

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



```

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

```

**pragma**

```

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

```

**Алгебра Pick**

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

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

**Newton**

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

**Catalan**

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

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

**Кол-во графов**

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

Количество связных помеченных графов

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

Количество помеченных графов с  $K$  компонентами связности

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

**Miller-Rabbin**

```

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

```

**Интегрирование по формуле Симпсона**

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

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

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

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

$$O(n^4).$$

**Простые числа**

```

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

```

**Числа для Фурье**

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

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

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

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

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

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

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

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

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

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

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

### Erdős–Gallai theorem

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

$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$  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 // stable gauss begins
104 // if at least one solution returns it in ans
105 int gauss (vector < vector<double> > a,
106   < vector<double> & ans) {
107   int n = (int) a.size();
108   int m = (int) a[0].size() - 1;
109
110   vector<int> where (m, -1);
111   for (int col=0, row=0; col<m && row<n; ++col) {
112     int sel = row;
113     for (int i=row; i<n; ++i)

```

```

11     if (abs (a[i][col]) > abs (a[sel][col])) 20
12         sel = i; 21     double d = sqrt(sqr(a) + sqr(b));
13     if (abs (a[sel][col]) < EPS) 22     a /= d, b /= d, c /= d;
14         continue; 23 }
15     for (int i=col; i<=m; ++i) 24 bool operator||(const Line& l) const { return
16         swap (a[sel][i], a[row][i]); 25     ↪ fabs(a * l.b - l.a * b) < EPS; }
17     where[col] = row; 26 double dist(const Point& p) const { return
18 27     ↪ fabs(a * p.x + b * p.y + c); }
19     for (int i=0; i<n; ++i) 28 Point operator^(const Line& l) const {
20         if (i != row) { 29     ↪ return {(l.c * b - c * l.b) / (a * l.b - l.a
21             double c = a[i][col] / a[row][col]; 30     ↪ * b),
22             for (int j=col; j<=m; ++j) 31     ↪ (l.c * a - c * l.a) / (l.a * b - a *
23                 a[i][j] -= a[row][j] * c; 32     ↪ l.b)};
24         } 33 }
25     ++row; 34 };
26 } 35 struct Circle {
27 36     Point c;
28 ans.assign (m, 0); 37     double r;
29     for (int i=0; i<m; ++i) 38     Circle(const Point& c, double r) : c(c), r(r)
30         if (where[i] != -1) 39     ↪ {}
31         ans[i] = a[where[i]][m] / a[where[i]][i]; 40     Circle(const Point& a, const Point& b, const
32     for (int i=0; i<n; ++i) { 41     ↪ Point& c) {
33         double sum = 0; 42         Point p1 = (a + b) * 0.5, p2 = (a + c) * 0.5;
34         for (int j=0; j<m; ++j) 43         Point q1 = p1 + (b - a).rotate(), q2 = p2 +
35             sum += ans[j] * a[i][j]; 44         ↪ (c - a).rotate();
36         if (abs (sum - a[i][m]) > EPS) 45         this->c = Line(p1, q1) ^ Line(p2, q2);
37             return 0; 46         r = (a - this->c).dist();
38     } 47     }
39 48 };
40     for (int i=0; i<m; ++i) 49 inline bool on_segment(const Point& p1, const
41         if (where[i] == -1) 50     ↪ Point& p2, const Point& x, bool strictly) {
42             return INF; 51     ↪ if (fabs((p1 - x) ^ (p2 - x)) > EPS)
43         return 1; 52         return false;
44     } 53     return (p1 - x) * (p2 - x) < (strictly ? - EPS
45 54     ↪ : EPS);
46 // stable gauss ends 55 }
47 // simple geometry begins 56 // in case intersection is not a segment
48 57 inline bool intersect_segments(const Point& p1,
49 58     ↪ const Point& p2, const Point& q1, const
50 59     ↪ Point& q2, Point& x) {
51 60     Line l1(p1, p2), l2(q1, q2);
52 61     if (l1 || l2) return false;
53 62     x = l1 ^ l2;
54 63     return on_segment(p1, p2, x, false);
55 64 }
56 65 // in case circles are not equal
57 66 inline bool intersect_circles(const Circle& c1,
58 67     ↪ const Circle& c2, Point& p1, Point& p2) {
59 68     double d = (c2.c - c1.c).dist();
60 69     if (d > c1.r + c2.r + EPS || d < fabs(c1.r -
61 70     ↪ c2.r) - EPS)
62 71         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 +
    ↪ * 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
1 // convex hull begins
2
3 struct Point {
4     int x, y;
5     Point operator-(const Point& p) const { return
    ↪ {x - p.x, y - p.y}; }
6     int64_t operator^(const Point& p) const {
    ↪ return x * 1ll * p.y - y * 1ll * p.x; }
7     int64_t dist() const { return x * 1ll * x + y *
    ↪ 1ll * y; }
8     bool operator<(const Point& p) const { return x
    ↪ != p.x ? x < p.x : y < p.y; }
9 };
10
11 // all point on convex hull are included
12 vector<Point> convex_hull(vector<Point> pt) {
13     int n = pt.size();
14     Point p0 = *std::min_element(pt.begin(),
    ↪ pt.end());
15     std::sort(pt.begin(), pt.end(), [&p0](const
    ↪ Point& a, const Point& b) {
16         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
30             int64_t cp = (p1 - p2) ^ (p - p1);
31             if (cp >= 0) break;
32             ch.pop_back();
33         }
34         ch.push_back(p);
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);

```



```

48     else
49         add_line(v + v + 2, m, r, a, b, nw);
50 }
51
52 ftype get(int v, int l, int r, int x) {
53     if(l == r - 1)
54         return f(line[v], x);
55     int m = (l + r) / 2;
56     if(x < m) {
57         return min(f(line[v], x), get(v + v + 1, l,
58             ↪ m, x));
59     } else {
60         return min(f(line[v], x), get(v + v + 2, m,
61             ↪ r, x));
62     }
63 } cdt(maxn);
64
65 // convex hull with stack
66
67 ftype cross(point a, point b) {
68     return (conj(a) * b).y();
69 }
70
71 vector<point> hull, vecs;
72
73 void add_line(ftype k, ftype b) {
74     point nw = {k, b};
75     while(!vecs.empty() && dot(vecs.back(), nw -
76         ↪ hull.back()) < 0) {
77         hull.pop_back();
78         vecs.pop_back();
79     }
80     if(!hull.empty()) {
81         vecs.push_back(1i * (nw - hull.back()));
82     }
83     hull.push_back(nw);
84 }
85
86 int get(ftype x) {
87     point query = {x, 1};
88     auto it = lower_bound(vecs.begin(), vecs.end(),
89         ↪ query, [](point a, point b) {
90         return cross(a, b) > 0;
91     });
92     return dot(query, hull[it - vecs.begin()]);
93 }
94
95 // convex hull trick ends
96
97 // heavy-light begins
98
99 int sz[maxn];
100
101 void dfs_sz(int v, int par = -1) {
102     sz[v] = 1;
103     for (int x : gr[v])
104         if (x != par) {
105             dfs_sz(x, v);
106             sz[v] += sz[x];
107         }
108
109     for (int i = 0; i < gr[v].size(); i++)
110         if (gr[v][i] != par)
111             if (sz[gr[v][i]] * 2 >= sz[v]) {
112                 swap(gr[v][i], gr[v][0]);
113                 break;
114             }
115 }
116
117 int rev[maxn];
118 int t_in[maxn];
119 int upper[maxn];
120 int par[maxn];
121 int dep[maxn];
122
123 int T = 0;
124
125 void dfs_build(int v, int uppr, int pr = -1) {
126     rev[T] = v;
127     t_in[v] = T++;
128     dep[v] = pr == -1 ? 0 : dep[pr] + 1;
129     par[v] = pr;
130     upper[v] = uppr;
131
132     bool first = true;
133
134     for (int x : gr[v])
135         if (x != pr) {
136             dfs_build(x, first ? upper[v] : x, v);
137             first = false;
138         }
139 }
140
141 struct interval {
142     int l;
143     int r;
144     bool inv; // should direction be reversed
145 };
146
147 // node-weighted hld
148 vector<interval> get_path(int a, int b) {
149     vector<interval> front;
150     vector<interval> back;
151
152     while (upper[a] != upper[b]) {
153         if (dep[upper[a]] > dep[upper[b]]) {
154             front.push_back({t_in[upper[a]], t_in[a],
155                 ↪ true});
156             a = par[upper[a]];
157         } else {
158             back.push_back({t_in[upper[b]], t_in[b],
159                 ↪ false});
160             b = par[upper[b]];
161         }
162     }
163
164     front.push_back({min(t_in[a], t_in[b]),
165         ↪ max(t_in[a], t_in[b]), t_in[a] > t_in[b]});
166     // for edge-weighted hld add:
167     ↪ "front.back().l++;";
168     front.insert(front.end(), back.rbegin(),
169         ↪ back.rend());

```



```

68     return front;
69 }
70
71 // heavy-light ends
72
73 // max flow begins
74
75 struct edge{
76     int from, to;
77     int c, f, num;
78     edge(int from, int to, int c, int
79         ↪ num):from(from), to(to), c(c), f(0),
80         ↪ num(num){}
81     edge(){}
82 };
83
84 const int max_n = 600;
85
86 edge eds[150000];
87 int num = 0;
88 int it[max_n];
89 vector<int> gr[max_n];
90 int s, t;
91 vector<int> d(max_n);
92
93 bool bfs(int k) {
94     queue<int> q;
95     q.push(s);
96     fill(d.begin(), d.end(), -1);
97     d[s] = 0;
98     while (!q.empty()) {
99         int v = q.front();
100         q.pop();
101         for (int x : gr[v]) {
102             int to = eds[x].to;
103             if (d[to] == -1 && eds[x].c - eds[x].f >=
104                 ↪ (1 << k)){
105                 d[to] = d[v] + 1;
106                 q.push(to);
107             }
108         }
109     }
110     return (d[t] != -1);
111 }
112
113 int dfs(int v, int flow, int k) {
114     if (flow < (1 << k))
115         return 0;
116     if (v == t)
117         return flow;
118     for (; it[v] < gr[v].size(); it[v]++) {
119         int num = gr[v][it[v]];
120         if (d[v] + 1 != d[num].to)
121             continue;
122         int res = dfs(eds[num].to, min(flow,
123             ↪ eds[num].c - eds[num].f), k);
124         if (res){
125             eds[num].f += res;
126             eds[num ^ 1].f -= res;
127             return res;
128         }
129     }
130     return 0;
131 }
132
133 void add(int fr, int to, int c, int nm) {
134     gr[fr].push_back(num);
135     eds[num++] = edge(fr, to, c, nm);
136     gr[to].push_back(num);
137     eds[num++] = edge(to, fr, 0, nm); //corrected c
138 }
139
140 int ans = 0;
141 for (int k = 30; k >= 0; k--)
142     while (bfs(k)) {
143         memset(it, 0, sizeof(it));
144         while (int res = dfs(s, 1e9 + 500, k))
145             ans += res;
146     }
147
148 // decomposition
149
150 int path_num = 0;
151 vector<int> paths[max_n];
152 int flows[max_n];
153
154 int decomp(int v, int flow) {
155     if (flow < 1)
156         return 0;
157     if (v == t) {
158         path_num++;
159         flows[path_num - 1] = flow;
160         return flow;
161     }
162     for (int i = 0; i < gr[v].size(); i++) {
163         int num = gr[v][i];
164         int res = decomp(eds[num].to, min(flow,
165             ↪ eds[num].f));
166         if (res) {
167             eds[num].f -= res;
168             paths[path_num -
169                 ↪ 1].push_back(eds[num].num);
170             return res;
171         }
172     }
173     return 0;
174 }
175
176 while (decomp(s, 1e9 + 5));
177
178 // max flow ends
179
180 // min-cost flow begins
181
182 long long ans = 0;
183 int mx = 2 * n + 2;
184
185 memset(upd, 0, sizeof(upd));
186 for (int i = 0; i < mx; i++)
187     dist[i] = inf;
188 dist[st] = 0;

```

```

10 queue<int> q;
11 q.push(st);
12 upd[st] = 1;
13 while (!q.empty()){
14     int v = q.front();
15     q.pop();
16     if (upd[v]){
17         for (int x : gr[v]) {
18             edge &e = edges[x];
19             if (e.c - e.f > 0 && dist[v] != inf &&
20                 ↪ dist[e.to] > dist[v] + e.w) {
21                 dist[e.to] = dist[v] + e.w;
22                 if (!upd[e.to])
23                     q.push(e.to);
24                 upd[e.to] = true;
25                 p[e.to] = x;
26             }
27         }
28         upd[v] = false;
29     }
30
31     for (int i = 0; i < k; i++){
32         for (int i = 0; i < mx; i++){
33             d[i] = inf;
34             d[st] = 0;
35             memset(used, false, sizeof(used));
36             set<pair<int, int> > s;
37             s.insert(make_pair(0, st));
38             for (int i = 0; i < mx; i++){
39                 int x;
40                 while (!s.empty() && used[(s.begin() ->
41                     ↪ second)]){
42                     s.erase(s.begin());
43                 }
44                 if (s.empty())
45                     break;
46                 x = s.begin() -> second;
47                 used[x] = true;
48                 s.erase(s.begin());
49                 for (int i = 0; i < gr[x].size(); i++){
50                     edge &e = edges[gr[x][i]];
51                     if (!used[e.to] && e.c - e.f > 0){
52                         if (d[e.to] > d[x] + (e.c - e.f) * e.w
53                             ↪ dist[x] - dist[e.to]){
54                             d[e.to] = d[x] + (e.c - e.f) * e.w +
55                                 ↪ dist[x] - dist[e.to];
56                             p[e.to] = gr[x][i];
57                             s.insert(make_pair(d[e.to], e.to));
58                         }
59                     }
60                 }
61                 dist[x] += d[x];
62             }
63             int pos = t;
64             while (pos != st){
65                 int id = p[pos];
66                 edges[id].f += 1;
67                 edges[id ^ 1].f -= 1;
68                 pos = edges[id].from;
69             }
70         }
71     }
72 }
73
74 // min-cost flow ends
75
76 // bad hungarian begins
77
78 fill(par, par + 301, -1);
79 fill(par2, par2 + 301, -1);
80
81 int ans = 0;
82 for (int v = 0; v < n; v++){
83     memset(useda, false, sizeof(useda));
84     memset(usedb, false, sizeof(usedb));
85     useda[v] = true;
86     for (int i = 0; i < n; i++){
87         w[i] = make_pair(a[v][i] + row[v] + col[i],
88             ↪ v);
89     }
90     memset(prev, 0, sizeof(prev));
91     int pos;
92     while (true){
93         pair<pair<int, int>, int> p =
94             ↪ make_pair(make_pair(1e9, 1e9), 1e9);
95         for (int i = 0; i < n; i++){
96             if (!usedb[i])
97                 p = min(p, make_pair(w[i], i));
98         }
99         for (int i = 0; i < n; i++){
100             if (!useda[i])
101                 row[i] += p.first.first;
102         }
103         for (int i = 0; i < n; i++){
104             if (!usedb[i]){
105                 col[i] -= p.first.first;
106                 w[i].first -= p.first.first;
107             }
108         }
109         ans += p.first.first;
110         usedb[p.second] = true;
111         prev[p.second] = p.first.second; //из второй
112             ↪ в первую
113         int x = par[p.second];
114         if (x == -1){
115             pos = p.second;
116             break;
117         }
118         useda[x] = true;
119         for (int j = 0; j < n; j++){
120             w[j] = min(w[j], {a[x][j] + row[x] +
121                 ↪ col[j], x});
122         }
123     }
124     while (pos != -1){
125         int nxt = par2[prev[pos]];
126         par[pos] = prev[pos];
127         par2[prev[pos]] = pos;
128         pos = nxt;
129     }
130 }
131
132 cout << ans << "\n";
133 for (int i = 0; i < n; i++)
134     cout << par[i] + 1 << " " << i + 1 << "\n";
135
136 // bad hungarian ends
137
138 // Edmonds O(n^3) begins

```

```

2
3 vector<int> gr[MAXN];
4 int match[MAXN], p[MAXN], base[MAXN], q[MAXN];
5 bool used[MAXN], blossom[MAXN];
6 int mark[MAXN];
7 int C = 1;
8
9 int lca(int a, int b) {
10     C++;
11     for (;;) {
12         a = base[a];
13         mark[a] = C;
14         if (match[a] == -1) break;
15         a = p[match[a]];
16     }
17
18     for (;;) {
19         b = base[b];
20         if (mark[b] == C) return b;
21         b = p[match[b]];
22     }
23 }
24
25 void mark_path(int v, int b, int children) {
26     while (base[v] != b) {
27         blossom[base[v]] = blossom[base[match[v]]]
28         ↳ true;
29         p[v] = children;
30         children = match[v];
31         v = p[match[v]];
32     }
33 }
34
35 int find_path(int root) {
36     memset(used, 0, sizeof(used));
37     memset(p, -1, sizeof p);
38     for (int i = 0; i < N; i++)
39         base[i] = i;
40
41     used[root] = true;
42     int qh = 0, qt = 0;
43     q[qt++] = root;
44     while (qh < qt) {
45         int v = q[qh++];
46         for (int to : gr[v]) {
47             if (base[v] == base[to] || match[v] == to)
48             ↳ continue;
49             if (to == root || match[to] != -1 &&
50             ↳ p[match[to]] != -1) {
51                 int curbase = lca(v, to);
52                 memset(blossom, 0, sizeof(blossom));
53                 mark_path(v, curbase, to);
54                 mark_path(to, curbase, v);
55                 for (int i = 0; i < N; i++)
56                     if (blossom[base[i]]) {
57                         base[i] = curbase;
58                         if (!used[i]) {
59                             used[i] = true;
60                             q[qt++] = i;
61                         }
62                     }
63             }
64         }
65     }
66
67     } else if (p[to] == -1) {
68         p[to] = v;
69         if (match[to] == -1)
70             return to;
71         to = match[to];
72         used[to] = true;
73         q[qt++] = to;
74     }
75 }
76
77 return -1;
78 }
79
80 memset(match, -1, sizeof match);
81 for (int i = 0; i < N; i++) {
82     if (match[i] == -1 && !gr[i].empty()) {
83         int v = find_path(i);
84         while (v != -1) {
85             int pv = p[v], ppv = match[pv];
86             match[v] = pv; match[pv] = v;
87             v = ppv;
88         }
89     }
90 }
91
92 // Edmonds O(n^3) ends
93
94 // string basis begins
95
96 vector<int> getZ(string s){
97     vector<int> z;
98     z.resize(s.size(), 0);
99     int l = 0, r = 0;
100     for (int i = 1; i < s.size(); i++){
101         if (i <= r)
102             z[i] = min(r - i + 1, z[i - 1]);
103         while (i + z[i] < s.size() && s[z[i]] == s[i
104         ↳ + z[i]])
105             z[i]++;
106         if (i + z[i] - 1 > r){
107             r = i + z[i] - 1;
108             l = i;
109         }
110     }
111     return z;
112 }
113
114 vector<int> getP(string s){
115     vector<int> p;
116     p.resize(s.size(), 0);
117     int k = 0;
118     for (int i = 1; i < s.size(); i++){
119         while (k > 0 && s[i] == s[k])
120             k = p[k - 1];
121         if (s[i] == s[k])
122             k++;
123         p[i] = k;
124     }
125     return p;
126 }

```

```

34 vector<int> getH(string s){
35     vector<int> h;
36     h.resize(s.size() + 1, 0);
37     for (int i = 0; i < s.size(); i++)
38         h[i + 1] = ((h[i] * 111 * pow) + s[i] - 'a'
39             ↪ 1) % mod;
40     return h;
41 }
42
43 int getHash(vector<int> &h, int l, int r){
44     int res = (h[r + 1] - h[l] * p[r - l + 1]) %
45         ↪ mod;
46     if (res < 0)
47         res += mod;
48     return res;
49 }
50
51 // string basis ends
52
53 // min cyclic shift begins
54
55 string min_cyclic_shift (string s) {
56     s += s;
57     int n = (int)s.length();
58     int i=0, ans=0;
59     while (i < n/2) {
60         ans = i;
61         int j=i+1, k=i;
62         while (j < n && s[k] <= s[j]) {
63             if (s[k] < s[j])
64                 k = i;
65             else
66                 ++k;
67             ++j;
68         }
69         while (i <= k) i += j - k;
70     }
71     return s.substr (ans, n/2);
72 }
73
74 // min cyclic shift ends
75
76 // suffix array O(n) begins
77
78 typedef vector<char> bits;
79
80 template<const int end>
81 void getBuckets(int *s, int *bkt, int n, int K) {
82     fill(bkt, bkt + K + 1, 0);
83     forn(i, n) bkt[s[i] + !end]++;
84     forn(i, K) bkt[i + 1] += bkt[i];
85 }
86
87 void induceSAL(bits &t, int *SA, int *s, int
88     ↪ *bkt, int n, int K) {
89     getBuckets<0>(s, bkt, n, K);
90     forn(i, n) {
91         int j = SA[i] - 1;
92         if (j >= 0 && !t[j])
93             SA[bkt[s[j]]++] = j;
94     }
95 }
96
97 void induceSAs(bits &t, int *SA, int *s, int
98     ↪ *bkt, int n, int K) {
99     getBuckets<1>(s, bkt, n, K);
100    for (int i = n - 1; i >= 0; i--) {
101        int j = SA[i] - 1;
102        if (j >= 0 && t[j])
103            SA[--bkt[s[j]]] = j;
104    }
105 }
106
107 void SA-IS(int *s, int *SA, int n, int K) { //
108     ↪ require last symbol is 0
109     #define isLMS(i) (i && t[i] && !t[i-1])
110     int i, j;
111     bits t(n);
112     t[n-1] = 1;
113     for (i = n - 3; i >= 0; i--)
114         t[i] = (s[i]<s[i+1] || (s[i]==s[i+1] &&
115             ↪ t[i+1]==1));
116     int bkt[K + 1];
117     getBuckets<1>(s, bkt, n, K);
118     fill(SA, SA + n, -1);
119     forn(i, n)
120         if (isLMS(i))
121             SA[--bkt[s[i]]] = i;
122     induceSAL(t, SA, s, bkt, n, K);
123     induceSAs(t, SA, s, bkt, n, K);
124     int n1 = 0;
125     forn(i, n)
126         if (isLMS(SA[i]))
127             SA[n1++] = SA[i];
128     fill(SA + n1, SA + n, -1);
129     int name = 0, prev = -1;
130     forn(i, n1) {
131         int pos = SA[i];
132         bool diff = false;
133         for (int d = 0; d < n; d++)
134             if (prev == -1 || s[pos+d] != s[prev+d] ||
135                 ↪ t[pos+d] != t[prev+d])
136                 diff = true, d = n;
137             else if (d > 0 && (isLMS(pos+d) ||
138                 ↪ isLMS(prev+d)))
139                 d = n;
140         if (diff)
141             name++, prev = pos;
142         SA[n1 + (pos >> 1)] = name - 1;
143     }
144     for (i = n - 1, j = n - 1; i >= n1; i--)
145         if (SA[i] >= 0)
146             SA[j--] = SA[i];
147     int *s1 = SA + n - n1;
148     if (name < n1)
149         SA-IS(s1, SA, n1, name - 1);
150     else
151         forn(i, n1)
152             SA[s1[i]] = i;
153     getBuckets<1>(s, bkt, n, K);
154     for (i = 1, j = 0; i < n; i++)
155         if (isLMS(i))
156             s1[j++] = i;
157     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
92 ↪ int l) { return (y [a] == y [b] && y [a + l]
93 ↪ == y [b + l]); }
94
95 void Sort () {
96     for (int i = 0; i < m; ++i) w[i] = 0;
97     for (int i = 0; i < N; ++i) ++w[x[y[i]]];
98     for (int i = 0; i < m - 1; ++i) w[i + 1] +=
99 ↪ w[i];
100    for (int i = N - 1; i >= 0; --i)
101 ↪ SA[--w[x[y[i]]]] = y[i];
102 }
103
104 void DA () {
105     for (int i = 0; i < N; ++i) x[i] = str[i], y[i]
106 ↪ = i;
107     Sort ();
108     for (int i, j = 1, p = 1; p < N; j <= 1, m =
109 ↪ p) {
110         for (p = 0, i = N - j; i < N; i++) y[p++] =
111 ↪ i;
112         for (int k = 0; k < N; ++k) if (SA[k] >= j)
113 ↪ y[p++] = SA[k] - j;
114         Sort();
115         for (swap (x, y), p = 1, x[SA[0]] = 0, i = 1;
116 ↪ i < N; ++i) x[SA [i]] = cmp (SA[i - 1],
117 ↪ SA[i], j) ? p - 1 : p++;
118     }
119 }
120
121 // common for all algorithms
122 void kasaiLCP () {
123     for (int i = 0; i < N; i++) c[SA[i]] = i;
124     for (int i = 0, j, k = 0; i < N; LCP [c[i++]]
125 ↪ k)
126         if (c [i] > 0) for (k ? k-- : 0, j = SA[c[i]]
127 ↪ - 1; str[i + k] == str[j + k]; k++);
128         else k = 0;
129 }
130
131 void suffixArray () { // require last symbol is
132 ↪ char(0)
133     m = 256;
134     N = str.size();
135     DA ();
136
137     kasaiLCP ();
138 }
139 // suffix array O(n log n) ends
140
141 // bad suffix automaton begins
142
143 struct node{
144     map<char, int> go;
145     int len, suff;
146     long long sum_in;
147     node(){ }
148 };
149
150 node v[max_n * 4];
151
152 int add_node(int max_len){
153     //v[number].sum_in = 0;
154     v[number].len = max_len;
155     v[number].suff = -1;
156     number++;
157     return number - 1;
158 }
159
160 int last = add_node(0);
161
162 void add_char(char c) {
163     int cur = last;
164     int new_node = add_node(v[cur].len + 1);
165     last = new_node;
166     while (cur != -1){
167         if (v[cur].go.count(c) == 0){
168             v[cur].go[c] = new_node;
169             //v[new_node].sum_in += v[cur].sum_in;
170             cur = v[cur].suff;
171             if (cur == -1)
172                 v[new_node].suff = 0;
173         }else{
174             int a = v[cur].go[c];
175             if (v[a].len == v[cur].len + 1){
176                 v[new_node].suff = a;
177             }else{
178                 int b = add_node(v[cur].len + 1);
179                 v[b].go = v[a].go;
180                 v[b].suff = v[a].suff;
181                 v[new_node].suff = b;
182                 while (cur != -1 && v[cur].go.count(c) !=
183 ↪ 0 && v[cur].go[c] == a){
184                     v[cur].go[c] = b;
185                     //v[a].sum_in -= v[cur].sum_in;
186                     //v[b].sum_in += v[cur].sum_in;
187                     cur = v[cur].suff;
188                 }
189                 v[a].suff = b;
190             }
191             return;
192         }
193     }
194 }
195 // bad suffix automaton ends
196
197 // aho-corasick begins

```

```

2
3 struct vertex {
4     int next[K];
5     bool leaf;
6     int p;
7     char pch;
8     int link;
9     int go[K];
10 };
11
12 vertex t[NMAX+1];
13 int sz;
14
15 void init() {
16     t[0].p = t[0].link = -1;
17     memset(t[0].next, 255, sizeof t[0].next);
18     memset(t[0].go, 255, sizeof t[0].go);
19     sz = 1;
20 }
21
22 void add_string(const string & s) {
23     int v = 0;
24     for (size_t i=0; i<s.length(); ++i) {
25         char c = s[i]-'a';
26         if (t[v].next[c] == -1) {
27             memset(t[sz].next, 255, sizeof t[sz].next);
28             memset(t[sz].go, 255, sizeof t[sz].go);
29             t[sz].link = -1;
30             t[sz].p = v;
31             t[sz].pch = c;
32             t[v].next[c] = sz++;
33         }
34         v = t[v].next[c];
35     }
36     t[v].leaf = true;
37 }
38
39 int go(int v, char c);
40
41 int get_link(int v) {
42     if (t[v].link == -1)
43         if (v == 0 || t[v].p == 0)
44             t[v].link = 0;
45         else
46             t[v].link = go(get_link(t[v].p), t[v].pch);
47     return t[v].link;
48 }
49
50 int go(int v, char c) {
51     if (t[v].go[c] == -1)
52         if (t[v].next[c] != -1)
53             t[v].go[c] = t[v].next[c];
54         else
55             t[v].go[c] = v == 0 ? 0 : go(get_link(v),
56                                     ↪ c);
57     return t[v].go[c];
58 }
59 // aho-corasick ends
60
61 // pollard begins
62
63 const int max_step = 4e5;
64
65 unsigned long long gcd(unsigned long long a,
66 ↪ unsigned long long b){
67     if (!a) return 1;
68     while (a) swap(a, b%=a);
69     return b;
70 }
71
72 unsigned long long get(unsigned long long a,
73 ↪ unsigned long long b){
74     if (a > b)
75         return a-b;
76     else
77         return b-a;
78 }
79
80 unsigned long long pollard(unsigned long long n){
81     unsigned long long x = (rand() + 1) % n, y = 1,
82 ↪ g;
83     int stage = 2, i = 0;
84     g = gcd(get(x, y), n);
85     while (g == 1) {
86         if (i == max_step)
87             break;
88         if (i == stage) {
89             y = x;
90             stage <= 1;
91         }
92         x = (x * (__int128)x + 1) % n;
93         i++;
94         g = gcd(get(x, y), n);
95     }
96     return g;
97 }
98
99 // pollard ends
100
101 // linear sieve begins
102
103 const int N = 1000000;
104
105 int pr[N + 1], sz = 0;
106 /* minimal prime, mobius function, euler function
107 ↪ */
108 int lp[N + 1], mu[N + 1], phi[N + 1];
109
110 lp[1] = mu[1] = phi[1] = 1;
111 for (int i = 2; i <= N; ++i) {
112     if (lp[i] == 0)
113         lp[i] = pr[sz++] = i;
114     for (int j = 0; j < sz && pr[j] <= lp[i] && i *
115 ↪ pr[j] <= N; ++j)
116         lp[i * pr[j]] = pr[j];
117
118     mu[i] = lp[i] == lp[i / lp[i]] ? 0 : -1 * mu[i]
119 ↪ / lp[i];
120     phi[i] = phi[i / lp[i]] * (lp[i] == lp[i /
121 ↪ lp[i]] ? lp[i] : lp[i] - 1);
122 }
123
124 // linear sieve ends

```

```

1  // discrete log in sqrt(p) begins
2
3  int k = sqrt((double)p) + 2;
4
5  for (int i = k; i >= 1; i--){
6      mp[bin(b, (i * 1ll * k) % (p-1), p)] = i;
7
8      bool answered = false;
9      int ans = INT32_MAX;
10     for (int i = 0; i <= k; i++){
11         int sum = (n * 1ll * bin(b, i, p)) % p;
12         if (mp.count(sum) != 0){
13             int an = mp[sum] * 1ll * k - i;
14             if (an < p)
15                 ans = min(an, ans);
16         }
17     }
18
19     // discrete log in sqrt(p) ends
20
21     // prime roots mod n begins
22
23     int num = 0;
24     long long phi = n, nn = n;
25     for (long long x:primes){
26         if (x*x>nn)
27             break;
28         if (nn % x == 0){
29             while (nn % x == 0)
30                 nn /= x;
31             phi -= phi/x;
32             num++;
33         }
34     }
35     if (nn != 1){
36         phi -= phi/nn;
37         num++;
38     }
39     if (!(num == 1 && n % 2 != 0) || n == 4 || n == 23
40         ↪ 2 || (num == 2 && n % 2 == 0 && n % 4 != 0))
41         ↪ {
42         cout << "-1\n";
43         continue;
44     }
45     vector<long long> v;
46     long long pp = phi;
47     for (long long x:primes){
48         if (x*x>pp)
49             break;
50         if (pp % x == 0){
51             while (pp % x == 0)
52                 pp /= x;
53             v.push_back(x);
54         }
55     }
56     if (pp != 1){
57         v.push_back(pp);
58     }
59     while (true){
60         long long a = primes[rand()%5000]%n;
61         if (gcd(a, n) != 1)
62             continue;
63
64         bool bb = false;
65         for (long long x:v)
66             if (pow(a, phi/x) == 1){
67                 bb = true;
68                 break;
69             }
70         if (!bb){
71             cout << a << "\n";
72             break;
73         }
74     }
75
76     // prime roots mod n ends
77
78     // simplex begins
79
80     const double EPS = 1e-9;
81
82     typedef vector<double> vdbl;
83
84     // n variables, m inequalities
85     // Ax <= b, c*x -> max, x >= 0
86     double simplex( int n, int m, const vector<vdbl>
87         ↪ &a0, const vdbl &b, const vdbl &c, vdbl &x )
88         ↪ {
89         // Ax + Ez = b, A[m]*x -> max
90         // x = 0, z = b, x >= 0, z >= 0
91         vector<vdbl> a(m + 2, vdbl(n + m + 2));
92         vector<int> p(m);
93         forn(i, n)
94             a[m + 1][i] = c[i];
95         forn(i, m) {
96             forn(j, n)
97                 a[i][j] = a0[i][j];
98             a[i][n + i] = 1;
99             a[i][m + n] = -1;
100            a[i][m + n + 1] = b[i];
101            p[i] = n + i;
102        }
103
104        // basis: enter "j", leave "ind+n"
105        auto pivot = [&]( int j, int ind ) {
106            double coef = a[ind][j];
107            assert(fabs(coef) > EPS);
108            forn(col, n + m + 2)
109                a[ind][col] /= coef;
110            forn(row, m + 2)
111                if (row != ind && fabs(a[row][j]) > EPS) {
112                    coef = a[row][j];
113                    forn(col, n + m + 2)
114                        a[row][col] -= a[ind][col] * coef;
115                    a[row][j] = 0; // reduce precision error
116                }
117            p[ind] = j;
118        };
119
120        // the Simplex itself
121        auto iterate = [&]( int nn ) {
122            for (int run = 1; run; ) {
123                run = 0;
124                forn(j, nn) {
125                    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     swap(a[m], a[m + 1]);
80     if (!iterate(n + m))
81         return INFINITY;
82     x = vdbl(n, 0);
83     forn(i, m)
84         if (p[i] < n)
85             x[p[i]] = a[i][n + m + 1];
86     return -a[m][n + m + 1];
87 }
88
89 // simplex usage:
90 vdbl x(n);
91 double result = simplex(n, m, a, b, c, x);
92 if (isinf(result))
93     puts("Unbounded");
94 else if (isnan(result))
95     puts("No solution");
96 else {
97     printf("%.9f :", result);
98     forn(i, n)
99         printf("%.9f", x[i]);
100     puts("");
101 }
102
103 // simplex ends
104
105 // sum over subsets begins
106 // fast subset convolution O(n 2^n)

```

```

3     for(int i = 0; i < (1<<N); ++i)
4         F[i] = A[i];
5     for(int i = 0; i < N; ++i) for(int mask = 0; mask
6         ↪ < (1<<N); ++mask){
7         if(mask & (1<<i))
8             F[mask] += F[mask^(1<<i)];
9     }
10 // sum over subsets ends
11
12 // algebra begins
13
14 Pick
15  $B + \Gamma / 2 - 1$ ,
16 где B - количество целочисленных точек внутри
17 ↪ многоугольника, а  $\Gamma$  - количество
18 ↪ целочисленных точек на границе
19 ↪ многоугольника.
20
21 Newton
22  $x_{i+1} = x_i - f(x_i) / f'(x_i)$ 
23
24 Catalan
25  $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$ 
26  $C_i = \frac{1}{n+1} \binom{2n}{n}$ 
27
28  $G_N = 2^{n(n-1)/2}$ 
29 Количество связанных помеченных графов
30 Conn_N =  $G_N - \frac{1}{N} \sum_{K=1}^{N-1} K \binom{N}{K} \text{Conn}_K$ 
31 ↪  $G_{N-K}$ 
32
33 Количество помеченных графов с K компонентами
34 ↪ связности
35  $D[N][K] = \sum_{S=1}^N \binom{N-1}{S-1} \text{Conn}_S D[N-S][K-1]$ 
36
37 Miller-Rabbin
38  $a = a^t$ 
39 FOR i = 1...s
40     if  $a^{2^i} = 1$  &&  $|a| \neq 1$ 
41         NOT PRIME
42      $a = a^{2^i}$ 
43 return  $a = 1$  ? PRIME : NOT PRIME
44
45 Интегрирование по формуле Симпсона
46  $\int_a^b f(x) dx \approx ?$ 
47  $x_i := a + ih, i = 0 \dots 2n$ 
48  $h = \frac{b-a}{2n}$ 
49
50  $\int_a^b f(x) dx \approx \frac{h}{3} (f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) +$ 
51     ↪  $2f(x_4) + \dots + 4f(x_{2n-1}) +$ 
52     ↪  $f(x_{2n}))$ 
53 Погрешность имеет порядок уменьшения как  $O(n^{-4})$ .
54
55 // algebra ends
56
57 // wavelet tree begins
58
59 struct wavelet_tree{

```

```

4   int lo, hi;
5   wavelet_tree *l, *r;
6   vi b;
7
8   //nos are in range [x,y]
9   //array indices are [from, to)
10  wavelet_tree(int *from, int *to, int x, int y){ }
11      lo = x, hi = y;
12      if(lo == hi or from >= to) return;
13
14      int mid = (lo+hi)/2;
15      auto f = [mid](int x){
16          return x <= mid;
17      };
18      b.reserve(to-from+1);
19      b.pb(0);
20      //b[i] = no of elements from first "i"
21      //elements that go to left node
22      for(auto it = from; it != to; it++)
23          b.pb(b.back() + f(*it));
24
25      //see how lambda function is used here
26      auto pivot = stable_partition(from, to, f);
27      l = new wavelet_tree(from, pivot, lo, mid);
28      r = new wavelet_tree(pivot, to, mid+1, hi);
29  }
30
31  //kth smallest element in [l, r]
32  int kth(int l, int r, int k){
33      if(l > r) return 0;
34      if(lo == hi) return lo;
35      //how many nos are there in left node from
36      // [l, r]
37      int inleft = b[r] - b[l-1];
38      int lb = b[l-1]; //amt of nos from first
39      // (l-1) nos that go in left
40      int rb = b[r]; //amt of nos from first (r)
41      // nos that go in left
42      //so [lb+1, rb] represents nos from [l, r]
43      // that go to left
44      if(k <= inleft) return this->l->kth(lb+1, rb, k);
45      // (l-1-lb) is amt of nos from first (l-1) nos
46      // that go to right
47      // (r-rb) is amt of nos from first (r) nos
48      // that go to right
49      //so [l-lb, r-rb] represents nos from [l, r]
50      // that go to right
51      return this->r->kth(l-lb, r-rb, k-inleft);
52  }
53
54  };
55
56  // wavelet tree ends
57
58  // berlecamp-massey begins
59
60  const int SZ = MAXN;
61
62  ll qp(ll a, ll b) {
63      ll x = 1;
64      a %= MOD;
65
66      while (b) {
67          if (b & 1) x = x * a % MOD;
68          a = a * a % MOD;
69          b >>= 1;
70      }
71      return x;
72  }
73
74  namespace linear_seq {
75      inline vector<int> BM(vector<int> x) {
76          vector<int> ls, cur;
77          int lf, ld;
78          for (int i = 0; i < int(x.size()); ++i) {
79              ll t = 0;
80              for (int j = 0; j < int(cur.size()); ++j)
81                  t = (t + x[i - j - 1] * (ll) cur[j]) %
82                      MOD;
83              if ((t - x[i]) % MOD == 0) continue;
84              if (!cur.size()) {
85                  cur.resize(i + 1);
86                  lf = i;
87                  ld = (t - x[i]) % MOD;
88                  continue;
89              }
90              ll k = -(x[i] - t) * qp(ld, MOD - 2) % MOD;
91              vector<int> c(i - lf - 1);
92              c.pb(k);
93              for (int j = 0; j < int(ls.size()); ++j)
94                  c.pb(-ls[j] * k % MOD);
95              if (c.size() < cur.size())
96                  c.resize(cur.size());
97              for (int j = 0; j < int(cur.size()); ++j)
98                  c[j] = (c[j] + cur[j]) % MOD;
99              if (i - lf + (int) ls.size() >= (int)
100                  cur.size())
101                  ls = cur, lf = i, ld = (t - x[i]) % MOD;
102              cur = c;
103          }
104          for (int i = 0; i < int(cur.size()); ++i)
105              cur[i] = (cur[i] % MOD + MOD) % MOD;
106          return cur;
107      }
108
109      int m;
110      ll a[SZ], h[SZ], t_[SZ], s[SZ], t[SZ];
111
112      inline void mull(ll* p, ll* q) {
113          for (int i = 0; i < m + m; ++i) t_[i] = 0;
114          for (int i = 0; i < m; ++i)
115              if (p[i])
116                  for (int j = 0; j < m; ++j)
117                      t_[i + j] = (t_[i + j] + p[i] * q[j]) %
118                          MOD;
119          for (int i = m + m - 1; i >= m; --i)
120              if (t_[i])
121                  for (int j = m - 1; ~j; --j)
122                      t_[i - j - 1] = (t_[i - j - 1] + t_[i]
123                          * h[j]) % MOD;
124          for (int i = 0; i < m; ++i) p[i] = t_[i];
125      }
126  }

```

```

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] *
76         ↪ a[i]) % MOD;
77     return (su % MOD + MOD) % MOD;
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]
85         ↪ = x[i];
86     return calc(n);
87 }
88 //b=a0/(1-p)
89 inline void calc_generating_function(const
90     ↪ vector<int>& b, vector<int>& p,
91     ↪ vector<int>& a0) {
92     p = BM(b);
93     a0.resize(p.size());
94     for (int i = 0; i < a0.size(); ++i) {
95         a0[i] = b[i];
96         for (int j = 0; j < i; ++j) {
97             a0[i] += MOD - (p[j] * 1ll * b[i - j -
98                 ↪ 1]) % MOD;
99             if (a0[i] > MOD) {
100                 a0[i] -= MOD;
101             }
102         }
103     }
104 // berlecamp-massey ends
105 // AND-FFT begins
106 void fast_fourier(vector<int>& a) { // AND-FFT.
107     for (int k = 1; k < SZ(a); k *= 2)
108     for (int start = 0; start < (1 << K); start +=
109         ↪ 2 * k) {
110         for (int off = 0; off < k; ++off) {
111             int a_val = a[start + off];
112             int b_val = a[start + k + off];
113
114             a[start + off] = b_val;
115             a[start + k + off] = add(a_val, b_val);
116         }
117     }
118 }
119
120 void inverse_fast_fourier(vector<int>& a) {
121     for (int k = 1; k < SZ(a); k *= 2)
122     for (int start = 0; start < (1 << K); start +=
123         ↪ 2 * k) {
124         for (int off = 0; off < k; ++off) {
125             int a_val = a[start + off];
126             int b_val = a[start + k + off];
127
128             a[start + off] = sub(b_val, a_val);
129             a[start + k + off] = a_val;
130         }
131     }
132 }
133 // AND-FFT ends
134 // 2-chinese begins
135 template <typename Info>
136 class DSU {
137 public:
138     DSU ( int n ) : jump (new int[n]), rank (new
139         ↪ int [n]), info (new Info [n]) {
140         for (int i = 0; i < n; i++) {
141             jump[i] = i;
142             rank[i] = 0;
143         }
144     }
145     Info& operator [] ( int x ) {
146         return info[get (x)];
147     }
148     void merge ( int a, int b, const Info &comment
149         ↪ ) {
150         a = get (a);
151         b = get (b);
152         if (rank[a] <= rank[b]) {
153             jump[a] = b;
154             rank[b] += rank[a] == rank[b];
155             info[b] = comment;
156         } else {
157             jump[b] = a;
158             info[a] = comment;
159         }
160     }
161 private:
162     int *jump, *rank;
163     Info *info;
164
165     int get ( int x ) {
166         return jump[x] == x ? x : (jump[x] = get
167             ↪ (jump[x]));
168     }
169 };
170
171 struct Treap {
172     int value, add;
173     int source, target, height;
174     int min_value, min_path;
175
176     Treap *left, *right;

```

```

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 Treap& operator += ( int sub ) {
51     add += sub;
52     return *this;
53 }
54
55 void push () {
56     if (!add)
57         return;
58     if (left) {
59         left->add += add;
60     }
61     if (right) {
62         right->add += add;
63     }
64     value += add;
65     min_value += add;
66     add = 0;
67 }
68
69 void recalc () {
70     min_value = value;
71     min_path = 0;
72     if (left && left->min_value + left->add <
73         ↪ min_value) {
74         min_value = left->min_value + left->add;
75         min_path = -1;
76     }
77     if (right && right->min_value + right->add <
78         ↪ min_value) {
79         min_value = right->min_value + right->add;
80         min_path = +1;
81     }
82 }
83 Treap* treap_merge ( Treap *x, Treap *y ) {
84     if (!x)
85         return y;
86     if (!y)
87         return x;
88     if (x->height < y->height) {
89         x->push ();
90         x->right = treap_merge (x->right, y);
91         x->recalc ();
92         return x;
93     } else {
94         y->push ();
95         y->left = treap_merge (x, y->left);
96         y->recalc ();
97         return y;
98     }
99 }
100 Treap* treap_getmin ( Treap *x, int &source, int
101     ↪ &target, int &value ) {
102     assert (x);
103     x->push ();
104     if (x->min_path == 0) {
105         // memory leak, sorry
106         source = x->source;
107         target = x->target;
108         value = x->value + x->add;
109         return treap_merge (x->left, x->right);
110     } else if (x->min_path == -1) {
111         x->left = treap_getmin (x->left, source,
112             ↪ target, value);
113         value += x->add;
114         x->recalc ();
115         return x;
116     } else if (x->min_path == +1) {
117         x->right = treap_getmin (x->right, source,
118             ↪ target, value);
119         value += x->add;
120         x->recalc ();
121         return x;
122     } else
123         assert (0);
124 }
125 Treap* treap_add ( Treap *x, int add ) {
126     if (!x)
127         return 0;
128     return &((*x) += add);
129 }
130
131 int main () {
132     int n, m;
133     while (scanf ("%d%d", &n, &m) == 2) {
134         Treap * g[n + 1];
135         for (int i = 0; i <= n; i++)
136             g[i] = 0;
137         for (int i = 1; i <= n; i++) {
138             int a;
139             assert (scanf ("%d", &a) == 1);
140             g[i] = treap_merge (g[i], new Treap (i, 0,
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];

```

```

155 vector<int> children[2 * n];
156 memset(c, 0, sizeof(c[0]) * n);
157 memset(parent, -1, sizeof(parent[0]) * 2 *
    ↪ n);
158 vector<int> finish;
159 for(int i = 0; i < n; i++) {
160     if(dsu[i].first == 0)
161         continue;
162     int u = i;
163     c[u] = 1;
164     while(true) {
165         int source, target, value;
166         dsu[u].second = treap_getmin(dsu[u].second,
    ↪ source, target, value);
167         if(dsu[target] == dsu[u])
168             continue;
169         treap_add(dsu[u].second, -value);
170         ans += value;
171         jump_from[dsu[u].first] = source;
172         jump[dsu[u].first] = target;
173         if(dsu[target].first == 0)
174             break;
175         if(!c[target]) {
176             c[target] = 1;
177             u = target;
178             continue;
179         }
180         assert(k < 2 * n);
181         int node = k++, t = target;
182         parent[dsu[u].first] = node;
183         children[node].push_back(dsu[u].first);
184         dsu[u].first = node;
185         Treap *v = dsu[u].second;
186         while(dsu[t].first != node) {
187             int next = jump[dsu[t].first];
188             parent[dsu[t].first] = node;
189             children[node].push_back(dsu[t].first);
190             v = treap_merge(v, dsu[t].second);
191             dsu.merge(u, t, make_pair(node, v));
192             t = next;
193         }
194     }
195     u = i;
196     while(dsu[u].first) {
197         int next = jump[dsu[u].first];
198         finish.push_back(dsu[u].first);
199         dsu.merge(u, 0, make_pair(0, (Treap *)0));
200         u = next;
201     }
202 }
203 bool ok[k];
204 int res[n];
205 memset(ok, 0, sizeof(ok[0]) * k);
206 memset(res, -1, sizeof(res[0]) * n);
207 function<void(int, int)> add_edge = [&ok,
    ↪ &parent, &res, &n] (int a, int b) {
208     assert(0 <= a && a < n);
209     assert(0 <= b && b < n);
210     assert(res[a] == -1);
211     res[a] = b;
212     while(a != -1 && !ok[a]) {
213         ok[a] = true;
214         a = parent[a];
215     }
216 };
217 function<void(int)> reach = [&ok, &reach,
    ↪ &children, &jump, &jump_from, &add_edge](
    ↪ int u) {
218     if(!ok[u])
219         add_edge(jump_from[u], jump[u]);
220     for(auto x : children[u])
221         reach(x);
222 };
223 for(auto x : finish)
224     reach(x);
225 printf("%d\n", ans);
226 for(int i = 1; i < n; i++)
227     printf("%d%c", res[i] ? res[i] : -1, "\n "[i
    ↪ < n - 1]);
228 }
229 return 0;
230 }
231
232 // 2-chinese ends
233
234 // general max weight match begins
235
236 #define DIST(e)
    ↪ (lab[e.u]+lab[e.v]-g[e.u][e.v].w*2)
237 using namespace std;
238 typedef long long ll;
239 const int N = 1023, INF = 1e9;
240 struct Edge {
241     int u, v, w;
242 } g[N][N];
243 int n, m, n_x, lab[N], match[N], slack[N], st[N],
    ↪ pa[N], flower_from[N][N], S[N], vis[N];
244 vector<int> flower[N];
245 deque<int> q;
246 void update_slack(int u, int x) {
247     if(!slack[x] || DIST(g[u][x]) <
    ↪ DIST(g[slack[x]][x])) slack[x] = u;
248 }
249 void set_slack(int x) {
250     slack[x] = 0;
251     for(int u = 1; u <= n; ++u)
252         if(g[u][x].w > 0 && st[u] != x && S[st[u]] ==
    ↪ 0) update_slack(u, x);
253 }
254 void q_push(int x) {
255     if(x <= n) return q.push_back(x);
256     for(int i = 0; i < flower[x].size(); i++)
257         q_push(flower[x][i]);
258 }
259 void set_st(int x, int b) {
260     st[x] = b;
261     if(x <= n) return;
262     for(int i = 0; i < flower[x].size(); ++i)
263         set_st(flower[x][i], b);
264 }
265 int get_pr(int b, int xr) {
266     int pr = find(flower[b].begin(),
    ↪ flower[b].end(), xr)-flower[b].begin();

```

```

32     if (pr % 2 == 1) {
33         reverse(flower[b].begin() + 1,
34             ↪ flower[b].end());
35     }
36     else return pr;
37 }
38 void set_match(int u, int v) {
39     match[u] = g[u][v].v;
40     if (u <= n) return;
41     Edge e = g[u][v];
42     int xr = flower_from[u][e.u], pr = get_pr(u,
43         ↪ xr);
44     for (int i = 0; i < pr; ++i)
45         ↪ set_match(flower[u][i], flower[u][i~1]);
46     set_match(xr, v);
47     rotate(flower[u].begin(), flower[u].begin()
48         ↪ +pr, flower[u].end());
49 }
50 void augment(int u, int v) {
51     int xnv = st[match[u]];
52     set_match(u, v);
53     if (!xnv) return;
54     set_match(xnv, st[pa[xnv]]);
55     augment(st[pa[xnv]], xnv);
56 }
57 int get_lca(int u, int v) {
58     static int t = 0;
59     for (++t; u || v; swap(u, v)) {
60         if (u == 0) continue;
61         if (vis[u] == t) return u;
62         vis[u] = t;
63         u = st[match[u]];
64         if (u) u = st[pa[u]];
65     }
66     return 0;
67 }
68 void add_blossom(int u, int lca, int v) {
69     int b = n+1;
70     while (b <= n_x && st[b]) ++b;
71     if (b > n_x) ++n_x;
72     lab[b] = 0, S[b] = 0;
73     match[b] = match[lca];
74     flower[b].clear();
75     flower[b].push_back(lca);
76     for (int x = u, y; x != lca; x = st[pa[y]])
77         flower[b].push_back(x), flower[b].push_back(y
78             ↪ = st[match[x]]), q_push(y);
79     reverse(flower[b].begin() + 1, flower[b].end());
80     for (int x = v, y; x != lca; x = st[pa[y]])
81         flower[b].push_back(x), flower[b].push_back(y
82             ↪ = st[match[x]]), q_push(y);
83     set_st(b, b);
84     for (int x = 1; x <= n_x; ++x) g[b][x].w =
85         ↪ g[x][b].w = 0;
86     for (int x = 1; x <= n; ++x) flower_from[b][x]
87         ↪ = 0;
88     for (int i = 0; i < flower[b].size(); ++i) {
89         int xs = flower[b][i];
90         for (int x = 1; x <= n_x; ++x)
91             if (g[b][x].w == 0 || DIST(g[xs][x]) <
92                 ↪ DIST(g[b][x]))
93                 g[b][x] = g[xs][x], g[x][b] = g[x][xs];
94         for (int x = 1; x <= n; ++x)
95             if (flower_from[xs][x]) flower_from[b][x] =
96                 ↪ xs;
97     }
98     set_slack(b);
99 }
100 void expand_blossom(int b) // S[b] == 1 {
101     for (int i = 0; i < flower[b].size(); ++i)
102         set_st(flower[b][i], flower[b][i]);
103     int xr = flower_from[b][g[b][pa[b]].u], pr =
104         ↪ get_pr(b, xr);
105     for (int i = 0; i < pr; i += 2) {
106         int xs = flower[b][i], xns = flower[b][i+1];
107         pa[xs] = g[xns][xs].u;
108         S[xs] = 1, S[xns] = 0;
109         slack[xs] = 0, set_slack(xns);
110         q_push(xns);
111     }
112     S[xr] = 1, pa[xr] = pa[b];
113     for (int i = pr+1; i < flower[b].size(); ++i) {
114         int xs = flower[b][i];
115         S[xs] = -1, set_slack(xs);
116     }
117     st[b] = 0;
118 }
119 bool on_found_Edge(const Edge &e) {
120     int u = st[e.u], v = st[e.v];
121     if (S[v] == -1) {
122         pa[v] = e.u, S[v] = 1;
123         int nu = st[match[v]];
124         slack[v] = slack[nu] = 0;
125         S[nu] = 0, q_push(nu);
126     }
127     else if (S[v] == 0) {
128         int lca = get_lca(u, v);
129         if (!lca) return augment(u, v), augment(v,
130             ↪ u), 1;
131         else add_blossom(u, lca, v);
132     }
133     return 0;
134 }
135 bool matching() {
136     fill(S, S+n_x+1, -1), fill(slack, slack+n_x+1,
137         ↪ 0);
138     q.clear();
139     for (int x = 1; x <= n_x; ++x)
140         if (st[x] == x && !match[x]) pa[x] = 0, S[x]
141             ↪ = 0, q_push(x);
142     if (q.empty()) return 0;
143     for (;;) {
144         while (q.size()) {
145             int u = q.front();
146             q.pop_front();
147             if (S[st[u]] == 1) continue;
148             for (int v = 1; v <= n; ++v)
149                 if (g[u][v].w > 0 && st[u] != st[v]) {
150                     if (DIST(g[u][v]) == 0) {
151                         if (on_found_Edge(g[u][v])) return 1;
152                     }
153                 }
154         }
155     }

```

```

139     }
140     else update_slack(u, st[v]);
141 }
142 }
143 int d = INF;
144 for (int b = n+1; b <= n_x; ++b)
145     if (st[b] == b && S[b] == 1) d = min(d,
146         ↪ lab[b]/2);
147 for (int x = 1; x <= n_x; ++x)
148     if (st[x] == x && slack[x]) {
149         if (S[x] == -1) d = min(d,
150             ↪ DIST(g[slack[x]][x]));
151         else if (S[x] == 0) d = min(d,
152             ↪ DIST(g[slack[x]][x])/2);
153     }
154 for (int u = 1; u <= n; ++u) {
155     if (S[st[u]] == 0) {
156         if (lab[u] <= d) return 0;
157         lab[u] -= d;
158     }
159     else if (S[st[u]] == 1) lab[u] += d;
160 }
161 for (int b = n+1; b <= n_x; ++b)
162     if (st[b] == b) {
163         if (S[st[b]] == 0) lab[b] += d*2;
164         else if (S[st[b]] == 1) lab[b] -= d*2;
165     }
166 q.clear();
167 for (int x = 1; x <= n_x; ++x)
168     if (st[x] == x && slack[x] && st[slack[x]]14
169         ↪ != x && DIST(g[slack[x]][x]) == 0)
170         if (on_found_Edge(g[slack[x]][x]))16
171             ↪ return
172             ↪ 1;
173 for (int b = n+1; b <= n_x; ++b)
174     if (st[b] == b && S[b] == 1 && lab[b] == 0)15
175         ↪ expand_blossom(b);
176 }
177 return 0;
178 }
179 pair < ll, int> weight_blossom() {
180     fill(match, match+n+1, 0);
181     n_x = n;
182     int n_matches = 0;
183     ll tot_weight = 0;
184     for (int u = 0; u <= n; ++u) st[u] = u,
185         ↪ flower[u].clear();
186     int w_max = 0;
187     for (int u = 1; u <= n; ++u)
188         for (int v = 1; v <= n; ++v) {
189             flower_from[u][v] = (u == v?u:0);
190             w_max = max(w_max, g[u][v].w);
191         }
192     for (int u = 1; u <= n; ++u) lab[u] = w_max;
193     while (matching()) ++n_matches;
194     for (int u = 1; u <= n; ++u)
195         if (match[u] && match[u] < u)
196             tot_weight += g[u][match[u]].w;
197     return make_pair(tot_weight, n_matches);
198 }
199 int main() {
200     cin>>n>>m;
201
202     for (int u = 1; u <= n; ++u)
203         for (int v = 1; v <= n; ++v)
204             g[u][v] = Edge {u, v, 0};
205     for (int i = 0, u, v, w; i < m; ++i) {
206         cin>>u>>v>>w;
207         g[u][v].w = g[v][u].w = w;
208     }
209     cout << weight_blossom().first << '\n';
210     for (int u = 1; u <= n; ++u) cout << match[u]
211         ↪ << ' ';
212 }
213
214 // general max weight match ends
215
216 // slow min circulation begins
217
218 struct Edge {
219     int a;
220     int b;
221     int cost;
222 };
223
224 vector<int> negative_cycle(int n, vector<Edge>
225     ↪ &edges) {
226     // O(nm), return ids of edges in negative cycle
227
228     vector<int> d(n);
229     vector<int> p(n, -1); // last edge ids
230
231     const int inf = 1e9;
232
233     int x = -1;
234     for (int i = 0; i < n; i++) {
235         x = -1;
236         for (int j = 0; j < edges.size(); j++) {
237             Edge &e = edges[j];
238
239             if (d[e.b] > d[e.a] + e.cost) {
240                 d[e.b] = max(-inf, d[e.a] + e.cost);
241                 p[e.b] = j;
242                 x = e.b;
243             }
244         }
245     }
246
247     if (x == -1)
248         return vector<int>(); // no negative cycle
249
250     for (int i = 0; i < n; i++)
251         x = edges[p[x]].a;
252
253     vector<int> result;
254     for (int cur = x; ; cur = edges[p[cur]].a) {
255         if (cur == x && result.size() > 0) break;
256         result.push_back(p[cur]);
257     }
258     reverse(result.begin(), result.end());
259
260     return result;
261 }

```



```

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

```

```

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

```

DM

xmodmap -e 'keycode 94='  
setxkbmap us

**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.**Fast subset convolution**

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

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

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

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

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

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

$$f_0(S) := f(S)$$

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

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

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

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

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

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

calculate using  $f_j!$