

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 = 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 // pollard ends

```

pragma

```

#pragma GCC optimize('O3,no-stack-protector')
#pragma GCC target('sse,sse2,sse4,ssse3,popcnt,abm,mmx,a

```

Алгебра Pick

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

где B — количество целочисленных точек внутри многоугольника, а Γ — 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

$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$ 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 // 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 // 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 = 1; i != N; ++i) {
35         if (i == (2 << last))
36             ++last;
37
38         rev[i] = rev[i ^ (1 << last)] | (1 << (LOG -
39             ↪ 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 += (2
50             ↪ << lvl))
51             for (int pos = 0; pos != (1 << lvl); ++pos)
52                 ↪ {
53                     cd x = a[start + pos];
54                     cd y = a[start + pos + (1 << lvl)];
55
56                     y *= W[pos << (LOG - 1 - lvl)];
57
58                     a[start + pos] = x + y;
59                     a[start + pos + (1 << lvl)] = x - y;
60                 }
61 }
62
63 void inv_fft(vector<cd>& a) {
64     fft(a);
65     std::reverse(a.begin() + 1, a.end());
66
67     for (cd& elem: a)
68         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];
108         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
62     ↪ rows[i].erase(std::remove_if(rows[i].begin(),
63   ↪ rows[i].end(), cond), 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 }
100
101 // fast gauss ends
102
103 // simple geometry begins
104
105 struct Point {
106     double x, y;
107     Point operator+(const Point& p) const { return
108   ↪ {x + p.x, y + p.y}; }
109     Point operator-(const Point& p) const { return
110   ↪ {x - p.x, y - p.y}; }
111     Point operator*(const double d) const { return
112   ↪ {x * d, y * d}; }

```

```

8   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 ? -EPS
    ↪ : EPS);
51 }

52
53 // in case intersection is not a segment
54
55 inline bool intersect_segments(const Point& p1,
    ↪ const Point& p2, const Point& q1, const
    ↪ Point& q2, Point& x) {
56     Line l1(p1, p2), l2(q1, q2);
57     if (l1 || l2) return false;
58     x = l1 ^ l2;
59     return on_segment(p1, p2, x, false);
60 }

61 // in case circles are not equal
62 inline bool intersect_circles(const Circle& c1,
    ↪ const Circle& c2, Point& p1, Point& p2) {
63     double d = (c2.c - c1.c).dist();
64     if (d > c1.r + c2.r + EPS || d < fabs(c1.r -
    ↪ c2.r) - EPS)
65         return false;
66     double cosa = (sqr(d) + sqr(c1.r) - sqr(c2.r))
    ↪ / (2 * c1.r * d);
67     double l = c1.r * cosa, h = sqrt(sqr(c1.r) -
    ↪ sqr(l));
68     Point v = (c2.c - c1.c) * (1 / d), p = c1.c + v
    ↪ * l;
69     p1 = p + v.rotate() * h, p2 = p - v.rotate() *
    ↪ h;
70     return true;
71 }

72
73 inline bool intersect_circle_and_line(const
    ↪ Circle& c, const Line& l, Point& p1, Point&
    ↪ p2) {
74     double d = l.dist(c.c);
75     if (d > c.r + EPS)
76         return false;
77     Point p = l.projection(c.c);
78     Point n{l.b, -l.a};
79     double h = sqrt(sqr(c.r) - sqr(l.dist(c.c)));
80     p1 = p + n * h, p2 = p - n * h;
81     return true;
82 }

83
84 // simple geometry ends
85
86 // convex hull begins
87
88 struct Point {
89     int x, y;
90     Point operator-(const Point& p) const { return
    ↪ {x - p.x, y - p.y}; }
91     int64_t operator^(const Point& p) const {
92         ↪ return x * 111 * p.y - y * 111 * p.x; }
93     int64_t dist() const { return x * 111 * x + y *
    ↪ 111 * y; }
94     bool operator<(const Point& p) const { return x
    ↪ != p.x ? x < p.x : y < p.y; }
95 };

96
97 // all point on convex hull are included
98 vector<Point> convex_hull(vector<Point> pt) {
99     int n = pt.size();
100    Point p0 = *std::min_element(pt.begin(),
    ↪ pt.end());

```

```

15 std::sort(pt.begin(), pt.end(), [&p0](const      32
    ↪ Point& a, const Point& b) {
16     int64_t cp = (a - p0) ^ (b - p0);          33
17     return cp != 0 ? cp > 0 : (a - p0).dist() < 34
    ↪ (b - p0).dist();
18 });                                             35
19
20 int i = n - 1;                                36
21 for (; i > 0 && ((pt[i] - p0) ^ (pt[i - 1] -    37
    ↪ p0)) == 0; i--);
22 std::reverse(pt.begin() + i, pt.end());        38
23
24 vector<Point> ch;                              39
25 for (auto& p : pt) {
26     while (ch.size() > 1) {
27         auto& p1 = ch[(int) ch.size() - 1];    40
28         auto& p2 = ch[(int) ch.size() - 2];    41
29         int64_t cp = (p1 - p2) ^ (p - p1);      42
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             add_line(v + v + 2, m, r, a, b, nw);
75             return;
76         }
77
78         bool lef = f(nw, l) < f(line[v], l);
79         bool mid = f(nw, m) < f(line[v], m);
80
81         if (mid) swap(line[v], nw);
82
83         if (l == r - 1)
84             return;
85
86         if (lef != mid)
87             add_line(v + v + 1, l, m, a, b, nw);
88         else
89             add_line(v + v + 2, m, r, a, b, nw);
90     }
91
92     ftype get(int v, int l, int r, int x) {
93         if (l == r - 1)
94             return f(line[v], x);
95         int m = (l + r) / 2;
96         if (x < m) {
97             return min(f(line[v], x), get(v + v + 1, l,
    ↪ m, x));
98         } else {
99             return min(f(line[v], x), get(v + v + 2, m,
    ↪ r, x));
100         }
101     }
102 } cdt(maxn);
103
104 // convex hull with stack
105
106 ftype cross(point a, point b) {
107     return (conj(a) * b).y();
108 }
109
110 vector<point> hull, vecs;
111
112 void add_line(ftype k, ftype b) {
113     point nw = {k, b};
114     while(!vecs.empty() && dot(vecs.back(), nw -
    ↪ hull.back()) < 0) {
115         hull.pop_back();
116         vecs.pop_back();
117     }
118     if(!hull.empty()) {
119         vecs.push_back(li * (nw - hull.back()));
120     }
121     hull.push_back(nw);
122 }
123
124 int get(ftype x) {
125     point query = {x, 1};
126     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;
140     int r;
141     bool inv; // should direction be reversed
142 };
143
144 // node-weighted hld
145 vector<interval> get_path(int a, int b) {
146     vector<interval> front;
147     vector<interval> back;
148
149     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         a = par[upper[a]];
60     } else {
61         back.push_back({t_in[upper[b]], t_in[b],
62             ↪ false});
63         b = par[upper[b]];
64     }
65
66     front.push_back({min(t_in[a], t_in[b]),
67         ↪ max(t_in[a], t_in[b]), t_in[a] > t_in[b]});
68     // for edge-weighted hld add:
69     ↪ "front.back().l++;";
70     front.insert(front.end(), back.rbegin(),
71         ↪ back.rend());
72
73     return front;
74 }
75
76 // heavy-light ends
77
78 // max flow begins
79
80 struct edge{
81     int from, to;
82     int c, f, num;
83     edge(int from, int to, int c, int
84         ↪ num):from(from), to(to), c(c), f(0),
85         ↪ num(num){}
86     edge(){}
87 };
88
89 const int max_n = 600;
90
91 edge eds[150000];
92 int num = 0;
93 int it[max_n];
94 vector<int> gr[max_n];
95 int s, t;
96 vector<int> d(max_n);
97
98 bool bfs(int k) {
99     queue<int> q;
100     q.push(s);
101     fill(d.begin(), d.end(), -1);
102     d[s] = 0;
103     while (!q.empty()) {
104         int v = q.front();
105         q.pop();
106         for (int x : gr[v]) {
107             int to = eds[x].to;
108             if (d[to] == -1 && eds[x].c - eds[x].f >=
109                 ↪ (1 << k)){
110                 d[to] = d[v] + 1;
111                 q.push(to);
112             }
113         }
114     }
115
116     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 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 -
94                 ↪ 1].push_back(eds[num].num);
95             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){

```

```

51     if (d[e.to] > d[x] + (e.c - e.f) * e.w +38         w[j] = min(w[j], {a[x][j] + row[x] +
    ↪ dist[x] - dist[e.to]){                               ↪ col[j], x});
52     d[e.to] = d[x] + (e.c - e.f) * e.w +39
    ↪ 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 // bad hungarian begins
72
73 fill(par, par + 301, -1);
74 fill(par2, par2 + 301, -1);
75
76 int ans = 0;
77 for (int v = 0; v < n; v++){
78     memset(useda, false, sizeof(useda));
79     memset(usedb, false, sizeof(usedb));
80     useda[v] = true;
81     for (int i = 0; i < n; i++)
82         w[i] = make_pair(a[v][i] + row[v] + col[i],
83             ↪ v);
84     memset(prev, 0, sizeof(prev));
85     int pos;
86     while (true){
87         pair<pair<int, int>, int> p =
88             ↪ make_pair(make_pair(1e9, 1e9), 1e9);
89         for (int i = 0; i < n; i++)
90             if (!usedb[i])
91                 p = min(p, make_pair(w[i], i));
92         for (int i = 0; i < n; i++)
93             if (!useda[i])
94                 row[i] += p.first.first;
95         for (int i = 0; i < n; i++)
96             if (!usedb[i]){
97                 col[i] -= p.first.first;
98                 w[i].first -= p.first.first;
99             }
100         ans += p.first.first;
101         usedb[p.second] = true;
102         prev[p.second] = p.first.second; //уз оторой
103             ↪ е непыю
104         int x = par[p.second];
105         if (x == -1){
106             pos = p.second;
107             break;
108         }
109         useda[x] = true;
110         for (int j = 0; j < n; j++)
111             w[j] = min(w[j], {a[x][j] + row[x] +
112                 ↪ col[j], x});
113     }
114     cout << ans << "\n";
115     for (int i = 0; i < n; i++)
116         cout << par[i] + 1 << " " << i + 1 << "\n";
117
118 // bad hungarian ends
119
120 // Edmonds 0(n^3) begins
121
122 vector<int> gr[MAXN];
123 int match[MAXN], p[MAXN], base[MAXN], q[MAXN];
124 bool used[MAXN], blossom[MAXN];
125 int mark[MAXN];
126 int C = 1;
127
128 int lca(int a, int b) {
129     C++;
130     for (;) {
131         a = base[a];
132         mark[a] = C;
133         if (match[a] == -1) break;
134         a = p[match[a]];
135     }
136     for (;) {
137         b = base[b];
138         if (mark[b] == C) return b;
139         b = p[match[b]];
140     }
141 }
142
143 void mark_path(int v, int b, int children) {
144     while (base[v] != b) {
145         blossom[base[v]] = blossom[base[match[v]]] =
146             ↪ true;
147         p[v] = children;
148         children = match[v];
149         v = p[match[v]];
150     }
151 }
152
153 int find_path(int root) {
154     memset(used, 0, sizeof(used));
155     memset(p, -1, sizeof p);
156     for (int i = 0; i < N; i++)
157         base[i] = i;
158
159     used[root] = true;
160     int qh = 0, qt = 0;
161     q[qt++] = root;
162     while (qh < qt) {
163         int v = q[qh++];

```

```

45     for (int to : gr[v]) {
46         if (base[v] == base[to] || match[v] == to)
47             ↪ continue;
48         if (to == root || match[to] != -1 &&
49             ↪ p[match[to]] != -1) {
50             int curbase = lca(v, to);
51             memset(blossom, 0, sizeof(blossom));
52             mark_path(v, curbase, to);
53             mark_path(to, curbase, v);
54             for (int i = 0; i < N; i++)
55                 if (blossom[base[i]]) {
56                     base[i] = curbase;
57                     if (!used[i]) {
58                         used[i] = true;
59                         q[qt++] = i;
60                     }
61                 }
62             } else if (p[to] == -1) {
63                 p[to] = v;
64                 if (match[to] == -1)
65                     return to;
66                 to = match[to];
67                 used[to] = true;
68                 q[qt++] = to;
69             }
70         }
71     }
72     return -1;
73 }
74
75 memset(match, -1, sizeof match);
76 for (int i = 0; i < N; i++) {
77     if (match[i] == -1 && !gr[i].empty()) {
78         int v = find_path(i);
79         while (v != -1) {
80             int pv = p[v], ppv = match[pv];
81             match[v] = pv; match[pv] = v;
82             v = ppv;
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 }
106
107 return z;
108 }
109
110 vector<int> getP(string s){
111     vector<int> p;
112     p.resize(s.size(), 0);
113     int k = 0;
114     for (int i = 1; i < s.size(); i++){
115         while (k > 0 && s[i] == s[k])
116             k = p[k - 1];
117         if (s[i] == s[k])
118             k++;
119         p[i] = k;
120     }
121     return p;
122 }
123
124 vector<int> getH(string s){
125     vector<int> h;
126     h.resize(s.size() + 1, 0);
127     for (int i = 0; i < s.size(); i++)
128         h[i + 1] = ((h[i] * 111 * pow) + s[i] - 'a' +
129             ↪ 1) % mod;
130     return h;
131 }
132
133 int getHash(vector<int> &h, int l, int r){
134     int res = (h[r + 1] - h[l] * p[r - l + 1]) %
135         ↪ mod;
136     if (res < 0)
137         res += mod;
138     return res;
139 }
140
141 // string basis ends
142
143 // min cyclic shift begins
144
145 string min_cyclic_shift (string s) {
146     s += s;
147     int n = (int) s.length();
148     int i=0, ans=0;
149     while (i < n/2) {
150         ans = i;
151         int j=i+1, k=i;
152         while (j < n && s[k] <= s[j]) {
153             if (s[k] < s[j])
154                 k = i;
155             else
156                 ++k;
157             ++j;
158         }
159         while (i <= k) i += j - k;
160     }
161     return s.substr (ans, n/2);
162 }
163
164 // min cyclic shift ends
165
166 // suffix array O(n) begins
167
168 typedef vector<char> bits;

```

```

4
5 template<const int end>
6 void getBuckets(int *s, int *bkt, int n, int K) {
7     fill(bkt, bkt + K + 1, 0);
8     forn(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 // common for all algorithms
111 void kasaiLCP () {
112     for (int i = 0; i < N; i++) c[SA[i]] = i;
113     for (int i = 0, j, k = 0; i < N; LCP [c[i++]]
114         ↪ k)
115         if (c [i] > 0) for (k ? k-- : 0, j = SA[c[i]
116             ↪ - 1]; str[i + k] == str[j + k]; k++);
117     else k = 0;
118 }
119 void suffixArray () { // require last symbol is
120     ↪ char(0)
121     m = 256;
122     N = str.size();
123     DA ();
124     kasaiLCP ();
125 }
126 // suffix array O(n log n) ends
127
128 // bad suffix automaton begins
129
130 struct node{
131     map<char, int> go;
132     int len, suff;
133     long long sum_in;
134     node(){
135
136     };
137
138     node v[max_n * 4];
139
140     int add_node(int max_len){
141         //v[number].sum_in = 0;
142         v[number].len = max_len;
143         v[number].suff = -1;
144         number++;
145         return number - 1;
146     }
147
148     int last = add_node(0);
149
150     void add_char(char c) {
151         int cur = last;
152         int new_node = add_node(v[cur].len + 1);
153         last = new_node;
154         while (cur != -1){
155             if (v[cur].go.count(c) == 0){
156                 v[cur].go[c] = new_node;
157                 //v[new_node].sum_in += v[cur].sum_in;
158                 cur = v[cur].suff;
159                 if (cur == -1)
160                     v[new_node].suff = 0;
161             }else{
162                 int a = v[cur].go[c];
163                 if (v[a].len == v[cur].len + 1){
164                     v[new_node].suff = a;
165                 }else{
166                     int b = add_node(v[cur].len + 1);
167                     v[b].go = v[a].go;
168                     v[b].suff = v[a].suff;
169                     v[new_node].suff = b;
170                 }
171             }
172         }
173     }
174
175     while (cur != -1 && v[cur].go.count(c) !=
176         ↪ 0 && v[cur].go[c] == a){
177         v[cur].go[c] = b;
178         //v[a].sum_in -= v[cur].sum_in;
179         //v[b].sum_in += v[cur].sum_in;
180         cur = v[cur].suff;
181     }
182     v[a].suff = b;
183 }
184 return;
185 }
186 }
187 // bad suffix automaton ends
188
189 // pollard begins
190
191 const int max_step = 4e5;
192
193 unsigned long long gcd(unsigned long long a,
194     ↪ unsigned long long b){
195     if (!a) return 1;
196     while (a) swap(a, b%=a);
197     return b;
198 }
199
200 unsigned long long get(unsigned long long a,
201     ↪ unsigned long long b){
202     if (a > b)
203         return a-b;
204     else
205         return b-a;
206 }
207
208 unsigned long long pollard(unsigned long long n){
209     unsigned long long x = (rand() + 1) % n, y = 1,
210     ↪ g;
211     int stage = 2, i = 0;
212     g = gcd(get(x, y), n);
213     while (g == 1) {
214         if (i == max_step)
215             break;
216         if (i == stage) {
217             y = x;
218             stage <= 1;
219         }
220         x = (x * (__int128)x + 1) % n;
221         i++;
222         g = gcd(get(x, y), n);
223     }
224     return g;
225 }
226
227 // pollard ends
228
229 // linear sieve begins
230
231 const int N = 1000000;
232
233 int pr[N + 1], sz = 0;

```

```

6  /* minimal prime, mobius function, euler function
   ↪ */
7  int lp[N + 1], mu[N + 1], phi[N + 1];
8
9  lp[1] = mu[1] = phi[1] = 1;
10 for (int i = 2; i <= N; ++i) {
11     if (lp[i] == 0)
12         lp[i] = pr[sz++] = i;
13     for (int j = 0; j < sz && pr[j] <= lp[i] && i *
   ↪ pr[j] <= N; ++j)
14         lp[i * pr[j]] = pr[j];
15
16     mu[i] = lp[i] == lp[i / lp[i]] ? 0 : -1 * mu[i
   ↪ / lp[i]];
17     phi[i] = phi[i / lp[i]] * (lp[i] == lp[i /
   ↪ lp[i]] ? lp[i] : lp[i] - 1);
18 }
19
20 // linear sieve ends
21
22 // discrete log in sqrt(p) begins
23
24 int k = sqrt((double)p) + 2;
25
26 for (int i = k; i >= 1; i--)
27     mp[bin(b, (i * 111 * k) % (p-1), p)] = i;
28
29 bool answered = false;
30 int ans = INT32_MAX;
31 for (int i = 0; i <= k; i++){
32     int sum = (n * 111 * bin(b, i, p)) % p;
33     if (mp.count(sum) != 0){
34         int an = mp[sum] * 111 * k - i;
35         if (an < p)
36             ans = min(an, ans);
37     }
38 }
39
40 // discrete log in sqrt(p) ends
41
42 // prime roots mod n begins
43
44 int num = 0;
45 long long phi = n, nn = n;
46 for (long long x:primes){
47     if (x*x>nn)
48         break;
49     if (nn % x == 0){
50         while (nn % x == 0)
51             nn /= x;
52         phi -= phi/x;
53         num++;
54     }
55 }
56
57 if (nn != 1){
58     phi -= phi/nn;
59     num++;
60 }
61
62 if (!(num == 1 && n % 2 != 0) || n == 4 || n == 23
   ↪ 2 || (num == 2 && n % 2 == 0 && n % 4 != 0))
   ↪ {
63     cout << "-1\n";
64
65     continue;
66 }
67
68 vector<long long> v;
69 long long pp = phi;
70 for (long long x:primes){
71     if (x*x>pp)
72         break;
73     if (pp % x == 0){
74         while (pp % x == 0)
75             pp /= x;
76         v.push_back(x);
77     }
78 }
79
80 if (pp != 1){
81     v.push_back(pp);
82 }
83
84 while (true){
85     long long a = primes[rand()%5000]%n;
86     if (gcd(a, n) != 1)
87         continue;
88     bool bb = false;
89     for (long long x:v)
90         if (pow(a, phi/x) == 1){
91             bb = true;
92             break;
93         }
94     if (!bb){
95         cout << a << "\n";
96         break;
97     }
98 }
99
100 // prime roots mod n ends
101
102 // simplex begins
103
104 const double EPS = 1e-9;
105
106 typedef vector<double> vdbl;
107
108 // n variables, m inequalities
109 // Ax <= b, c*x -> max, x >= 0
110 double simplex( int n, int m, const vector<vdbl>
   ↪ &a0, const vdbl &b, const vdbl &c, vdbl &x )
   ↪ {
111     // Ax + Ez = b, A[m]*x -> max
112     // x = 0, z = b, x >= 0, z >= 0
113     vector<vdbl> a(m + 2, vdbl(n + m + 2));
114     vector<int> p(m);
115     forn(i, n)
116         a[m + 1][i] = c[i];
117     forn(i, m) {
118         forn(j, n)
119             a[i][j] = a0[i][j];
120         a[i][n + i] = 1;
121         a[i][m + n] = -1;
122         a[i][m + n + 1] = b[i];
123         p[i] = n + i;
124     }
125
126     // basis: enter "j", leave "ind+n"
127     auto pivot = [&]( int j, int ind ) {

```

```

27     double coef = a[ind][j];
28     assert(fabs(coef) > EPS);
29     forn(col, n + m + 2)
30         a[ind][col] /= coef;
31     forn(row, m + 2)
32         if (row != ind && fabs(a[row][j]) > EPS) {
33             coef = a[row][j];
34             forn(col, n + m + 2)
35                 a[row][col] -= a[ind][col] * coef;
36             a[row][j] = 0; // reduce precision error
37         }
38     p[ind] = j;
39 };
40
41 // the Simplex itself
42 auto iterate = [&](int nn) {
43     for (int run = 1; run; ) {
44         run = 0;
45         forn(j, nn) {
46             if (a[m][j] > EPS) { // strictly positive
47                 run = 1;
48                 double mi = INFINITY, t;
49                 int ind = -1;
50                 forn(i, m)
51                     if (a[i][j] > EPS && (t = a[i][n + m]
52                                     ↪ + 1) / a[i][j]) < mi - EPS)
53                         mi = t, ind = i;
54                 if (ind == -1)
55                     return false;
56                 pivot(j, ind);
57             }
58         }
59         return true;
60     };
61
62     int mi = min_element(b.begin(), b.end()) -
63     ↪ b.begin();
64     if (b[mi] < -EPS) {
65         a[m][n + m] = -1;
66         pivot(n + m, mi);
67         assert(iterate(n + m + 1));
68         if (a[m][m + n + 1] > EPS) // optimal value
69             ↪ is positive
70             return NAN;
71         forn(i, m)
72             if (p[i] == m + n) {
73                 int j = 0;
74                 while (find(p.begin(), p.end(), j) !=
75                     ↪ p.end() || fabs(a[i][j]) < EPS)
76                     j++, assert(j < m + n);
77                 pivot(j, i);
78             }
79     }
80     swap(a[m], a[m + 1]);
81     if (!iterate(n + m))
82         return INFINITY;
83     x = vdbl(n, 0);
84     forn(i, m)
85         if (p[i] < n)
86             x[p[i]] = a[i][n + m + 1];
87
88     return -a[m][n + m + 1];
89 }
90
91 // simplex usage:
92 vdbl x(n);
93 double result = simplex(n, m, a, b, c, x);
94 if (isinf(result))
95     puts("Unbounded");
96 else if (isnan(result))
97     puts("No solution");
98 else {
99     printf("%.9f :", result);
100     forn(i, n)
101         printf("%.9f", x[i]);
102     puts("");
103 }
104
105 // simplex ends
106 // sum over subsets begins
107 // fast subset convolution O(n 2^n)
108 for (int i = 0; i < (1<<N); ++i)
109     F[i] = A[i];
110 for (int i = 0; i < N; ++i) for (int mask = 0; mask
111     ↪ < (1<<N); ++mask){
112     if (mask & (1<<i))
113         F[mask] += F[mask^(1<<i)];
114 }
115 // sum over subsets ends
116
117 // algebra begins
118
119 Pick
120  $B + \frac{\Gamma}{2} - 1$ ,
121 где  $B$  - количество целочисленных точек внутри
122 ↪ многоугольника, а  $\Gamma$  - количество
123 ↪ целочисленных точек на границе
124 ↪ многоугольника.
125
126 Newton
127  $x_{i+1} = x_i - f(x_i) / f'(x_i)$ 
128
129 Catalan
130  $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$ 
131  $C_i = \frac{1}{n+1} \binom{2n}{n}$ 
132
133  $G_N = 2^{n(n-1)/2}$ 
134 Количество связанных помеченных графов
135  $Conn_N = G_N - \sum_{K=1}^{N-1} \binom{N}{K} Conn_K G_{N-K}$ 
136 ↪  $K$ 
137
138 Количество помеченных графов с  $K$  компонентами
139 ↪ связности
140  $D[N][K] = \sum_{S=1}^N \binom{N-1}{S-1} Conn_S D[N-S][K-1]$ 
141 ↪  $Conn_S D[N-S][K-1]$ 
142
143 Miller-Rabbin
144  $a = a^t$ 
145 FOR  $i = 1 \dots s$ 
146     if  $a^2 = 1$  &&  $|a| \neq 1$ 
147         NOT PRIME

```



```

26     a=a^2
27     return a==1 ? PRIME : NOT PRIME
28
29     Интегрирование по формуле Симпсона
30     \int_a^b f(x)dx - ?
31     x_i := a+ih, i=0\dots 2n
32     h = \frac{b-a}{2n}
33
34     \int =
35     \rightarrow (f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+2f(x_4)+\dots+4f(x_{2n-1})+f(x_{2n}))
36     \rightarrow \frac{h}{3}
37     Погрешность имеет порядок уменьшения как O(n^4)
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"
60             \rightarrow elements that go to left node
61             for(auto it = from; it != to; it++){
62                 b.pb(b.back() + f(*it));
63             }
64
65             //see how lambda function is used here
66             auto pivot = stable_partition(from, to, f);
67             l = new wavelet_tree(from, pivot, lo, mid);
68             r = new wavelet_tree(pivot, to, mid+1, hi);
69         }
70
71         //kth smallest element in [l, r]
72         int kth(int l, int r, int k){
73             if(l > r) return 0;
74             if(lo == hi) return lo;
75             //how many nos are there in left node from
76             \rightarrow [l, r]
77             int inleft = b[r] - b[l-1];
78             int lb = b[l-1]; //amt of nos from first
79             \rightarrow (l-1) nos that go in left
80             int rb = b[r]; //amt of nos from first (r)
81             \rightarrow nos that go in left
82             //so [lb+1, rb] represents nos from [l, r]
83             \rightarrow that go to left
84             if(k <= inleft) return this->l->kth(lb+1, rb, k);
85             \rightarrow , k);
86
87         // (l-1-lb) is amt of nos from first (l-1) nos
88         \rightarrow that go to right
89         // (r-rb) is amt of nos from first (r) nos
90         \rightarrow that go to right
91         //so [l-lb, r-rb] represents nos from [l, r]
92         \rightarrow that go to right
93         return this->r->kth(l-lb, r-rb, k-inleft);
94     }
95 }
96 // wavelet tree ends
97 // berlecamp-massey begins
98
99 const int SZ = MAXN;
100
101 ll qp(ll a, ll b) {
102     ll x = 1;
103     a %= MOD;
104     while (b) {
105         if (b & 1) x = x * a % MOD;
106         a = a * a % MOD;
107         b >>= 1;
108     }
109     return x;
110 }
111
112 namespace linear_seq {
113     inline vector<int> BM(vector<int> x) {
114         vector<int> ls, cur;
115         int lf, ld;
116         for (int i = 0; i < int(x.size()); ++i) {
117             ll t = 0;
118             for (int j = 0; j < int(cur.size()); ++j)
119                 t = (t + x[i - j - 1] * (ll) cur[j]) % MOD;
120             if ((t - x[i]) % MOD == 0) continue;
121             if (!cur.size()) {
122                 cur.resize(i + 1);
123                 lf = i;
124                 ld = (t - x[i]) % MOD;
125                 continue;
126             }
127             ll k = -(x[i] - t) * qp(ld, MOD - 2) % MOD;
128             vector<int> c(i - lf - 1);
129             c.pb(k);
130             for (int j = 0; j < int(ls.size()); ++j)
131                 c.pb(-ls[j] * k % MOD);
132             if (c.size() < cur.size())
133                 \rightarrow c.resize(cur.size());
134             for (int j = 0; j < int(cur.size()); ++j)
135                 c[j] = (c[j] + cur[j]) % MOD;
136             if (i - lf + (int) ls.size() >= (int) cur.size())
137                 \rightarrow cur.size()
138                 ls = cur, lf = i, ld = (t - x[i]) % MOD;
139             cur = c;
140         }
141         for (int i = 0; i < int(cur.size()); ++i)
142             cur[i] = (cur[i] % MOD + MOD) % MOD;
143         return cur;
144     }
145 }

```

```

47
48 int m;
49 ll a[SZ], h[SZ], t_[SZ], s[SZ], t[SZ];
50
51 inline void mull(ll* p, ll* q) {
52     for (int i = 0; i < m + m; ++i) t_[i] = 0;
53     for (int i = 0; i < m; ++i)
54         if (p[i])
55             for (int j = 0; j < m; ++j)
56                 t_[i + j] = (t_[i + j] + p[i] * q[j]) %
57                     MOD;
58     for (int i = m + m - 1; i >= m; --i)
59         if (t_[i])
60             for (int j = m - 1; ~j; --j)
61                 t_[i - j - 1] = (t_[i - j - 1] + t_[i]
62                     * h[j]) % MOD;
63     for (int i = 0; i < m; ++i) p[i] = t_[i];
64 }
65
66 inline ll calc(ll K) {
67     for (int i = m; ~i; --i)
68         s[i] = t[i] = 0;
69     s[0] = 1;
70     if (m != 1) t[1] = 1; else t[0] = h[0];
71     while (K) {
72         if (K & 1) mull(s, t);
73         mull(t, t);
74         K >>= 1;
75     }
76     ll su = 0;
77     for (int i = 0; i < m; ++i) su = (su + s[i]
78         * a[i]) % MOD;
79     return (su % MOD + MOD) % MOD;
80 }
81
82 inline int work(vector<int> x, ll n) {
83     if (n < int(x.size())) return x[n];
84     vector<int> v = BM(x);
85     m = v.size();
86     if (!m) return 0;
87     for (int i = 0; i < m; ++i) h[i] = v[i], a[i]
88         = x[i];
89     return calc(n);
90 }
91
92 //b=a0/(1-p)
93 inline void calc_generating_function(const
94     vector<int>& b, vector<int>& p,
95     vector<int>& a0) {
96     p = BM(b);
97     a0.resize(p.size());
98     for (int i = 0; i < a0.size(); ++i) {
99         a0[i] = b[i];
100         for (int j = 0; j < i; ++j) {
101             a0[i] += MOD - (p[j] * 1ll * b[i - j -
102                 1]) % MOD;
103             if (a0[i] > MOD) {
104                 a0[i] -= MOD;
105             }
106         }
107     }
108 }
109
110 }
111 }
112
113 // berlecamp-massey ends
114
115 // AND-FFT begins
116
117 void fast_fourier(vector<int>& a) { // AND-FFT.
118     for (int k = 1; k < SZ(a); k *= 2)
119         for (int start = 0; start < (1 << K); start +=
120             2 * k) {
121             for (int off = 0; off < k; ++off) {
122                 int a_val = a[start + off];
123                 int b_val = a[start + k + off];
124
125                 a[start + off] = b_val;
126                 a[start + k + off] = add(a_val, b_val);
127             }
128         }
129 }
130
131 void inverse_fast_fourier(vector<int>& a) {
132     for (int k = 1; k < SZ(a); k *= 2)
133         for (int start = 0; start < (1 << K); start +=
134             2 * k) {
135             for (int off = 0; off < k; ++off) {
136                 int a_val = a[start + off];
137                 int b_val = a[start + k + off];
138
139                 a[start + off] = sub(b_val, a_val);
140                 a[start + k + off] = a_val;
141             }
142         }
143 }
144
145 // AND-FFT ends
146
147 // 2-chinese begins
148
149 template <typename Info>
150 class DSU {
151 public:
152     DSU ( int n ) : jump (new int[n]), rank (new
153         int [n]), info (new Info [n]) {
154         for (int i = 0; i < n; i++) {
155             jump[i] = i;
156             rank[i] = 0;
157         }
158     }
159     Info& operator [] ( int x ) {
160         return info[get (x)];
161     }
162     void merge ( int a, int b, const Info &comment
163         ) {
164         a = get (a);
165         b = get (b);
166         if (rank[a] <= rank[b]) {
167             jump[a] = b;
168             rank[b] += rank[a] == rank[b];
169             info[b] = comment;
170         } else {
171             jump[b] = a;
172         }
173     }
174 };

```

```

24     info[a] = comment;
25 }
26 }
27 private:
28 int *jump, *rank;
29 Info *info;
30
31 int get ( int x ) {
32     return jump[x] == x ? x : (jump[x] = get
33         ↪ (jump[x]));
34 };
35
36 struct Treap {
37     int value, add;
38     int source, target, height;
39     int min_value, min_path;
40
41     Treap *left, *right;
42
43     Treap ( int _source, int _target, int _value )
44         ↪ : value (_value), add (0), source
45         ↪ (_source), target (_target) {
46         height = rand ();
47         min_value = value, min_path = 0;
48         left = right = 0;
49     }
50
51     Treap& operator += ( int sub ) {
52         add += sub;
53         return *this;
54     }
55
56     void push () {
57         if (!add)
58             return;
59         if (left) {
60             left->add += add;
61         }
62         if (right) {
63             right->add += add;
64         }
65         value += add;
66         min_value += add;
67         add = 0;
68     }
69
70     void recalc () {
71         min_value = value;
72         min_path = 0;
73         if (left && left->min_value + left->add <
74             ↪ min_value) {
75             min_value = left->min_value + left->add;
76             min_path = -1;
77         }
78         if (right && right->min_value + right->add <
79             ↪ min_value) {
80             min_value = right->min_value + right->add;
81             min_path = +1;
82         }
83     }
84
85     Treap* treap_merge ( Treap *x, Treap *y ) {
86         if (!x)
87             return y;
88         if (!y)
89             return x;
90         if (x->height < y->height) {
91             x->push ();
92             x->right = treap_merge (x->right, y);
93             x->recalc ();
94             return x;
95         } else {
96             y->push ();
97             y->left = treap_merge (x, y->left);
98             y->recalc ();
99             return y;
100         }
101     }
102
103     Treap* treap_getmin ( Treap *x, int &source, int
104         ↪ &target, int &value ) {
105         assert (x);
106         x->push ();
107         if (x->min_path == 0) {
108             // memory leak, sorry
109             source = x->source;
110             target = x->target;
111             value = x->value + x->add;
112             return treap_merge (x->left, x->right);
113         } else if (x->min_path == -1) {
114             x->left = treap_getmin (x->left, source,
115                 ↪ target, value);
116             value += x->add;
117             x->recalc ();
118             return x;
119         } else if (x->min_path == +1) {
120             x->right = treap_getmin (x->right, source,
121                 ↪ target, value);
122             value += x->add;
123             x->recalc ();
124             return x;
125         } else
126             assert (0);
127     }
128
129     Treap* treap_add ( Treap *x, int add ) {
130         if (!x)
131             return 0;
132         return &((*x) += add);
133     }
134
135     int main () {
136         int n, m;
137         while (scanf ("%d%d", &n, &m) == 2) {
138             Treap *g[n + 1];
139             for (int i = 0; i <= n; i++)
140                 g[i] = 0;
141             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,
141         ↪ a));
142 }
143 n++;
144 for (int i = 0; i < m; i++) {
145     int a, b, c;
146     assert (scanf ("%d%d%d", &a, &b, &c) == 3);
147     g[b] = treap_merge (g[b], new Treap (b, a,
148         ↪ c));
149 }
150 DSU <pair <int, Treap*> > dsu (n + 1);
151 for (int i = 0; i < n; i++) {
152     dsu[i] = make_pair (i, g[i]);
153 }
154 int ans = 0, k = n;
155 int jump[2 * n], jump_from[2 * n], parent[2 *
156     ↪ n], c[n];
157 vector <int> children[2 * n];
158 memset (c, 0, sizeof (c[0]) * n);
159 memset (parent, -1, sizeof (parent[0]) * 2 *
160     ↪ n);
161 vector <int> finish;
162 for (int i = 0; i < n; i++) {
163     if (dsu[i].first == 0)
164         continue;
165     int u = i;
166     c[u] = 1;
167     while (true) {
168         int source, target, value;
169         dsu[u].second = treap_getmin (dsu[u].second,
170             ↪ source, target, value);
171         if (dsu[target] == dsu[u])
172             continue;
173         treap_add (dsu[u].second, -value);
174         ans += value;
175         jump_from[dsu[u].first] = source;
176         jump[dsu[u].first] = target;
177         if (dsu[target].first == 0)
178             break;
179         if (!c[target]) {
180             c[target] = 1;
181             u = target;
182             continue;
183         }
184     }
185     assert (k < 2 * n);
186     int node = k++, t = target;
187     parent[dsu[u].first] = node;
188     children[node].push_back (dsu[u].first);
189     dsu[u].first = node;
190     Treap *v = dsu[u].second;
191     while (dsu[t].first != node) {
192         int next = jump[dsu[t].first];
193         parent[dsu[t].first] = node;
194         children[node].push_back (dsu[t].first);
195         v = treap_merge (v, dsu[t].second);
196         dsu.merge (u, t, make_pair (node, v));
197         t = next;
198     }
199 }
200
201 }
202 u = i;
203 while (dsu[u].first) {
204     int next = jump[dsu[u].first];
205     finish.push_back (dsu[u].first);
206     dsu.merge (u, 0, make_pair (0, (Treap *)0));
207     u = next;
208 }
209 bool ok[k];
210 int res[n];
211 memset (ok, 0, sizeof (ok[0]) * k);
212 memset (res, -1, sizeof (res[0]) * n);
213 function <void (int, int)> add_edge = [&ok,
214     ↪ &parent, &res, &n] ( int a, int b ) {
215     assert (0 <= a && a < n);
216     assert (0 <= b && b < n);
217     assert (res[a] == -1);
218     res[a] = b;
219     while (a != -1 && !ok[a]) {
220         ok[a] = true;
221         a = parent[a];
222     }
223 };
224 function <void (int)> reach = [&ok, &reach,
225     ↪ &children, &jump, &jump_from, &add_edge] (
226     ↪ int u ) {
227     if (!ok[u])
228         add_edge (jump_from[u], jump[u]);
229     for (auto x : children[u])
230         reach (x);
231 };
232 for (auto x : finish)
233     reach (x);
234 printf ("%d\n", ans);
235 for (int i = 1; i < n; i++)
236     printf ("%d%c", res[i] ? res[i] : -1, "\n "[i
237         ↪ < n - 1]);
238 }
239 return 0;
240 }
241
242 // 2-chinese ends
243
244 // slow min circulation begins
245
246 struct Edge {
247     int a;
248     int b;
249     int cost;
250 };
251
252 vector<int> negative_cycle(int n, vector<Edge>
253     ↪ &edges) {
254     // O(nm), return ids of edges in negative cycle
255
256     vector<int> d(n);
257     vector<int> p(n, -1); // last edge ids
258
259     const int inf = 1e9;
260
261     int x = -1;

```

```

18     for (int i = 0; i < n; i++) {
19         x = -1;
20         for (int j = 0; j < edges.size(); j++) {
21             Edge &e = edges[j];
22
23             if (d[e.b] > d[e.a] + e.cost) {
24                 d[e.b] = max(-inf, d[e.a] + e.cost);
25                 p[e.b] = j;
26                 x = e.b;
27             }
28         }
29     }
30
31     if (x == -1)
32         return vector<int>(); // no negative cycle
33
34     for (int i = 0; i < n; i++)
35         x = edges[p[x]].a;
36
37     vector<int> result;
38     for (int cur = x; ; cur = edges[p[cur]].a) {
39         if (cur == x && result.size() > 0) break;
40         result.push_back(p[cur]);
41     }
42     reverse(result.begin(), result.end());
43
44     return result;
45 }
46
47 vector<int> min_avg_cycle(int n, vector<Edge>
48 ↪ &edges) {
49     const int inf = 1e3;
50
51     for (auto &e : edges)
52         e.cost *= n * n;
53
54     int l = -inf;
55     int r = inf;
56     while (l + 1 < r) {
57         int m = (l + r) / 2;
58         for (auto &e : edges)
59             e.cost -= m;
60
61         if (negative_cycle(n, edges).empty())
62             l = m;
63         else
64             r = m;
65
66         for (auto &e : edges)
67             e.cost += m;
68     }
69
70     if (r >= 0) // if only negative needed
71         return vector<int>();
72
73     for (auto &e : edges)
74         e.cost -= r;
75
76     vector<int> result = negative_cycle(n, edges);
77
78     for (auto &e : edges)
79         e.cost += r;
80
81     for (auto &e : edges)
82         e.cost /= n * n;
83
84     return result;
85 }
86
87 struct edge {
88     int from, to;
89     int c, f, cost;
90 };
91
92 const int max_n = 200;
93
94 vector<int> gr[max_n];
95 vector<edge> edges;
96
97 void add(int fr, int to, int c, int cost) {
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}); //
102    ↪ single
103 }
104
105 void calc_min_circulation(int n) {
106     while (true) {
107         vector<Edge> eds;
108         vector<int> origin;
109
110         for (int i = 0; i < edges.size(); i++) {
111             edge &e = edges[i];
112             if (e.c - e.f > 0) {
113                 eds.push_back({e.from, e.to, e.cost});
114                 origin.push_back(i);
115             }
116         }
117
118         vector<int> cycle = negative_cycle(n, eds);
119
120         if (cycle.empty())
121             break;
122
123         for (auto id : cycle) {
124             int x = origin[id];
125             edges[x].f += 1;
126             edges[x ^ 1].f -= 1;
127         }
128     }
129
130     // slow min circulation ends
131
132     // fast hashtable begins
133
134     #include <ext/pb_ds/assoc_container.hpp>
135     using namespace __gnu_pbds;
136     gp_hash_table<int, int> table;
137
138     1
139     2
140     3
141     4
142     5
143     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

```

DM

Кол-во корней деревьев:

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

Кол-во эйлеровых циклов:

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

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