

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

```

```

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

```

```

79     if (!ok) {
80         // det = 0;
81         return false;
82     }
83 }
84 }
85
86 res[n] = sub(0, 1);
87 for (int i = n - 1; i >= 0; i--) {
88     int k = zro[i];
89     for (auto& el : rows[i])
90         if (el.first != k)
91             res[p[k]] = sub(res[p[k]],
92                             ↪ mult(el.second, res[p[el.first]]));
93 }
94 return true;
95 }
96
97 // fast gauss ends
98
99 // stable gauss begins
100 // if at least one solution returns it in ans
101 int gauss (vector < vector<double> > a,
102 ↪ vector<double> & ans) {
103     int n = (int) a.size();
104     int m = (int) a[0].size() - 1;
105
106     vector<int> where (m, -1);
107     for (int col=0, row=0; col<m && row<n; ++col) {
108         int sel = row;
109         for (int i=row; i<n; ++i)
110             if (abs (a[i][col]) > abs (a[sel][col]))
111                 sel = i;
112         if (abs (a[sel][col]) < EPS)
113             continue;
114         for (int i=col; i<=m; ++i)
115             swap (a[sel][i], a[row][i]);
116         where[col] = row;
117
118         for (int i=0; i<n; ++i)
119             if (i != row) {
120                 double c = a[i][col] / a[row][col];
121                 for (int j=col; j<=m; ++j)
122                     a[i][j] -= a[row][j] * c;
123             }
124         ++row;
125     }
126 }
127
128 ans.assign (m, 0);
129 for (int i=0; i<m; ++i)
130     if (where[i] != -1)
131         ans[i] = a[where[i]][m] / a[where[i]][i];
132 for (int i=0; i<n; ++i) {
133     double sum = 0;
134     for (int j=0; j<m; ++j)
135         sum += ans[j] * a[i][j];
136     if (abs (sum - a[i][m]) > EPS)
137         return 0;
138 }
139
140 for (int i=0; i<m; ++i)

```

```

41     if (where[i] == -1)
42         return INF;
43     return 1;
44 }
45
46 // stable gauss ends
47
48 // simple geometry begins
49
50 struct Point {
51     double x, y;
52     Point operator+(const Point& p) const { return
53         ↪ {x + p.x, y + p.y}; }
54     Point operator-(const Point& p) const { return
55         ↪ {x - p.x, y - p.y}; }
56     Point operator*(const double d) const { return
57         ↪ {x * d, y * d}; }
58     Point rotate() const { return {y, -x}; }
59     double operator*(const Point& p) const { return
60         ↪ x * p.x + y * p.y; }
61     double operator^(const Point& p) const { return
62         ↪ x * p.y - y * p.x; }
63     double dist() const { return sqrt(x * x + y *
64         ↪ y); }
65 };
66
67 struct Line {
68     double a, b, c;
69     Line(const Point& p1, const Point& p2) {
70         a = p1.y - p2.y;
71         b = p2.x - p1.x;
72         c = - a * p1.x - b * p1.y;
73
74         double d = sqrt(sqr(a) + sqr(b));
75         a /= d, b /= d, c /= d;
76     }
77     bool operator||(const Line& l) const { return
78         ↪ fabs(a * l.b - l.a * b) < EPS; }
79     double dist(const Point& p) const { return
80         ↪ fabs(a * p.x + b * p.y + c); }
81     Point operator^(const Line& l) const {
82         return {(l.c * b - c * l.b) / (a * l.b - l.a
83         ↪ * b),
84                 (l.c * a - c * l.a) / (l.a * b - a *
85         ↪ l.b)};
86     }
87     Point projection(const Point& p) const {
88         return p - Point{a, b} * (a * p.x + b * p.y +
89         ↪ c);
90     }
91 };
92
93 struct Circle {
94     Point c;
95     double r;
96     Circle(const Point& c, double r) : c(c), r(r)
97     ↪ {}
98     Circle(const Point& a, const Point& b, const
99     ↪ Point& c) {
100         Point p1 = (a + b) * 0.5, p2 = (a + c) * 0.5;
101         Point q1 = p1 + (b - a).rotate(), q2 = p2 +
102         ↪ (c - a).rotate();

```

```

42     this->c = Line(p1, q1) ^ Line(p2, q2);
43     r = (a - this->c).dist();
44 }
45 };
46
47 inline bool on_segment(const Point& p1, const
    ↪ Point& p2, const Point& x, bool strictly) {
48     if (fabs((p1 - x) ^ (p2 - x)) > EPS)
49         return false;
50     return (p1 - x) * (p2 - x) < (strictly ? - EPS
    ↪ : EPS);
51 }
52
53 // in case intersection is not a segment
54 inline bool intersect_segments(const Point& p1,
    ↪ const Point& p2, const Point& q1, const
    ↪ Point& q2, Point& x) {
55     Line l1(p1, p2), l2(q1, q2);
56     if (l1 || l2) return false;
57     x = l1 ^ l2;
58     return on_segment(p1, p2, x, false);
59 }
60
61 // in case circles are not equal
62 inline bool intersect_circles(const Circle& c1,
    ↪ const Circle& c2, Point& p1, Point& p2) {
63     double d = (c2.c - c1.c).dist();
64     if (d > c1.r + c2.r + EPS || d < fabs(c1.r -
    ↪ c2.r) - EPS)
65         return false;
66     double cosa = (sqr(d) + sqr(c1.r) - sqr(c2.r))
    ↪ / (2 * c1.r * d);
67     double l = c1.r * cosa, h = sqrt(sqr(c1.r) -
    ↪ sqr(l));
68     Point v = (c2.c - c1.c) * (1 / d), p = c1.c +
    ↪ * 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
7     int64_t operator^(const Point& p) const {
    ↪ return x * 111 * p.y - y * 111 * p.x; }
8     int64_t dist() const { return x * 111 * x + y *
    ↪ 111 * y; }
9     bool operator<(const Point& p) const { return x
    ↪ != p.x ? x < p.x : y < p.y; }
10 };
11
12 // all point on convex hull are included
13 vector<Point> convex_hull(vector<Point> pt) {
14     int n = pt.size();
15     Point p0 = *std::min_element(pt.begin(),
    ↪ pt.end());
16     std::sort(pt.begin(), pt.end(), [&p0](const
    ↪ Point& a, const Point& b) {
17         int64_t cp = (a - p0) ^ (b - p0);
18         return cp != 0 ? cp > 0 : (a - p0).dist() <
    ↪ (b - p0).dist();
19     });
20
21     int i = n - 1;
22     for (; i > 0 && ((pt[i] - p0) ^ (pt[i - 1] -
    ↪ p0)) == 0; i--);
23     std::reverse(pt.begin() + i, pt.end());
24
25     vector<Point> ch;
26     for (auto& p : pt) {
27         while (ch.size() > 1) {
28             auto& p1 = ch[(int) ch.size() - 1];
29             auto& p2 = ch[(int) ch.size() - 2];
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
37     return ch;
38 }
39
40 // convex hull ends
41
42 // convex hull trick begins
43
44 typedef long long ftype;
45 typedef complex<ftype> point;
46 #define x real
47 #define y imag
48
49 ftype dot(point const& a, point const& b) {
50     return (conj(a) * b).x();
51 }
52
53 ftype f(point const& a, int x) {
54     return dot(a, {compressed[x], 1});
55     //return dot(a, {x, 1});
56 }
57
58 int pos = 0;
59
60 // (x, y) -> (k, b) -> kb + x
61 struct li_chao { // for min

```



```

21     vector<point> line;
22
23     li_chao(int maxn) {
24         line.resize(4 * maxn, {0, inf});
25     }
26
27     void add_line(int v, int l, int r, int a, int
    ↪ b, point nw) {
28         if (r <= a || b <= l) return; // remove if no
    ↪ [a, b] query
29
30         int m = (l + r) >> 1;
31
32         if (!(a <= l && r <= b)) { // remove if no
    ↪ [a, b] query
33             add_line(v + v + 1, l, m, a, b, nw);
34             add_line(v + v + 2, m, r, a, b, nw);
35             return;
36         }
37
38         bool lef = f(nw, l) < f(line[v], l);
39         bool mid = f(nw, m) < f(line[v], m);
40
41         if (mid) swap(line[v], nw);
42
43         if (l == r - 1)
44             return;
45
46         if (lef != mid)
47             add_line(v + v + 1, l, m, a, b, nw);
48         else
49             add_line(v + v + 2, m, r, a, b, nw);
50     }
51
52     ftype get(int v, int l, int r, int x) {
53         if (l == r - 1)
54             return f(line[v], x);
55         int m = (l + r) / 2;
56         if (x < m) {
57             return min(f(line[v], x), get(v + v + 1, l,
    ↪ m, x));
58         } else {
59             return min(f(line[v], x), get(v + v + 2, m,
    ↪ r, x));
60         }
61     }
62
63 } cdt(maxn);
64
65 // convex hull with stack
66
67 ftype cross(point a, point b) {
68     return (conj(a) * b).y();
69 }
70
71 vector<point> hull, vecs;
72
73 void add_line(ftype k, ftype b) {
74     point nw = {k, b};
75     while(!vecs.empty() && dot(vecs.back(), nw -
    ↪ hull.back()) < 0) {
76         hull.pop_back();
77         vecs.pop_back();
78     }
79     if(!hull.empty()) {
80         vecs.push_back(li * (nw - hull.back()));
81     }
82     hull.push_back(nw);
83 }
84
85 int get(ftype x) {
86     point query = {x, 1};
87     auto it = lower_bound(vecs.begin(), vecs.end(),
    ↪ query, [](point a, point b) {
88         return cross(a, b) > 0;
89     });
90     return dot(query, hull[it - vecs.begin()]);
91 }
92
93 // convex hull trick ends
94
95 // heavy-light begins
96
97 int sz[maxn];
98
99 void dfs_sz(int v, int par = -1) {
100     sz[v] = 1;
101     for (int x : gr[v])
102         if (x != par) {
103             dfs_sz(x, v);
104             sz[v] += sz[x];
105         }
106     for (int i = 0; i < gr[v].size(); i++)
107         if (gr[v][i] != par)
108             if (sz[gr[v][i]] * 2 >= sz[v]) {
109                 swap(gr[v][i], gr[v][0]);
110                 break;
111             }
112 }
113
114 int rev[maxn];
115 int t_in[maxn];
116 int upper[maxn];
117 int par[maxn];
118 int dep[maxn];
119
120 int T = 0;
121
122 void dfs_build(int v, int uppr, int pr = -1) {
123     rev[T] = v;
124     t_in[v] = T++;
125     dep[v] = pr == -1 ? 0 : dep[pr] + 1;
126     par[v] = pr;
127     upper[v] = uppr;
128
129     bool first = true;
130
131     for (int x : gr[v])
132         if (x != pr) {
133             dfs_build(x, first ? upper[v] : x, v);
134             first = false;
135         }
136 }

```



```

43
44 struct interval {
45     int l;
46     int r;
47     bool inv; // should direction be reversed
48 };
49
50 // node-weighted hld
51 vector<interval> get_path(int a, int b) {
52     vector<interval> front;
53     vector<interval> back;
54
55     while (upper[a] != upper[b]) {
56         if (dep[upper[a]] > dep[upper[b]]) {
57             front.push_back({t_in[upper[a]], t_in[a],
58                 ↪ true});
59             a = par[upper[a]];
60         } else {
61             back.push_back({t_in[upper[b]], t_in[b],
62                 ↪ false});
63             b = par[upper[b]];
64         }
65     }
66
67     front.push_back({min(t_in[a], t_in[b]),
68         ↪ max(t_in[a], t_in[b]), t_in[a] > t_in[b]});
69     // for edge-weighted hld add:
70     ↪ "front.back().l++;";
71     front.insert(front.end(), back.rbegin(),
72         ↪ back.rend());
73
74     return front;
75 }
76
77 // heavy-light ends
78
79 // max flow begins
80
81 struct edge{
82     int from, to;
83     int c, f, num;
84     edge(int from, int to, int c, int
85         ↪ num):from(from), to(to), c(c), f(0),
86         ↪ num(num){}
87     edge(){}
88 };
89
90 const int max_n = 600;
91
92 edge eds[150000];
93 int num = 0;
94 int it[max_n];
95 vector<int> gr[max_n];
96 int s, t;
97 vector<int> d(max_n);
98
99 bool bfs(int k) {
100     queue<int> q;
101     q.push(s);
102     fill(d.begin(), d.end(), -1);
103     d[s] = 0;
104     while (!q.empty()) {
105
106         int v = q.front();
107         q.pop();
108         for (int x : gr[v]) {
109             int to = eds[x].to;
110             if (d[to] == -1 && eds[x].c - eds[x].f >=
111                 ↪ (1 << k)){
112                 d[to] = d[v] + 1;
113                 q.push(to);
114             }
115         }
116     }
117
118     return (d[t] != -1);
119 }
120
121 int dfs(int v, int flow, int k) {
122     if (flow < (1 << k))
123         return 0;
124     if (v == t)
125         return flow;
126     return flow;
127     for (; it[v] < gr[v].size(); it[v]++) {
128         int num = gr[v][it[v]];
129         if (d[v] + 1 != d[num].to)
130             continue;
131         int res = dfs(eds[num].to, min(flow,
132             ↪ eds[num].c - eds[num].f), k);
133         if (res){
134             eds[num].f += res;
135             eds[num ^ 1].f -= res;
136             return res;
137         }
138     }
139     return 0;
140 }
141
142 void add(int fr, int to, int c, int nm) {
143     gr[fr].push_back(num);
144     eds[num++] = edge(fr, to, c, nm);
145     gr[to].push_back(num);
146     eds[num++] = edge(to, fr, 0, nm); //corrected c
147 }
148
149 int ans = 0;
150 for (int k = 30; k >= 0; k--)
151     while (bfs(k)) {
152         memset(it, 0, sizeof(it));
153         while (int res = dfs(s, 1e9 + 500, k))
154             ans += res;
155     }
156
157 // decomposition
158
159 int path_num = 0;
160 vector<int> paths[max_n];
161 int flows[max_n];
162
163 int decomp(int v, int flow) {
164     if (flow < 1)
165         return 0;
166     if (v == t) {

```

```

84     path_num++;
85     flows[path_num - 1] = flow;
86     return flow;
87 }
88 for (int i = 0; i < gr[v].size(); i++) {
89     int num = gr[v][i];
90     int res = decomp(eds[num].to, min(flow,
91         ↪ eds[num].f));
92     if (res) {
93         eds[num].f -= res;
94         paths[path_num -
95             ↪ 1].push_back(eds[num].num);
96         return res;
97     }
98 }
99 return 0;
100 while (decomp(s, 1e9 + 5));
101
102 // max flow ends
103
104 // min-cost flow begins
105
106 1 long long ans = 0;
107 2 int mx = 2 * n + 2;
108
109 3 memset(upd, 0, sizeof(upd));
110 4 for (int i = 0; i < mx; i++)
111     dist[i] = inf;
112 5 dist[st] = 0;
113 6 queue<int> q;
114 7 q.push(st);
115 8 upd[st] = 1;
116 9 while (!q.empty()){
117 10     int v = q.front();
118 11     q.pop();
119 12     if (upd[v]){
120 13         for (int x : gr[v]) {
121 14             edge &e = edges[x];
122 15             if (e.c - e.f > 0 && dist[v] != inf &&
123 16                 ↪ dist[e.to] > dist[v] + e.w) {
124 17                 dist[e.to] = dist[v] + e.w;
125 18                 if (!upd[e.to])
126 19                     q.push(e.to);
127 20                 upd[e.to] = true;
128 21                 p[e.to] = x;
129 22             }
130 23         }
131 24     }
132 25     upd[v] = false;
133 26 }
134 27 }
135 28 }
136 29 }
137 30
138 31 for (int i = 0; i < k; i++){
139 32     for (int i = 0; i < mx; i++)
140 33         d[i] = inf;
141 34     d[st] = 0;
142 35     memset(used, false, sizeof(used));
143 36     set<pair<int, int>> s;
144 37     s.insert(make_pair(0, st));
145 38     for (int i = 0; i < mx; i++){
146 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
64 int pos = t;
65 while (pos != st){
66     int id = p[pos];
67     edges[id].f += 1;
68     edges[id ^ 1].f -= 1;
69     pos = edges[id].from;
70 }
71
72 // min-cost flow ends
73
74 // min circulation begins
75 class Solver2 { // Min-cost circulation
76     struct Edge {
77         int to, ne, w, c;
78     };
79     vector<Edge> es;
80     vi firs;
81     int curRes;
82 public:
83     Solver2(int n) : es(), firs(n, -1),
84     curRes(0) {}
85     // from, to, capacity (max.flow), cost
86     int adde(int a, int b, int w, int c) {
87         Edge e;
88         e.to = b; e.ne = firs[a];
89         e.w = w; e.c = c;
90         es.pb(e);
91         firs[a] = sz(es) - 1;
92
93         e.to = a; e.ne = firs[b];
94         e.w = 0; e.c = -c;
95         es.pb(e);
96         firs[b] = sz(es) - 1;
97         return sz(es) - 2;
98     }
99     // increase capacity of edge 'id' by 'w'
100     void ince(int id, int w) {
101         es[id].w += w;

```

```

29 }
30 int solve() {
31     const int n = sz(firs);
32
33     for (;;) {
34         vi d(n, 0), fre(n, -1);
35         vi chd(n, -1);
36
37         int base = -1;
38         for (int step = 0; step < n; step++) {
39             for (int i = 0; i < sz(es); i++) if
40                 ↪ (es[i].w > 0) {
41                 int b = es[i].to;
42                 int a = es[i ^ 1].to;
43                 if (d[b] <= d[a] + es[i].c) continue;
44                 d[b] = d[a] + es[i].c;
45                 fre[b] = i;
46                 if (step == n - 1)
47                     base = b;
48             }
49             if (base < 0) break;
50
51             vi seq;
52             vb was(n, false);
53             for (int x = base;; x = es[fre[x] ^ 1].to)
54                 ↪ {
55                 if (!was[x]) {
56                     seq.pb(x);
57                     was[x] = true;
58                 } else {
59                     seq.erase(
60                         seq.begin(),
61                         find(seq.begin(), seq.end(),
62                             x
63                         ));
64                     break;
65                 }
66             }
67             for (int i = 0; i < sz(seq); i++) {
68                 int v = seq[i];
69                 int eid = fre[v];
70                 assert(es[eid].w > 0);
71                 es[eid].w--;
72                 es[eid ^ 1].w++;
73                 curRes += es[eid].c;
74             }
75             return curRes;
76         }
77     };
78     // slow min circulation ends
79
80     // global cut begins
81     // g[i][j] = g[j][i] is sum of edges between i
82     ↪ and j
83     // ans is value of mincut
84     ll g[maxn][maxn], w[maxn];
85     ll ans;
86     bool ex[maxn], inA[maxn];
87     int n;
88     void iterate(int curN){
89         int prev = -1;
90         memset(inA, 0, sizeof(inA));
91         memset(w, 0, sizeof(w));
92         for (int it = 0; it < curN; it++) {
93             int best = -1;
94             for (int i = 0; i < n; i++)
95                 if (ex[i] && !inA[i]
96                     && (best == -1 || w[i] > w[best]))
97                     best = i;
98             assert(best != -1);
99             if (it == curN - 1) {
100                 ans = min(ans, w[best]);
101                 for (int i = 0; i < n; i++){
102                     g[i][prev] += g[best][i];
103                     g[prev][i] = g[i][prev];
104                 }
105                 ex[best] = false;
106             } else {
107                 inA[best] = true;
108                 for (int i = 0; i < n; i++)
109                     w[i] += g[best][i];
110                 prev = best;
111             }
112         }
113     }
114 }
115
116 void solve(){
117     ans = INF;
118     for (int i = n; i > 1; i--)
119         iterate(i);
120     cout << ans << endl;
121 }
122 // global cut ends
123
124 // bad hungarian begins
125
126 fill(par, par + 301, -1);
127 fill(par2, par2 + 301, -1);
128
129 int ans = 0;
130 for (int v = 0; v < n; v++){
131     memset(useda, false, sizeof(useda));
132     memset(usedb, false, sizeof(usedb));
133     useda[v] = true;
134     for (int i = 0; i < n; i++)
135         w[i] = make_pair(a[v][i] + row[v] + col[i],
136             ↪ v);
137     memset(prev, 0, sizeof(prev));
138     int pos;
139     while (true){
140         pair<pair<int, int>, int> p =
141             ↪ make_pair(make_pair(1e9, 1e9), 1e9);
142         for (int i = 0; i < n; i++)
143             if (!usedb[i])
144                 p = min(p, make_pair(w[i], i));
145         for (int i = 0; i < n; i++)
146             if (!useda[i])
147                 row[i] += p.first.first;
148         for (int i = 0; i < n; i++)
149             if (!usedb[i]){
150                 col[i] -= p.first.first;
151                 w[i].first -= p.first.first;
152             }
153     }

```

```

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

```

```

7   for (int i = 1; i < s.size(); i++){
8       if (i <= r)
9           z[i] = min(r - i + 1, z[i - 1]);
10      while (i + z[i] < s.size() && s[z[i]] == s[i
11          ↪ + z[i]])
12          z[i]++;
13      if (i + z[i] - 1 > r){
14          r = i + z[i] - 1;
15          l = i;
16      }
17  }
18  return z;
19 }
20 vector<int> getP(string s){
21     vector<int> p;
22     p.resize(s.size(), 0);
23     int k = 0;
24     for (int i = 1; i < s.size(); i++){
25         while (k > 0 && s[i] == s[k])
26             k = p[k - 1];
27         if (s[i] == s[k])
28             k++;
29         p[i] = k;
30     }
31     return p;
32 }
33
34 vector<int> getH(string s){
35     vector<int> h;
36     h.resize(s.size() + 1, 0);
37     for (int i = 0; i < s.size(); i++)
38         h[i + 1] = ((h[i] * 111 * pow) + s[i] - '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     }
70 }
71
72 while (i <= k) i += j - k;
73 }
74 return s.substr (ans, n/2);
75 }
76
77 // min cyclic shift ends
78
79 // suffix array O(n) begins
80
81 typedef vector<char> bits;
82
83 template<const int end>
84 void getBuckets(int *s, int *bkt, int n, int K) {
85     fill(bkt, bkt + K + 1, 0);
86     forn(i, n) bkt[s[i] + !end]++;
87     forn(i, K) bkt[i + 1] += bkt[i];
88 }
89
90 void induceSAL(bits &t, int *SA, int *s, int
91     ↪ *bkt, int n, int K) {
92     getBuckets<0>(s, bkt, n, K);
93     forn(i, n) {
94         int j = SA[i] - 1;
95         if (j >= 0 && !t[j])
96             SA[bkt[s[j]]++] = j;
97     }
98 }
99
100 void induceSAs(bits &t, int *SA, int *s, int
101     ↪ *bkt, int n, int K) {
102     getBuckets<1>(s, bkt, n, K);
103     for (int i = n - 1; i >= 0; i--) {
104         int j = SA[i] - 1;
105         if (j >= 0 && t[j])
106             SA[--bkt[s[j]]] = j;
107     }
108 }
109
110 void SA_IS(int *s, int *SA, int n, int K) { //
111     ↪ require last symbol is 0
112     #define isLMS(i) (i && t[i] && !t[i-1])
113     int i, j;
114     bits t(n);
115     t[n-1] = 1;
116     for (i = n - 3; i >= 0; i--)
117         t[i] = (s[i]<s[i+1] || (s[i]==s[i+1] &&
118             ↪ t[i+1]==1));
119     int bkt[K + 1];
120     getBuckets<1>(s, bkt, n, K);
121     fill(SA, SA + n, -1);
122     forn(i, n)
123         if (isLMS(i))
124             SA[--bkt[s[i]]] = i;
125     induceSAL(t, SA, s, bkt, n, K);
126     induceSAs(t, SA, s, bkt, n, K);
127     int n1 = 0;
128     forn(i, n)
129         if (isLMS(SA[i]))
130             SA[n1++] = SA[i];
131     fill(SA + n1, SA + n, -1);
132     int name = 0, prev = -1;
133     forn(i, n1) {
134         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] ||
54             t[pos+d] != t[prev+d])
55             diff = true, d = n;
56         else if (d > 0 && (isLMS(pos+d) ||
57             isLMS(prev+d)))
58             d = n;
59     if (diff)
60         name++, prev = pos;
61     SA[n1 + (pos >> 1)] = name - 1;
62 }
63 for (i = n - 1, j = n - 1; i >= n1; i--)
64     if (SA[i] >= 0)
65         SA[j--] = SA[i];
66 int *s1 = SA + n - n1;
67 if (name < n1)
68     SA_IS(s1, SA, n1, name - 1);
69 else
70     forn(i, n1)
71         SA[s1[i]] = i;
72 getBuckets<1>(s, bkt, n, K);
73 for (i = 1, j = 0; i < n; i++)
74     if (isLMS(i))
75         s1[j++] = i;
76 forn(i, n1)
77     SA[i] = s1[SA[i]];
78 fill(SA + n1, SA + n, -1);
79 for (i = n1 - 1; i >= 0; i--) {
80     j = SA[i], SA[i] = -1;
81     SA[--bkt[s[j]]] = j;
82 }
83 induceSA1(t, SA, s, bkt, n, K);
84 induceSAs(t, SA, s, bkt, n, K);
85 }
86 // suffix array O(n) ends
87
88 // suffix array O(n log n) begins
89 string str;
90 int N, m, SA [MAX_N], LCP [MAX_N];
91 int x [MAX_N], y [MAX_N], w [MAX_N], c [MAX_N];
92
93 inline bool cmp (const int a, const int b, const
94     int l) { return (y [a] == y [b] && y [a + l]
95     == y [b + l]); }
96
97 void Sort () {
98     for (int i = 0; i < m; ++i) w[i] = 0;
99     for (int i = 0; i < N; ++i) ++w[x[y[i]]];
100     for (int i = 0; i < m - 1; ++i) w[i + 1] +=
101         w[i];
102     for (int i = N - 1; i >= 0; --i)
103         SA[--w[x[y[i]]]] = y[i];
104 }
105
106 void DA () {
107     for (int i = 0; i < N; ++i) x[i] = str[i], y[i]
108         = i;
109     Sort ();
110     for (int i, j = 1, p = 1; p < N; j <= 1, m =
111         p) {
112         for (p = 0, i = N - j; i < N; i++) y[p++] =
113             i;
114         for (int k = 0; k < N; ++k) if (SA[k] >= j)
115             y[p++] = SA[k] - j;
116         Sort();
117         for (swap (x, y), p = 1, x[SA[0]] = 0, i = 1;
118             i < N; ++i) x[SA [i]] = cmp (SA[i - 1],
119             SA[i], j) ? p - 1 : p++;
120     }
121 }
122
123 // common for all algorithms
124 void kasaiLCP () {
125     for (int i = 0; i < N; i++) c[SA[i]] = i;
126     for (int i = 0, j, k = 0; i < N; LCP [c[i++]] =
127         k)
128         if (c [i] > 0) for (k ? k-- : 0, j = SA[c[i]
129             - 1]; str[i + k] == str[j + k]; k++);
130         else k = 0;
131 }
132
133 void suffixArray () { // require last symbol is
134     char(0)
135     m = 256;
136     N = str.size();
137     DA ();
138     kasaiLCP ();
139 }
140 // suffix array O(n log n) ends
141
142 // bad suffix automaton begins
143
144 struct node{
145     map<char, int> go;
146     int len, suff;
147     long long sum_in;
148     node(){
149
150     };
151
152     node v[max_n * 4];
153
154     int add_node(int max_len){
155         //v[number].sum_in = 0;
156         v[number].len = max_len;
157         v[number].suff = -1;
158         number++;
159         return number - 1;
160     }
161 }
162
163 int last = add_node(0);
164
165 void add_char(char c) {
166     int cur = last;
167     int new_node = add_node(v[cur].len + 1);
168     last = new_node;
169     while (cur != -1){
170         if (v[cur].go.count(c) == 0){
171             v[cur].go[c] = new_node;
172             //v[new_node].sum_in += v[cur].sum_in;
173             cur = v[cur].suff;
174             if (cur == -1)
175                 v[new_node].suff = 0;
176         }
177     }
178 }

```

```

33     }else{
34         int a = v[cur].go[c];
35         if (v[a].len == v[cur].len + 1){
36             v[new_node].suff = a;
37         }else{
38             int b = add_node(v[cur].len + 1);
39             v[b].go = v[a].go;
40             v[b].suff = v[a].suff;
41             v[new_node].suff = b;
42             while (cur != -1 && v[cur].go.count(c) != 0 && v[cur].go[c] == a){
43                 v[cur].go[c] = b;
44                 //v[a].sum_in -= v[cur].sum_in;
45                 //v[b].sum_in += v[cur].sum_in;
46                 cur = v[cur].suff;
47             }
48             v[a].suff = b;
49         }
50         return;
51     }
52 }
53 }
54
55 // bad suffix automaton ends
56
57 // aho-corasick begins
58
59 struct vertex {
60     int next[K];
61     bool leaf;
62     int p;
63     char pch;
64     int link;
65     int go[K];
66 };
67
68 vertex t[NMAX+1];
69 int sz;
70
71 void init() {
72     t[0].p = t[0].link = -1;
73     memset(t[0].next, 255, sizeof t[0].next);
74     memset(t[0].go, 255, sizeof t[0].go);
75     sz = 1;
76 }
77
78 void add_string(const string & s) {
79     int v = 0;
80     for (size_t i=0; i<s.length(); ++i) {
81         char c = s[i]-'a';
82         if (t[v].next[c] == -1) {
83             memset(t[sz].next, 255, sizeof t[sz].next);
84             memset(t[sz].go, 255, sizeof t[sz].go);
85             t[sz].link = -1;
86             t[sz].p = v;
87             t[sz].pch = c;
88             t[v].next[c] = sz++;
89         }
90         v = t[v].next[c];
91     }
92     t[v].leaf = true;
93 }
94
95 int go(int v, char c);
96
97 int get_link(int v) {
98     if (t[v].link == -1)
99         if (v == 0 || t[v].p == 0)
100             t[v].link = 0;
101     else
102         t[v].link = go(get_link(t[v].p), t[v].pch);
103     return t[v].link;
104 }
105
106 int go(int v, char c) {
107     if (t[v].go[c] == -1)
108         if (t[v].next[c] != -1)
109             t[v].go[c] = t[v].next[c];
110     else
111         t[v].go[c] = v == 0 ? 0 : go(get_link(v),
112                                     c);
113     return t[v].go[c];
114 }
115
116 // aho-corasick ends
117
118 // pollard begins
119
120 const int max_step = 4e5;
121
122 unsigned long long gcd(unsigned long long a,
123                        unsigned long long b){
124     if (!a) return 1;
125     while (a) swap(a, b%=a);
126     return b;
127 }
128
129 unsigned long long get(unsigned long long a,
130                        unsigned long long b){
131     if (a > b)
132         return a-b;
133     else
134         return b-a;
135 }
136
137 unsigned long long pollard(unsigned long long n){
138     unsigned long long x = (rand() + 1) % n, y = 1,
139                             g;
140     int stage = 2, i = 0;
141     g = gcd(get(x, y), n);
142     while (g == 1) {
143         if (i == max_step)
144             break;
145         if (i == stage) {
146             y = x;
147             stage <= 1;
148         }
149         x = (x * (__int128)x + 1) % n;
150         i++;
151         g = gcd(get(x, y), n);
152     }
153     return g;
154 }

```



```

36 // pollard ends

1 // linear sieve begins
2
3 const int N = 1000000;
4
5 int pr[N + 1], sz = 0;
6 /* minimal prime, mobius function, euler function
   ↪ */
7 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

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 * 111 * 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 * 111 * bin(b, i, p)) % p;
12     if (mp.count(sum) != 0){
13         int an = mp[sum] * 111 * k - i;
14         if (an < p)
15             ans = min(an, ans);
16     }
17 }
18
19 // discrete log in sqrt(p) ends

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

1 // simplex begins
2
3 const double EPS = 1e-9;
4
5 typedef vector<double> vdbl;
6
7 // n variables, m inequalities
8 // Ax <= b, c*x -> max, x >= 0
9 double simplex( int n, int m, const vector<vdbl>
   ↪ &a0, const vdbl &b, const vdbl &c, vdbl &x )
   ↪ {
10     // Ax + Ez = b, A[m]*x -> max
11     // x = 0, z = b, x >= 0, z >= 0
12     vector<vdbl> a(m + 2, vdbl(n + m + 2));
13     vector<int> p(m);
14     forn(i, n)
15         a[m + 1][i] = c[i];
16     forn(i, m) {
17         forn(j, n)
18             a[i][j] = a0[i][j];
19         a[i][n + i] = 1;

```

```

20     a[i][m + n] = -1;
21     a[i][m + n + 1] = b[i];
22     p[i] = n + i;
23 }
24
25 // basis: enter "j", leave "ind+n"
26 auto pivot = [&](int j, int ind) {
27     double coef = a[ind][j];
28     assert(fabs(coef) > EPS);
29     forn(col, n + m + 2)
30         a[ind][col] /= coef;
31     forn(row, m + 2)
32         if (row != ind && fabs(a[row][j]) > EPS) {
33             coef = a[row][j];
34             forn(col, n + m + 2)
35                 a[row][col] -= a[ind][col] * coef;
36             a[row][j] = 0; // reduce precision error
37         }
38     p[ind] = j;
39 };
40
41 // the Simplex itself
42 auto iterate = [&](int nn) {
43     for (int run = 1; run; ) {
44         run = 0;
45         forn(j, nn) {
46             if (a[m][j] > EPS) { // strictly positive
47                 run = 1;
48                 double mi = INFINITY, t;
49                 int ind = -1;
50                 forn(i, m)
51                     if (a[i][j] > EPS && (t = a[i][n + m
52                         ↪ + 1] / a[i][j]) < mi - EPS)
53                         mi = t, ind = i;
54                 if (ind == -1)
55                     return false;
56                 pivot(j, ind);
57             }
58         }
59         return true;
60     };
61
62     int mi = min_element(b.begin(), b.end()) -
63     ↪ b.begin();
64     if (b[mi] < -EPS) {
65         a[m][n + m] = -1;
66         pivot(n + m, mi);
67         assert(iterate(n + m + 1));
68         if (a[m][m + n + 1] > EPS) // optimal value
69             ↪ is positive
70             return NAN;
71         forn(i, m)
72             if (p[i] == m + n) {
73                 int j = 0;
74                 while (find(p.begin(), p.end(), j) !=
75                     ↪ p.end() || fabs(a[i][j]) < EPS)
76                     j++, assert(j < m + n);
77                 pivot(j, i);
78             }
79     }
80 }
81
82 // simplex usage:
83 vdbl x(n);
84 double result = simplex(n, m, a, b, c, x);
85 if (isinf(result))
86     puts("Unbounded");
87 else if (isnan(result))
88     puts("No solution");
89 else {
90     printf("%.9f :", result);
91     forn(i, n)
92         printf(" %.9f", x[i]);
93     puts("");
94 }
95
96 // simplex ends
97 // sum over subsets begins
98 // fast subset convolution O(2^n)
99 for (int i = 0; i < (1<<N); ++i)
100     F[i] = A[i];
101 for (int i = 0; i < N; ++i) for (int mask = 0; mask
102     ↪ < (1<<N); ++mask){
103     if (mask & (1<<i))
104         F[mask] += F[mask^(1<<i)];
105 }
106 // sum over subsets ends
107
108 // algebra begins
109
110 Pick
111  $B + \frac{\Gamma}{2} - 1$ ,
112 где  $B$  - количество целочисленных точек внутри
113     ↪ многоугольника, а  $\Gamma$  - количество
114     ↪ целочисленных точек на границе
115     ↪ многоугольника.
116
117 Newton
118  $x_{i+1} = x_i - f(x_i) / f'(x_i)$ 
119
120 Catalan
121  $C_n = \sum_{limits_{k=0}^{n-1}} C_k C_{n-1-k}$ 
122  $C_i = \frac{1}{n+1} \binom{2n}{n}$ 
123
124  $G_N = 2^{n(n-1)/2}$ 
125 Количество связанных помеченных графов
126  $Conn_N = G_N - \sum_{limits_{K=1}^{N-1}} K \binom{N}{K} Conn_K$ 
127     ↪  $G_{N-K}$ 
128
129 Количество помеченных графов с  $K$  компонентами
130     ↪ связности

```

```

19 D[N][K] = \sum_{limits_{S=1}^N} \binom{N-1}{S-1}
    \rightarrow Conn_S D[N-S][K-1]
20
21 Miller-Rabbin
22 a=a^t
23 FOR i = 1...s
24     if a^2=1 && |a|!=1
25         NOT PRIME
26     a=a^2
27 return a==1 ? PRIME : NOT PRIME
28
29
30 Интегрирование по формуле Симпсона
31 \int_a^b f(x)dx \approx ?
32 x_i := a+ih, i=0 \ldots 2n
33 h = \frac{b-a}{2n}
34
35 \int = (f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) +
    \rightarrow 2f(x_4) + \ldots + 4f(x_{2n-1})) +
    \rightarrow f(x_{2n})) \frac{h}{3}
36 Погрешность имеет порядок уменьшения как O(n^4).
37
38 // algebra ends
39
40 // wavelet tree begins
41
42 struct wavelet_tree{
43     int lo, hi;
44     wavelet_tree *l, *r;
45     vi b;
46
47     //nos are in range [x,y]
48     //array indices are [from, to)
49     wavelet_tree(int *from, int *to, int x, int y){
50         lo = x, hi = y;
51         if(lo == hi or from >= to) return;
52
53         int mid = (lo+hi)/2;
54         auto f = [mid](int x){
55             return x <= mid;
56         };
57         b.reserve(to-from+1);
58         b.pb(0);
59         //b[i] = no of elements from first "i"
60         \rightarrow elements that go to left node
61         for(auto it = from; it != to; it++)
62             b.pb(b.back() + f(*it));
63
64         //see how lambda function is used here
65         auto pivot = stable_partition(from, to, f);
66         l = new wavelet_tree(from, pivot, lo, mid);
67         r = new wavelet_tree(pivot, to, mid+1, hi);
68     }
69
70     //kth smallest element in [l, r]
71     int kth(int l, int r, int k){
72         if(l > r) return 0;
73         if(lo == hi) return lo;
74         //how many nos are there in left node from
75         \rightarrow [l, r]
76         int inleft = b[r] - b[l-1];

```

```

    int lb = b[l-1]; //amt of nos from first
    \rightarrow (l-1) nos that go in left
    int rb = b[r]; //amt of nos from first (r)
    \rightarrow nos that go in left
    //so [lb+1, rb] represents nos from [l, r]
    \rightarrow that go to left
    if(k <= inleft) return this->l->kth(lb+1, rb
    \rightarrow , k);

    // (l-1-lb) is amt of nos from first (l-1) nos
    \rightarrow that go to right
    // (r-rb) is amt of nos from first (r) nos
    \rightarrow that go to right
    //so [l-lb, r-rb] represents nos from [l, r]
    \rightarrow that go to right
    return this->r->kth(l-lb, r-rb, k-inleft);
    }
    };

    // wavelet tree ends

    // berlecamp-massey begins
    1
    2
    3 const int SZ = MAXN;
    4
    5 ll qp(ll a, ll b) {
    6     ll x = 1;
    7     a %= MOD;
    8     while (b) {
    9         if (b & 1) x = x * a % MOD;
    10        a = a * a % MOD;
    11        b >>= 1;
    12    }
    13    return x;
    14 }

    15 namespace linear_seq {
    16     inline vector<int> BM(vector<int> x) {
    17         vector<int> ls, cur;
    18         int lf, ld;
    19         for (int i = 0; i < int(x.size()); ++i) {
    20             ll t = 0;
    21             for (int j = 0; j < int(cur.size()); ++j)
    22                 t = (t + x[i - j - 1] * (ll) cur[j]) %
    23                     MOD;
    24             if ((t - x[i]) % MOD == 0) continue;
    25             if (!cur.size()) {
    26                 cur.resize(i + 1);
    27                 lf = i;
    28                 ld = (t - x[i]) % MOD;
    29                 continue;
    30             }
    31             ll k = -(x[i] - t) * qp(ld, MOD - 2) % MOD;
    32             vector<int> c(i - lf - 1);
    33             c.pb(k);
    34             for (int j = 0; j < int(ls.size()); ++j)
    35                 c.pb(-ls[j] * k % MOD);
    36             if (c.size() < cur.size())
    37                 \rightarrow c.resize(cur.size());
    38             for (int j = 0; j < int(cur.size()); ++j)
    39                 c[j] = (c[j] + cur[j]) % MOD;

```

```

39     if (i - lf + (int) ls.size() >= (int)
40         ↪ cur.size())
41         ls = cur, lf = i, ld = (t - x[i]) % MOD;
42     cur = c;
43 }
44 for (int i = 0; i < int(cur.size()); ++i)
45     cur[i] = (cur[i] % MOD + MOD) % MOD;
46 return cur;
47 }
48
49 int m;
50 ll a[SZ], h[SZ], t_[SZ], s[SZ], t[SZ];
51
52 inline void mull(ll* p, ll* q) {
53     for (int i = 0; i < m + m; ++i) t_[i] = 0;
54     for (int i = 0; i < m; ++i)
55         if (p[i])
56             for (int j = 0; j < m; ++j)
57                 t_[i + j] = (t_[i + j] + p[i] * q[j])
58                     ↪ MOD;
59     for (int i = m + m - 1; i >= m; --i)
60         if (t_[i])
61             for (int j = m - 1; ~j; --j)
62                 t_[i - j - 1] = (t_[i - j - 1] + t_[i]
63                     ↪ * h[j]) % MOD;
64     for (int i = 0; i < m; ++i) p[i] = t_[i];
65 }
66
67 inline ll calc(ll K) {
68     for (int i = m; ~i; --i)
69         s[i] = t[i] = 0;
70     s[0] = 1;
71     if (m != 1) t[1] = 1; else t[0] = h[0];
72     while (K) {
73         if (K & 1) mull(s, t);
74         mull(t, t);
75         K >>= 1;
76     }
77     ll su = 0;
78     for (int i = 0; i < m; ++i) su = (su + s[i]
79         ↪ a[i]) % MOD;
80     return (su % MOD + MOD) % MOD;
81 }
82
83 inline int work(vector<int> x, ll n) {
84     if (n < int(x.size())) return x[n];
85     vector<int> v = BM(x);
86     m = v.size();
87     if (!m) return 0;
88     for (int i = 0; i < m; ++i) h[i] = v[i], a[i]
89         ↪ = x[i];
90     return calc(n);
91 }
92
93 //b=a0/(1-p)
94 inline void calc_generating_function(const
95     ↪ vector<int>& b, vector<int>& p,
96     ↪ vector<int>& a0) {
97     p = BM(b);
98     a0.resize(p.size());
99     for (int i = 0; i < a0.size(); ++i) {
100
101         a0[i] = b[i];
102         for (int j = 0; j < i; ++j) {
103             a0[i] += MOD - (p[j] * 111 * b[i - j -
104                 ↪ 1]) % MOD;
105             if (a0[i] > MOD) {
106                 a0[i] -= MOD;
107             }
108         }
109     }
110 }
111
112 // berlecamp-massey ends
113
114 // AND-FFT begins
115
116 void fast_fourier(vector<int>& a) { // AND-FFT.
117     for (int k = 1; k < SZ(a); k *= 2)
118         for (int start = 0; start < (1 << K); start +=
119             ↪ 2 * k) {
120             for (int off = 0; off < k; ++off) {
121                 int a_val = a[start + off];
122                 int b_val = a[start + k + off];
123
124                 a[start + off] = b_val;
125                 a[start + k + off] = add(a_val, b_val);
126             }
127         }
128 }
129
130 void inverse_fast_fourier(vector<int>& a) {
131     for (int k = 1; k < SZ(a); k *= 2)
132         for (int start = 0; start < (1 << K); start +=
133             ↪ 2 * k) {
134             for (int off = 0; off < k; ++off) {
135                 int a_val = a[start + off];
136                 int b_val = a[start + k + off];
137
138                 a[start + off] = sub(b_val, a_val);
139                 a[start + k + off] = a_val;
140             }
141         }
142 }
143
144 // AND-FFT ends
145
146 // 2-chinese begins
147
148 template <typename Info>
149 class DSU {
150 public:
151     DSU ( int n ) : jump (new int[n]), rank (new
152         ↪ int [n]), info (new Info [n]) {
153         for (int i = 0; i < n; i++) {
154             jump[i] = i;
155             rank[i] = 0;
156         }
157     }
158     Info& operator [] ( int x ) {
159         return info[get (x)];
160     }
161 }

```

```

15 void merge ( int a, int b, const Info &comment72
    ↪ ) {
16     a = get (a);
17     b = get (b);
18     if (rank[a] <= rank[b]) {
19         jump[a] = b;
20         rank[b] += rank[a] == rank[b];
21         info[b] = comment;
22     } else {
23         jump[b] = a;
24         info[a] = comment;
25     }
26 }
27 private:
28 int *jump, *rank;
29 Info *info;
30
31 int get ( int x ) {
32     return jump[x] == x ? x : (jump[x] = get
    ↪ (jump[x]));
33 }
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 )01
    ↪ : value (_value), add (0), source
    ↪ (_source), target (_target) {
44         height = rand ();
45         min_value = value, min_path = 0;
46         left = right = 0;
47     }
48
49     Treap& operator += ( int sub ) {
50         add += sub;
51         return *this;
52     }
53
54     void push () {
55         if (!add)
56             return;
57         if (left) {
58             left->add += add;
59         }
60         if (right) {
61             right->add += add;
62         }
63         value += add;
64         min_value += add;
65         add = 0;
66     }
67
68     void recalc () {
69         min_value = value;
70         min_path = 0;
71
72         if (left && left->min_value + left->add <
    ↪ min_value) {
73             min_value = left->min_value + left->add;
74             min_path = -1;
75         }
76         if (right && right->min_value + right->add <
    ↪ min_value) {
77             min_value = right->min_value + right->add;
78             min_path = +1;
79         }
80     }
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
101 Treap* treap_getmin ( Treap *x, int &source, int
    ↪ &target, int &value ) {
102     assert (x);
103     x->push ();
104     if (x->min_path == 0) {
105         // memory leak, sorry
106         source = x->source;
107         target = x->target;
108         value = x->value + x->add;
109         return treap_merge (x->left, x->right);
110     } else if (x->min_path == -1) {
111         x->left = treap_getmin (x->left, source,
    ↪ target, value);
112         value += x->add;
113         x->recalc ();
114         return x;
115     } else if (x->min_path == +1) {
116         x->right = treap_getmin (x->right, source,
    ↪ target, value);
117         value += x->add;
118         x->recalc ();
119         return x;
120     } else
121         assert (0);
122 }
123
124 Treap* treap_add ( Treap *x, int add ) {
125     if (!x)
126         return 0;
127     return &((*x) += add);

```

```

128 }
129
130
131 int main () {
132     int n, m;
133     while (scanf ("%d%d", &n, &m) == 2) {
134         Treap * g[n + 1];
135         for (int i = 0; i <= n; i++)
136             g[i] = 0;
137         for (int i = 1; i <= n; i++) {
138             int a;
139             assert (scanf ("%d", &a) == 1);
140             g[i] = treap_merge (g[i], new Treap (i, 0,
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
155         int ans = 0, k = n;
156         int jump[2 * n], jump_from[2 * n], parent[2 *
157             ↪ n], c[n];
158         vector <int> children[2 * n];
159         memset (c, 0, sizeof (c[0]) * n);
160         memset (parent, -1, sizeof (parent[0]) * 2 *
161             ↪ n);
162         vector <int> finish;
163         for (int i = 0; i < n; i++) {
164             if (dsu[i].first == 0)
165                 continue;
166             int u = i;
167             c[u] = 1;
168             while (true) {
169                 int source, target, value;
170                 dsu[u].second = treap_getmin (dsu[u].second,
171                 ↪ source, target, value);
172                 if (dsu[target] == dsu[u])
173                     continue;
174                 treap_add (dsu[u].second, -value);
175                 ans += value;
176                 jump_from[dsu[u].first] = source;
177                 jump[dsu[u].first] = target;
178                 if (dsu[target].first == 0)
179                     break;
180                 if (!c[target]) {
181                     c[target] = 1;
182                     u = target;
183                     continue;
184                 }
185             }
186             assert (k < 2 * n);
187             int node = k++, t = target;
188             parent[dsu[u].first] = node;
189             children[node].push_back (dsu[u].first);
190
191             dsu[u].first = node;
192             Treap *v = dsu[u].second;
193             while (dsu[t].first != node) {
194                 int next = jump[dsu[t].first];
195                 parent[dsu[t].first] = node;
196                 children[node].push_back (dsu[t].first);
197                 v = treap_merge (v, dsu[t].second);
198                 dsu.merge (u, t, make_pair (node, v));
199                 t = next;
200             }
201             u = i;
202             while (dsu[u].first) {
203                 int next = jump[dsu[u].first];
204                 finish.push_back (dsu[u].first);
205                 dsu.merge (u, 0, make_pair (0, (Treap *)0));
206                 u = next;
207             }
208         }
209         bool ok[k];
210         int res[n];
211         memset (ok, 0, sizeof (ok[0]) * k);
212         memset (res, -1, sizeof (res[0]) * n);
213         function <void (int, int)> add_edge = [&ok,
214             ↪ &parent, &res, &n] ( int a, int b ) {
215             assert (0 <= a && a < n);
216             assert (0 <= b && b < n);
217             assert (res[a] == -1);
218             res[a] = b;
219             while (a != -1 && !ok[a]) {
220                 ok[a] = true;
221                 a = parent[a];
222             }
223         };
224         function <void (int)> reach = [&ok, &reach,
225             ↪ &children, &jump, &jump_from, &add_edge] (
226             ↪ int u ) {
227             if (!ok[u])
228                 add_edge (jump_from[u], jump[u]);
229             for (auto x : children[u])
230                 reach (x);
231         };
232         for (auto x : finish)
233             reach (x);
234         printf ("%d\n", ans);
235         for (int i = 1; i < n; i++)
236             printf ("%d%c", res[i] ? res[i] : -1, "\n "[i
237                 ↪ < n - 1]);
238         return 0;
239     }
240 }
241
242 // 2-chinese ends
243
244 // general max weight match begins
245
246 #define DIST(e)
247     ↪ (lab[e.u]+lab[e.v]-g[e.u][e.v].w*2)
248 using namespace std;
249 typedef long long ll;
250 const int N = 1023, INF = 1e9;
251 struct Edge {

```

```

8     int u, v, w;
9 } g[N][N];
10 int n, m, n_x, lab[N], match[N], slack[N], st[N];
11     pa[N], flower_from[N][N], S[N], vis[N];
12 vector < int > flower[N];
13 deque < int > q;
14 void update_slack(int u, int x) {
15     if (!slack[x] || DIST(g[u][x]) <
16         DIST(g[slack[x]][x])) slack[x] = u;
17 }
18 void set_slack(int x) {
19     slack[x] = 0;
20     for (int u = 1; u <= n; ++u)
21         if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
22             update_slack(u, x);
23 }
24 void q_push(int x) {
25     if (x <= n) return q.push_back(x);
26     for (int i = 0; i < flower[x].size(); i++)
27         q.push(flower[x][i]);
28 }
29 void set_st(int x, int b) {
30     st[x] = b;
31     if (x <= n) return;
32     for (int i = 0; i < flower[x].size(); ++i)
33         set_st(flower[x][i], b);
34 }
35 int get_pr(int b, int xr) {
36     int pr = find(flower[b].begin(),
37         flower[b].end(), xr) - flower[b].begin();
38     if (pr % 2 == 1) {
39         reverse(flower[b].begin() + 1,
40             flower[b].end());
41         return (int) flower[b].size() - pr;
42     }
43     else return pr;
44 }
45 void set_match(int u, int v) {
46     match[u] = g[u][v].v;
47     if (u <= n) return;
48     Edge e = g[u][v];
49     int xr = flower_from[u][e.u], pr = get_pr(u,
50         xr);
51     for (int i = 0; i < pr; ++i)
52         set_match(flower[u][i], flower[u][i+1]);
53     set_match(xr, v);
54     rotate(flower[u].begin(), flower[u].begin()
55         + pr, flower[u].end());
56 }
57 void augment(int u, int v) {
58     int xnv = st[match[u]];
59     set_match(u, v);
60     if (!xnv) return;
61     set_match(xnv, st[pa[xnv]]);
62     augment(st[pa[xnv]], xnv);
63 }
64 int get_lca(int u, int v) {
65     static int t = 0;
66     for (++t; u || v; swap(u, v)) {
67         if (u == 0) continue;
68         if (vis[u] == t) return u;
69     }
70     vis[u] = t;
71     u = st[match[u]];
72     if (u) u = st[pa[u]];
73     return 0;
74 }
75 void add_blossom(int u, int lca, int v) {
76     int b = n+1;
77     while (b <= n_x && st[b] != 0) ++b;
78     if (b > n_x) ++n_x;
79     lab[b] = 0, S[b] = 0;
80     match[b] = match[lca];
81     flower[b].clear();
82     flower[b].push_back(lca);
83     for (int x = u, y; x != lca; x = st[pa[y]])
84         flower[b].push_back(x), flower[b].push_back(y)
85         = st[match[x]], q_push(y);
86     reverse(flower[b].begin() + 1, flower[b].end());
87     for (int x = v, y; x != lca; x = st[pa[y]])
88         flower[b].push_back(x), flower[b].push_back(y)
89         = st[match[x]], q_push(y);
90     set_st(b, b);
91     for (int x = 1; x <= n_x; ++x) g[b][x].w =
92         g[x][b].w = 0;
93     for (int x = 1; x <= n; ++x) flower_from[b][x]
94         = 0;
95     for (int i = 0; i < flower[b].size(); ++i) {
96         int xs = flower[b][i];
97         for (int x = 1; x <= n_x; ++x)
98             if (g[b][x].w == 0 || DIST(g[xs][x]) <
99                 DIST(g[b][x]))
100                 g[b][x] = g[xs][x], g[x][b] = g[x][xs];
101         for (int x = 1; x <= n; ++x)
102             if (flower_from[xs][x]) flower_from[b][x] =
103                 xs;
104     }
105     set_slack(b);
106 }
107 void expand_blossom(int b) // S[b] == 1 {
108     for (int i = 0; i < flower[b].size(); ++i)
109         set_st(flower[b][i], flower[b][i]);
110     int xr = flower_from[b][g[b][pa[b]].u], pr =
111         get_pr(b, xr);
112     for (int i = 0; i < pr; i += 2) {
113         int xs = flower[b][i], xns = flower[b][i+1];
114         pa[xs] = g[xns][xs].u;
115         S[xs] = 1, S[xns] = 0;
116         slack[xs] = 0, set_slack(xns);
117         q_push(xns);
118     }
119     S[xr] = 1, pa[xr] = pa[b];
120     for (int i = pr+1; i < flower[b].size(); ++i) {
121         int xs = flower[b][i];
122         S[xs] = -1, set_slack(xs);
123     }
124     st[b] = 0;
125 }
126 bool on_found_Edge(const Edge &e) {
127     int u = st[e.u], v = st[e.v];
128     if (S[v] == -1) {
129         pa[v] = e.u, S[v] = 1;

```



```

113     int nu = st[match[v]];
114     slack[v] = slack[nu] = 0;
115     S[nu] = 0, q_push(nu);
116 }
117 else if (S[v] == 0) {
118     int lca = get_lca(u, v);
119     if (!lca) return augment(u, v), augment(v,
120         ↪ u), 1;
121     else add_blossom(u, lca, v);
122 }
123 return 0;
124 }
125 bool matching() {
126     fill(S, S+n_x+1, -1), fill(slack, slack+n_x+1,
127         ↪ 0);
128     q.clear();
129     for (int x = 1; x <= n_x; ++x)
130         if (st[x] == x && !match[x]) pa[x] = 0, S[x]
131             ↪ = 0, q_push(x);
132     if (q.empty()) return 0;
133     for (;;) {
134         while (q.size()) {
135             int u = q.front();
136             q.pop_front();
137             if (S[st[u]] == 1) continue;
138             for (int v = 1; v <= n; ++v)
139                 if (g[u][v].w>0 && st[u] != st[v]) {
140                     if (DIST(g[u][v]) == 0) {
141                         if (on_found_Edge(g[u][v])) return
142                             ↪ 1;
143                     }
144                     else update_slack(u, st[v]);
145                 }
146             }
147             int d = INF;
148             for (int b = n+1; b <= n_x; ++b)
149                 if (st[b] == b && S[b] == 1) d = min(d,
150                     ↪ lab[b]/2);
151             for (int x = 1; x <= n_x; ++x)
152                 if (st[x] == x && slack[x]) {
153                     if (S[x] == -1) d = min(d,
154                         ↪ DIST(g[slack[x]][x]));
155                     else if (S[x] == 0) d = min(d,
156                         ↪ DIST(g[slack[x]][x])/2);
157                 }
158             for (int u = 1; u <= n; ++u) {
159                 if (S[st[u]] == 0) {
160                     if (lab[u] <= d) return 0;
161                     lab[u] -= d;
162                 }
163                 else if (S[st[u]] == 1) lab[u] += d;
164             }
165             for (int b = n+1; b <= n_x; ++b)
166                 if (st[b] == b) {
167                     if (S[st[b]] == 0) lab[b] += d*2;
168                     else if (S[st[b]] == 1) lab[b] -= d*2;
169                 }
170             q.clear();
171             for (int x = 1; x <= n_x; ++x)
172                 if (st[x] == x && slack[x] && st[slack[x]]
173                     ↪ != x && DIST(g[slack[x]][x]) == 0)
174                     if (on_found_Edge(g[slack[x]][x])) return
175                         ↪ 1;
176             for (int b = n+1; b <= n_x; ++b)
177                 if (st[b] == b && S[b] == 1 && lab[b] == 0)
178                     ↪ expand_blossom(b);
179             }
180         }
181     pair < ll, int> weight_blossom() {
182         fill(match, match+n+1, 0);
183         n_x = n;
184         int n_matches = 0;
185         ll tot_weight = 0;
186         for (int u = 0; u <= n; ++u) st[u] = u,
187             ↪ flower[u].clear();
188         int w_max = 0;
189         for (int u = 1; u <= n; ++u)
190             for (int v = 1; v <= n; ++v) {
191                 flower_from[u][v] = (u == v?u:0);
192                 w_max = max(w_max, g[u][v].w);
193             }
194         for (int u = 1; u <= n; ++u) lab[u] = w_max;
195         while (matching()) ++n_matches;
196         for (int u = 1; u <= n; ++u)
197             if (match[u] && match[u] < u)
198                 tot_weight += g[u][match[u]].w;
199         return make_pair(tot_weight, n_matches);
200     }
201     int main() {
202         cin>>n>>m;
203         for (int u = 1; u <= n; ++u)
204             for (int v = 1; v <= n; ++v)
205                 g[u][v] = Edge {u, v, 0};
206         for (int i = 0, u, v, w; i < m; ++i) {
207             cin>>u>>v>>w;
208             g[u][v].w = g[v][u].w = w;
209         }
210         cout << weight_blossom().first << '\n';
211         for (int u = 1; u <= n; ++u) cout << match[u]
212             ↪ << ' ';
213     }
214     // general max weighet match ends
215     // fast hashtable begins
216     #include <ext/pb_ds/assoc_container.hpp>
217     using namespace __gnu_pbds;
218     gp_hash_table<int, int> table;
219     const int RANDOM = chrono ::
220         ↪ high_resolution_clock ::
221         ↪ now().time_since_epoch().count();
222     struct chash {
223         int operator()(int x) { return hash<int>{}(x ^
224             ↪ RANDOM); }
225     };
226     gp_hash_table<key, int, chash> table;
227     // 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  $f_j!$