

Team Reference of St. Petersburg Campus of Higher School of Economics.  
SPb HSE: Abstract Economists  
(Ermilov, Fedorov, Labutin)



```

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 = 1,
22     ↪ 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,

```

**Алгебра Pick**

$$B + \Gamma / 2 - 1 = \text{AREA},$$

где  $B$  — количество целочисленных точек внутри многоугольника, а  $\Gamma$  — количество целочисленных точек на границе многоугольника.

**Newton**

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

**Catalan**

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

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

**Кол-во графов**

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

Количество связных 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

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

**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

```

**Интегрирование по формуле Симпсона**

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

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

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

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

$$O(n^4).$$

**Простые числа**

```

1009,1013;10007,10009;100003,100019
1000003,1000033;10000019,10000079
100000007,100000037
10000000019,10000000033
1000000000039,1000000000061
10000000000031,10000000000067
1000000000000061,1000000000000069
10000000000000003,10000000000000009

```

**Числа для Фурье**

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

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

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

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

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

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

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

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

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

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

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

### 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

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

```

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 // sk fast allocation ends
12
13
14
15
16 // sk fast read-write begins
17
18 inline int readChar();
19 template <class T = int> inline T readInt();
20 template <class T> inline void writeInt( T x,
21     ↪ char end = 0 );
22 inline void writeChar( int x );
23 inline void writeWord( const char *s );
24
25 /** Read */
26
27 static const int buf_size = 2048;
28
29 inline int getChar() {
30     static char buf[buf_size];
31     static int len = 0, pos = 0;
32     if (pos == len)
33         pos = 0, len = fread(buf, 1, buf_size,
34             ↪ stdin);
35     if (pos == len)
36         return -1;
37     return buf[pos++];
38 }
39
40 inline int readWord(char * buffer) {
41     int c = getChar();
42     while (c <= 32) {
43         c = getChar();
44     }
45
46     int len = 0;
47     while (c > 32) {
48         *buffer = (char) c;
49         c = getChar();
50         buffer++;
51         len++;
52     }
53     return len;
54 }
55
56 inline int readChar() {
57     int c = getChar();
58     while (c <= 32)
59         c = getChar();
60     return c;

```

```

59 }
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, stdout),
108             ↪ 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 // kto begins
18 //return pair(nmod,nr)
19 //nr%mod1=r1, nr%mod2=r2
20 //nmod=mod1*mod2/gcd(mod1,mod2)
21 //if input incosistent return mp(-1,-1)
22 pll kto( ll mod1, ll r1, ll mod2, ll r2 )
23 {
24     ll d=__gcd(mod1,mod2);
25     if (r1%d!=r2%d)
26         return mp(-1,-1);
27     ll rd=r1%d;
28     mod1/=d, mod2/=d, r1/=d, r2/=d;
29
30     if (mod1<mod2)
31         swap(mod1,mod2), swap(r1,r2);
32
33     ll k=(r2-r1)%mod2;
34     if (k<0)
35         k+=mod2;
36
37     ll x, y;
38     gcdex(mod1,mod2,x,y);
39     x%=mod2;
40     if (x<0)
41         x+=mod2;
42     k*=x, k%=mod2;
43     return mp(mod1*mod2*d, (k*mod1+r1)*d+rd);
44 }
45 // kto ends
46
47 // FFT begins
48
49 const int MAX_LOG = 17;
50 const int MAXN = (1 << MAX_LOG);
51 int LOG = MAX_LOG;
52 int N = MAXN;
53
54 typedef std::complex<double> cd;
55
56 int rev[MAXN];
57 cd W[MAXN];
58
59 void precalc() {
60     const double pi = std::acos(-1);
61     for (int i = 0; i != N; ++i)
62         W[i] = cd(std::cos(2 * pi * i / N),
63             ↪ std::sin(2 * pi * i / N));
64
65     int last = 0;
66     for (int i = 1; i != N; ++i) {
67         if (i == (2 << last))
68             ++last;
69     }

```

```

22         rev[i] = rev[i ^ (1 << last)] | (1 << (LOG
23         ↪ 1 - last));
24     }
25 }
26
27 void fft(vector<cd>& a) {
28     for (int i = 0; i != N; ++i)
29         if (i < rev[i])
30             std::swap(a[i], a[rev[i]]);
31
32     for (int lvl = 0; lvl != LOG; ++lvl)
33         for (int start = 0; start != N; start += (2
34         ↪ << lvl))
35             for (int pos = 0; pos != (1 << lvl); ++pos) // FFT ends
36                 {
37                     cd x = a[start + pos];
38                     cd y = a[start + pos + (1 << lvl)];
39
40                     y *= W[pos << (LOG - 1 - lvl)];
41
42                     a[start + pos] = x + y;
43                     a[start + pos + (1 << lvl)] = x - y;
44                 }
45 }
46
47 void inv_fft(vector<cd>& a) {
48     fft(a);
49     std::reverse(a.begin() + 1, a.begin() + N);
50
51     for (cd& elem: a)
52         elem /= N;
53 }
54
55 vector<cd> mul_fft(vector<cd> a, vector<cd> b,
56 ↪ int n = N) {
57     if (N != n) {
58         N = n;
59         LOG = round(log2(N));
60         precalc();
61     }
62     fft(a);
63     fft(b);
64
65     vector<cd> c(n);
66     for (int i = 0; i < n; i++) {
67         c[i] = a[i] * b[i];
68     }
69
70     inv_fft(c);
71     return c;
72 }
73
74 vector<cd> inv_poly(vector<cd>& a, int n = N) {
75     if (n == 1) {
76         vector<cd> res(1);
77         res[0] = cd(1) / a[0];
78         return res;
79     }
80
81     vector<cd> r = inv_poly(a, n / 2);
82     r.resize(n);
83     vector<cd> q = mul_fft(a, r, n);
84     for (int i = 0; i < n / 2; i++) {
85         q[i] = -q[n / 2 + i];
86         q[n / 2 + i] = 0;
87     }
88     vector<cd> c = mul_fft(q, r, n);
89     for (int i = n / 2; i < n; i++) {
90         r[i] = c[i - n / 2];
91     }
92     return r;
93 }
94
95 // fast gauss begins
96
97 using elem_t = int;
98 // a[i][rows[i][j].first]=rows[i][j].second;
99 ↪ b[i]=a[i][n]
100
101 bool gauss(vector<vector<pair<int, elem_t>>>
102 ↪ rows, vector<elem_t> &res) {
103     int n = rows.size();
104
105     res.resize(n + 1, 0);
106     vector<int> p(n + 1);
107     iota(p.begin(), p.end(), 0);
108     vector<int> toZero(n + 1, -1);
109     vector<int> zro(n + 1);
110     vector<elem_t> a(n + 1);
111
112     // optional: sort rows
113
114     sort(p.begin(), p.begin() + n, [&rows](int i,
115 ↪ int j) { return rows[i].size() <
116 ↪ rows[j].size(); });
117     vector<int> invP(n + 1);
118     vector<vector<pair<int, elem_t>>> rs(n);
119     for (int i = 0; i < n; i++) {
120         invP[p[i]] = i;
121         rs[i] = rows[p[i]];
122     }
123     for (int i = 0; i < n; i++) {
124         rows[i] = rs[i];
125         for (auto& el: rows[i]) {
126             if (el.first < n) {
127                 el.first = invP[el.first];
128             }
129         }
130     }
131
132     for (int i = 0; i < n; i++) {
133         for (auto& el: rows[i]) {
134             a[el.first] = el.second;
135         }
136         while (true) {
137             int k = -1;
138             for (auto& el: rows[i]) {
139                 if (!isZero(a[el.first]) &&
140 ↪ toZero[el.first] != -1 &&

```

```

42         (k == -1 || toZero[el.first] <
43         ↪ toZero[k])) {
44     }
45 }
46 if (k == -1)
47     break;
48
49 int j = toZero[k];
50 elem_t c = a[k];
51
52 for (auto el: rows[j]) {
53     if (isZero(a[el.first]))
54         rows[i].emplace_back(el.first, 0);
55     a[el.first] = sub(a[el.first], mult(c,
56     ↪ el.second));
57 }
58
59 auto cond = [&a](const pair<int, elem_t>&
60 ↪ p) { return isZero(a[p.first]); };
61 rows[i].erase(std::remove_if
62 ↪ (rows[i].begin(), rows[i].end(), cond),
63 ↪ rows[i].end());
64 }
65
66 bool ok = false;
67 for (auto& el: rows[i]) {
68     if (el.first < n && !isZero(a[el.first])) {
69         toZero[el.first] = i;
70         zro[i] = el.first;
71         // det = (det * a[el.first]) % MOD;
72
73         elem_t c = divM(1, a[el.first]);
74         for (auto& el: rows[i]) {
75             el.second = mult(a[el.first], c);
76             a[el.first] = 0;
77         }
78
79         ok = true;
80         break;
81     }
82 }
83
84 if (!ok) {
85     // det = 0;
86     return false;
87 }
88
89 res[n] = sub(0, 1);
90 for (int i = n - 1; i >= 0; i--) {
91     int k = zro[i];
92     for (auto& el: rows[i])
93         if (el.first != k)
94             res[p[k]] = sub(res[p[k]],
95             ↪ mult(el.second, res[p[el.first]]));
96 }
97
98 return true;
99 }

```

```

97 // fast gauss ends
98
99 // stable gauss begins
100 // if at least one solution returns it in ans
101 int gauss (vector < vector<double> > a,
102 ↪ vector<double> & ans) {
103     int n = (int) a.size();
104     int m = (int) a[0].size() - 1;
105
106     vector<int> where (m, -1);
107     for (int col=0, row=0; col<m && row<n; ++col) {
108         int sel = row;
109         for (int i=row; i<n; ++i)
110             if (abs (a[i][col]) > abs (a[sel][col]))
111                 sel = i;
112         if (abs (a[sel][col]) < EPS)
113             continue;
114         for (int i=col; i<=m; ++i)
115             swap (a[sel][i], a[row][i]);
116         where[col] = row;
117
118         for (int i=0; i<n; ++i)
119             if (i != row) {
120                 double c = a[i][col] / a[row][col];
121                 for (int j=col; j<=m; ++j)
122                     a[i][j] -= a[row][j] * c;
123             }
124         ++row;
125     }
126
127     ans.assign (m, 0);
128     for (int i=0; i<m; ++i)
129         if (where[i] != -1)
130             ans[i] = a[where[i]][m] / a[where[i]][i];
131     for (int i=0; i<n; ++i) {
132         double sum = 0;
133         for (int j=0; j<m; ++j)
134             sum += ans[j] * a[i][j];
135         if (abs (sum - a[i][m]) > EPS)
136             return 0;
137     }
138
139     for (int i=0; i<m; ++i)
140         if (where[i] == -1)
141             return INF;
142     return 1;
143 }
144
145 // stable gauss ends
146
147 // simple geometry begins
148
149 struct Point {
150     double x, y;
151     Point operator+(const Point& p) const { return
152     ↪ {x + p.x, y + p.y}; }
153     Point operator-(const Point& p) const { return
154     ↪ {x - p.x, y - p.y}; }
155     Point operator*(const double d) const { return
156     ↪ {x * d, y * d}; }
157     Point rotate() const { return {y, -x}; }

```

```

9  double operator*(const Point& p) const { return
    ↪ x * p.x + y * p.y; }
10 double operator^(const Point& p) const { return
    ↪ x * p.y - y * p.x; }
11 double dist() const { return sqrt(x * x + y * y); }
12 };
13
14 struct Line {
15     double a, b, c;
16     Line(const Point& p1, const Point& p2) {
17         a = p1.y - p2.y;
18         b = p2.x - p1.x;
19         c = - a * p1.x - b * p1.y;
20
21         double d = sqrt(sqr(a) + sqr(b));
22         a /= d, b /= d, c /= d;
23     }
24     bool operator||(const Line& l) const { return
    ↪ fabs(a * l.b - l.a * b) < EPS; }
25     double dist(const Point& p) const { return
    ↪ fabs(a * p.x + b * p.y + c); }
26     Point operator^(const Line& l) const {
27         return {(l.c * b - c * l.b) / (a * l.b - l.a *
    ↪ b),
28                 (l.c * a - c * l.a) / (l.a * b - a *
    ↪ l.b)};
29     }
30     Point projection(const Point& p) const {
31         return p - Point{a, b} * (a * p.x + b * p.y +
    ↪ c);
32     }
33 };
34
35 struct Circle {
36     Point c;
37     double r;
38     Circle(const Point& c, double r) : c(c), r(r) {}
39     Circle(const Point& a, const Point& b, const
    ↪ Point& c) {
40         Point p1 = (a + b) * 0.5, p2 = (a + c) * 0.5;
41         Point q1 = p1 + (b - a).rotate(), q2 = p2 +
    ↪ (c - a).rotate();
42         this->c = Line(p1, q1) ^ Line(p2, q2);
43         r = (a - this->c).dist();
44     }
45 };
46
47 inline bool on_segment(const Point& p1, const
    ↪ Point& p2, const Point& x, bool strictly) {
48     if (fabs((p1 - x) ^ (p2 - x)) > EPS)
49         return false;
50     return (p1 - x) * (p2 - x) < (strictly ? - EPS10
    ↪ : EPS);
51 }
52
53 // in case intersection is not a segment
54 inline bool intersect_segments(const Point& p1,
    ↪ const Point& p2, const Point& q1, const
    ↪ Point& q2, Point& x) {
    Line l1(p1, p2), l2(q1, q2);
    if (l1 || l2) return false;
    x = l1 ^ l2;
    return on_segment(p1, p2, x, false);
}
// in case circles are not equal
inline bool intersect_circles(const Circle& c1,
    ↪ const Circle& c2, Point& p1, Point& p2) {
    double d = (c2.c - c1.c).dist();
    if (d > c1.r + c2.r + EPS || d < fabs(c1.r -
    ↪ c2.r) - EPS)
        return false;
    double cosa = (sqr(d) + sqr(c1.r) - sqr(c2.r))
    ↪ / (2 * c1.r * d);
    double l = c1.r * cosa, h = sqrt(sqr(c1.r) -
    ↪ sqr(l));
    Point v = (c2.c - c1.c) * (1 / d), p = c1.c + v
    ↪ * l;
    p1 = p + v.rotate() * h, p2 = p - v.rotate() *
    ↪ h;
    return true;
}
inline bool intersect_circle_and_line(const
    ↪ Circle& c, const Line& l, Point& p1, Point&
    ↪ p2) {
    double d = l.dist(c.c);
    if (d > c.r + EPS)
        return false;
    Point p = l.projection(c.c);
    Point n{l.b, -l.a};
    double h = sqrt(sqr(c.r) - sqr(l.dist(c.c)));
    p1 = p + n * h, p2 = p - n * h;
    return true;
}
// simple geometry ends
// convex hull begins
struct Point {
    int x, y;
    Point operator-(const Point& p) const { return
    ↪ {x - p.x, y - p.y}; }
    int64_t operator^(const Point& p) const {
    ↪ return x * 1ll * p.y - y * 1ll * p.x; }
    int64_t dist() const { return x * 1ll * x + y *
    ↪ 1ll * y; }
    bool operator<(const Point& p) const { return x
    ↪ != p.x ? x < p.x : y < p.y; }
};
// all point on convex hull are included
vector<Point> convex_hull(vector<Point> pt) {
    int n = pt.size();
    Point p0 = *std::min_element(pt.begin(),
    ↪ pt.end());
    std::sort(pt.begin(), pt.end(), [&p0](const
    ↪ Point& a, const Point& b) {
        int64_t cp = (a - p0) ^ (b - p0);

```



```

17     return cp != 0 ? cp > 0 : (a - p0).dist() <
    ↪ (b - p0).dist();
18 });
19
20 int i = n - 1;
21 for (; i > 0 && ((pt[i] - p0) ^ (pt[i - 1] -
    ↪ p0)) == 0; i--);
22 std::reverse(pt.begin() + i, pt.end());
23
24 vector<Point> ch;
25 for (auto& p : pt) {
26     while (ch.size() > 1) {
27         auto& p1 = ch[(int) ch.size() - 1];
28         auto& p2 = ch[(int) ch.size() - 2];
29         int64_t cp = (p1 - p2) ^ (p - p1);
30         if (cp >= 0) break;
31         ch.pop_back();
32     }
33     ch.push_back(p);
34 }
35
36 return ch;
37 }
38
39 // convex hull ends
40
41 // convex hull trick begins
42
43 typedef long long ftype;
44 typedef complex<ftype> point;
45 #define x real
46 #define y imag
47
48 ftype dot(point const& a, point const& b) {
49     return (conj(a) * b).x();
50 }
51
52 ftype f(point const& a, int x) {
53     return dot(a, {compressed[x], 1});
54     //return dot(a, {x, 1});
55 }
56
57 int pos = 0;
58
59 // (x, y) -> (k, b) -> kb + x
60 struct li_chao { // for min
61     vector<point> line;
62
63     li_chao(int maxn) {
64         line.resize(4 * maxn, {0, inf});
65     }
66
67     void add_line(int v, int l, int r, int a, int
    ↪ b, point nw) {
68         if (r <= a || b <= l) return; // remove if no
    ↪ [a, b] query
69
70         int m = (l + r) >> 1;
71
72         if (!(a <= l && r <= b)) { // remove if no
    ↪ [a, b] query
73             add_line(v + v + 1, l, m, a, b, nw);
74
75             add_line(v + v + 2, m, r, a, b, nw);
76             return;
77         }
78
79         bool lef = f(nw, l) < f(line[v], l);
80         bool mid = f(nw, m) < f(line[v], m);
81
82         if (mid) swap(line[v], nw);
83
84         if (l == r - 1)
85             return;
86
87         if (lef != mid)
88             add_line(v + v + 1, l, m, a, b, nw);
89         else
90             add_line(v + v + 2, m, r, a, b, nw);
91     }
92
93     ftype get(int v, int l, int r, int x) {
94         if (l == r - 1)
95             return f(line[v], x);
96         int m = (l + r) / 2;
97         if (x < m) {
98             return min(f(line[v], x), get(v + v + 1, l,
    ↪ m, x));
99         } else {
100             return min(f(line[v], x), get(v + v + 2, m,
    ↪ r, x));
101         }
102     }
103 } cdt(maxn);
104
105 // convex hull with stack
106
107 ftype cross(point a, point b) {
108     return (conj(a) * b).y();
109 }
110
111 vector<point> hull, vecs;
112
113 void add_line(ftype k, ftype b) {
114     point nw = {k, b};
115     while (!vecs.empty() && dot(vecs.back(), nw -
    ↪ hull.back()) < 0) {
116         hull.pop_back();
117         vecs.pop_back();
118     }
119     if (!hull.empty()) {
120         vecs.push_back(li * (nw - hull.back()));
121     }
122     hull.push_back(nw);
123 }
124
125 int get(ftype x) {
126     point query = {x, 1};
127     auto it = lower_bound(vecs.begin(), vecs.end(),
    ↪ query, [](point a, point b) {
128         return cross(a, b) > 0;
129     });
130     return dot(query, hull[it - vecs.begin()]);

```



```

91 }
92
93 // convex hull trick ends
94
1 // heavy-light begins
2
3 int sz[maxn];
4
5 void dfs_sz(int v, int par = -1) {
6     sz[v] = 1;
7     for (int x : gr[v])
8         if (x != par) {
9             dfs_sz(x, v);
10            sz[v] += sz[x];
11        }
12    for (int i = 0; i < gr[v].size(); i++)
13        if (gr[v][i] != par)
14            if (sz[gr[v][i]] * 2 >= sz[v]) {
15                swap(gr[v][i], gr[v][0]);
16                break;
17            }
18 }
19
20 int rev[maxn];
21 int t_in[maxn];
22 int upper[maxn];
23 int par[maxn];
24 int dep[maxn];
25
26 int T = 0;
27
28 void dfs_build(int v, int uppr, int pr = -1) {
29     rev[T] = v;
30     t_in[v] = T++;
31     dep[v] = pr == -1 ? 0 : dep[pr] + 1;
32     par[v] = pr;
33     upper[v] = uppr;
34
35     bool first = true;
36
37     for (int x : gr[v])
38         if (x != pr) {
39             dfs_build(x, first ? upper[v] : x, v);
40             first = false;
41         }
42 }
43
44 struct interval {
45     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]], t_in[a],
58                             ↪ true});
59
58             a = par[upper[a]];
59         } else {
60             back.push_back({t_in[upper[b]], t_in[b],
61                             ↪ false});
62             b = par[upper[b]];
63         }
64
65     front.push_back({min(t_in[a], t_in[b]),
66                     ↪ max(t_in[a], t_in[b]), t_in[a] > t_in[b]});
67     // for edge-weighted hld add:
68     ↪ "front.back().l++;";
69     front.insert(front.end(), back.rbegin(),
70                 ↪ back.rend());
71
72     return front;
73 }
74
75 // heavy-light ends
76
1 // max flow begins
2
3 struct edge{
4     int from, to;
5     int c, f, num;
6     edge(int from, int to, int c, int
7         ↪ num):from(from), to(to), c(c), f(0),
8         ↪ num(num){}
9     edge(){}
10 };
11
12 const int max_n = 600;
13
14 edge eds[150000];
15 int num = 0;
16 int it[max_n];
17 vector<int> gr[max_n];
18 int s, t;
19 vector<int> d(max_n);
20
21 bool bfs(int k) {
22     queue<int> q;
23     q.push(s);
24     fill(d.begin(), d.end(), -1);
25     d[s] = 0;
26     while (!q.empty()) {
27         int v = q.front();
28         q.pop();
29         for (int x : gr[v]) {
30             int to = eds[x].to;
31             if (d[to] == -1 && eds[x].c - eds[x].f >=
32                 ↪ (1 << k)){
33                 d[to] = d[v] + 1;
34                 q.push(to);
35             }
36         }
37     }
38
39     return (d[t] != -1);
40 }
41
42 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         eds[num].f += res;
51         eds[num ^ 1].f -= res;
52         return res;
53     }
54 }
55 return 0;
56 }
57
58 void add(int fr, int to, int c, int nm) {
59     gr[fr].push_back(num);
60     eds[num++] = edge(fr, to, c, nm);
61     gr[to].push_back(num);
62     eds[num++] = edge(to, fr, 0, nm); //corrected
63 }
64
65 int ans = 0;
66 for (int k = 30; k >= 0; k--)
67     while (bfs(k)) {
68         memset(it, 0, sizeof(it));
69         while (int res = dfs(s, 1e9 + 500, k))
70             ans += res;
71     }
72
73 // decomposition
74
75 int path_num = 0;
76 vector<int> paths[max_n];
77 int flows[max_n];
78
79 int decomp(int v, int flow) {
80     if (flow < 1)
81         return 0;
82     if (v == t) {
83         path_num++;
84         flows[path_num - 1] = flow;
85         return flow;
86     }
87     for (int i = 0; i < gr[v].size(); i++) {
88         int num = gr[v][i];
89         int res = decomp(eds[num].to, min(flow,
90             ↪ eds[num].f));
91         if (res) {
92             eds[num].f -= res;
93             paths[path_num - 1].push_back(eds[num].num);
94             return res;
95         }
96     }
97     return 0;
98 }
99
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) * e.w +
157                     ↪ dist[x] - dist[e.to]){
158                     d[e.to] = d[x] + (e.c - e.f) * e.w +
159                         ↪ dist[x] - dist[e.to];

```

```

53         p[e.to] = gr[x][i];
54         s.insert(make_pair(d[e.to], e.to));
55     }
56 }
57 }
58 dist[x] += d[x];
59 }
60 int pos = t;
61 while (pos != st){
62     int id = p[pos];
63     edges[id].f += 1;
64     edges[id ^ 1].f -= 1;
65     pos = edges[id].from;
66 }
67 }
68
69 // min-cost flow ends
70
71 // min circulation begins
72 class Solver2 { // Min-cost circulation
73     struct Edge {
74         int to, ne, w, c;
75     };
76     vector<Edge> es;
77     vi firs;
78     int curRes;
79 public:
80     Solver2(int n) : es(), firs(n, -1),
81     curRes(0) {}
82     // from, to, capacity (max.flow), cost
83     int adde(int a, int b, int w, int c) {
84         Edge e;
85         e.to = b; e.ne = firs[a];
86         e.w = w; e.c = c;
87         es.pb(e);
88         firs[a] = sz(es) - 1;
89
90         e.to = a; e.ne = firs[b];
91         e.w = 0; e.c = -c;
92         es.pb(e);
93         firs[b] = sz(es) - 1;
94         return sz(es) - 2;
95     }
96     // increase capacity of edge 'id' by 'w'
97     void ince(int id, int w) {
98         es[id].w += w;
99     }
100     int solve() {
101         const int n = sz(firs);
102
103         for (;;) {
104             vi d(n, 0), fre(n, -1);
105             vi chd(n, -1);
106
107             int base = -1;
108             for (int step = 0; step < n; step++) {
109                 for (int i = 0; i < sz(es); i++) if
110                 ↪ (es[i].w > 0) {
111                     int b = es[i].to;
112                     int a = es[i ^ 1].to;
113                     if (d[b] <= d[a] + es[i].c) continue;
114                     d[b] = d[a] + es[i].c;
115
116                     fre[b] = i;
117                     if (step == n - 1)
118                         base = b;
119                 }
120             }
121             if (base < 0) break;
122
123             vi seq;
124             vb was(n, false);
125             for (int x = base;; x = es[fre[x] ^ 1].to)
126                 ↪ {
127                     if (!was[x]) {
128                         seq.pb(x);
129                         was[x] = true;
130                     } else {
131                         seq.erase(
132                             seq.begin(),
133                             find(seq.begin(), seq.end(),
134                                 x
135                             ));
136                         break;
137                     }
138                 }
139             for (int i = 0; i < sz(seq); i++) {
140                 int v = seq[i];
141                 int eid = fre[v];
142                 assert(es[eid].w > 0);
143                 es[eid].w--;
144                 es[eid ^ 1].w++;
145                 curRes += es[eid].c;
146             }
147             return curRes;
148         }
149     };
150     // min circulation ends
151
152     // global cut begins
153     // g[i][j] = g[j][i] is sum of edges between i
154     ↪ and j
155     // ans is value of mincut
156     ll g[maxn][maxn], w[maxn];
157     ll ans;
158     bool ex[maxn], inA[maxn];
159     int n;
160     void iterate(int curN){
161         int prev = -1;
162         memset(inA, 0, sizeof(inA));
163         memset(w, 0, sizeof(w));
164         for (int it = 0; it < curN; it++) {
165             int best = -1;
166             for (int i = 0; i < n; i++)
167                 if (ex[i] && !inA[i]
168                     && (best == -1 || w[i] > w[best]))
169                     best = i;
170             assert(best != -1);
171             if (it == curN - 1) {
172                 ans = min(ans, w[best]);
173                 for (int i = 0; i < n; i++){
174                     g[i][prev] += g[best][i];
175                     g[prev][i] = g[i][prev];
176                 }
177             }
178         }
179     }

```

```

25     ex[best] = false;
26 } else {
27     inA[best] = true;
28     for (int i = 0; i < n; i++)
29         w[i] += g[best][i];
30     prev = best;
31 }
32 }
33 }
34 void solve(){
35     ans = INF;
36     for (int i = n; i > 1; i--)
37         iterate(i);
38     cout << ans << endl;
39 }
40 // global cut ends
41
42 // bad hungarian begins
43
44 fill(par, par + 301, -1);
45 fill(par2, par2 + 301, -1);
46
47 int ans = 0;
48 for (int v = 0; v < n; v++){
49     memset(useda, false, sizeof(useda));
50     memset(usedb, false, sizeof(usedb));
51     useda[v] = true;
52     for (int i = 0; i < n; i++)
53         w[i] = make_pair(a[v][i] + row[v] + col[i],
54             ↪ v);
55     memset(prev, 0, sizeof(prev));
56     int pos;
57     while (true){
58         pair<pair<int, int>, int> p =
59             ↪ make_pair(make_pair(1e9, 1e9), 1e9);
60         for (int i = 0; i < n; i++)
61             if (!usedb[i])
62                 p = min(p, make_pair(w[i], i));
63         for (int i = 0; i < n; i++)
64             if (!useda[i])
65                 row[i] += p.first.first;
66         for (int i = 0; i < n; i++)
67             if (!usedb[i]){
68                 col[i] -= p.first.first;
69                 w[i].first -= p.first.first;
70             }
71         ans += p.first.first;
72         usedb[p.second] = true;
73         prev[p.second] = p.first.second; //из аномой
74             ↪ в непыю
75         int x = par[p.second];
76         if (x == -1){
77             pos = p.second;
78             break;
79         }
80         useda[x] = true;
81         for (int j = 0; j < n; j++)
82             w[j] = min(w[j], {a[x][j] + row[x] +
83                 ↪ col[j], x});
84     }
85 }
86 while (pos != -1){
87     int nxt = par2[prev[pos]];
88     par[pos] = prev[pos];
89     par2[prev[pos]] = pos;
90     pos = nxt;
91 }
92 }
93 cout << ans << "\n";
94 for (int i = 0; i < n; i++)
95     cout << par[i] + 1 << " " << i + 1 << "\n";
96
97 // bad hungarian ends
98
99 // Edmonds O(n^3) begins
100
101 vector<int> gr[MAXN];
102 int match[MAXN], p[MAXN], base[MAXN], q[MAXN];
103 bool used[MAXN], blossom[MAXN];
104 int mark[MAXN];
105 int C = 1;
106
107 int lca(int a, int b) {
108     C++;
109     for (;) {
110         a = base[a];
111         mark[a] = C;
112         if (match[a] == -1) break;
113         a = p[match[a]];
114     }
115
116     for (;) {
117         b = base[b];
118         if (mark[b] == C) return b;
119         b = p[match[b]];
120     }
121 }
122
123 void mark_path(int v, int b, int children) {
124     while (base[v] != b) {
125         blossom[base[v]] = blossom[base[match[v]]] =
126             ↪ true;
127         p[v] = children;
128         children = match[v];
129         v = p[match[v]];
130     }
131 }
132
133 int find_path(int root) {
134     memset(used, 0, sizeof(used));
135     memset(p, -1, sizeof p);
136     for (int i = 0; i < N; i++)
137         base[i] = i;
138
139     used[root] = true;
140     int qh = 0, qt = 0;
141     q[qt++] = root;
142     while (qh < qt) {
143         int v = q[qh++];
144         for (int to : gr[v]) {
145             if (base[v] == base[to] || match[v] == to)
146                 ↪ continue;
147             if (to == root || match[to] != -1 &&
148                 ↪ p[match[to]] != -1) {

```

```

48     int curbase = lca(v, to);
49     memset(blossom, 0, sizeof(blossom));
50     mark_path(v, curbase, to);
51     mark_path(to, curbase, v);
52     for (int i = 0; i < N; i++)
53         if (blossom[base[i]]) {
54             base[i] = curbase;
55             if (!used[i]) {
56                 used[i] = true;
57                 q[qt++] = i;
58             }
59         }
60     } else if (p[to] == -1) {
61         p[to] = v;
62         if (match[to] == -1)
63             return to;
64         to = match[to];
65         used[to] = true;
66         q[qt++] = to;
67     }
68 }
69 }
70
71 return -1;
72 }
73
74 memset(match, -1, sizeof match);
75 for (int i = 0; i < N; i++) {
76     if (match[i] == -1 && !gr[i].empty()) {
77         int v = find_path(i);
78         while (v != -1) {
79             int pv = p[v], ppv = match[pv];
80             match[v] = pv; match[pv] = v;
81             v = ppv;
82         }
83     }
84 }
85
86 // Edmonds O(n^3) ends
87
88 // string basis begins
89
90 vector<int> getZ(string s){
91     vector<int> z;
92     z.resize(s.size(), 0);
93     int l = 0, r = 0;
94     for (int i = 1; i < s.size(); i++){
95         if (i <= r)
96             z[i] = min(r - i + 1, z[i - l]);
97         while (i + z[i] < s.size() && s[z[i]] == s[i
98             ↵ + z[i]])
99             z[i]++;
100         if (i + z[i] - 1 > r){
101             r = i + z[i] - 1;
102             l = i;
103         }
104     }
105     return z;
106 }
107
108 vector<int> getP(string s){
109     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] - 'a' +
39             ↵ 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             if (s[k] < s[j])
64                 k = i;
65             else
66                 ++k;
67             ++j;
68         }
69         while (i <= k) i += j - k;
70     }
71     return s.substr (ans, n/2);
72 }
73
74 // min cyclic shift ends
75
76 // suffix array O(n) begins
77
78 typedef vector<char> bits;
79
80 template<const int end>
81 void getBuckets(int *s, int *bkt, int n, int K) {
82     fill(bkt, bkt + K + 1, 0);
83     for (i, n) bkt[s[i] + !end]++;

```

```

9     forn(i, K) bkt[i + 1] += bkt[i];
10 }
11 void induceSA1(bits &t, int *SA, int *s, int
    ↪ *bkt, int n, int K) {
12     getBuckets<0>(s, bkt, n, K);
13     forn(i, n) {
14         int j = SA[i] - 1;
15         if (j >= 0 && !t[j])
16             SA[bkt[s[j]]++] = j;
17     }
18 }
19 void induceSAs(bits &t, int *SA, int *s, int
    ↪ *bkt, int n, int K) {
20     getBuckets<1>(s, bkt, n, K);
21     for (int i = n - 1; i >= 0; i--) {
22         int j = SA[i] - 1;
23         if (j >= 0 && t[j])
24             SA[--bkt[s[j]]] = j;
25     }
26 }
27
28 void SA_IS(int *s, int *SA, int n, int K) { //
    ↪ require last symbol is 0
29 #define isLMS(i) (i && t[i] && !t[i-1])
30     int i, j;
31     bits t(n);
32     t[n-1] = 1;
33     for (i = n - 3; i >= 0; i--)
34         t[i] = (s[i]<s[i+1] || (s[i]==s[i+1] &&
            ↪ t[i+1]==1));
35     int bkt[K + 1];
36     getBuckets<1>(s, bkt, n, K);
37     fill(SA, SA + n, -1);
38     forn(i, n)
39         if (isLMS(i))
40             SA[--bkt[s[i]]] = i;
41     induceSA1(t, SA, s, bkt, n, K);
42     induceSAs(t, SA, s, bkt, n, K);
43     int n1 = 0;
44     forn(i, n)
45         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] != s[prev+d] ||
                ↪ t[pos+d] != t[prev+d])
54                 diff = true, d = n;
55             else if (d > 0 && (isLMS(pos+d) ||
                ↪ isLMS(prev+d)))
56                 d = n;
57         if (diff)
58             name++, prev = pos;
59         SA[n1 + (pos >> 1)] = name - 1;
60     }
61     for (i = n - 1, j = n - 1; i >= n1; i--)
62         if (SA[i] >= 0)
63             SA[j--] = SA[i];
64     int *s1 = SA + n - n1;
65     if (name < n1)
66         SA_IS(s1, SA, n1, name - 1);
67     else
68         forn(i, n1)
69             SA[s1[i]] = i;
70     getBuckets<1>(s, bkt, n, K);
71     for (i = 1, j = 0; i < n; i++)
72         if (isLMS(i))
73             s1[j++] = i;
74     forn(i, n1)
75         SA[i] = s1[SA[i]];
76     fill(SA + n1, SA + n, -1);
77     for (i = n1 - 1; i >= 0; i--) {
78         j = SA[i], SA[i] = -1;
79         SA[--bkt[s[j]]] = j;
80     }
81     induceSA1(t, SA, s, bkt, n, K);
82     induceSAs(t, SA, s, bkt, n, K);
83 }
84 // suffix array O(n) ends
85
86 // suffix array O(n log n) begins
87 string str;
88 int N, m, SA [MAX_N], LCP [MAX_N];
89 int x [MAX_N], y [MAX_N], w [MAX_N], c [MAX_N];
90
91 inline bool cmp (const int a, const int b, const
    ↪ int l) { return (y [a] == y [b] && y [a + 1]
    ↪ == y [b + 1]); }
92
93 void Sort () {
94     for (int i = 0; i < m; ++i) w[i] = 0;
95     for (int i = 0; i < N; ++i) ++w[x[y[i]]];
96     for (int i = 0; i < m - 1; ++i) w[i + 1] +=
        ↪ w[i];
97     for (int i = N - 1; i >= 0; --i)
98         ↪ SA[--w[x[y[i]]]] = y[i];
99 }
100 void DA () {
101     for (int i = 0; i < N; ++i) x[i] = str[i], y[i]
        ↪ = i;
102     Sort ();
103     for (int i, j = 1, p = 1; p < N; j <= 1, m =
        ↪ p) {
104         for (p = 0, i = N - j; i < N; i++) y[p++] =
            ↪ i;
105         for (int k = 0; k < N; ++k) if (SA[k] >= j)
            ↪ y[p++] = SA[k] - j;
106         Sort();
107         for (swap (x, y), p = 1, x[SA[0]] = 0, i = 1;
            ↪ i < N; ++i) x[SA [i]] = cmp (SA[i - 1],
            ↪ SA[i], j) ? p - 1 : p++;
108     }
109 }
110
111 // common for all algorithms
112 void kasaiLCP () {
113     for (int i = 0; i < N; i++) c[SA[i]] = i;

```

```

114   for (int i = 0, j, k = 0; i < N; LCP [c[i++]] 46
    ↪ k) 47
115   if (c [i] > 0) for (k ? k-- : 0, j = SA[c[i] 48
    ↪ - 1]; str[i + k] == str[j + k]; k++); 49
116   else k = 0; 50
117 } 51
118 52
119 void suffixArray () { // require last symbol is 53
    ↪ char(0) 54
120   m = 256; 55 // bad suffix automaton ends
121   N = str.size();
122   DA (); 1 // aho-corasick begins
123   kasaiLCP (); 2
124 } 3
125 // suffix array O(n log n) ends 4
1 // bad suffix automaton begins 5
3 struct node{ 6
4   map<char, int> go; 7
5   int len, suff; 8
6   long long sum_in; 9
7   node(){} 10
8 }; 11
9 12
10 node v[max_n * 4]; 13
11 14
12 int add_node(int max_len){ 15
13   //v[number].sum_in = 0; 16
14   v[number].len = max_len; 17
15   v[number].suff = -1; 18
16   number++; 19
17   return number - 1; 20
18 } 21
19 22
20 int last = add_node(0); 23
21 24
22 void add_char(char c) { 25
23   int cur = last; 26
24   int new_node = add_node(v[cur].len + 1); 27
25   last = new_node; 28
26   while (cur != -1){ 29
27     if (v[cur].go.count(c) == 0){ 30
28       v[cur].go[c] = new_node; 31
29       //v[new_node].sum_in += v[cur].sum_in; 32
30       cur = v[cur].suff; 33
31       if (cur == -1) 34
32         v[new_node].suff = 0; 35
33     }else{ 36
34       int a = v[cur].go[c]; 37
35       if (v[a].len == v[cur].len + 1){ 38
36         v[new_node].suff = a; 39
37       }else{ 40
38         int b = add_node(v[cur].len + 1); 41
39         v[b].go = v[a].go; 42
40         v[b].suff = v[a].suff; 43
41         v[new_node].suff = b; 44
42         while (cur != -1 && v[cur].go.count(c) != 45
    ↪ 0 && v[cur].go[c] == a){ 46
43           v[cur].go[c] = b; 47
44           //v[a].sum_in -= v[cur].sum_in; 48
45           //v[b].sum_in += v[cur].sum_in; 49
    cur = v[cur].suff; 50
    } 51
    } 52
    } 53
    } 54
    } 55
    } 56
    } 57
    } 58
    } 59
    } 60
    } 61
    } 62
    } 63
    } 64
    } 65
    } 66
    } 67
    } 68
    } 69
    } 70
    } 71
    } 72
    } 73
    } 74
    } 75
    } 76
    } 77
    } 78
    } 79
    } 80
    } 81
    } 82
    } 83
    } 84
    } 85
    } 86
    } 87
    } 88
    } 89
    } 90
    } 91
    } 92
    } 93
    } 94
    } 95
    } 96
    } 97
    } 98
    } 99
    } 100
    } 101
    } 102
    } 103
    } 104
    } 105
    } 106
    } 107
    } 108
    } 109
    } 110
    } 111
    } 112
    } 113
    } 114
    } 115
    } 116
    } 117
    } 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
    } 501
    } 502
    } 503
    } 504
    } 505
    } 506
    } 507
    } 508
    } 509
    } 510
    } 511
    } 512
    } 513
    } 514
    } 515
    } 516
    } 517
    } 518
    } 519
    } 520
    } 521
    } 522
    } 523
    } 524
    } 525
    } 526
    } 527
    } 528
    } 529
    } 530
    } 531
    } 532
    } 533
    } 534
    } 535
    } 536
    } 537
    } 538
    } 539
    } 540
    } 541
    } 542
    } 543
    } 544
    } 545
    } 546
    } 547
    } 548
    } 549
    } 550
    } 551
    } 552
    } 553
    } 554
    } 555
    } 556
    } 557
    } 558
    } 559
    } 560
    } 561
    } 562
    } 563
    } 564
    } 565
    } 566
    } 567
    } 568
    } 569
    } 570
    } 571
    } 572
    } 573
    } 574
    } 575
    } 576
    } 577
    } 578
    } 579
    } 580
    } 581
    } 582
    } 583
    } 584
    } 585
    } 586
    } 587
    } 588
    } 589
    } 590
    } 591
    } 592
    } 593
    } 594
    } 595
    } 596
    } 597
    } 598
    } 599
    } 600
    } 601
    } 602
    } 603
    } 604
    } 605
    } 606
    } 607
    } 608
    } 609
    } 610
    } 611
    } 612
    } 613
    } 614
    } 615
    } 616
    } 617
    } 618
    } 619
    } 620
    } 621
    } 622
    } 623
    } 624
    } 625
    } 626
    } 627
    } 628
    } 629
    } 630
    } 631
    } 632
    } 633
    } 634
    } 635
    } 636
    } 637
    } 638
    } 639
    } 640
    } 641
    } 642
    } 643
    } 644
    } 645
    } 646
    } 647
    } 648
    } 649
    } 650
    } 651
    } 652
    } 653
    } 654
    } 655
    } 656
    } 657
    } 658
    } 659
    } 660
    } 661
    } 662
    } 663
    } 664
    } 665
    } 666
    } 667
    } 668
    } 669
    } 670
    } 671
    } 672
    } 673
    } 674
    } 675
    } 676
    } 677
    } 678
    } 679
    } 680
    } 681
    } 682
    } 683
    } 684
    } 685
    } 686
    } 687
    } 688
    } 689
    } 690
    } 691
    } 692
    } 693
    } 694
    } 695
    } 696
    } 697
    } 698
    } 699
    } 700
    } 701
    } 702
    } 703
    } 704
    } 705
    } 706
    } 707
    } 708
    } 709
    } 710
    } 711
    } 712
    } 713
    } 714
    } 715
    } 716
    } 717
    } 718
    } 719
    } 720
    } 721
    } 722
    } 723
    } 724
    } 725
    } 726
    } 727
    } 728
    } 729
    } 730
    } 731
    } 732
    } 733
    } 734
    } 735
    } 736
    } 737
    } 738
    } 739
    } 740
    } 741
    } 742
    } 743
    } 744
    } 745
    } 746
    } 747
    } 748
    } 749
    } 750
    } 751
    } 752
    } 753
    } 754
    } 755
    } 756
    } 757
    } 758
    } 759
    } 760
    } 761
    } 762
    } 763
    } 764
    } 765
    } 766
    } 767
    } 768
    } 769
    } 770
    } 771
    } 772
    } 773
    } 774
    } 775
    } 776
    } 777
    } 778
    } 779
    } 780
    } 781
    } 782
    } 783
    } 784
    } 785
    } 786
    } 787
    } 788
    } 789
    } 790
    } 791
    } 792
    } 793
    } 794
    } 795
    } 796
    } 797
    } 798
    } 799
    } 800
    } 801
    } 802
    } 803
    } 804
    } 805
    } 806
    } 807
    } 808
    } 809
    } 810
    } 811
    } 812
    } 813
    } 814
    } 815
    } 816
    } 817
    } 818
    } 819
    } 820
    } 821
    } 822
    } 823
    } 824
    } 825
    } 826
    } 827
    } 828
    } 829
    } 830
    } 831
    } 832
    } 833
    } 834
    } 835
    } 836
    } 837
    } 838
    } 839
    } 840
    } 841
    } 842
    } 843
    } 844
    } 845
    } 846
    } 847
    } 848
    } 849
    } 850
    } 851
    } 852
    } 853
    } 854
    } 855
    } 856
    } 857
    } 858
    } 859
    } 860
    } 861
    } 862
    } 863
    } 864
    } 865
    } 866
    } 867
    } 868
    } 869
    } 870
    } 871
    } 872
    } 873
    } 874
    } 875
    } 876
    } 877
    } 878
    } 879
    } 880
    } 881
    } 882
    } 883
    } 884
    } 885
    } 886
    } 887
    } 888
    } 889
    } 890
    } 891
    } 892
    } 893
    } 894
    } 895
    } 896
    } 897
    } 898
    } 899
    } 900
    } 901
    } 902
    } 903
    } 904
    } 905
    } 906
    } 907
    } 908
    } 909
    } 910
    } 911
    } 912
    } 913
    } 914
    } 915
    } 916
    } 917
    } 918
    } 919
    } 920
    } 921
    } 922
    } 923
    } 924
    } 925
    } 926
    } 927
    } 928
    } 929
    } 930
    } 931
    } 932
    } 933
    } 934
    } 935
    } 936
    } 937
    } 938
    } 939
    } 940
    } 941
    } 942
    } 943
    } 944
    } 945
    } 946
    } 947
    } 948
    } 949
    } 950
    } 951
    } 952
    } 953
    } 954
    } 955
    } 956
    } 957
    } 958
    } 959
    } 960
    } 961
    } 962
    } 963
    } 964
    } 965
    } 966
    } 967
    } 968
    } 969
    } 970
    } 971
    } 972
    } 973
    } 974
    } 975
    } 976
    } 977
    } 978
    } 979
    } 980
    } 981
    } 982
    } 983
    } 984
    } 985
    } 986
    } 987
    } 988
    } 989
    } 990
    } 991
    } 992
    } 993
    } 994
    } 995
    } 996
    } 997
    } 998
    } 999
    } 1000
    } 1001
    } 1002
    } 1003
    } 1004
    } 1005
    } 1006
    } 1007
    } 1008
    } 1009
    } 1010
    } 1011
    } 1012
    } 1013
    } 1014
    } 1015
    } 1016
    } 1017
    } 1018
    } 1019
    } 1020
    } 1021
    } 1022
    } 1023
    } 1024
    } 1025
    } 1026
    } 1027
    } 1028
    } 1029
    } 1030
    } 1031
    } 1032
    } 1033
    } 1034
    } 1035
    } 1036
    } 1037
    } 1038
    } 1039
    } 1040
    } 1041
    } 1042
    } 1043
    } 1044
    } 1045
    } 1046
    } 1047
    } 1048
    } 1049
    } 1050
    } 1051
    } 1052
    } 1053
    } 1054
    } 1055
    } 1056
    } 1057
    } 1058
    } 1059
    } 1060
    } 1061
    } 1062
    } 1063
    } 1064
    } 1065
    } 1066
    } 1067
    } 1068
    } 1069
    } 1070
    } 1071
    } 1072
    } 1073
    } 1074
    } 1075
    } 1076
    } 1077
    } 1078
    } 1079
    } 1080
    } 1081
    } 1082
    } 1083
    } 1084
    } 1085
    } 1086
    } 1087
    } 1088
    } 1089
    } 1090
    } 1091
    } 1092
    } 1093
    } 1094
    } 1095
    } 1096
    } 1097
    } 1098
    } 1099
    } 1100
    } 1101
    } 1102
    } 1103
    } 1104
    } 1105
    } 1106
    } 1107
    } 1108
    } 1109
    } 1110
    } 1111
    } 1112
    } 1113
    } 1114
    } 1115
    } 1116
    } 1117
    } 1118
    } 1119
    } 1120
    } 1121
    } 1122
    } 1123
    } 1124
    } 1125
    } 1126
    } 1127
    } 1128
    } 1129
    } 1130
    } 1131
    } 1132
    } 1133
    } 1134
    } 1135
    } 1136
    } 1137
    } 1138
    } 1139
    } 1140
    } 1141
    } 1142
    } 1143
    } 1144
    } 1145
    } 1146
    } 1147
    } 1148
    } 1149
    } 1150
    } 1151
    } 1152
    } 1153
    } 1154
    } 1155
    } 1156
    } 1157
    } 1158
    } 1159
    } 1160
    } 1161
    } 1162
    } 1163
    } 1164
    } 1165
    } 1166
    } 1167
    } 1168
    } 1169
    } 1170
    } 1171
    } 1172
    } 1173
    } 1174
    } 1175
    } 1176
    } 1177
    } 1178
    } 1179
    } 1180
    } 1181
    } 1182
    } 1183
    } 1184
    } 1185
    } 1186
    } 1187
    } 1188
    } 1189
    } 1190
    } 1191
    } 1192
    } 1193
    } 1194
    } 1195
    } 1196
    } 1197
    } 1198
    } 1199
    } 1200
    } 1201
    } 1202
    } 1203
    } 1204
    } 1205
    } 1206
    } 1207
    } 1208
    } 1209
    } 1210
    } 1211
    } 1212
    } 1213
    } 1214
    } 1215
    } 1216
    } 1217
    } 1218
    } 1219
    } 1220
    } 1221
    } 1222
    } 1223
    } 1224
    } 1225
    } 1226
    } 1227
    } 1228
    } 1229
    } 1230
    } 1231
    } 1232
    } 1233
    } 1234
    } 1235
    } 1236
    } 1237
    } 1238
    } 1239
    } 1240
    } 1241
    } 1242
    } 1243
    } 1244
    } 1245
    } 1246
    } 1247
    } 1248
    } 1249
    } 1250
    } 1251
    } 1252
    } 1253
    } 1254
    } 1255
    } 1256
    } 1257
    } 1258
    } 1259
    } 1260
    } 1261
    } 1262
    } 1263
    } 1264
    } 1265
    } 1266
    } 1267
    } 1268
    } 1269
    } 1270
    } 1271
    } 1272
    } 1273
    } 1274
    } 1275
    } 1276
    } 1277
    } 1278
    } 1279
    } 1280
    } 1281
    } 1282
    } 1283
    } 1284
    } 1285
    } 1286
    } 1287
    } 1288
    } 1289
    } 1290
    } 1291
    } 1292
    } 1293
    } 1294
    } 1295
    } 1296
    } 1297
    } 1298
    } 1299
    } 1300
    } 1301
    } 1302
    } 1303
    } 1304
    } 1305
    } 1306
    } 1307
    } 1308
    } 1309
    } 1310
    } 1311
    } 1312
    } 1313
    } 1314
    } 1315
    } 1316
    } 1317
    } 1318
    } 1319
    } 1320
    } 1321
    } 1322
    } 1323
    } 1324
    } 1325
    } 1326
    } 1327
    } 1328
    } 1329
    } 1330
    } 1331
    } 1332
    } 1333
    } 1334
    } 1335
    } 1336
    } 1337
    } 1338
    } 1339
    } 1340
    } 1341
    } 1342
    } 1343
    } 1344
    } 1345
    } 1346
    } 1347
    } 1348
    } 1349
    } 1350
    } 1351
    } 1352
    } 1353
    } 1354
    } 1355
    } 1356
    } 1357
    } 1358
    } 1359
    } 1360
    } 1361
    } 1362
    } 1363
    } 1364
    } 1365
    } 1366
    } 1367
    } 1368
    } 1369
    } 1370
    } 1371
    } 1372
    } 1373
    } 1374
    } 1375
    } 1376
    } 1377
    } 1378
    } 1379
    } 1380
    } 1381
    } 1382
    } 1383
    } 1384
    } 1385
    } 1386
    } 1387
    } 1388
    } 1389
    } 1390
    } 1391
    } 1392
    } 1393
    } 1394
    } 1395
    } 1396
    } 1397
    } 1398
    } 1399
    } 1400
    } 1401
    } 1402
    } 1403
    } 1404
    } 1405
    } 1406
    } 1407
    } 1408
    } 1409
    } 1410
    } 1411
    } 1412
    } 1413
    } 1414
    } 1415
    } 1416
    } 1417
    } 1418
    } 1419
    } 1420
    } 1421
    } 1422
    } 1423
    } 1424
    } 1425
    } 1426
    } 1427
    } 1428
    } 1429
    } 1430
    } 1431
    } 1432
    } 1433
    } 1434
    } 1435
    } 1436
    } 1437
    } 1438
    } 1439
    } 1440
    } 1441
    } 1442
    } 1443
    } 1444
    } 1445
    } 1446
    } 1447
    } 1448
    } 1449
    } 1450
    } 1451
    } 1452
    } 1453
    } 1454
    } 1455
    } 1456
    } 1457
    } 1458
    } 1459
    } 1460
    } 1461
    } 1462
    } 1463
    } 1464
    } 1465
    } 1466
    } 1467
    } 1468
    } 1469
    } 1470
    } 1471
    } 1472
    } 1473
    } 1474
    } 1475
    } 1476
    } 1477
    } 1478
    } 1479
    } 1480
    } 1481
    } 1482
    } 1483
    } 1484
    } 1485
    } 1486
    } 1487
    } 1488
    } 1489
    } 1490
    } 1491
    } 1492
    } 1493
    } 1494
    } 1495
    } 1496
    } 1497
    } 1498
    } 1499
    } 1500
    } 1501
    } 1502
    } 1503
    } 1504
    } 1505
    } 1506
    } 1507
    } 1508
    } 1509
    } 1510
    } 1511
    } 1512
    } 1513
    } 1514
    } 1515
    } 1516
    } 1517
    } 1518
    } 1519
    } 1520
    } 1521
    } 1522
    } 1523
    } 1524
    } 1525
    } 1526
    } 1527
    } 1528
    } 1529
    } 1530
    } 1531
    } 1532
    } 1533
    } 1534
    } 1535
    } 1536
    } 1537
    } 1538
    } 1539
    } 1540
    } 1541
    } 1542
    } 1543
    } 1544
    } 1545
    } 1546
    } 1547
    } 1548
    } 1549
    } 1550
    } 1551
    } 1552
    } 1553
    } 1554
    } 1555
    } 1556
    } 1557
    } 1558
    } 1559
    } 1560
    } 1561
    } 1562
    } 1563
    } 1564
    } 1565
    } 1566
    } 1567
    } 1568
    } 1569
    } 1570
    } 1571
    } 1572
    } 1573
    } 1574
    } 1575
    } 1576
    } 1577
    } 1578
    } 1579
    } 1580
    } 1581
    } 1582
    } 1583
    } 1584
    } 1585
    } 1586
    } 1587
    } 1588
    } 1589
    } 1590
    } 1591
    } 1592
    } 1593
    } 1594
    } 1595
    } 1596
    } 1597
    } 1598
```



```

52     if (t[v].next[c] != -1)
53         t[v].go[c] = t[v].next[c];
54     else
55         t[v].go[c] = v == 0 ? 0 : go(get_link(v),
56             ↪ c);
57     return t[v].go[c];
58 }
59 // aho-corasick ends

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 unsigned long long get(unsigned long long a,
12     ↪ unsigned long long b){
13     if (a > b)
14         return a-b;
15     else
16         return b-a;
17 }
18 unsigned long long pollard(unsigned long long n){
19     unsigned long long x = (rand() + 1) % n, y = 1;
20     ↪ g;
21     int stage = 2, i = 0;
22     g = gcd(get(x, y), n);
23     while (g == 1) {
24         if (i == max_step)
25             break;
26         if (i == stage) {
27             y = x;
28             stage <= 1;
29         }
30         x = (x * (__int128)x + 1) % n;
31         i++;
32         g = gcd(get(x, y), n);
33     }
34     return g;
35 }
36 // pollard ends

1 // linear sieve begins
2
3 const int N = 1000000;
4
5 int pr[N + 1], sz = 0;
6 /* minimal prime, mobius function, euler function
7     ↪ */
8 int lp[N + 1], mu[N + 1], phi[N + 1];
9
10 lp[1] = mu[1] = phi[1] = 1;
11 for (int i = 2; i <= N; ++i) {
12     if (lp[i] == 0)
13         lp[i] = pr[sz++] = i;
14     for (int j = 0; j < sz && pr[j] <= lp[i] && i *
15         ↪ pr[j] <= N; ++j)
16         lp[i * pr[j]] = pr[j];
17
18     mu[i] = lp[i] == lp[i / lp[i]] ? 0 : -1 * mu[i]
19         ↪ / lp[i]];
20     phi[i] = phi[i / lp[i]] * (lp[i] == lp[i /
21         ↪ lp[i]] ? lp[i] : lp[i] - 1);
22 }
23 // linear sieve ends

1 // discrete log in sqrt(p) begins
2
3 int k = sqrt((double)p) + 2;
4
5 for (int i = k; i >= 1; i--)
6     mp[bin(b, (i * 1ll * k) % (p-1), p)] = i;
7
8 bool answered = false;
9 int ans = INT32_MAX;
10 for (int i = 0; i <= k; i++){
11     int sum = (n * 1ll * bin(b, i, p)) % p;
12     if (mp.count(sum) != 0){
13         int an = mp[sum] * 1ll * k - i;
14         if (an < p)
15             ans = min(an, ans);
16     }
17 }
18 // discrete log in sqrt(p) ends

1 // prime roots mod n begins
2
3 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 ==
20     ↪ 2 || (num == 2 && n % 2 == 0 && n % 4 != 0))
21     ↪ {
22     cout << "-1\n";
23     continue;
24 }
25 vector<long long> v;
26 long long pp = phi;
27 for (long long x:primes){
28     if (x*x>pp)
29         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
53 // prime roots mod n ends
54
55 // simplex begins
56
57 const double EPS = 1e-9;
58
59 typedef vector<double> vdbl;
60
61 // n variables, m inequalities
62 // Ax <= b, c*x -> max, x >= 0
63 double simplex( int n, int m, const vector<vdbl>
64     ↪ &a0, const vdbl &b, const vdbl &c, vdbl &x )
65     ↪ {
66     // Ax + Ez = b, A[m]*x -> max
67     // x = 0, z = b, x >= 0, z >= 0
68     vector<vdbl> a(m + 2, vdbl(n + m + 2));
69     vector<int> p(m);
70     forn(i, n)
71         a[m + 1][i] = c[i];
72     forn(i, m) {
73         forn(j, n)
74             a[i][j] = a0[i][j];
75         a[i][n + i] = 1;
76         a[i][m + n] = -1;
77         a[i][m + n + 1] = b[i];
78         p[i] = n + i;
79     }
80
81 // basis: enter "j", leave "ind+n"
82 auto pivot = [&]( int j, int ind ) {
83     double coef = a[ind][j];
84     assert(fabs(coef) > EPS);
85     forn(col, n + m + 2)
86         a[ind][col] /= coef;
87     forn(row, m + 2)
88         if (row != ind && fabs(a[row][j]) > EPS) {
89             coef = a[row][j];
90
91             forn(col, n + m + 2)
92                 a[row][col] -= a[ind][col] * coef;
93             a[row][j] = 0; // reduce precision error
94         }
95     p[ind] = j;
96 };
97
98 // the Simplex itself
99 auto iterate = [&]( int nn ) {
100     for (int run = 1; run; ) {
101         run = 0;
102         forn(j, nn) {
103             if (a[m][j] > EPS) { // strictly positive
104                 run = 1;
105                 double mi = INFINITY, t;
106                 int ind = -1;
107                 forn(i, m)
108                     if (a[i][j] > EPS && (t = a[i][n + m]
109                         ↪ + 1] / a[i][j]) < mi - EPS)
110                         mi = t, ind = i;
111                 if (ind == -1)
112                     return false;
113                 return true;
114                 pivot(j, ind);
115             }
116         }
117     }
118     return true;
119 };
120
121 int mi = min_element(b.begin(), b.end()) -
122     ↪ b.begin();
123 if (b[mi] < -EPS) {
124     a[m][n + m] = -1;
125     pivot(n + m, mi);
126     assert(iterate(n + m + 1));
127     if (a[m][m + n + 1] > EPS) // optimal value
128         ↪ is positive
129         return NAN;
130     forn(i, m)
131         if (p[i] == m + n) {
132             int j = 0;
133             while (find(p.begin(), p.end(), j) !=
134                 ↪ p.end() || fabs(a[i][j]) < EPS)
135                 j++, assert(j < m + n);
136             pivot(j, i);
137         }
138     swap(a[m], a[m + 1]);
139     if (!iterate(n + m))
140         return INFINITY;
141     x = vdbl(n, 0);
142     forn(i, m)
143         if (p[i] < n)
144             x[p[i]] = a[i][n + m + 1];
145     return -a[m][n + m + 1];
146 }
147
148 // simplex usage:
149 vdbl x(n);
150 double result = simplex(n, m, a, b, c, x);
151 if (isinf(result))

```

```

91 puts("Unbounded");
92 else if (isnan(result))
93 puts("No solution");
94 else {
95 printf("%.9f :", result);
96 for(i, n)
97 printf(" %.9f", x[i]);
98 puts("");
99 }
100
101 // simplex ends
102
1 // sum over subsets begins
2 // fast subset convolution  $O(n 2^n)$ 
3 for(int i = 0; i < (1<<N); ++i)
4 F[i] = A[i];
5 for(int i = 0; i < N; ++i) for(int mask = 0; mask
  ↳ < (1<<N); ++mask){
6 if(mask & (1<<i))
7 F[mask] += F[mask^(1<<i)];
8 }
9 // sum over subsets ends
10
11 // algebra begins
12
13 Pick
14  $B + \frac{\Gamma}{2} - 1$ ,
15 где  $B$  - количество целочисленных точек внутри
  ↳ многоугольника, а  $\Gamma$  - количество
  ↳ целочисленных точек на границе
  ↳ многоугольника.
16
17 Newton
18  $x_{i+1} = x_i - f(x_i)/f'(x_i)$ 
19
20 Catalan
21  $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$ 
22  $C_i = \frac{1}{n+1} \binom{2n}{n}$ 
23
24  $G_N = 2^{n(n-1)/2}$ 
25 Количество связанных помеченных графов
26  $Conn_N = G_N - \frac{1}{N} \sum_{k=1}^{N-1} k \binom{N}{k} Conn_k$ 
  ↳  $G_{N-K}$ 
27
28 Количество помеченных графов с  $K$  компонентами
  ↳ связности
29  $D[N][K] = \sum_{S=1}^N \binom{N-1}{S-1} D[N-S][K-1]$ 
  ↳  $Conn_S D[N-S][K-1]$ 
30
31 Miller-Rabbin
32  $a = a^t$ 
33 FOR  $i = 1 \dots s$ 
34 if  $a^{i/2} = 1$  &&  $|a| \neq 1$ 
35 NOT PRIME
36  $a = a^2$ 
37 return  $a = 1$  ? PRIME : NOT PRIME
38
39 Интегрирование по формуле Симпсона
40  $\int_a^b f(x) dx \approx ?$ 

```

```

32 x_i := a+ih, i=0...2n
33 h =  $\frac{b-a}{2n}$ 
34
35  $\int = (f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) +$ 
  ↳  $2f(x_4) + \dots + 4f(x_{2n-1}) +$ 
  ↳  $f(x_{2n})) \frac{h}{3}$ 
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, int y){
50 lo = x, hi = y;
51 if(lo == hi or from >= to) return;
52
53 int mid = (lo+hi)/2;
54 auto f = [mid](int x){
55 return x <= mid;
56 };
57 b.reserve(to-from+1);
58 b.pb(0);
59 //b[i] = no of elements from first "i"
  ↳ elements that go to left node
60 for(auto it = from; it != to; it++)
61 b.pb(b.back() + f(*it));
62
63 //see how lambda function is used here
64 auto pivot = stable_partition(from, to, f);
65 l = new wavelet_tree(from, pivot, lo, mid);
66 r = new wavelet_tree(pivot, to, mid+1, hi);
67 }
68
69 //kth smallest element in [l, r]
70 int kth(int l, int r, int k){
71 if(l > r) return 0;
72 if(lo == hi) return lo;
73 //how many nos are there in left node from
  ↳ [l, r]
74 int inleft = b[r] - b[l-1];
75 int lb = b[l-1]; //amt of nos from first
  ↳ (l-1) nos that go in left
76 int rb = b[r]; //amt of nos from first (r)
  ↳ nos that go in left
77 //so [lb+1, rb] represents nos from [l, r]
  ↳ that go to left
78 if(k <= inleft) return this->l->kth(lb+1, rb
  ↳ , k);
79
80 //(l-1-lb) is amt of nos from first (l-1) nos
  ↳ that go to right
81 //(r-rb) is amt of nos from first (r) nos
  ↳ that go to right

```

```

43 //so [l-lb, r-rb] represents nos from [l, r]
44 ↪ that go to right
45 return this->r->kth(l-lb, r-rb, k-inleft);
46 };
47
48 // wavelet tree ends
49
50 // berlecamp-massey begins
51
52 const int SZ = MAXN;
53
54 ll qp(ll a, ll b) {
55     ll x = 1;
56     a %= MOD;
57     while (b) {
58         if (b & 1) x = x * a % MOD;
59         a = a * a % MOD;
60         b >>= 1;
61     }
62     return x;
63 }
64
65 namespace linear_seq {
66     inline vector<int> BM(vector<int> x) {
67         vector<int> ls, cur;
68         int lf, ld;
69         for (int i = 0; i < int(x.size()); ++i) {
70             ll t = 0;
71             for (int j = 0; j < int(cur.size()); ++j)
72                 t = (t + x[i - j - 1] * (ll) cur[j]) %
73                     ↪ MOD;
74             if ((t - x[i]) % MOD == 0) continue;
75             if (!cur.size()) {
76                 cur.resize(i + 1);
77                 lf = i;
78                 ld = (t - x[i]) % MOD;
79                 continue;
80             }
81             ll k = -(x[i] - t) * qp(ld, MOD - 2) % MOD;
82             vector<int> c(i - lf - 1);
83             c.pb(k);
84             for (int j = 0; j < int(ls.size()); ++j)
85                 c.pb(-ls[j] * k % MOD);
86             if (c.size() < cur.size())
87                 ↪ c.resize(cur.size());
88             for (int j = 0; j < int(cur.size()); ++j)
89                 c[j] = (c[j] + cur[j]) % MOD;
90             if (i - lf + (int) ls.size() >= (int)
91                 ↪ cur.size())
92                 ls = cur, lf = i, ld = (t - x[i]) % MOD;
93             cur = c;
94         }
95         for (int i = 0; i < int(cur.size()); ++i)
96             cur[i] = (cur[i] % MOD + MOD) % MOD;
97         return cur;
98     }
99 }
100
101 int m;
102 ll a[SZ], h[SZ], t_[SZ], s[SZ], t[SZ];
103
104 inline void mull(ll* p, ll* q) {
105     for (int i = 0; i < m + m; ++i) t_[i] = 0;
106     for (int i = 0; i < m; ++i)
107         if (p[i])
108             for (int j = 0; j < m; ++j)
109                 t_[i + j] = (t_[i + j] + p[i] * q[j]) %
110                     ↪ MOD;
111     for (int i = m + m - 1; i >= m; --i)
112         if (t_[i])
113             for (int j = m - 1; ~j; --j)
114                 t_[i - j - 1] = (t_[i - j - 1] + t_[i]
115                     ↪ * h[j]) % MOD;
116     for (int i = 0; i < m; ++i) p[i] = t_[i];
117 }
118
119 inline ll calc(ll K) {
120     for (int i = m; ~i; --i)
121         s[i] = t[i] = 0;
122     s[0] = 1;
123     if (m != 1) t[1] = 1; else t[0] = h[0];
124     while (K) {
125         if (K & 1) mull(s, t);
126         mull(t, t);
127         K >>= 1;
128     }
129     ll su = 0;
130     for (int i = 0; i < m; ++i) su = (su + s[i] *
131         ↪ a[i]) % MOD;
132     return (su % MOD + MOD) % MOD;
133 }
134
135 inline int work(vector<int> x, ll n) {
136     if (n < int(x.size())) return x[n];
137     vector<int> v = BM(x);
138     m = v.size();
139     if (!m) return 0;
140     for (int i = 0; i < m; ++i) h[i] = v[i], a[i]
141         ↪ = x[i];
142     return calc(n);
143 }
144
145 //b=a0/(1-p)
146 inline void calc_generating_function(const
147     ↪ vector<int>& b, vector<int>& p,
148     ↪ vector<int>& a0) {
149     p = BM(b);
150     a0.resize(p.size());
151     for (int i = 0; i < a0.size(); ++i) {
152         a0[i] = b[i];
153         for (int j = 0; j < i; ++j) {
154             a0[i] += MOD - (p[j] * 1ll * b[i - j -
155                 ↪ 1]) % MOD;
156             if (a0[i] > MOD) {
157                 a0[i] -= MOD;
158             }
159         }
160     }
161 }
162
163 // berlecamp-massey ends
164
165 // AND-FFT begins

```

```

2
3 void fast_fourier(vector<int>& a) { // AND-FFT.
4     for (int k = 1; k < SZ(a); k *= 2)
5         for (int start = 0; start < (1 << K); start +=
6             ↪ 2 * k) {
7             for (int off = 0; off < k; ++off) {
8                 int a_val = a[start + off];
9                 int b_val = a[start + k + off];
10
11                 a[start + off] = b_val;
12                 a[start + k + off] = add(a_val, b_val);
13             }
14         }
15
16 void inverse_fast_fourier(vector<int>& a) {
17     for (int k = 1; k < SZ(a); k *= 2)
18         for (int start = 0; start < (1 << K); start +=
19             ↪ 2 * k) {
20             for (int off = 0; off < k; ++off) {
21                 int a_val = a[start + off];
22                 int b_val = a[start + k + off];
23
24                 a[start + off] = sub(b_val, 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 &comment
47         ↪ ) {
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     }

```

```

86     if (!y)
87     return x;
88     if (x->height < y->height) {
89     x->push ();
90     x->right = treap_merge (x->right, y);
91     x->recalc ();
92     return x;
93     } else {
94     y->push ();
95     y->left = treap_merge (x, y->left);
96     y->recalc ();
97     return y;
98     }
99 }

100
101 Treap* treap_getmin ( Treap *x, int &source, int
    ↪ &target, int &value ) {
102     assert (x);
103     x->push ();
104     if (x->min_path == 0) {
105     // memory leak, sorry
106     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,
    ↪ target, value);
112     value += x->add;
113     x->recalc ();
114     return x;
115     } else if (x->min_path == +1) {
116     x->right = treap_getmin (x->right, source,
    ↪ target, value);
117     value += x->add;
118     x->recalc ();
119     return x;
120     } else
121     assert (0);
122 }

123
124 Treap* treap_add ( Treap *x, int add ) {
125     if (!x)
126     return 0;
127     return &((*x) += add);
128 }

129
130
131 int main () {
132     int n, m;
133     while (scanf ("%d%d", &n, &m) == 2) {
134     Treap * g[n + 1];
135     for (int i = 0; i <= n; i++)
136     g[i] = 0;
137     for (int i = 1; i <= n; i++) {
138     int a;
139     assert (scanf ("%d", &a) == 1);
140     g[i] = treap_merge (g[i], new Treap (i, 0,
    ↪ a));
141     }
142     n++;
143     for (int i = 0; i < m; i++) {
144     int a, b, c;
145     assert (scanf ("%d%d%d", &a, &b, &c) == 3);
146     g[b] = treap_merge (g[b], new Treap (b, a,
    ↪ c));
147     }
148     DSU <pair <int, Treap*> > dsu (n + 1);
149     for (int i = 0; i < n; i++) {
150     dsu[i] = make_pair (i, g[i]);
151     }
152
153     int ans = 0, k = n;
154     int jump[2 * n], jump_from[2 * n], parent[2 *
    ↪ n], c[n];
155     vector <int> children[2 * n];
156     memset (c, 0, sizeof (c[0]) * n);
157     memset (parent, -1, sizeof (parent[0]) * 2 *
    ↪ n);
158     vector <int> finish;
159     for (int i = 0; i < n; i++) {
160     if (dsu[i].first == 0)
161     continue;
162     int u = i;
163     c[u] = 1;
164     while (true) {
165     int source, target, value;
166     dsu[u].second = treap_getmin (dsu[u].second,
    ↪ source, target, value);
167     if (dsu[target] == dsu[u])
168     continue;
169     treap_add (dsu[u].second, -value);
170     ans += value;
171     jump_from[dsu[u].first] = source;
172     jump[dsu[u].first] = target;
173     if (dsu[target].first == 0)
174     break;
175     if (!c[target]) {
176     c[target] = 1;
177     u = target;
178     continue;
179     }
180     assert (k < 2 * n);
181     int node = k++, t = target;
182     parent[dsu[u].first] = node;
183     children[node].push_back (dsu[u].first);
184     dsu[u].first = node;
185     Treap *v = dsu[u].second;
186     while (dsu[t].first != node) {
187     int next = jump[dsu[t].first];
188     parent[dsu[t].first] = node;
189     children[node].push_back (dsu[t].first);
190     v = treap_merge (v, dsu[t].second);
191     dsu.merge (u, t, make_pair (node, v));
192     t = next;
193     }
194     }
195     u = i;
196     while (dsu[u].first) {
197     int next = jump[dsu[u].first];
198     finish.push_back (dsu[u].first);
199     dsu.merge (u, 0, make_pair (0, (Treap *)0));

```

```

200     u = next;
201 }
202 }
203 bool ok[k];
204 int res[n];
205 memset(ok, 0, sizeof(ok[0]) * k);
206 memset(res, -1, sizeof(res[0]) * n);
207 function <void(int, int)> add_edge = [&ok,
    ↪ &parent, &res, &n] (int a, int b) {
208     assert(0 <= a && a < n);
209     assert(0 <= b && b < n);
210     assert(res[a] == -1);
211     res[a] = b;
212     while(a != -1 && !ok[a]) {
213         ok[a] = true;
214         a = parent[a];
215     }
216 };
217 function <void(int)> reach = [&ok, &reach,
    ↪ &children, &jump, &jump_from, &add_edge](
    ↪ int u) {
218     if(!ok[u])
219         add_edge(jump_from[u], jump[u]);
220     for(auto x : children[u])
221         reach(x);
222 };
223 for(auto x : finish)
224     reach(x);
225 printf("%d\n", ans);
226 for(int i = 1; i < n; i++)
227     printf("%d%c", res[i] ? res[i] : -1, "\n "[i
    ↪ < n - 1]);
228 }
229 return 0;
230 }
231
232 // 2-chinese ends
233
234 // general max weight match begins
235
236 #define DIST(e)
    ↪ (lab[e.u]+lab[e.v]-g[e.u][e.v].w*2)
237 using namespace std;
238 typedef long long ll;
239 const int N = 1023, INF = 1e9;
240 struct Edge {
241     int u, v, w;
242 } g[N][N];
243 int n, m, n_x, lab[N], match[N], slack[N], st[N],
    ↪ pa[N], flower_from[N][N], S[N], vis[N];
244 vector<int> flower[N];
245 deque<int> q;
246 void update_slack(int u, int x) {
247     if(!slack[x] || DIST(g[u][x]) <
    ↪ DIST(g[slack[x]][x])) slack[x] = u;
248 }
249 void set_slack(int x) {
250     slack[x] = 0;
251     for(int u = 1; u <= n; ++u)
252         if(g[u][x].w > 0 && st[u] != x && S[st[u]] ==
    ↪ 0) update_slack(u, x);
253 }
254 void q_push(int x) {
255     if(x <= n) return q.push_back(x);
256     for(int i = 0; i < flower[x].size(); i++)
    ↪ q_push(flower[x][i]);
257 }
258 void set_st(int x, int b) {
259     st[x] = b;
260     if(x <= n) return;
261     for(int i = 0; i < flower[x].size(); ++i)
    ↪ set_st(flower[x][i], b);
262 }
263 int get_pr(int b, int xr) {
264     int pr = find(flower[b].begin(),
    ↪ flower[b].end(), xr)-flower[b].begin();
265     if(pr % 2 == 1) {
266         reverse(flower[b].begin() + 1,
    ↪ flower[b].end());
267         return (int) flower[b].size()-pr;
268     }
269     else return pr;
270 }
271 void set_match(int u, int v) {
272     match[u] = g[u][v].v;
273     if(u <= n) return;
274     Edge e = g[u][v];
275     int xr = flower_from[u][e.u], pr = get_pr(u,
    ↪ xr);
276     for(int i = 0; i < pr; ++i)
    ↪ set_match(flower[u][i], flower[u][i^1]);
277     set_match(xr, v);
278     rotate(flower[u].begin(), flower[u].begin()
    ↪ +pr, flower[u].end());
279 }
280 void augment(int u, int v) {
281     int xnv = st[match[u]];
282     set_match(u, v);
283     if(!xnv) return;
284     set_match(xnv, st[pa[xnv]]);
285     augment(st[pa[xnv]], xnv);
286 }
287 int get_lca(int u, int v) {
288     static int t = 0;
289     for(++t; u || v; swap(u, v)) {
290         if(u == 0) continue;
291         if(vis[u] == t) return u;
292         vis[u] = t;
293         u = st[match[u]];
294         if(u) u = st[pa[u]];
295     }
296     return 0;
297 }
298 void add_blossom(int u, int lca, int v) {
299     int b = n+1;
300     while(b <= n_x && st[b] == b) ++b;
301     if(b > n_x) ++n_x;
302     lab[b] = 0, S[b] = 0;
303     match[b] = match[lca];
304     flower[b].clear();
305     flower[b].push_back(lca);
306     for(int x = u, y; x != lca; x = st[pa[y]])

```



```

74     flower[b].push_back(x), flower[b].push_back(y)
75     ↪ = st[match[x]], q_push(y);
76     reverse(flower[b].begin() + 1, flower[b].end());
77     for (int x = v, y; x != lca; x = st[pa[y]])
78         flower[b].push_back(x), flower[b].push_back(y)
79         ↪ = st[match[x]], q_push(y);
80     set_st(b, b);
81     for (int x = 1; x <= n_x; ++x) g[b][x].w =
82     ↪ g[x][b].w = 0;
83     for (int x = 1; x <= n; ++x) flower_from[b][x]
84     ↪ = 0;
85     for (int i = 0; i < flower[b].size(); ++i) {
86         int xs = flower[b][i];
87         for (int x = 1; x <= n_x; ++x)
88             if (g[b][x].w == 0 || DIST(g[xs][x]) <
89             ↪ DIST(g[b][x]))
90                 g[b][x] = g[xs][x], g[x][b] = g[x][xs];
91         for (int x = 1; x <= n; ++x)
92             if (flower_from[xs][x]) flower_from[b][x]
93             ↪ xs;
94     }
95     set_slack(b);
96 }
97 void expand_blossom(int b) // S[b] == 1 {
98     for (int i = 0; i < flower[b].size(); ++i)
99         set_st(flower[b][i], flower[b][i]);
100     int xr = flower_from[b][g[b][pa[b]].u], pr =
101     ↪ get_pr(b, xr);
102     for (int i = 0; i < pr; i += 2) {
103         int xs = flower[b][i], xns = flower[b][i+1];
104         pa[xs] = g[xns][xs].u;
105         S[xs] = 1, S[xns] = 0;
106         slack[xs] = 0, set_slack(xns);
107         q_push(xns);
108     }
109     S[xr] = 1, pa[xr] = pa[b];
110     for (int i = pr+1; i < flower[b].size(); ++i) {
111         int xs = flower[b][i];
112         S[xs] = -1, set_slack(xs);
113     }
114     st[b] = 0;
115 }
116 bool on_found_Edge(const Edge &e) {
117     int u = st[e.u], v = st[e.v];
118     if (S[v] == -1) {
119         pa[v] = e.u, S[v] = 1;
120         int nu = st[match[v]];
121         slack[v] = slack[nu] = 0;
122         S[nu] = 0, q_push(nu);
123     }
124     else if (S[v] == 0) {
125         int lca = get_lca(u, v);
126         if (!lca) return augment(u, v), augment(v,
127         ↪ u), 1;
128         else add_blossom(u, lca, v);
129     }
130     return 0;
131 }
132 bool matching() {
133     fill(S, S+n_x+1, -1), fill(slack, slack+n_x+1,
134     ↪ 0);
135     q.clear();
136     for (int x = 1; x <= n_x; ++x)
137         if (st[x] == x && !match[x]) pa[x] = 0, S[x]
138         ↪ = 0, q_push(x);
139     if (q.empty()) return 0;
140     for (;;) {
141         while (q.size()) {
142             int u = q.front();
143             q.pop_front();
144             if (S[st[u]] == 1) continue;
145             for (int v = 1; v <= n; ++v)
146                 if (g[u][v].w>0 && st[u] != st[v]) {
147                     if (DIST(g[u][v]) == 0) {
148                         if (on_found_Edge(g[u][v])) return 1;
149                     }
150                     else update_slack(u, st[v]);
151                 }
152         }
153         int d = INF;
154         for (int b = n+1; b <= n_x; ++b)
155             if (st[b] == b && S[b] == 1) d = min(d,
156             ↪ lab[b]/2);
157         for (int x = 1; x <= n_x; ++x)
158             if (st[x] == x && slack[x]) {
159                 if (S[x] == -1) d = min(d,
160                 ↪ DIST(g[slack[x]][x]));
161                 else if (S[x] == 0) d = min(d,
162                 ↪ DIST(g[slack[x]][x])/2);
163             }
164         for (int u = 1; u <= n; ++u) {
165             if (S[st[u]] == 0) {
166                 if (lab[u] <= d) return 0;
167                 lab[u] -= d;
168             }
169             else if (S[st[u]] == 1) lab[u] += d;
170         }
171         for (int b = n+1; b <= n_x; ++b)
172             if (st[b] == b) {
173                 if (S[st[b]] == 0) lab[b] += d*2;
174                 else if (S[st[b]] == 1) lab[b] -= d*2;
175             }
176         q.clear();
177         for (int x = 1; x <= n_x; ++x)
178             if (st[x] == x && slack[x] && st[slack[x]]
179             ↪ != x && DIST(g[slack[x]][x]) == 0)
180                 if (on_found_Edge(g[slack[x]][x])) return
181                 ↪ 1;
182         for (int b = n+1; b <= n_x; ++b)
183             if (st[b] == b && S[b] == 1 && lab[b] == 0)
184                 ↪ expand_blossom(b);
185     }
186     return 0;
187 }
188 pair < ll, int> weight_blossom() {
189     fill(match, match+n+1, 0);
190     n_x = n;
191     int n_matches = 0;
192     ll tot_weight = 0;
193     for (int u = 0; u <= n; ++u) st[u] = u,
194     ↪ flower[u].clear();
195     int w_max = 0;

```

```

179 for (int u = 1; u <= n; ++u)
180     for (int v = 1; v <= n; ++v) {
181         flower_from[u][v] = (u == v?u:0);
182         w_max = max(w_max, g[u][v].w);
183     }
184 for (int u = 1; u <= n; ++u) lab[u] = w_max;
185 while (matching()) ++n_matches;
186 for (int u = 1; u <= n; ++u)
187     if (match[u] && match[u] < u)
188         tot_weight += g[u][match[u]].w;
189 return make_pair(tot_weight, n_matches);
190 }
191 int main() {
192     cin>>n>>m;
193     for (int u = 1; u <= n; ++u)
194         for (int v = 1; v <= n; ++v)
195             g[u][v] = Edge {u, v, 0};
196     for (int i = 0, u, v, w; i < m; ++i) {
197         cin>>u>>v>>w;
198         g[u][v].w = g[v][u].w = w;
199     }
200     cout << weight_blossom().first << '\n';
201     for (int u = 1; u <= n; ++u) cout << match[u]
202     ↪ << ' ';
203 }

```

*// general max weighted match ends*

*// planar begins*

```

2 // обход граней
3
4 int n; // число вершин
5 vector < vector<int> > g; // граф
6 vector < vector<char> > used (n);
7 for (int i=0; i<n; ++i)
8     used[i].resize (g[i].size());
9 for (int i=0; i<n; ++i)
10     for (size_t j=0; j<g[i].size(); ++j)
11         if (!used[i][j]) {
12             used[i][j] = true;
13             int v = g[i][j], pv = i;
14             vector<int> facet;
15             for (;;) {
16                 facet.push_back (v);
17                 vector<int>::iterator it = find
18                 ↪ (g[v].begin(), g[v].end(), pv);
19                 //vector<int>::iterator it =
20                 ↪ lower_bound(g[v].begin(), g[v].end(),
21                 ↪ pv, cmp_ang(v));
22                 // cmp_ang(v) -- true, если меньше
23                 ↪ полярный угол относ v
24                 if (++it == g[v].end()) it =
25                 ↪ g[v].begin();
26                 if (used[v][it-g[v].begin()]) break;
27                 used[v][it-g[v].begin()] = true;
28                 pv = v, v = *it;
29             }
30             ... вывод facet - текущей грани ...
31         }
32     }

```

*// построение планарного графа*

```

33 struct point {

```

```

34 double x, y;
35 bool operator< (const point & p) const {
36     return x < p.x - EPS || abs (x - p.x) < EPS
37     ↪ && y < p.y - EPS;
38 }
39 };
40 map<point,int> ids;
41 vector<point> p;
42 vector < vector<int> > g;
43
44 int get_point_id (point pt) {
45     if (!ids.count(pt)) {
46         ids[pt] = (int)p.size();
47         p.push_back (pt);
48         g.resize (g.size() + 1);
49     }
50     return ids[p];
51 }
52
53 void intersect (pair<point,point> a,
54     ↪ pair<point,point> b, vector<point> & res) {
55     ... стандартная процедура, пересекает два
56     ↪ отрезка a и b и закидывает результат в res
57     ↪ ...
58     ... если отрезки перекрываются, то закидывает
59     ↪ те концы, которые попали внутрь первого
60     ↪ отрезка ...
61 }

```

```

62 int main() {
63     // входные данные
64     int m;
65     vector < pair<point,point> > a (m);
66     ... чтение ...
67
68     // построение графа
69     for (int i=0; i<m; ++i) {
70         vector<point> cur;
71         for (int j=0; j<m; ++j)
72             intersect (a[i], a[j], cur);
73         sort (cur.begin(), cur.end());
74         for (size_t j=0; j+1<cur.size(); ++j) {
75             int x = get_id (cur[j]), y = get_id
76             ↪ (cur[j+1]);
77             if (x != y) {
78                 g[x].push_back (y);
79                 g[y].push_back (x);
80             }
81         }
82     }
83     int n = (int) g.size();
84     // сортировка по углу и удаление кратных рёбер
85     for (int i=0; i<n; ++i) {
86         sort (g[i].begin(), g[i].end(), cmp_ang (i));
87         g[i].erase (unique (g[i].begin(),
88             ↪ g[i].end()), g[i].end());
89     }
90 }
91 // planar ends

```

```

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

```

xmodmap -e 'keycode 94='  
setxkbmap us

**DM****Кол-во корневых деревьев:**

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

**Кол-во эйлеровых циклов:**

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

**Пентагональная теорема Эйлера:**

$$\prod_{n=1}^{\infty} (1 - x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k (x^{k(3k-1)/2} + x^{k(3k+1)/2})$$

**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.

**Fast subset convolution**

$$(f * g)(S) := \sum_{T \subseteq S} f(T)g(S \setminus T)$$

$$\hat{f}(X) := \sum_{S \subseteq X} f(S)$$

$$f(S) = \sum_{X \subseteq S} (-1)^{|S \setminus X|} \hat{f}(X)$$

$$\hat{f}_0(X) := f(X)$$

$$\hat{f}_j(X) = \begin{cases} \hat{f}_{j-1}(X) & \text{if } j \notin X \\ \hat{f}_{j-1}(X \setminus j) + \hat{f}_{j-1}(X) & \text{if } j \in X \end{cases}$$

$$\hat{f}_n(X) == \hat{f}(X)$$

$$f_0(S) := \hat{f}(S)$$

$$f_j(S) = \begin{cases} f_{j-1}(S) & \text{if } j \notin S \\ -f_{j-1}(S \setminus j) + f_{j-1}(S) & \text{if } j \in S \end{cases}$$

$$f_n(S) == f(S)$$

$$\hat{f}(k, X) := \sum_{S \subseteq X, |S|=k} f(S)$$

$$f(S) == \hat{f}(|S|, S)$$

$$(\hat{f} \otimes \hat{g})(k, X) := \sum_{j=0}^k \hat{f}(j, X) \hat{g}(k-j, X)$$

$$(f * g)(S) = \sum_{X \subseteq S} (-1)^{|S \setminus X|} (\hat{f} \otimes \hat{g})(|S|, X)$$

calculate using  $f_j!$