[i] -= p.first.first;
26             w[i].first -= p.first.first;
27         }
28         ans += p.first.first;
29         usedb[p.second] = true;
30         prev[p.second] = p.first.second; //us
31         ↪ эмопой в nepeyю
32         int x = par[p.second];
33         if (x == -1){
34             pos = p.second;
35             break;
36         }
37         useda[x] = true;
38         for (int j = 0; j < n; j++)
39             w[j] = min(w[j], {a[x][j] + row[x]
40             ↪ col[j], x});
41     }
42     while (pos != -1){
43         int nxt = par2[prev[pos]];
44         par[pos] = prev[pos];
45         par2[prev[pos]] = pos;
46         pos = nxt;
47     }
48     cout << ans << "\n";
49     for (int i = 0; i < n; i++)
50         cout << par[i] + 1 << " " << i + 1 << "\n";
51
52     // bad hungarian ends
53
54     // Edmonds O(n^3) begins
55
56     vector<int> gr[MAXN];
57     int match[MAXN], p[MAXN], base[MAXN], q[MAXN];
58     bool used[MAXN], blossom[MAXN];
59     int mark[MAXN];
60     int C = 1;
61
62     int lca(int a, int b) {
63         C++;
64         for (;;) {
65             a = base[a];
66             mark[a] = C;
67             if (match[a] == -1) break;
68             a = p[match[a]];
69         }
70
71         for (;;) {
72             b = base[b];
73             if (mark[b] == C) return b;
74             b = p[match[b]];
75         }
76     }
77
78     void mark_path(int v, int b, int children) {
79         while (base[v] != b) {
80             blossom[base[v]] =
81             ↪ blossom[base[match[v]]] = true;
82             p[v] = children;
83             children = match[v];
84         }
85     }
86
87     v = p[match[v]];
88 }
89
90 int find_path(int root) {
91     memset(used, 0, sizeof(used));
92     memset(p, -1, sizeof p);
93     for (int i = 0; i < N; i++)
94         base[i] = i;
95
96     used[root] = true;
97     int qh = 0, qt = 0;
98     q[qt++] = root;
99     while (qh < qt) {
100         int v = q[qh++];
101         for (int to : gr[v]) {
102             if (base[v] == base[to] || match[v]
103             ↪ == to) continue;
104             if (to == root || match[to] != -1 &&
105             ↪ p[match[to]] != -1) {
106                 int curbase = lca(v, to);
107                 memset(blossom, 0,
108                 ↪ sizeof(blossom));
109                 mark_path(v, curbase, to);
110                 mark_path(to, curbase, v);
111                 for (int i = 0; i < N; i++)
112                     if (blossom[base[i]]) {
113                         base[i] = curbase;
114                         if (!used[i]) {
115                             used[i] = true;
116                             q[qt++] = i;
117                         }
118                     }
119             } else if (p[to] == -1) {
120                 p[to] = v;
121                 if (match[to] == -1)
122                     return to;
123                 to = match[to];
124                 used[to] = true;
125                 q[qt++] = to;
126             }
127         }
128     }
129     return -1;
130 }
131
132 memset(match, -1, sizeof match);
133 for (int i = 0; i < N; i++) {
134     if (match[i] == -1 && !gr[i].empty()) {
135         int v = find_path(i);
136         while (v != -1) {
137             int pv = p[v], ppv = match[pv];
138             match[v] = pv; match[pv] = v;
139             v = ppv;
140         }
141     }
142 }
143
144 // Edmonds O(n^3) ends
145
146 // string basis begins

```

```

2
3 vector<int> getZ(string s){
4     vector<int> z;
5     z.resize(s.size(), 0);
6     int l = 0, r = 0;
7     for (int i = 1; i < s.size(); i++){
8         if (i <= r)
9             z[i] = min(r - i + 1, z[i - 1]);
10        while (i + z[i] < s.size() && s[z[i]] ==
11            ↪ s[i + z[i]])
12            z[i]++;
13        if (i + z[i] - 1 > r){
14            r = i + z[i] - 1;
15            l = i;
16        }
17    }
18    return z;
19 }
20 vector<int> getP(string s){
21     vector<int> p;
22     p.resize(s.size(), 0);
23     int k = 0;
24     for (int i = 1; i < s.size(); i++){
25         while (k > 0 && s[i] == s[k])
26             k = p[k - 1];
27         if (s[i] == s[k])
28             k++;
29         p[i] = k;
30     }
31     return p;
32 }
33
34 vector<int> getH(string s){
35     vector<int> h;
36     h.resize(s.size() + 1, 0);
37     for (int i = 0; i < s.size(); i++)
38         h[i + 1] = ((h[i] * 111 * pow) + s[i] -
39             ↪ 'a' + 1) % mod;
40     return h;
41 }
42
43 int getHash(vector<int> &h, int l, int r){
44     int res = (h[r + 1] - h[l] * p[r - l + 1]) %
45     ↪ mod;
46     if (res < 0)
47         res += mod;
48     return res;
49 }
50
51 // string basis ends
52
53 // min cyclic shift begins
54
55 string min_cyclic_shift (string s) {
56     s += s;
57     int n = (int) s.length();
58     int i=0, ans=0;
59     while (i < n/2) {
60         ans = i;
61         int j=i+1, k=i;
62         while (j < n && s[k] <= s[j]) {
63
64
65             if (s[k] < s[j])
66                 k = i;
67             else
68                 ++k;
69             ++j;
70         }
71         while (i <= k) i += j - k;
72     }
73     return s.substr (ans, n/2);
74 }
75
76 // min cyclic shift ends
77
78 // suffix array O(n) begins
79
80 typedef vector<char> bits;
81
82 template<const int end>
83 void getBuckets(int *s, int *bkt, int n, int K) {
84     fill(bkt, bkt + K + 1, 0);
85     forn(i, n) bkt[s[i] + !end]++;
86     forn(i, K) bkt[i + 1] += bkt[i];
87 }
88
89 void induceSA1(bits &t, int *SA, int *s, int
90     ↪ *bkt, int n, int K) {
91     getBuckets<0>(s, bkt, n, K);
92     forn(i, n) {
93         int j = SA[i] - 1;
94         if (j >= 0 && !t[j])
95             SA[bkt[s[j]]++] = j;
96     }
97 }
98
99 void induceSAs(bits &t, int *SA, int *s, int
100     ↪ *bkt, int n, int K) {
101     getBuckets<1>(s, bkt, n, K);
102     for (int i = n - 1; i >= 0; i--) {
103         int j = SA[i] - 1;
104         if (j >= 0 && t[j])
105             SA[--bkt[s[j]]] = j;
106     }
107 }
108
109 void SA_IS(int *s, int *SA, int n, int K) { //
110     ↪ require last symbol is 0
111     #define isLMS(i) (i && t[i] && !t[i-1])
112     int i, j;
113     bits t(n);
114     t[n-1] = 1;
115     for (i = n - 3; i >= 0; i--)
116         t[i] = (s[i]<s[i+1] || (s[i]==s[i+1] &&
117             ↪ t[i+1]==1));
118     int bkt[K + 1];
119     getBuckets<1>(s, bkt, n, K);
120     fill(SA, SA + n, -1);
121     forn(i, n)
122         if (isLMS(i))
123             SA[--bkt[s[i]]] = i;
124     induceSA1(t, SA, s, bkt, n, K);
125     induceSAs(t, SA, s, bkt, n, K);
126     int n1 = 0;
127     forn(i, n)
128         if (isLMS(SA[i]))

```

```

46         SA[n1++] = SA[i];
47     fill(SA + n1, SA + n, -1);
48     int name = 0, prev = -1;
49     forn(i, n1) {
50         int pos = SA[i];
51         bool diff = false;
52         for (int d = 0; d < n; d++)
53             if (prev == -1 || s[pos+d] !=
54                 ↪ s[prev+d] || t[pos+d] !=
55                 ↪ t[prev+d])
56                 diff = true, d = n;
57             else if (d > 0 && (isLMS(pos+d) ||
58                 ↪ isLMS(prev+d)))
59                 d = n;
60         if (diff)
61             name++, prev = pos;
62         SA[n1 + (pos >> 1)] = name - 1;
63     }
64     for (i = n - 1, j = n - 1; i >= n1; i--)
65         if (SA[i] >= 0)
66             SA[j--] = SA[i];
67     int *s1 = SA + n - n1;
68     if (name < n1)
69         SA_IS(s1, SA, n1, name - 1);
70     else
71         forn(i, n1)
72             SA[s1[i]] = i;
73     getBuckets<1>(s, bkt, n, K);
74     for (i = 1, j = 0; i < n; i++)
75         if (isLMS(i))
76             s1[j++] = i;
77     forn(i, n1)
78         SA[i] = s1[SA[i]];
79     fill(SA + n1, SA + n, -1);
80     for (i = n1 - 1; i >= 0; i--) {
81         j = SA[i], SA[i] = -1;
82         SA[--bkt[s[j]]] = j;
83     }
84     induceSA1(t, SA, s, bkt, n, K);
85     induceSAs(t, SA, s, bkt, n, K);
86 }
87 // suffix array O(n) ends
88
89 // suffix array O(n log n) begins
90 string str;
91 int N, m, SA [MAX_N], LCP [MAX_N];
92 int x [MAX_N], y [MAX_N], w [MAX_N], c [MAX_N];
93
94 inline bool cmp (const int a, const int b, const
95     ↪ int l) { return (y [a] == y [b] && y [a + l]
96     ↪ == y [b + l]); }
97
98 void Sort () {
99     for (int i = 0; i < m; ++i) w[i] = 0;
100     for (int i = 0; i < N; ++i) ++w[x[y[i]]];
101     for (int i = 0; i < m - 1; ++i) w[i + 1] +=
102         ↪ w[i];
103     for (int i = N - 1; i >= 0; --i)
104         ↪ SA[--w[x[y[i]]]] = y[i];
105 }
106
107 void DA () {
108     for (int i = 0; i < N; ++i) x[i] = str[i],
109         ↪ y[i] = i;
110     Sort ();
111     for (int i, j = 1, p = 1; p < N; j <= 1, m =
112         ↪ p) {
113         for (p = 0, i = N - j; i < N; i++) y[p++]
114             ↪ = i;
115         for (int k = 0; k < N; ++k) if (SA[k] >=
116             ↪ j) y[p++] = SA[k] - j;
117         Sort();
118         for (swap (x, y), p = 1, x[SA[0]] = 0, i
119             ↪ = 1; i < N; ++i) x[SA [i]] = cmp
120             ↪ (SA[i - 1], SA[i], j) ? p - 1 : p++;
121     }
122 }
123
124 // common for all algorithms
125 void kasaiLCP () {
126     for (int i = 0; i < N; i++) c[SA[i]] = i;
127     for (int i = 0, j, k = 0; i < N; LCP [c[i++]]
128         ↪ = k)
129         if (c [i] > 0) for (k ? k-- : 0, j =
130             ↪ SA[c[i] - 1]; str[i + k] == str[j +
131             ↪ k]; k++);
132         else k = 0;
133 }
134
135 void suffixArray () { // require last symbol is
136     ↪ char(0)
137     m = 256;
138     N = str.size();
139     DA ();
140     kasaiLCP ();
141 }
142
143 // suffix array O(n log n) ends
144
145 // bad suffix automaton begins
146
147 struct node{
148     map<char, int> go;
149     int len, suff;
150     long long sum_in;
151     node(){ }
152 };
153
154 node v[max_n * 4];
155
156 int add_node(int max_len){
157     //v[number].sum_in = 0;
158     v[number].len = max_len;
159     v[number].suff = -1;
160     number++;
161     return number - 1;
162 }
163
164 int last = add_node(0);
165
166 void add_char(char c) {
167     int cur = last;
168     int new_node = add_node(v[cur].len + 1);
169     last = new_node;
170 }

```

```

26 while (cur != -1){
27     if (v[cur].go.count(c) == 0){
28         v[cur].go[c] = new_node;
29         //v[new_node].sum_in +=
        ↪ v[cur].sum_in;
30         cur = v[cur].suff;
31         if (cur == -1)
32             v[new_node].suff = 0;
33     }else{
34         int a = v[cur].go[c];
35         if (v[a].len == v[cur].len + 1){
36             v[new_node].suff = a;
37         }else{
38             int b = add_node(v[cur].len + 1);
39             v[b].go = v[a].go;
40             v[b].suff = v[a].suff;
41             v[new_node].suff = b;
42             while (cur != -1 &&
        ↪ v[cur].go.count(c) != 0 &&
        ↪ v[cur].go[c] == a){
43                 v[cur].go[c] = b;
44                 //v[a].sum_in -=
        ↪ v[cur].sum_in;
45                 //v[b].sum_in +=
        ↪ v[cur].sum_in;
46                 cur = v[cur].suff;
47             }
48             v[a].suff = b;
49         }
50         return;
51     }
52 }
53 }
54
55 // bad suffix automaton ends
56
57 // pollard begins
58
59 const int max_step = 4e5;
60
61 unsigned long long gcd(unsigned long long a,
62     ↪ unsigned long long b){
63     if (!a) return 1;
64     while (a) swap(a, b%=a);
65     return b;
66 }
67
68 unsigned long long get(unsigned long long a,
69     ↪ unsigned long long b){
70     if (a > b)
71         return a-b;
72     else
73         return b-a;
74 }
75
76 unsigned long long pollard(unsigned long long n){
77     unsigned long long x = (rand() + 1) % n, y =
        ↪ 1, g;
78     int stage = 2, i = 0;
79     g = gcd(get(x, y), n);
80     while (g == 1) {
81         if (i == max_step)
82             break;
83         if (i == stage) {
84             y = x;
85             stage <= 1;
86         }
87         x = (x * (__int128)x + 1) % n;
88         i++;
89         g = gcd(get(x, y), n);
90     }
91     return g;
92 }
93
94 // pollard ends
95
96 // linear sieve begins
97
98 const int N = 1000000;
99
100 int pr[N + 1], sz = 0;
101 /* minimal prime, mobius function, euler function
    ↪ */
102 int lp[N + 1], mu[N + 1], phi[N + 1];
103
104 lp[1] = mu[1] = phi[1] = 1;
105 for (int i = 2; i <= N; ++i) {
106     if (lp[i] == 0)
107         lp[i] = pr[sz++] = i;
108     for (int j = 0; j < sz && pr[j] <= lp[i]
        ↪ && i * pr[j] <= N; ++j)
109         lp[i * pr[j]] = pr[j];
110
111     mu[i] = lp[i] == lp[i / lp[i]] ? 0 : -1 *
        ↪ mu[i / lp[i]];
112     phi[i] = phi[i / lp[i]] * (lp[i] == lp[i /
        ↪ lp[i]] ? lp[i] : lp[i] - 1);
113 }
114
115 // linear sieve ends
116
117 // discrete log in sqrt(p) begins
118
119 int k = sqrt((double)p) + 2;
120
121 for (int i = k; i >= 1; i--)
122     mp[bin(b, (i * 1ll * k) % (p-1), p)] = i;
123
124 bool answered = false;
125 int ans = INT32_MAX;
126 for (int i = 0; i <= k; i++){
127     int sum = (n * 1ll * bin(b, i, p)) % p;
128     if (mp.count(sum) != 0){
129         int an = mp[sum] * 1ll * k - i;
130         if (an < p)
131             ans = min(an, ans);
132     }
133 }
134
135 // discrete log in sqrt(p) ends
136
137 // prime roots mod n begins
138
139 int num = 0;

```

```

4  long long phi = n, nn = n;
5  for (long long x:primes){
6      if (x*x>nn)
7          break;
8      if (nn % x == 0){
9          while (nn % x == 0)
10             nn /= x;
11             phi -= phi/x;
12             num++;
13     }
14 }
15 if (nn != 1){
16     phi -= phi/nn;
17     num++;
18 }
19 if (!(num == 1 && n % 2 != 0) || n == 4 || n == 2 ||
    2 || (num == 2 && n % 2 == 0 && n % 4 != 0)){
20     cout << "-1\n";
21     continue;
22 }
23 vector<long long> v;
24 long long pp = phi;
25 for (long long x:primes){
26     if (x*x>pp)
27         break;
28     if (pp % x == 0){
29         while (pp % x == 0)
30             pp /= x;
31         v.push_back(x);
32     }
33 }
34 if (pp != 1){
35     v.push_back(pp);
36 }
37 while (true){
38     long long a = primes[rand()%5000]%n;
39     if (gcd(a, n) != 1)
40         continue;
41     bool bb = false;
42     for (long long x:v)
43         if (pow(a, phi/x) == 1){
44             bb = true;
45             break;
46         }
47     if (!bb){
48         cout << a << "\n";
49         break;
50     }
51 }
52 // prime roots mod n ends
53
54 // simplex begins
55
56 const double EPS = 1e-9;
57
58 typedef vector<double> vdbl;
59
60 // n variables, m inequalities
61 // Ax <= b, c*x -> max, x >= 0

```

```

9  double simplex( int n, int m, const vector<vdbl>
    &a0, const vdbl &b, const vdbl &c, vdbl &x )
10  {
11      // Ax + Ez = b, A[m]*x -> max
12      // x = 0, z = b, x >= 0, z >= 0
13      vector<vdbl> a(m + 2, vdbl(n + m + 2));
14      vector<int> p(m);
15      forn(i, n)
16          a[m + 1][i] = c[i];
17      forn(i, m) {
18          forn(j, n)
19              a[i][j] = a0[i][j];
20          a[i][n + i] = 1;
21          a[i][m + n] = -1;
22          a[i][m + n + 1] = b[i];
23          p[i] = n + i;
24      }
25
26      // basis: enter "j", leave "ind+n"
27      auto pivot = [&]( int j, int ind ) {
28          double coef = a[ind][j];
29          assert(fabs(coef) > EPS);
30          forn(col, n + m + 2)
31              a[ind][col] /= coef;
32          forn(row, m + 2)
33              if (row != ind &&
34                  fabs(a[row][j]) >
35                  EPS) {
36              coef = a[row][j];
37              forn(col, n + m +
38                  2)
39                  a[row][col]
40                      <- -=
41                      <- a[ind][col]
42                      <- *
43                      <- coef;
44              a[row][j] = 0; //
45              <- reduce
46              <- precision
47              <- error
48          }
49      };
50      p[ind] = j;
51
52      // the Simplex itself
53      auto iterate = [&]( int nn ) {
54          for (int run = 1; run; ) {
55              run = 0;
56              forn(j, nn) {
57                  if (a[m][j] >
58                      EPS) { //
59                      <- strictly
60                      <- positive
61                      run = 1;
62                      double mi
63                          <- =
64                          <- INFINITY,
65                          <- t;
66                      int ind =
67                          <- -1;

```

```

50     forn(i, 74                                     pivot(j, i);
51         ↪ m) 75                                     }
52         if 76                                     }
53         77 ↪ (a[i][j] swap(a[m], a[m + 1]));
54         78 ↪ > if (!iterate(n + m))
55         79 ↪ EPS return INFINITY;
56         80 ↪ && x = vdbl(n, 0);
57         81 ↪ (t forn(i, m)
58         82 ↪ = if (p[i] < n)
59         83 ↪ a[i][n x[p[i]] = a[i][n + m +
60         ↪ + ↪ 1];
61         84 ↪ m return -a[m][n + m + 1];
62         85 ↪ }+
63         86 ↪ 1]
64         87 ↪ // simplex usage:
65         88 ↪ val f(m);
66         89 ↪ double result = simplex(n, m, a, b, c, x);
67         90 ↪ if (isinf(result))
68         91 ↪ - puts("Unbounded");
69         92 ↪ eEPS if (isnan(result))
70         93 ↪ mi puts("No solution");
71         94 ↪ else { ↪ =
72         95 ↪ ↪ printf("%.9f :", result);
73         96 ↪ ↪ find(i, n)
74         97 ↪ ↪ = printf(" %.9f", x[i]);
75         98 ↪ ↪ puts("");
76     if (ind 99 }
77     ↪ == 100
78     ↪ -1) 101 // simplex ends
79     return
80     ↪ 1 ↪ // false; over subsets begins
81     pivot(j, 2 // fast subset convolution O(n 2^n)
82     ↪ ind) 3 for(int i = 0; i < (1<<N); ++i)
83     ↪ 4 F[i] = A[i];
84     ↪ 5 for(int i = 0; i < N; ++i) for(int mask = 0; mask
85     ↪ ↪ < (1<<N); ++mask){
86     ↪ 6 if(mask & (1<<i))
87     ↪ 7 F[mask] += F[mask^(1<<i)];
88     ↪ 8 }
89     int mi = min_element(b.begin(), b.end()) 9 // sum over subsets ends
90     ↪ - b.begin();
91     if (b[mi] < -EPS) { 1 // algebra begins
92     ↪ 2
93     ↪ 3 Pick
94     ↪ 4 B + Γ / 2 - 1,
95     ↪ 5 где B - количество целочисленных точек внутри
96     ↪ ↪ многоугольника, а Γ - количество
97     ↪ ↪ целочисленных точек на границе
98     ↪ ↪ многоугольника.
99     forn(i, m) 6
100     ↪ if (p[i] == m + n) { 7
101     ↪ ↪ int j = 0; 8 Newton
102     ↪ ↪ while 9 x_{i+1}=x_i-f(x_i)/f'(x_i)
103     ↪ ↪ ↪ (find(p.begin(), 10
104     ↪ ↪ ↪ p.end(), j) 11 Catalan
105     ↪ ↪ ↪ != p.end() || 12 C_n=\sum_{limits_{k=0}^{n-1}} C_{k} C_{n-1-k}
106     ↪ ↪ ↪ fabs(a[i][j]) 13 C_i=\frac{1}{n+1} \binom{2n}{n}
107     ↪ ↪ ↪ < EPS) 14
108     ↪ ↪ ↪ j++, 15
109     ↪ ↪ ↪ ↪ assert(G_N:=2^{n(n-1)/2}
110     ↪ ↪ ↪ ↪ ↪ Количество связанных помеченных графов
111     ↪ ↪ ↪ ↪ ↪ Conn_N = G_N - \frac{1}{N} \sum_{limits_{K=1}^{N-1}}
112     ↪ ↪ ↪ ↪ ↪ ↪ K \binom{N}{K} Conn_K G_{N-K}

```

```

17
18 Количество помеченных графов с K компонентами
   ↳ связности
19  $D[N][K] = \sum_{limits_{\{S=1\}}^N \binom{N-1}{S-1} \text{Conn}_S D[N-S][K-1]$ 
   ↳ Conn_S D[N-S][K-1]
20
21 Miller-Rabbin
22  $a = a^t$ 
23 FOR  $i = 1 \dots s$ 
24   if  $a^{2^i} \neq 1$  &&  $|a| \neq 1$ 
25     NOT PRIME
26    $a = a^2$ 
27 return  $a = 1 ? \text{PRIME} : \text{NOT PRIME}$ 
28
29
30 Интегрирование по формуле Симпсона
31  $\int_a^b f(x) dx \approx ?$ 
32  $x_i := a + ih, i = 0 \dots 2n$ 
33  $h = \frac{b-a}{2n}$ 
34
35  $\int_a^b f(x) dx \approx \frac{h}{3} (f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \dots + 4f(x_{2n-1}) + f(x_{2n}))$ 
36 Погрешность имеет порядок уменьшения как  $O(n^{-4})$ .
37
38 // algebra ends
39
40 // wavelet tree begins
41
42 struct wavelet_tree{
43   int lo, hi;
44   wavelet_tree *l, *r;
45   vi b;
46
47   //nos are in range [x,y]
48   //array indices are [from, to]
49   wavelet_tree(int *from, int *to, int x,
50     ↳ int y){
51     lo = x, hi = y;
52     if(lo == hi or from >= to)
53       ↳ return;
54
55     int mid = (lo+hi)/2;
56     auto f = [mid](int x){
57       return x <= mid;
58     };
59     b.reserve(to-from+1);
60     b.pb(0);
61     //b[i] = no of elements from
62     ↳ first "i" elements that go to
63     ↳ left node
64     for(auto it = from; it != to;
65       ↳ it++)
66       b.pb(b.back() + f(*it));
67
68     //see how lambda function is used
69     ↳ here
70     auto pivot =
71     ↳ stable_partition(from, to,
72     ↳ f);
73
74     l = new wavelet_tree(from, pivot,
75     ↳ lo, mid);
76     r = new wavelet_tree(pivot, to,
77     ↳ mid+1, hi);
78   }
79
80   //kth smallest element in [l, r]
81   int kth(int l, int r, int k){
82     if(l > r) return 0;
83     if(lo == hi) return lo;
84     //how many nos are there in left
85     ↳ node from [l, r]
86     int inleft = b[r] - b[l-1];
87     int lb = b[l-1]; //amt of nos
88     ↳ from first (l-1) nos that go
89     ↳ in left
90     int rb = b[r]; //amt of nos from
91     ↳ first (r) nos that go in left
92     //so [lb+1, rb] represents nos
93     ↳ from [l, r] that go to left
94     if(k <= inleft) return
95     ↳ this->l->kth(lb+1, rb, k);
96     //((l-1)-lb) is amt of nos from
97     ↳ first (l-1) nos that go to
98     ↳ right
99     //((r)-rb) is amt of nos from first
100    ↳ (r) nos that go to right
101    //so [l-lb, r-rb] represents nos
102    ↳ from [l, r] that go to right
103    return this->r->kth(l-lb, r-rb,
104    ↳ k-inleft);
105  }
106 };
107
108 // wavelet tree ends
109
110 // berlecamp-massey begins
111 const int SZ = MAXN;
112 ll qp(ll a, ll b) {
113   ll x = 1;
114   a %= MOD;
115   while (b) {
116     if (b & 1) x = x * a % MOD;
117     a = a * a % MOD;
118     b >>= 1;
119   }
120   return x;
121 }
122
123 namespace linear_seq {
124   inline vector<int> BM(vector<int> x) {
125     vector<int> ls, cur;
126     int lf, ld;
127     for (int i = 0; i < int(x.size()); ++i) {
128       ll t = 0;
129       for (int j = 0; j < int(cur.size());
130         ↳ ++j)
131         t = (t + x[i - j - 1] * (ll)
132         ↳ cur[j]) % MOD;
133     }
134   }
135 }

```



```

24     if ((t - x[i]) % MOD == 0) continue;75
25     if (!cur.size()) {
26         cur.resize(i + 1);76
27         lf = i;77
28         ld = (t - x[i]) % MOD;78
29         continue;79
30     }
31     ll k = -(x[i] - t) * qp(ld, MOD - 2);81
32     ↪ % MOD;82
33     vector<int> c(i - lf - 1);83
34     c.pb(k);84
35     for (int j = 0; j < int(ls.size());85
36         ↪ ++j)
37         c.pb(-ls[j] * k % MOD);86
38     if (c.size() < cur.size())87
39         ↪ c.resize(cur.size());88
40     for (int j = 0; j < int(cur.size());89
41         ↪ ++j)
42         c[j] = (c[j] + cur[j]) % MOD;
43     if (i - lf + (int) ls.size() >= (int)91
44         ↪ cur.size())
45         ls = cur, lf = i, ld = (t - x[i]);92
46         ↪ % MOD;93
47     cur = c;94
48 }95
49 for (int i = 0; i < int(cur.size()); ++i)96
50     cur[i] = (cur[i] % MOD + MOD) % MOD;97
51 return cur;98
52 }99
53
54 int m;100
55 ll a[SZ], h[SZ], t_[SZ], s[SZ], t[SZ];101
56
57 inline void mull(ll* p, ll* q) {103
58     for (int i = 0; i < m + m; ++i) t_[i] =104
59     ↪ 0;
60     for (int i = 0; i < m; ++i)
61         if (p[i])
62             for (int j = 0; j < m; ++j)
63                 t_[i + j] = (t_[i + j] + p[i]
64                     ↪ * q[j]) % MOD;
65     for (int i = m + m - 1; i >= m; --i)
66         if (t_[i])
67             for (int j = m - 1; ~j; --j)
68                 t_[i - j - 1] = (t_[i - j -
69                     ↪ 1] + t_[i] * h[j]) % MOD;
70     for (int i = 0; i < m; ++i) p[i] = t_[i];10
71 }11
72
73 inline ll calc(ll K) {12
74     for (int i = m; ~i; --i)13
75         s[i] = t[i] = 0;14
76     s[0] = 1;15
77     if (m != 1) t[1] = 1; else t[0] = h[0];16
78     while (K) {17
79         if (K & 1) mull(s, t);18
80         mull(t, t);19
81         K >>= 1;20
82     }21
83     ll su = 0;22

```

```

for (int i = 0; i < m; ++i) su = (su +
    ↪ s[i] * a[i]) % MOD;
return (su % MOD + MOD) % MOD;
}

inline int work(vector<int> x, ll n) {
    if (n < int(x.size())) return x[n];
    vector<int> v = BM(x);
    m = v.size();
    if (!m) return 0;
    for (int i = 0; i < m; ++i) h[i] = v[i],
    ↪ a[i] = x[i];
    return calc(n);
}

//b=a0/(1-p)
inline void calc_generating_function(const
    ↪ vector<int>& b, vector<int>& p,
    ↪ vector<int>& a0) {
    p = BM(b);
    a0.resize(p.size());
    for (int i = 0; i < a0.size(); ++i) {
        a0[i] = b[i];
        for (int j = 0; j < i; ++j) {
            a0[i] += MOD - (p[j] * 1ll * b[i
            ↪ - j - 1]) % MOD;
            if (a0[i] > MOD) {
                a0[i] -= MOD;
            }
        }
    }
}

// berlecamp-massey ends

// AND-FFT begins

void fast_fourier(vector<int>& a) { // AND-FFT.
    for (int k = 1; k < SZ(a); k *= 2)
        for (int start = 0; start < (1 << K);
            ↪ start += 2 * k) {
            for (int off = 0; off < k; ++off) {
                int a_val = a[start + off];
                int b_val = a[start + k + off];

                a[start + off] = b_val;
                a[start + k + off] = add(a_val,
                    ↪ b_val);
            }
        }
}

void inverse_fast_fourier(vector<int>& a) {
    for (int k = 1; k < SZ(a); k *= 2)
        for (int start = 0; start < (1 << K);
            ↪ start += 2 * k) {
            for (int off = 0; off < k; ++off) {
                int a_val = a[start + off];
                int b_val = a[start + k + off];

```

```

23         a[start + off] = sub(b_val,
24             ↪ a_val);
25         a[start + k + off] = a_val;
26     }
27 }
28
29 // AND-FFT ends
30
31 // 2-chinese begins
32
33 template <typename Info>
34 class DSU {
35 public:
36     DSU ( int n ) : jump (new int[n]), rank (new
37         ↪ int [n]), info (new Info [n]) {
38         for (int i = 0; i < n; i++) {
39             jump[i] = i;
40             rank[i] = 0;
41         }
42     }
43     Info& operator [] ( int x ) {
44         return info[get (x)];
45     }
46     void merge ( int a, int b, const Info
47         ↪ &comment ) {
48         a = get (a);
49         b = get (b);
50         if (rank[a] <= rank[b]) {
51             jump[a] = b;
52             rank[b] += rank[a] == rank[b];
53             info[b] = comment;
54         } else {
55             jump[b] = a;
56             info[a] = comment;
57         }
58     }
59 private:
60     int *jump, *rank;
61     Info *info;
62
63     int get ( int x ) {
64         return jump[x] == x ? x : (jump[x] = get
65             ↪ (jump[x]));
66     }
67 };
68
69 struct Treap {
70     int value, add;
71     int source, target, height;
72     int min_value, min_path;
73
74     Treap *left, *right;
75
76     Treap ( int _source, int _target, int _value )
77         ↪ : value (_value), add (0), source
78         ↪ (_source), target (_target) {
79         height = rand ();
80         min_value = value, min_path = 0;
81         left = right = 0;
82     }
83
84     Treap& operator += ( int sub ) {
85         add += sub;
86         return *this;
87     }
88
89     void push () {
90         if (!add)
91             return;
92         if (left) {
93             left->add += add;
94         }
95         if (right) {
96             right->add += add;
97         }
98         value += add;
99         min_value += add;
100         add = 0;
101     }
102
103     void recalc () {
104         min_value = value;
105         min_path = 0;
106         if (left && left->min_value + left->add <
107             ↪ min_value) {
108             min_value = left->min_value + left->add;
109             min_path = -1;
110         }
111         if (right && right->min_value + right->add <
112             ↪ min_value) {
113             min_value = right->min_value + right->add;
114             min_path = +1;
115         }
116     }
117
118     Treap* treap_merge ( Treap *x, Treap *y ) {
119         if (!x)
120             return y;
121         if (!y)
122             return x;
123         if (x->height < y->height) {
124             x->push ();
125             x->right = treap_merge (x->right, y);
126             x->recalc ();
127             return x;
128         } else {
129             y->push ();
130             y->left = treap_merge (x, y->left);
131             y->recalc ();
132             return y;
133         }
134     }
135
136     Treap* treap_getmin ( Treap *x, int &source, int
137         ↪ &target, int &value ) {
138         assert (x);
139         x->push ();
140         if (x->min_path == 0) {
141             // memory leak, sorry
142             source = x->source;

```

```

107     target = x->target;
108     value = x->value + x->add;
109     return treap_merge (x->left, x->right);
110 } else if (x->min_path == -1) {
111     x->left = treap_getmin (x->left, source,
112         ↪ target, value);
113     value += x->add;
114     x->recalc ();
115     return x;
116 } else if (x->min_path == +1) {
117     x->right = treap_getmin (x->right, source,
118         ↪ target, value);
119     value += x->add;
120     x->recalc ();
121     return x;
122 } else
123     assert (0);
124 }
125
126 Treap* treap_add ( Treap *x, int add ) {
127     if (!x)
128         return 0;
129     return &((*x) += add);
130 }
131
132 int main () {
133     int n, m;
134     while (scanf ("%d%d", &n, &m) == 2) {
135         Treap * g[n + 1];
136         for (int i = 0; i <= n; i++)
137             g[i] = 0;
138         for (int i = 1; i <= n; i++) {
139             int a;
140             assert (scanf ("%d", &a) == 1);
141             g[i] = treap_merge (g[i], new Treap (i, 0,
142                 ↪ a));
143         }
144         n++;
145         for (int i = 0; i < m; i++) {
146             int a, b, c;
147             assert (scanf ("%d%d%d", &a, &b, &c) == 3);
148             g[b] = treap_merge (g[b], new Treap (b, a,
149                 ↪ c));
150         }
151         DSU <pair <int, Treap*> > dsu (n + 1);
152         for (int i = 0; i < n; i++) {
153             dsu[i] = make_pair (i, g[i]);
154         }
155
156         int ans = 0, k = n;
157         int jump[2 * n], jump_from[2 * n], parent[2 * n]
158             ↪ n], c[n];
159         vector <int> children[2 * n];
160         memset (c, 0, sizeof (c[0]) * n);
161         memset (parent, -1, sizeof (parent[0]) * 2 * n);
162         vector <int> finish;
163         for (int i = 0; i < n; i++) {
164             if (dsu[i].first == 0)
165                 continue;
166
167             int u = i;
168             c[u] = 1;
169             while (true) {
170                 int source, target, value;
171                 dsu[u].second = treap_getmin
172                     ↪ (dsu[u].second, source, target,
173                     ↪ value);
174                 if (dsu[target] == dsu[u])
175                     continue;
176                 treap_add (dsu[u].second, -value);
177                 ans += value;
178                 jump_from[dsu[u].first] = source;
179                 jump[dsu[u].first] = target;
180                 if (dsu[target].first == 0)
181                     break;
182                 if (!c[target]) {
183                     c[target] = 1;
184                     u = target;
185                     continue;
186                 }
187                 assert (k < 2 * n);
188                 int node = k++, t = target;
189                 parent[dsu[u].first] = node;
190                 children[node].push_back (dsu[u].first);
191                 dsu[u].first = node;
192                 Treap *v = dsu[u].second;
193                 while (dsu[t].first != node) {
194                     int next = jump[dsu[t].first];
195                     parent[dsu[t].first] = node;
196                     children[node].push_back
197                         ↪ (dsu[t].first);
198                     v = treap_merge (v, dsu[t].second);
199                     dsu.merge (u, t, make_pair (node, v));
200                     t = next;
201                 }
202             }
203             u = i;
204             while (dsu[u].first) {
205                 int next = jump[dsu[u].first];
206                 finish.push_back (dsu[u].first);
207                 dsu.merge (u, 0, make_pair (0, (Treap
208                     ↪ *)0));
209                 u = next;
210             }
211         }
212         bool ok[k];
213         int res[n];
214         memset (ok, 0, sizeof (ok[0]) * k);
215         memset (res, -1, sizeof (res[0]) * n);
216         function <void (int, int)> add_edge = [&ok,
217             ↪ &parent, &res, &n] ( int a, int b ) {
218             assert (0 <= a && a < n);
219             assert (0 <= b && b < n);
220             assert (res[a] == -1);
221             res[a] = b;
222             while (a != -1 && !ok[a]) {
223                 ok[a] = true;
224                 a = parent[a];
225             }
226         };

```

```

217 function <void (int)> reach = [&ok, &reach, 39
    ↪ &children, &jump, &jump_from, &add_edge](
    ↪ int u ) { 41
218     if (!ok[u]) 42
219         add_edge (jump_from[u], jump[u]); 43
220     for (auto x : children[u]) 44
221         reach (x); 45
222 } 46
223 for (auto x : finish) 47
224     reach (x);
225 printf ("%d\n", ans); 48
226 for (int i = 1; i < n; i++) 49
227     printf ("%d%c", res[i] ? res[i] : -1, "\n" 50
    ↪ "[i < n - 1]); 51
228 } 52
229 return 0; 53
230 } 54
231 55
232 // 2-chinese ends 56
57 // slow min circulation begins 58
59 60
61 struct Edge { 62
63     int a; 64
65     int b; 66
67     int cost; 68
69 }; 69
70 71
72 vector<int> negative_cycle(int n, vector<Edge> 72
    ↪ &edges) { 73
74     // O(nm), return ids of edges in negative 74
    ↪ cycle 75
76 76
77     vector<int> d(n); 77
78     vector<int> p(n, -1); // last edge ids 78
79 79
80     const int inf = 1e9; 80
81 81
82     int x = -1; 82
83     for (int i = 0; i < n; i++) { 83
84         x = -1; 84
85         for (int j = 0; j < edges.size(); j++) { 85
86             Edge &e = edges[j]; 86
87 87
88             if (d[e.b] > d[e.a] + e.cost) { 88
89                 d[e.b] = max(-inf, d[e.a] + 89
    ↪ e.cost); 90
91                 p[e.b] = j; 91
92                 x = e.b; 92
93             } 93
94         } 94
95     } 95
96 96
97     if (x == -1) 97
98         return vector<int>(); // no negative 98
    ↪ cycle 99
99 99
100     for (int i = 0; i < n; i++) 100
101         x = edges[p[x]].a; 101
102 102
103     vector<int> result; 103
104     for (int cur = x; ; cur = edges[p[cur]].a) { 104
105         if (cur == x && result.size() > 0) break; 105
106         result.push_back(p[cur]); 106
107     } 107
108     reverse(result.begin(), result.end()); 108
109     return result; 109
110 } 110
111 111
112 vector<int> min_avg_cycle(int n, vector<Edge> 112
    ↪ &edges) { 113
114     const int inf = 1e3; 114
115 115
116     for (auto &e : edges) 116
117         e.cost *= n * n; 117
118 118
119     int l = -inf; 119
120     int r = inf; 120
121     while (l + 1 < r) { 121
122         int m = (l + r) / 2; 122
123         for (auto &e : edges) 123
124             e.cost -= m; 124
125 125
126         if (negative_cycle(n, edges).empty()) 126
127             l = m; 127
128         else 128
129             r = m; 129
130 130
131         for (auto &e : edges) 131
132             e.cost += m; 132
133     } 133
134 134
135     if (r >= 0) // if only negative needed 135
136         return vector<int>(); 136
137 137
138     for (auto &e : edges) 138
139         e.cost -= r; 139
140 140
141     vector<int> result = negative_cycle(n, 141
    ↪ edges); 142
143 142
144     for (auto &e : edges) 144
145         e.cost += r; 145
146 146
147     for (auto &e : edges) 147
148         e.cost /= n * n; 148
149 149
150     return result; 150
151 } 151
152 152
153 struct edge { 153
154     int from, to; 154
155     int c, f, cost; 155
156 }; 156
157 157
158 const int max_n = 200; 158
159 159
160 vector<int> gr[max_n]; 160
161 vector<edge> edges; 161
162 162
163 void add(int fr, int to, int c, int cost) { 163

```

```

98     gr[fr].push_back(edges.size());
99     edges.push_back({fr, to, c, 0, cost});
100    gr[to].push_back(edges.size());
101    edges.push_back({to, fr, 0, 0, -cost}); //
    ↪ single
102 }
103
104 void calc_min_circulation(int n) {
105     while (true) {
106         vector<Edge> eds;
107         vector<int> origin;
108
109         for (int i = 0; i < edges.size(); i++) {
110             edge &e = edges[i];
111             if (e.c - e.f > 0) {
112                 eds.push_back({e.from, e.to,
113                               ↪ e.cost});
114                 origin.push_back(i);
115             }
116         }
117
118         vector<int> cycle = negative_cycle(n,
119     ↪ eds);
120
121         if (cycle.empty())
122             break;
123
124         for (auto id : cycle) {
125             int x = origin[id];
126             edges[x].f += 1;
127             edges[x ^ 1].f -= 1;
128         }
129     }
130 }
131 // slow min circulation ends

```

DM

Кол-во корневых деревьев:

$$t(G) = \frac{1}{n} \lambda_2 \dots \lambda_n \quad (\lambda_1 = 0)$$

Кол-во эйлеровых циклов:

$$e(D) = t^-(D, x) \cdot \prod_{y \in D} (outdeg(y) - 1)!$$

```

1 // fast hashtable begins
2
3 #include <ext/pb_ds/assoc_container.hpp>
4 using namespace __gnu_pbds;
5 gp_hash_table<int, int> table;
6
7 const int RANDOM =
    ↪ chrono::high_resolution_clock::now().time_since_epoch().count();
8 struct chash {
9     int operator()(int x) { return hash<int>{}(x
    ↪ ^ RANDOM); }
10 };
11 gp_hash_table<key, int, chash> table;
12
13 // fast hashtable ends

```

Наличие совершенного паросочетания:

$T$  – матрица с нулями на диагонали. Если есть ребро  $(i, j)$ , то  $a_{i,j} := x_{i,j}$ ,  $a_{j,i} = -x_{i,j}$   
 $\det(T) = 0 \Leftrightarrow$  нет совершенного паросочетания.

## **Whitespace code FFT**