

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
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 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

$$Conn_N = G_N - \frac{1}{N} \sum_{K=1}^{N-1} K \binom{N}{K} Conn_K G_{N-K}$$

Количество помеченных графов с K компонентами связности

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

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 // 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
109 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 // return true;
98 // }
99
100 // fast gauss ends
101
102 // simple geometry begins
103
104 struct Point {
105     double x, y;
106     Point operator+(const Point& p) const { return
107 // {x + p.x, y + p.y}; }
108     Point operator-(const Point& p) const { return
109 // {x - p.x, y - p.y}; }
110     Point operator*(const double d) const { return
111 // {x * d, y * d}; }
112     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);
55
56  // in case circles are not equal
57  inline bool intersect_circles(const Circle& c1,
    ↪ const Circle& c2, Point& p1, Point& p2) {
58      double d = (c2.c - c1.c).dist();
59      if (d > c1.r + c2.r + EPS || d < fabs(c1.r -
    ↪ c2.r) - EPS)
60          return false;
61      double cosa = (sqr(d) + sqr(c1.r) - sqr(c2.r))
    ↪ / (2 * c1.r * d);
62      double l = c1.r * cosa, h = sqrt(sqr(c1.r) -
    ↪ sqr(l));
63      Point v = (c2.c - c1.c) * (1 / d), p = c1.c + v
    ↪ * l;
64      p1 = p + v.rotate() * h, p2 = p - v.rotate() *
    ↪ h;
65      return true;
66  }
67
68  inline bool intersect_circle_and_line(const
    ↪ Circle& c, const Line& l, Point& p1, Point&
    ↪ p2) {
69      double d = l.dist(c.c);
70      if (d > c.r + EPS)
71          return false;
72      Point p = l.projection(c.c);
73      Point n{l.b, -l.a};
74      double h = sqrt(sqr(c.r) - sqr(l.dist(c.c)));
75      p1 = p + n * h, p2 = p - n * h;
76      return true;
77  }
78
79  // simple geometry ends
80
81  // convex hull begins
82
83  struct Point {
84      int x, y;
85      Point operator-(const Point& p) const { return
    ↪ {x - p.x, y - p.y}; }
86      int64_t operator^(const Point& p) const {
    ↪ return x * 1ll * p.y - y * 1ll * p.x; }
87      int64_t dist() const { return x * 1ll * x + y *
    ↪ 1ll * y; }
88      bool operator<(const Point& p) const { return x
    ↪ != p.x ? x < p.x : y < p.y; }
89  };
90
91  // all point on convex hull are included
92  vector<Point> convex_hull(vector<Point> pt) {
93      int n = pt.size();
94      Point p0 = *std::min_element(pt.begin(),
    ↪ pt.end());
95      std::sort(pt.begin(), pt.end(), [&p0](const
    ↪ Point& a, const Point& b) {
96          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
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     for (int x : gr[v])
131         if (x != pr) {
132             dfs_build(x, first ? upper[v] : x, v);
133             first = false;
134         }
135 }
136
137 struct interval {
138     int l;
139     int r;
140     bool inv; // should direction be reversed
141 };
142
143 // node-weighted hld
144 vector<interval> get_path(int a, int b) {
145     vector<interval> front;
146     vector<interval> back;
147
148     while (upper[a] != upper[b]) {
149         if (dep[upper[a]] > dep[upper[b]]) {
150             front.push_back({t_in[upper[a]], t_in[a],
151                             ↪ true});
152             a = par[upper[a]];
153         } else {
154             back.push_back({t_in[upper[b]], t_in[b],
155                             ↪ false});
156             b = par[upper[b]];
157         }
158     }
159     front.push_back({min(t_in[a], t_in[b]),
160                     ↪ max(t_in[a], t_in[b]), t_in[a] > t_in[b]});
161     // for edge-weighted hld add:
162     ↪ "front.back().l++;";
163     front.insert(front.end(), back.rbegin(),
164                 ↪ back.rend());
165
166     return front;
167 }
168
169 // heavy-light ends
170
171 // max flow begins
172
173 struct edge{
174     int from, to;
175     int c, f, num;
176     edge(int from, int to, int c, int
177         ↪ num):from(from), to(to), c(c), f(0),
178         ↪ num(num){}
179     edge(){}
180 };
181
182 const int max_n = 600;
183
184 edge eds[150000];
185 int num = 0;
186 int it[max_n];
187 vector<int> gr[max_n];
188 int s, t;
189 vector<int> d(max_n);
190
191 bool bfs(int k) {
192     queue<int> q;
193     q.push(s);
194     fill(d.begin(), d.end(), -1);
195     d[s] = 0;
196     while (!q.empty()) {
197         int v = q.front();
198         q.pop();
199         for (int x : gr[v]) {
200             int to = eds[x].to;
201             if (d[to] == -1 && eds[x].c - eds[x].f >=
202                 ↪ (1 << k)){
203                 d[to] = d[v] + 1;
204                 q.push(to);
205             }
206         }
207     }
208     return (d[t] != -1);
209 }
210
211 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 // 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 верной
103             ⇨ в цепью
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 }
115
116 while (pos != -1){
117     int nxt = par2[prev[pos]];
118     par[pos] = prev[pos];
119     par2[prev[pos]] = pos;
120     pos = nxt;
121 }
122 }
123 cout << ans << "\n";
124 for (int i = 0; i < n; i++)
125     cout << par[i] + 1 << " " << i + 1 << "\n";
126
127 // bad hungarian ends
128
129 // Edmonds O(n^3) begins
130
131 vector<int> gr[MAXN];
132 int match[MAXN], p[MAXN], base[MAXN], q[MAXN];
133 bool used[MAXN], blossom[MAXN];
134 int mark[MAXN];
135 int C = 1;
136
137 int lca(int a, int b) {
138     C++;
139     for (;;) {
140         a = base[a];
141         mark[a] = C;
142         if (match[a] == -1) break;
143         a = p[match[a]];
144     }
145
146     for (;;) {
147         b = base[b];
148         if (mark[b] == C) return b;
149         b = p[match[b]];
150     }
151 }
152
153 void mark_path(int v, int b, int children) {
154     while (base[v] != b) {
155         blossom[base[v]] = blossom[base[match[v]]] =
156             ⇨ true;
157         p[v] = children;
158         children = match[v];
159         v = p[match[v]];
160     }
161 }
162
163 int find_path(int root) {
164     memset(used, 0, sizeof(used));
165     memset(p, -1, sizeof p);
166     for (int i = 0; i < N; i++)
167         base[i] = i;
168
169     used[root] = true;
170     int qh = 0, qt = 0;
171     q[qt++] = root;
172     while (qh < qt) {
173         int v = q[qh++];
174         for (int to : gr[v]) {
175             if (base[v] == base[to] || match[v] == to)
176                 ⇨ continue;

```

```

47     if (to == root || match[to] != -1 &&
48         ↪ p[match[to]] != -1) {
49         int curbase = lca(v, to);
50         memset(blossom, 0, sizeof(blossom));
51         mark_path(v, curbase, to);
52         mark_path(to, curbase, v);
53         for (int i = 0; i < N; i++)
54             if (blossom[base[i]]) {
55                 base[i] = curbase;
56                 if (!used[i]) {
57                     used[i] = true;
58                     q[qt++] = i;
59                 }
60             }
61         else if (p[to] == -1) {
62             p[to] = v;
63             if (match[to] == -1)
64                 return to;
65             to = match[to];
66             used[to] = true;
67             q[qt++] = to;
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;
110     p.resize(s.size(), 0);
111     int k = 0;
112     for (int i = 1; i < s.size(); i++){
113         while (k > 0 && s[i] == s[k])
114             k = p[k - 1];
115         if (s[i] == s[k])
116             k++;
117         p[i] = k;
118     }
119     return p;
120 }
121
122 vector<int> getH(string s){
123     vector<int> h;
124     h.resize(s.size() + 1, 0);
125     for (int i = 0; i < s.size(); i++)
126         h[i + 1] = ((h[i] * 111 * pow) + s[i] - 'a' +
127             ↪ 1) % mod;
128     return h;
129 }
130
131 int getHash(vector<int> &h, int l, int r){
132     int res = (h[r + 1] - h[l] * p[r - l + 1]) %
133         ↪ mod;
134     if (res < 0)
135         res += mod;
136     return res;
137 }
138
139 // string basis ends
140
141 // min cyclic shift begins
142
143 string min_cyclic_shift (string s) {
144     s += s;
145     int n = (int) s.length();
146     int i=0, ans=0;
147     while (i < n/2) {
148         ans = i;
149         int j=i+1, k=i;
150         while (j < n && s[k] <= s[j]) {
151             if (s[k] < s[j])
152                 k = i;
153             else
154                 ++k;
155             ++j;
156         }
157         while (i <= k) i += j - k;
158     }
159     return s.substr (ans, n/2);
160 }
161
162 // min cyclic shift ends
163
164 // suffix array O(n) begins
165
166 typedef vector<char> bits;
167
168 template<const int end>
169 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)
    ↪ SA[--w[x[y[i]]]] = y[i];
98  }
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++]]
    ↪ k)
115         if (c [i] > 0) for (k ? k-- : 0, j = SA[c[i]
    ↪ - 1]; str[i + k] == str[j + k]; k++);
116     else k = 0;
117 }
118
119 void suffixArray () { // require last symbol is
    ↪ char(0)
120     m = 256;
121     N = str.size();
122     DA ();
123     kasaiLCP ();
124 }
125 // suffix array O(n log n) ends
1
// bad suffix automaton begins
2
3 struct node{
4     map<char, int> go;
5     int len, suff;
6     long long sum_in;
7     node(){
8     };
9
10 node v[max_n * 4];
11
12 int add_node(int max_len){
13     //v[number].sum_in = 0;
14     v[number].len = max_len;
15     v[number].suff = -1;
16     number++;
17     return number - 1;
18 }
19
20 int last = add_node(0);
21
22 void add_char(char c) {
23     int cur = last;
24     int new_node = add_node(v[cur].len + 1);
25     last = new_node;
26     while (cur != -1){
27         if (v[cur].go.count(c) == 0){
28             v[cur].go[c] = new_node;
29             //v[new_node].sum_in += v[cur].sum_in;
30             cur = v[cur].suff;
31             if (cur == -1)
32                 v[new_node].suff = 0;
33         }else{
34             int a = v[cur].go[c];
35             if (v[a].len == v[cur].len + 1){
36                 v[new_node].suff = a;
37             }else{
38                 int b = add_node(v[cur].len + 1);
39                 v[b].go = v[a].go;
40                 v[b].suff = v[a].suff;
41                 v[new_node].suff = b;
42                 while (cur != -1 && v[cur].go.count(c) !=
    ↪ 0 && v[cur].go[c] == a){
43                     v[cur].go[c] = b;
44                     //v[a].sum_in -= v[cur].sum_in;
45                     //v[b].sum_in += v[cur].sum_in;
46                     cur = v[cur].suff;
47                 }
48                 v[a].suff = b;
49             }
50             return;
51         }
52     }
53 }
54
55 // bad suffix automaton ends
56
// pollard begins
1
2
3 const int max_step = 4e5;
4
5 unsigned long long gcd(unsigned long long a,
    ↪ unsigned long long b){
6     if (!a) return 1;
7     while (a) swap(a, b%=a);
8     return b;
9 }
10
11 unsigned long long get(unsigned long long a,
    ↪ unsigned long long b){
12     if (a > b)
13         return a-b;
14     else
15         return b-a;
16 }
17
18 unsigned long long pollard(unsigned long long n){
19     unsigned long long x = (rand() + 1) % n, y = 1,
    ↪ g;
20     int stage = 2, i = 0;
21     g = gcd(get(x, y), n);
22     while (g == 1) {
23         if (i == max_step)
24             break;
25         if (i == stage) {
26             y = x;
27             stage <= 1;
28         }
29         x = (x * (__int128)x + 1) % n;
30         i++;
31         g = gcd(get(x, y), n);
32     }
33     return g;
34 }
35
36 // pollard ends
37
// linear sieve begins
1
2
3 const int N = 1000000;
4
5 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 * 1ll * 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 * 1ll * bin(b, i, p)) % p;
33     if (mp.count(sum) != 0){
34         int an = mp[sum] * 1ll * 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 ||
63     2 || (num == 2 && n % 2 == 0 && n % 4 != 0)){
64     cout << "-1\n";
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>
111     &a0, const vdbl &b, const vdbl &c, vdbl &x )
112 {
113     // Ax + Ez = b, A[m]*x -> max
114     // x = 0, z = b, x >= 0, z >= 0
115     vector<vdbl> a(m + 2, vdbl(n + m + 2));
116     vector<int> p(m);
117     forn(i, n)
118         a[m + 1][i] = c[i];
119     forn(i, m) {
120         forn(j, n)
121             a[i][j] = a0[i][j];
122         a[i][n + i] = 1;
123         a[i][m + n] = -1;
124         a[i][m + n + 1] = b[i];
125         p[i] = n + i;
126     }
127
128     // basis: enter "j", leave "ind+n"
129     auto pivot = [&]( int j, int ind ) {
130         double coef = a[ind][j];
131         assert(fabs(coef) > EPS);
132         forn(col, n + m + 2)
133             a[ind][col] /= coef;
134         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     return -a[m][n + m + 1];
88 }
89
90 // simplex usage:
91 vdbl x(n);
92
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 связных помеченных графов
135  $Conn_N = G_N - \sum_{K=1}^{N-1} K \binom{N}{K} Conn_K$ 
136 ↪  $G_{N-K}$ 
137
138 Количество помеченных графов с K компонентами
139 ↪ связности
140  $D[N][K] = \sum_{S=1}^{N-1} \binom{N-1}{S-1} Conn_S D[N-S][K-1]$ 
141
142 Miller-Rabbin
143  $a = a^t$ 
144 FOR  $i = 1 \dots s$ 
145     if  $a^2 = 1$  &&  $|a| \neq 1$ 
146         NOT PRIME
147      $a = a^2$ 
148 return  $a == 1$  ? PRIME : NOT PRIME

```



```

30 Интегрирование по формуле Симпсона
31  $\int_a^b f(x)dx = ?$ 
32  $x_i := a + ih, i=0 \dots 2n$ 
33  $h = \frac{b-a}{2n}$ 
34
35  $\int = (f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) +$ 
36  $\rightarrow 2f(x_4) + \dots + 4f(x_{2n-1}) +$ 
37  $\rightarrow f(x_{2n})) \frac{h}{3}$ 
38 Порешность имеет порядок уменьшения как  $O(n^4)$ .
39
40 // algebra ends
41
42 // wavelet tree begins
43
44 struct wavelet_tree{
45     int lo, hi;
46     wavelet_tree *l, *r;
47     vi b;
48
49     //nos are in range [x,y]
50     //array indices are [from, to)
51     wavelet_tree(int *from, int *to, int x, int y){
52         lo = x, hi = y;
53         if(lo == hi or from >= to) return;
54
55         int mid = (lo+hi)/2;
56         auto f = [mid](int x){
57             return x <= mid;
58         };
59         b.reserve(to-from+1);
60         b.pb(0);
61         //b[i] = no of elements from first "i"
62         //elements that go to left node
63         for(auto it = from; it != to; it++){
64             b.pb(b.back() + f(*it));
65
66         //see how lambda function is used here
67         auto pivot = stable_partition(from, to, f);
68         l = new wavelet_tree(from, pivot, lo, mid);
69         r = new wavelet_tree(pivot, to, mid+1, hi);
70     }
71
72     //kth smallest element in [l, r]
73     int kth(int l, int r, int k){
74         if(l > r) return 0;
75         if(lo == hi) return lo;
76         //how many nos are there in left node from
77         // [l, r]
78         int inleft = b[r] - b[l-1];
79         int lb = b[l-1]; //amt of nos from first
80         // (l-1) nos that go in left
81         int rb = b[r]; //amt of nos from first (r)
82         // nos that go in left
83         //so [lb+1, rb] represents nos from [l, r]
84         //that go to left
85         if(k <= inleft) return this->l->kth(lb+1, rb
86         // , k);
87
88         //(l-1-lb) is amt of nos from first (l-1) nos
89         //that go to right
90
91         //(r-rb) is amt of nos from first (r) nos
92         //that go to right
93         //so [l-lb, r-rb] represents nos from [l, r]
94         //that go to right
95         return this->r->kth(l-lb, r-rb, k-inleft);
96     }
97 };
98
99 // wavelet tree ends
100 // berlecamp-massey begins
101
102 const int SZ = MAXN;
103
104 ll qp(ll a, ll b) {
105     ll x = 1;
106     a %= MOD;
107     while (b) {
108         if (b & 1) x = x * a % MOD;
109         a = a * a % MOD;
110         b >>= 1;
111     }
112     return x;
113 }
114
115 namespace linear_seq {
116     inline vector<int> BM(vector<int> x) {
117         vector<int> ls, cur;
118         int lf, ld;
119         for (int i = 0; i < int(x.size()); ++i) {
120             ll t = 0;
121             for (int j = 0; j < int(cur.size()); ++j)
122                 t = (t + x[i - j - 1] * (ll) cur[j]) %
123                 MOD;
124             if ((t - x[i]) % MOD == 0) continue;
125             if (!cur.size()) {
126                 cur.resize(i + 1);
127                 lf = i;
128                 ld = (t - x[i]) % MOD;
129                 continue;
130             }
131             ll k = -(x[i] - t) * qp(ld, MOD - 2) % MOD;
132             vector<int> c(i - lf - 1);
133             c.pb(k);
134             for (int j = 0; j < int(ls.size()); ++j)
135                 c.pb(-ls[j] * k % MOD);
136             if (c.size() < cur.size())
137                 c.resize(cur.size());
138             for (int j = 0; j < int(cur.size()); ++j)
139                 c[j] = (c[j] + cur[j]) % MOD;
140             if (i - lf + (int) ls.size() >= (int)
141                 cur.size())
142                 ls = cur, lf = i, ld = (t - x[i]) % MOD;
143             cur = c;
144         }
145         for (int i = 0; i < int(cur.size()); ++i)
146             cur[i] = (cur[i] % MOD + MOD) % MOD;
147         return cur;
148     }
149
150 int m;
151 ll a[SZ], h[SZ], t_[SZ], s[SZ], t[SZ];

```

```

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

```

```

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
86 Treap* treap_merge ( Treap *x, Treap *y ) {
87     if (!x)
88         return y;
89     if (!y)
90         return x;
91     if (x->height < y->height) {
92         x->push ();
93         x->right = treap_merge (x->right, y);
94         x->recalc ();
95         return x;
96     } else {
97         y->push ();
98         y->left = treap_merge (x, y->left);
99         y->recalc ();
100         return y;
101     }
102 }
103
104 Treap* treap_getmin ( Treap *x, int &source, int
105     ↪ &target, int &value ) {
106     assert (x);
107     x->push ();
108     if (x->min_path == 0) {
109         // memory leak, sorry
110         source = x->source;
111         target = x->target;
112         value = x->value + x->add;
113         return treap_merge (x->left, x->right);
114     } else if (x->min_path == -1) {
115         x->left = treap_getmin (x->left, source,
116             ↪ target, value);
117         value += x->add;
118         x->recalc ();
119         return x;
120     } else if (x->min_path == +1) {
121         x->right = treap_getmin (x->right, source,
122             ↪ target, value);
123         value += x->add;
124         x->recalc ();
125         return x;
126     } else
127         assert (0);
128 }
129
130 Treap* treap_add ( Treap *x, int add ) {
131     if (!x)
132         return 0;
133     return &((*x) += add);
134 }
135
136 int main () {
137     int n, m;
138     while (scanf ("%d%d", &n, &m) == 2) {
139         Treap *g[n + 1];
140         for (int i = 0; i <= n; i++)
141             g[i] = 0;
142         for (int i = 1; i <= n; i++) {
143             int a;
144             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 u = i;
201 while (dsu[u].first) {
202     int next = jump[dsu[u].first];
203     finish.push_back (dsu[u].first);
204     dsu.merge (u, 0, make_pair (0, (Treap *)0));
205     u = next;
206 }
207 bool ok[k];
208 int res[n];
209 memset (ok, 0, sizeof (ok[0]) * k);
210 memset (res, -1, sizeof (res[0]) * n);
211 function <void (int, int)> add_edge = [&ok,
212     ↪ &parent, &res, &n] ( int a, int b ) {
213     assert (0 <= a && a < n);
214     assert (0 <= b && b < n);
215     assert (res[a] == -1);
216     res[a] = b;
217     while (a != -1 && !ok[a]) {
218         ok[a] = true;
219         a = parent[a];
220     }
221 };
222 function <void (int)> reach = [&ok, &reach,
223     ↪ &children, &jump, &jump_from, &add_edge](
224     ↪ int u ) {
225     if (!ok[u])
226         add_edge (jump_from[u], jump[u]);
227     for (auto x : children[u])
228         reach (x);
229 };
230 for (auto x : finish)
231     reach (x);
232 printf ("%d\n", ans);
233 for (int i = 1; i < n; i++)
234     printf ("%d%c", res[i] ? res[i] : -1, "\n "[i
235         ↪ < n - 1]);
236 return 0;
237 }
238 // 2-chinese ends
239 // general max weight match begins
240 #define DIST(e)
241     ↪ (lab[e.u]+lab[e.v]-g[e.u][e.v].w*2)
242 using namespace std;
243 typedef long long ll;
244 const int N = 1023, INF = 1e9;
245 struct Edge {
246     int u, v, w;
247 } g[N][N];
248 int n, m, n_x, lab[N], match[N], slack[N], st[N],
249     ↪ pa[N], flower_from[N][N], S[N], vis[N];
250 vector < int> flower[N];
251 deque < int> q;
252 void update_slack(int u, int x) {
253     if (!slack[x] || DIST(g[u][x]) <
254         ↪ DIST(g[slack[x]][x])) slack[x] = u;
255 }
256 void set_slack(int x) {
257     slack[x] = 0;

```

```

18     for (int u = 1; u <= n; ++u)
19         if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0) update_slack(u, x);
20     }
21     void q_push(int x) {
22         if (x <= n) return q.push_back(x);
23         for (int i = 0; i < flower[x].size(); i++)
24             q.push(flower[x][i]);
25     }
26     void set_st(int x, int b) {
27         st[x] = b;
28         if (x <= n) return;
29         for (int i = 0; i < flower[x].size(); ++i)
30             set_st(flower[x][i], b);
31     }
32     int get_pr(int b, int xr) {
33         int pr = find(flower[b].begin(),
34             flower[b].end(), xr) - flower[b].begin();
35         if (pr % 2 == 1) {
36             reverse(flower[b].begin() + 1,
37                 flower[b].end());
38             return (int) flower[b].size() - pr;
39         }
40         else return pr;
41     }
42     void set_match(int u, int v) {
43         match[u] = g[u][v].v;
44         if (u <= n) return;
45         Edge e = g[u][v];
46         int xr = flower_from[u][e.u], pr = get_pr(u,
47             xr);
48         for (int i = 0; i < pr; ++i)
49             set_match(flower[u][i], flower[u][i+1]);
50         set_match(xr, v);
51         rotate(flower[u].begin(), flower[u].begin()
52             + pr, flower[u].end());
53     }
54     void augment(int u, int v) {
55         int xnv = st[match[u]];
56         set_match(u, v);
57         if (!xnv) return;
58         set_match(xnv, st[pa[xnv]]);
59         augment(st[pa[xnv]], xnv);
60     }
61     int get_lca(int u, int v) {
62         static int t = 0;
63         for (++t; u || v; swap(u, v)) {
64             if (u == 0) continue;
65             if (vis[u] == t) return u;
66             vis[u] = t;
67             u = st[match[u]];
68             if (u) u = st[pa[u]];
69         }
70         return 0;
71     }
72     void add_blossom(int u, int lca, int v) {
73         int b = n+1;
74         while (b <= n_x && st[b]) ++b;
75         if (b > n_x) ++n_x;
76         lab[b] = 0, S[b] = 0;
77         match[b] = match[lca];
78         flower[b].clear();
79         flower[b].push_back(lca);
80         for (int x = u, y; x != lca; x = st[pa[y]])
81             flower[b].push_back(x), flower[b].push_back(y)
82             y = st[match[x]];
83         q.push(y);
84         reverse(flower[b].begin() + 1, flower[b].end());
85         for (int x = v, y; x != lca; x = st[pa[y]])
86             flower[b].push_back(x), flower[b].push_back(y)
87             y = st[match[x]];
88         q.push(y);
89         set_st(b, b);
90         for (int x = 1; x <= n_x; ++x) g[b][x].w =
91             g[x][b].w = 0;
92         for (int x = 1; x <= n; ++x) flower_from[b][x]
93             = 0;
94         for (int i = 0; i < flower[b].size(); ++i) {
95             int xs = flower[b][i];
96             for (int x = 1; x <= n_x; ++x)
97                 if (g[b][x].w == 0 || DIST(g[xs][x]) <
98                     DIST(g[b][x]))
99                     g[b][x] = g[xs][x], g[x][b] = g[x][xs];
100             for (int x = 1; x <= n; ++x)
101                 if (flower_from[xs][x]) flower_from[b][x] =
102                     xs;
103         }
104         set_slack(b);
105     }
106     void expand_blossom(int b) // S[b] == 1 {
107         for (int i = 0; i < flower[b].size(); ++i)
108             set_st(flower[b][i], flower[b][i]);
109         int xr = flower_from[b][g[b][pa[b]].u], pr =
110             get_pr(b, xr);
111         for (int i = 0; i < pr; i += 2) {
112             int xs = flower[b][i], xns = flower[b][i+1];
113             pa[xs] = g[xns][xs].u;
114             S[xs] = 1, S[xns] = 0;
115             slack[xs] = 0, set_slack(xns);
116             q.push(xns);
117         }
118         S[xr] = 1, pa[xr] = pa[b];
119         for (int i = pr+1; i < flower[b].size(); ++i) {
120             int xs = flower[b][i];
121             S[xs] = -1, set_slack(xs);
122         }
123         st[b] = 0;
124     }
125     bool on_found_Edge(const Edge &e) {
126         int u = st[e.u], v = st[e.v];
127         if (S[v] == -1) {
128             pa[v] = e.u, S[v] = 1;
129             int nu = st[match[v]];
130             slack[v] = slack[nu] = 0;
131             S[nu] = 0, q.push(nu);
132         }
133         else if (S[v] == 0) {
134             int lca = get_lca(u, v);
135             if (!lca) return augment(u, v), augment(v,
136                 u), 1;
137             else add_blossom(u, lca, v);
138         }
139         return 0;
140     }

```

```

124 bool matching() {
125     fill(S, S+n_x+1, -1), fill(slack, slack+n_x+1,
126         ↪ 0);
127     q.clear();
128     for (int x = 1; x <= n_x; ++x)
129         if (st[x] == x && !match[x]) pa[x] = 0, S[x]
130         ↪ = 0, q.push(x);
131     if (q.empty()) return 0;
132     for (;;) {
133         while (q.size()) {
134             int u = q.front();
135             q.pop_front();
136             if (S[st[u]] == 1) continue;
137             for (int v = 1; v <= n; ++v)
138                 if (g[u][v].w>0 && st[u] != st[v]) {
139                     if (DIST(g[u][v]) == 0) {
140                         if (on_found_Edge(g[u][v])) return 1;
141                     }
142                     else update_slack(u, st[v]);
143                 }
144             }
145             int d = INF;
146             for (int b = n+1; b <= n_x; ++b)
147                 if (st[b] == b && S[b] == 1) d = min(d,
148                     ↪ lab[b]/2);
149             for (int x = 1; x <= n_x; ++x)
150                 if (st[x] == x && slack[x]) {
151                     if (S[x] == -1) d = min(d,
152                         ↪ DIST(g[slack[x]][x]));
153                     else if (S[x] == 0) d = min(d,
154                         ↪ DIST(g[slack[x]][x])/2);
155                 }
156             for (int u = 1; u <= n; ++u) {
157                 if (S[st[u]] == 0) {
158                     if (lab[u] <= d) return 0;
159                     lab[u] -= d;
160                 }
161                 else if (S[st[u]] == 1) lab[u] += d;
162             }
163             for (int b = n+1; b <= n_x; ++b)
164                 if (st[b] == b) {
165                     if (S[st[b]] == 0) lab[b] += d*2;
166                     else if (S[st[b]] == 1) lab[b] -= d*2;
167                 }
168             q.clear();
169             for (int x = 1; x <= n_x; ++x)
170                 if (st[x] == x && slack[x] && st[slack[x]]
171                     ↪ != x && DIST(g[slack[x]][x]) == 0)
172                     if (on_found_Edge(g[slack[x]][x])) return
173                     ↪ 1;
174             for (int b = n+1; b <= n_x; ++b)
175                 if (st[b] == b && S[b] == 1 && lab[b] == 0)
176                     ↪ expand_blossom(b);
177             }
178         return 0;
179     }
180 }
181
182 pair < ll, int> weight_blossom() {
183     fill(match, match+n+1, 0);
184     n_x = n;
185     int n_matches = 0;
186     ll tot_weight = 0;
187
188     for (int u = 0; u <= n; ++u) st[u] = u,
189         ↪ flower[u].clear();
190     int w_max = 0;
191     for (int u = 1; u <= n; ++u)
192         for (int v = 1; v <= n; ++v) {
193             flower_from[u][v] = (u == v?u:0);
194             w_max = max(w_max, g[u][v].w);
195         }
196     for (int u = 1; u <= n; ++u) lab[u] = w_max;
197     while (matching()) ++n_matches;
198     for (int u = 1; u <= n; ++u)
199         if (match[u] && match[u] < u)
200             tot_weight += g[u][match[u]].w;
201     return make_pair(tot_weight, n_matches);
202 }
203
204 int main() {
205     cin>>n>>m;
206     for (int u = 1; u <= n; ++u)
207         for (int v = 1; v <= n; ++v)
208             g[u][v] = Edge {u, v, 0};
209     for (int i = 0, u, v, w; i < m; ++i) {
210         cin>>u>>v>>w;
211         g[u][v].w = g[v][u].w = w;
212     }
213     cout << weight_blossom().first << '\n';
214     for (int u = 1; u <= n; ++u) cout << match[u]
215         ↪ << ' ';
216 }
217
218 // general max weight match ends
219
220 // slow min circulation begins
221
222 struct Edge {
223     int a;
224     int b;
225     int cost;
226 };
227
228 vector<int> negative_cycle(int n, vector<Edge>
229     ↪ &edges) {
230     // O(nm), return ids of edges in negative cycle
231
232     vector<int> d(n);
233     vector<int> p(n, -1); // last edge ids
234
235     const int inf = 1e9;
236
237     int x = -1;
238     for (int i = 0; i < n; i++) {
239         x = -1;
240         for (int j = 0; j < edges.size(); j++) {
241             Edge &e = edges[j];
242
243             if (d[e.b] > d[e.a] + e.cost) {
244                 d[e.b] = max(-inf, d[e.a] + e.cost);
245                 p[e.b] = j;
246                 x = e.b;
247             }
248         }
249     }
250 }

```

```

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>
↳ &edges) {
48     const int inf = 1e3;
49
50     for (auto &e : edges)
51         e.cost *= n * n;
52
53     int l = -inf;
54     int r = inf;
55     while (l + 1 < r) {
56         int m = (l + r) / 2;
57         for (auto &e : edges)
58             e.cost -= m;
59
60         if (negative_cycle(n, edges).empty())
61             l = m;
62         else
63             r = m;
64
65         for (auto &e : edges)
66             e.cost += m;
67     }
68
69     if (r >= 0) // if only negative needed
70         return vector<int>();
71
72     for (auto &e : edges)
73         e.cost -= r;
74
75     vector<int> result = negative_cycle(n, edges);
76
77     for (auto &e : edges)
78         e.cost += r;
79
80     for (auto &e : edges)
81         e.cost /= n * n;
82
83     return result;
84 }
85
86 struct edge {
87     int from, to;
88     int c, f, cost;
89 };
90
91 const int max_n = 200;
92
93 vector<int> gr[max_n];
94 vector<edge> edges;
95
96 void add(int fr, int to, int c, int cost) {
97     gr[fr].push_back(edges.size());
98     edges.push_back({fr, to, c, 0, cost});
99     gr[to].push_back(edges.size());
100     edges.push_back({to, fr, 0, 0, -cost}); //
↳ single
101 }
102
103 void calc_min_circulation(int n) {
104     while (true) {
105         vector<Edge> eds;
106         vector<int> origin;
107
108         for (int i = 0; i < edges.size(); i++) {
109             edge &e = edges[i];
110             if (e.c - e.f > 0) {
111                 eds.push_back({e.from, e.to, e.cost});
112                 origin.push_back(i);
113             }
114         }
115
116         vector<int> cycle = negative_cycle(n, eds);
117
118         if (cycle.empty())
119             break;
120
121         for (auto id : cycle) {
122             int x = origin[id];
123             edges[x].f += 1;
124             edges[x ^ 1].f -= 1;
125         }
126     }
127 }
128
129 // slow min circulation ends
130
131 // fast hashtable begins
132
133 #include <ext/pb_ds/assoc_container.hpp>
134 using namespace __gnu_pbds;
135 gp_hash_table<int, int> table;
136
137 const int RANDOM = chrono ::
↳ high_resolution_clock ::
↳ now().time_since_epoch().count();
138 struct chash {
139     int operator()(int x) { return hash<int>{}(x ^
↳ RANDOM); }
140 };
141 gp_hash_table<key, int, chash> table;
142
143 // fast hashtable ends

```


DM

Кол-во корневых деревьев:

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

Кол-во эйлеровых циклов:

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

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.

```
xmodmap -e 'keycode 94='
setxkbmap us
```