Pollard

Team Reference

```cpp
// pollard begins

const int max_step = 4e5;

unsigned long long gcd(unsigned long long a, unsigned long long b){
    if (!a) return 1;
    while (a) swap(a, b%=a);
    return b;
}

unsigned long long get(unsigned long long a, unsigned long long b){
    if (a > b)
        return a-b;
    else
        return b-a;
}

unsigned long long pollard(unsigned long long n){
    unsigned long long x = (rand() + 1) % n, y = 1, g;
    int stage = 2, i = 0;
    g = gcd(get(x, y), n);
    while (g == 1) {
        if (i == max_step)
            break;
        if (i == stage) {
            y = x;
            stage <<= 1;
        }
        x = (x * (__int128)x + 1) % n;
        i++;
        g = gcd(get(x, y), n);
    }
    return g;
}

// pollard ends
```

pragma

```
#pragma GCC optimize(''O3,no-stack-protector'')
#pragma GCC target(''sse,sse2,sse4,ssse3,popcnt,abm,mmx,avx,tune=native'')
```

Алгебра Pick

В + Г / 2 − 1 = AREA,
где В — количество целочисленных точек внутри многоугольника, а Г — количество целочисленных точек на границе многоугольника.

Newton

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

Catalan

$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$
$C_i = \frac{1}{n+1}\binom{2n}{n}$

Кол-во графов

$G_N := 2^{n(n-1)/2}$
Количество связных помеченных графов
$Conn_N = G_N - \frac{1}{N}\sum_{K=1}^{N-1} K\binom{N}{K} Conn_K G_{N-K}$

Количество помеченных графов с K компонентами связности
$D[N][K] = \sum_{S=1}^{N} \binom{N-1}{S-1} Conn_S D[N-S][K-1]$

Miller-Rabbin

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

Интегрирование по формуле Симпсона

$\int_a^b f(x)dx$?
$x_i := a + ih, i = 0 \dots 2n$
$h = \frac{b-a}{2n}$

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

Простые числа

1009,1013;10007,10009;100003,100019
1000003,1000033;10000019,10000079
100000007,100000037
10000000019,10000000033
1000000000039,1000000000061
100000000000031,100000000000067
10000000000000061,10000000000000069
100000000000000003,1000000000000000009

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

Erdős–Gallai theorem

A sequence of non-negative integers $d_1 \geq \cdots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + \cdots + d_n$ $d_1 + \cdots + d_n$ 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 $1 \leq k \leq n$.

```cpp
// sk fast allocation begins
const int MAX_MEM = 5e8;
int mpos = 0;
char mem[MAX_MEM];
inline void * operator new ( size_t n ) {
    assert((mpos += n) <= MAX_MEM);
    return (void *)(mem + mpos - n);
}
inline void operator delete ( void * ) noexcept { } // must have!

// sk fast allocation ends



// sk fast read-write begins

inline int readChar();
template <class T = int> inline T readInt();
template <class T> inline void writeInt( T x, char end = 0 );
inline void writeChar( int x );
inline void writeWord( const char *s );

/** Read */

static const int buf_size = 2048;

inline int getChar() {
    static char buf[buf_size];
    static int len = 0, pos = 0;
    if (pos == len)
        pos = 0, len = fread(buf, 1, buf_size, stdin);
    if (pos == len)
        return -1;
    return buf[pos++];
}

inline int readWord(char * buffer) {
    int c = getChar();
    while (c <= 32) {
        c = getChar();
    }

    int len = 0;
    while (c > 32) {
        *buffer = (char) c;
        c = getChar();
        buffer++;
        len++;
    }
    return len;
}

inline int readChar() {
    int c = getChar();
    while (c <= 32)
        c = getChar();
    return c;
}

template <class T>
inline T readInt() {
    int s = 1, c = readChar();
    T x = 0;
    if (c == '-')
        s = -1, c = getChar();
    while ('0' <= c && c <= '9')
        x = x * 10 + c - '0', c = getChar();
    return s == 1 ? x : -x;
}

/** Write */

static int write_pos = 0;
static char write_buf[buf_size];

inline void writeChar( int x ) {
    if (write_pos == buf_size)
        fwrite(write_buf, 1, buf_size, stdout), write_pos = 0;
    write_buf[write_pos++] = x;
}

template <class T>
inline void writeInt( T x, char end ) {
    if (x < 0)
        writeChar('-'), x = -x;

    char s[24];
    int n = 0;
    while (x || !n)
        s[n++] = '0' + x % 10, x /= 10;
    while (n--)
        writeChar(s[n]);
    if (end)
        writeChar(end);
}

inline void writeWord( const char *s ) {
    while (*s)
        writeChar(*s++);
}

struct Flusher {
    ~Flusher() {
        if (write_pos)
            fwrite(write_buf, 1, write_pos, stdout), write_pos = 0;
    }
} flusher;


// sk fast read-write ends
```

```cpp
// extended euclid begins

int gcd (int a, int b, int & x, int & y) {
        if (a == 0) {
                x = 0; y = 1;
                return b;
```

```
        }
        int x1, y1;
        int d = gcd (b%a, a, x1, y1);
        x = y1 - (b / a) * x1;
        y = x1;
        return d;
}

// extended euclid ends

// FFT begins

const int LOG = 19;
const int N = (1 << LOG);

typedef std::complex<double> cd;

int rev[N];
cd W[N];

void precalc() {
    const double pi = std::acos(-1);
    for (int i = 0; i != N; ++i)
        W[i] = cd(std::cos(2 * pi * i / N),
        ↪  std::sin(2 * pi * i / N));

    int last = 0;
    for (int i = 1; i != N; ++i) {
        if (i == (2 << last))
            ++last;

        rev[i] = rev[i ^ (1 << last)] | (1 <<
        ↪  (LOG - 1 - last));
    }
}

void fft(vector<cd>& a) {
    for (int i = 0; i != N; ++i)
        if (i < rev[i])
            std::swap(a[i], a[rev[i]]);

    for (int lvl = 0; lvl != LOG; ++lvl)
        for (int start = 0; start != N; start +=
        ↪  (2 << lvl))
            for (int pos = 0; pos != (1 << lvl);
            ↪  ++pos) {
                cd x = a[start + pos];
                cd y = a[start + pos + (1 <<
                ↪  lvl)];

                y *= W[pos << (LOG - 1 - lvl)];

                a[start + pos] = x + y;
                a[start + pos + (1 << lvl)] = x -
                ↪  y;
            }
}

void inv_fft(vector<cd>& a) {
    fft(a);
    std::reverse(a.begin() + 1, a.end());

    for (cd& elem: a)
        elem /= N;
}

// FFT ends

// fast gauss begins

using elem_t = int;
// a[i][rows[i][j].first]=rows[i][j].second;
↪  b[i]=a[i][n]
bool gauss(vector<vector<pair<int, elem_t>>>
↪  rows, vector<elem_t> &res) {
    int n = rows.size();

    res.resize(n + 1, 0);
    vector<int> p(n + 1);
    iota(p.begin(), p.end(), 0);
    vector<int> toZero(n + 1, -1);
    vector<int> zro(n + 1);
    vector<elem_t> a(n + 1);

    // optional: sort rows

    sort(p.begin(), p.begin() + n, [&rows](int i,
    ↪  int j) { return rows[i].size() <
    ↪  rows[j].size(); });
    vector<int> invP(n + 1);
    vector<vector<pair<int, elem_t>>> rs(n);
    for (int i = 0; i < n; i++) {
        invP[p[i]] = i;
        rs[i] = rows[p[i]];
    }
    for (int i = 0; i < n; i++) {
        rows[i] = rs[i];
        for (auto& el: rows[i]) {
            if (el.first < n) {
                el.first = invP[el.first];
            }
        }
    }


    for (int i = 0; i < n; i++) {
        for (auto& el: rows[i]) {
            a[el.first] = el.second;
        }
        while (true) {
            int k = -1;
            for (auto& el: rows[i]) {
                if (!isZero(a[el.first]) &&
                ↪  toZero[el.first] != -1 &&
                    (k == -1 || toZero[el.first]
                    ↪  < toZero[k])) {
                    k = el.first;
                }
            }
            if (k == -1)
                break;

            int j = toZero[k];
            elem_t c = a[k];
```

```
            for (auto el: rows[j]) {
                if (isZero(a[el.first]))
                    rows[i].emplace_back(el.first, 0);
                a[el.first] = sub(a[el.first], mult(c, el.second));
            }

            auto cond = [&a](const pair<int, elem_t>& p) { return isZero(a[p.first]); };

            rows[i].erase(std::remove_if(rows[i].begin(), rows[i].end(), cond), rows[i].end());
        }

        bool ok = false;
        for (auto& el: rows[i]) {
            if (el.first < n && !isZero(a[el.first])) {
                toZero[el.first] = i;
                zro[i] = el.first;
//                det = (det * a[el.first]) % MOD;

                elem_t c = divM(1, a[el.first]);
                for (auto& el: rows[i]) {
                    el.second = mult(a[el.first], c);
                    a[el.first] = 0;
                }

                ok = true;
                break;
            }
        }

        if (!ok) {
//            det = 0;
            return false;
        }
    }

    res[n] = sub(0, 1);
    for (int i = n - 1; i >= 0; i--) {
        int k = zro[i];
        for (auto& el : rows[i])
            if (el.first != k)
                res[p[k]] = sub(res[p[k]], mult(el.second, res[p[el.first]]));
    }

    return true;
}

// fast gauss ends
```

```
// simple geometry begins

struct Point {
    double x, y;
    Point operator+(const Point& p) const { return {x + p.x, y + p.y}; }
    Point operator-(const Point& p) const { return {x - p.x, y - p.y}; }
    Point operator*(const double d) const { return {x * d, y * d}; }
    Point rotate() const { return {y, -x}; }
    double operator*(const Point& p) const { return x * p.x + y * p.y; }
    double operator^(const Point& p) const { return x * p.y - y * p.x; }
    double dist() const { return sqrt(x * x + y * y); }
};

struct Line {
    double a, b, c;
    Line(const Point& p1, const Point& p2) {
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = - a * p1.x - b * p1.y;

        double d = sqrt(sqr(a) + sqr(b));
        a /= d, b /= d, c /= d;
    }
    bool operator||(const Line& l) const { return fabs(a * l.b - l.a * b) < EPS; }
    double dist(const Point& p) const { return fabs(a * p.x + b * p.y + c); }
    Point operator^(const Line& l) const {
        return {(l.c * b - c * l.b) / (a * l.b - l.a * b),
                (l.c * a - c * l.a) / (l.a * b - a * l.b)};
    }
    Point projection(const Point& p) const {
        return p - Point{a, b} * (a * p.x + b * p.y + c);
    }
};

struct Circle {
    Point c;
    double r;
    Circle(const Point& c, double r) : c(c), r(r) {}
    Circle(const Point& a, const Point& b, const Point& c) {
        Point p1 = (a + b) * 0.5, p2 = (a + c) * 0.5;
        Point q1 = p1 + (b - a).rotate(), q2 = p2 + (c - a).rotate();
        this->c = Line(p1, q1) ^ Line(p2, q2);
        r = (a - this->c).dist();
    }
};
```

```cpp
inline bool on_segment(const Point& p1, const Point& p2, const Point& x, bool strictly) {
    if (fabs((p1 - x) ^ (p2 - x)) > EPS)
        return false;
    return (p1 - x) * (p2 - x) < (strictly ? -EPS : EPS);
}

// in case intersection is not a segment
inline bool intersect_segments(const Point& p1, const Point& p2, const Point& q1, const Point& q2, Point& x) {
    Line l1(p1, p2), l2(q1, q2);
    if (l1 || l2) return false;
    x = l1 ^ l2;
    return on_segment(p1, p2, x, false);
}

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

inline bool intersect_circle_and_line(const Circle& c, const Line& l, Point& p1, Point& p2) {
    double d = l.dist(c.c);
    if (d > c.r + EPS)
        return false;
    Point p = l.projection(c.c);
    Point n{l.b, -l.a};
    double h = sqrt(sqr(c.r) - sqr(l.dist(c.c)));
    p1 = p + n * h, p2 = p - n * h;
    return true;
}

// simple geometry ends
```

```cpp
// convex hull begins

struct Point {
    int x, y;
    Point operator-(const Point& p) const { return {x - p.x, y - p.y}; }
    int64_t operator^(const Point& p) const { return x * 1ll * p.y - y * 1ll * p.x; }
    int64_t dist() const { return x * 1ll * x + y * 1ll * y; }
    bool operator<(const Point& p) const { return x != p.x ? x < p.x : y < p.y; }
};

// all point on convex hull are included
vector<Point> convex_hull(vector<Point> pt) {
    int n = pt.size();
    Point p0 = *std::min_element(pt.begin(), pt.end());
    std::sort(pt.begin(), pt.end(), [&p0](const Point& a, const Point& b) {
        int64_t cp = (a - p0) ^ (b - p0);
        return cp != 0 ? cp > 0 : (a - p0).dist() < (b - p0).dist();
    });

    int i = n - 1;
    for (; i > 0 && ((pt[i] - p0) ^ (pt[i - 1] - p0)) == 0; i--);
    std::reverse(pt.begin() + i, pt.end());

    vector<Point> ch;
    for (auto& p : pt) {
        while (ch.size() > 1) {
            auto& p1 = ch[(int) ch.size() - 1];
            auto& p2 = ch[(int) ch.size() - 2];
            int64_t cp = (p1 - p2) ^ (p - p1);
            if (cp >= 0) break;
            ch.pop_back();
        }
        ch.push_back(p);
    }

    return ch;
}

// convex hull ends
```

```cpp
// convex hull trick begins

typedef long long ftype;
typedef complex<ftype> point;
#define x real
#define y imag

ftype dot(point const& a, point const& b) {
    return (conj(a) * b).x();
}

ftype f(point const& a, int x) {
    return dot(a, {compressed[x], 1});
    //return dot(a, {x, 1});
}

int pos = 0;

// (x, y) -> (k, b) -> kb + x
struct li_chao { // for min
    vector<point> line;

    li_chao(int maxn) {
        line.resize(4 * maxn, {0, inf});
```

```
    }

    void add_line(int v, int l, int r, int a, int b, point nw) {
        if (r <= a || b <= l) return; // remove if no [a, b) query

        int m = (l + r) >> 1;

        if (!(a <= l && r <= b)) { // remove if no [a, b) query
            add_line(v + v + 1, l, m, a, b, nw);
            add_line(v + v + 2, m, r, a, b, nw);
            return;
        }

        bool lef = f(nw, l) < f(line[v], l);
        bool mid = f(nw, m) < f(line[v], m);

        if (mid) swap(line[v], nw);

        if (l == r - 1)
            return;

        if (lef != mid)
            add_line(v + v + 1, l, m, a, b, nw);
        else
            add_line(v + v + 2, m, r, a, b, nw);
    }

    ftype get(int v, int l, int r, int x) {
        if(l == r - 1)
            return f(line[v], x);
        int m = (l + r) / 2;
        if(x < m) {
            return min(f(line[v], x), get(v + v + 1, l, m, x));
        } else {
            return min(f(line[v], x), get(v + v + 2, m, r, x));
        }
    }

} cdt(maxn);

// convex hull with stack

ftype cross(point a, point b) {
    return (conj(a) * b).y();
}

vector<point> hull, vecs;

void add_line(ftype k, ftype b) {
    point nw = {k, b};
    while(!vecs.empty() && dot(vecs.back(), nw - hull.back()) < 0) {
        hull.pop_back();
        vecs.pop_back();
    }
    if(!hull.empty()) {
        vecs.push_back(1i * (nw - hull.back()));
    }
    hull.push_back(nw);
}

int get(ftype x) {
    point query = {x, 1};
    auto it = lower_bound(vecs.begin(), vecs.end(), query, [](point a, point b) {
        return cross(a, b) > 0;
    });
    return dot(query, hull[it - vecs.begin()]);
}

// convex hull trick ends

// heavy-light begins

int sz[maxn];

void dfs_sz(int v, int par = -1) {
    sz[v] = 1;
    for (int x : gr[v])
        if (x != par) {
            dfs_sz(x, v);
            sz[v] += sz[x];
        }
    for (int i = 0; i < gr[v].size(); i++)
        if (gr[v][i] != par)
        if (sz[gr[v][i]] * 2 >= sz[v]) {
            swap(gr[v][i], gr[v][0]);
            break;
        }
}

int rev[maxn];
int t_in[maxn];
int upper[maxn];;
int par[maxn];
int dep[maxn];

int T = 0;

void dfs_build(int v, int uppr, int pr = -1) {
    rev[T] = v;
    t_in[v] = T++;
    dep[v] = pr == -1 ? 0 : dep[pr] + 1;
    par[v] = pr;
    upper[v] = uppr;

    bool first = true;

    for (int x : gr[v])
        if (x != pr) {
            dfs_build(x, first ? upper[v] : x, v);
            first = false;
        }
}

struct interval {
    int l;
```

```
    int r;
    bool inv; // should direction be reversed
};

// node-weighted hld
vector<interval> get_path(int a, int b) {
    vector<interval> front;
    vector<interval> back;

    while (upper[a] != upper[b]) {
        if (dep[upper[a]] > dep[upper[b]]) {
            front.push_back({t_in[upper[a]], t_in[a], true});
            a = par[upper[a]];
        } else {
            back.push_back({t_in[upper[b]], t_in[b], false});
            b = par[upper[b]];
        }
    }

    front.push_back({min(t_in[a], t_in[b]), max(t_in[a], t_in[b]), t_in[a] > t_in[b]});
    // for edge-weighted hld add: "front.back().l++;"
    front.insert(front.end(), back.rbegin(), back.rend());

    return front;
}

// heavy-light ends
```

```
// max flow begins

struct edge{
    int from, to;
    int c, f, num;
    edge(int from, int to, int c, int num):from(from), to(to), c(c), f(0), num(num){}
    edge(){}
};

const int max_n =  600;

edge eds[150000];
int num = 0;
int it[max_n];
vector<int> gr[max_n];
int s, t;
vector<int> d(max_n);

bool bfs(int k) {
    queue<int> q;
    q.push(s);
    fill(d.begin(), d.end(), -1);
    d[s] = 0;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int x : gr[v]) {
            int to = eds[x].to;
            if (d[to] == -1 && eds[x].c - eds[x].f >= (1 << k)){
                d[to] = d[v] + 1;
                q.push(to);
            }
        }
    }

    return (d[t] != -1);
}

int dfs(int v, int flow, int k) {
    if (flow < (1 << k))
        return 0;
    if (v == t)
        return flow;
    for (; it[v] < gr[v].size(); it[v]++) {
        int num = gr[v][it[v]];
        if (d[v] + 1 != d[num].to])
            continue;
        int res = dfs(eds[num].to, min(flow, eds[num].c - eds[num].f), k);
        if (res){
            eds[num].f += res;
            eds[num ^ 1].f -= res;
            return res;
        }
    }
    return 0;
}

void add(int fr, int to, int c, int nm) {
    gr[fr].push_back(num);
    eds[num++] = edge(fr, to, c, nm);
    gr[to].push_back(num);
    eds[num++] = edge(to, fr, 0, nm); //corrected c
}

int ans = 0;
    for (int k = 30; k >= 0; k--)
        while (bfs(k)) {
            memset(it, 0, sizeof(it));
            while (int res = dfs(s, 1e9 + 500, k))
                ans += res;
        }


// decomposition

int path_num = 0;
vector<int> paths[max_n];
int flows[max_n];

int decomp(int v, int flow) {
    if (flow < 1)
        return 0;
    if (v == t) {
```

```
        path_num++;
        flows[path_num - 1] = flow;
        return flow;
    }
    for (int i = 0; i < gr[v].size(); i++) {
        int num = gr[v][i];
        int res = decomp(eds[num].to, min(flow, eds[num].f));
        if (res)  {
            eds[num].f -= res;
            paths[path_num - 1].push_back(eds[num].num);
            return res;
        }
    }
    return 0;
}

while (decomp(s, 1e9 + 5));

// max flow ends
```

```
// min-cost flow begins

long long ans = 0;
int mx = 2 * n + 2;

memset(upd, 0, sizeof(upd));
for (int i = 0; i < mx; i++)
    dist[i] = inf;
dist[st] = 0;
queue<int> q;
q.push(st);
upd[st] = 1;
while (!q.empty()){
    int v = q.front();
    q.pop();
    if (upd[v]){
        for (int x : gr[v])        {
            edge &e = edges[x];
            if (e.c - e.f > 0 && dist[v] != inf && dist[e.to] > dist[v] + e.w) {
                dist[e.to] = dist[v] + e.w;
                if (!upd[e.to])
                    q.push(e.to);
                upd[e.to] = true;
                p[e.to] = x;
            }
        }
        upd[v] = false;
    }
}

for (int i = 0; i < k; i++){
    for (int i = 0; i < mx; i++)
        d[i] = inf;
    d[st] = 0;
    memset(used, false, sizeof(used));
    set<pair<int, int> > s;
    s.insert(make_pair(0, st));
    for (int i = 0; i < mx; i++){
        int x;
        while (!s.empty() && used[(s.begin() -> second)]){
            s.erase(s.begin());
        }
        if (s.empty())
            break;
        x = s.begin() -> second;
        used[x] = true;
        s.erase(s.begin());
        for (int i = 0; i < gr[x].size(); i++){
            edge &e = edges[gr[x][i]];
            if (!used[e.to] && e.c - e.f > 0){
                if (d[e.to] > d[x] + (e.c - e.f) * e.w + dist[x] - dist[e.to]){
                    d[e.to] = d[x] + (e.c - e.f) * e.w + dist[x] - dist[e.to];
                    p[e.to] = gr[x][i];
                    s.insert(make_pair(d[e.to], e.to));
                }
            }
        }
        dist[x] += d[x];
    }
    int pos = t;
    while (pos != st){
            int id = p[pos];
            edges[id].f += 1;
        edges[id ^ 1].f -= 1;
        pos = edges[id].from;
    }
}

// min-cost flow ends
```

```
// bad hungarian begins

fill(par, par + 301, -1);
fill(par2, par2 + 301, -1);

int ans = 0;
for (int v = 0; v < n; v++){
    memset(useda, false, sizeof(useda));
    memset(usedb, false, sizeof(usedb));
    useda[v] = true;
    for (int i = 0; i < n; i++)
        w[i] = make_pair(a[v][i] + row[v] + col[i], v);
    memset(prev, 0, sizeof(prev));
    int pos;
    while (true){
        pair<pair<int, int>, int> p = make_pair(make_pair(1e9, 1e9), 1e9);
        for (int i = 0; i < n; i++)
            if (!usedb[i])
                p = min(p, make_pair(w[i], i));
        for (int i = 0; i < n; i++)
            if (!useda[i])
                row[i] += p.first.first;
        for (int i = 0; i < n; i++)
```

```
            if (!usedb[i]){
                col[i] -= p.first.first;
                w[i].first -= p.first.first;
            }
        ans += p.first.first;
        usedb[p.second] = true;
        prev[p.second] = p.first.second; //из второй в первую
        int x = par[p.second];
        if (x == -1){
            pos = p.second;
            break;
        }
        useda[x] = true;
        for (int j = 0; j < n; j++)
            w[j] = min(w[j], {a[x][j] + row[x] - col[j], x});

    }
    while (pos != -1){
        int nxt = par2[prev[pos]];
        par[pos] = prev[pos];
        par2[prev[pos]] = pos;
        pos = nxt;
    }
}
cout << ans << "\n";
for (int i = 0; i < n; i++)
    cout << par[i] + 1 << " " << i + 1 << "\n";

// bad hungarian ends
```

```
// Edmonds O(n^3) begins

vector<int> gr[MAXN];
int match[MAXN], p[MAXN], base[MAXN], q[MAXN];
bool used[MAXN], blossom[MAXN];
int mark[MAXN];
int C = 1;

int lca(int a, int b) {
    C++;
    for (;;) {
        a = base[a];
        mark[a] = C;
        if (match[a] == -1) break;
        a = p[match[a]];
    }

    for (;;) {
        b = base[b];
        if (mark[b] == C) return b;
        b = p[match[b]];
    }
}

void mark_path(int v, int b, int children) {
    while (base[v] != b) {
        blossom[base[v]] = blossom[base[match[v]]] = true;
        p[v] = children;
        children = match[v];
        v = p[match[v]];
    }
}

int find_path(int root) {
    memset(used, 0, sizeof(used));
    memset(p, -1, sizeof p);
    for (int i = 0; i < N; i++)
        base[i] = i;

    used[root] = true;
    int qh = 0, qt = 0;
    q[qt++] = root;
    while (qh < qt) {
        int v = q[qh++];
        for (int to : gr[v]) {
            if (base[v] == base[to] || match[v] == to) continue;
            if (to == root || match[to] != -1 && p[match[to]] != -1) {
                int curbase = lca(v, to);
                memset(blossom, 0, sizeof(blossom));
                mark_path(v, curbase, to);
                mark_path(to, curbase, v);
                for (int i = 0; i < N; i++)
                    if (blossom[base[i]]) {
                        base[i] = curbase;
                        if (!used[i]) {
                            used[i] = true;
                            q[qt++] = i;
                        }
                    }
            } else if (p[to] == -1) {
                p[to] = v;
                if (match[to] == -1)
                    return to;
                to = match[to];
                used[to] = true;
                q[qt++] = to;
            }
        }
    }

    return -1;
}

memset(match, -1, sizeof match);
    for (int i = 0; i < N; i++) {
        if (match[i] == -1 && !gr[i].empty()) {
            int v = find_path(i);
            while (v != -1) {
                int pv = p[v], ppv = match[pv];
                match[v] = pv; match[pv] = v;
                v = ppv;
            }
        }
    }

// Edmonds O(n^3) ends
```

```
// string basis begins
```

```

vector<int> getZ(string s){
    vector<int> z;
    z.resize(s.size(), 0);
    int l = 0, r = 0;
    for (int i = 1; i < s.size(); i++){
        if (i <= r)
            z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < s.size() && s[z[i]] ==
        ↪  s[i + z[i]])
            z[i]++;
        if (i + z[i] - 1 > r){
            r = i + z[i] - 1;
            l = i;
        }
    }
    return z;
}

vector<int> getP(string s){
    vector<int> p;
    p.resize(s.size(), 0);
    int k = 0;
    for (int i = 1; i < s.size(); i++){
        while (k > 0 && s[i] == s[k])
            k = p[k - 1];
        if (s[i] == s[k])
            k++;
        p[i] = k;
    }
    return p;
}

vector<int> getH(string s){
    vector<int> h;
    h.resize(s.size() + 1, 0);
    for (int i = 0; i < s.size(); i++)
        h[i + 1] = ((h[i] * 1ll * pow) + s[i] -
        ↪  'a' + 1) % mod;
    return h;
}

int getHash(vector<int> &h, int l, int r){
    int res = (h[r + 1] - h[l] * p[r - l + 1]) %
    ↪  mod;
    if (res < 0)
        res += mod;
    return res;
}

// string basis ends

// min cyclic shift begins

string min_cyclic_shift (string s) {
        s += s;
        int n = (int) s.length();
        int i=0, ans=0;
        while (i < n/2) {
                ans = i;
                int j=i+1, k=i;
                while (j < n && s[k] <= s[j]) {
                        if (s[k] < s[j])
                                k = i;
                        else
                                ++k;
                        ++j;
                }
                while (i <= k)  i += j - k;
        }
        return s.substr (ans, n/2);
}

// min cyclic shift ends

// suffix array O(n) begins

typedef vector<char> bits;

template<const int end>
void getBuckets(int *s, int *bkt, int n, int K) {
    fill(bkt, bkt + K + 1, 0);
    forn(i, n) bkt[s[i] + !end]++;
    forn(i, K) bkt[i + 1] += bkt[i];
}
void induceSAl(bits &t, int *SA, int *s, int
↪  *bkt, int n, int K) {
    getBuckets<0>(s, bkt, n, K);
    forn(i, n) {
        int j = SA[i] - 1;
        if (j >= 0 && !t[j])
            SA[bkt[s[j]]++] = j;
    }
}
void induceSAs(bits &t, int *SA, int *s, int
↪  *bkt, int n, int K) {
    getBuckets<1>(s, bkt, n, K);
    for (int i = n - 1; i >= 0; i--) {
        int j = SA[i] - 1;
        if (j >= 0 && t[j])
            SA[--bkt[s[j]]] = j;
    }
}

void SA_IS(int *s, int *SA, int n, int K) { //
↪  require last symbol is 0
#define isLMS(i) (i && t[i] && !t[i-1])
    int i, j;
    bits t(n);
    t[n-1] = 1;
    for (i = n - 3; i >= 0; i--)
        t[i] = (s[i]<s[i+1] || (s[i]==s[i+1] &&
        ↪  t[i+1]==1));
    int bkt[K + 1];
    getBuckets<1>(s, bkt, n, K);
    fill(SA, SA + n, -1);
    forn(i, n)
        if (isLMS(i))
            SA[--bkt[s[i]]] = i;
    induceSAl(t, SA, s, bkt, n, K);
    induceSAs(t, SA, s, bkt, n, K);
    int n1 = 0;
    forn(i, n)
        if (isLMS(SA[i]))
```

```
            SA[n1++] = SA[i];
    fill(SA + n1, SA + n, -1);
    int name = 0, prev = -1;
    forn(i, n1) {
        int pos = SA[i];
        bool diff = false;
        for (int d = 0; d < n; d++)
            if (prev == -1 || s[pos+d] !=
            ↪ s[prev+d] || t[pos+d] !=
            ↪ t[prev+d])
                diff = true, d = n;
            else if (d > 0 && (isLMS(pos+d) ||
            ↪ isLMS(prev+d)))
                d = n;
        if (diff)
            name++, prev = pos;
        SA[n1 + (pos >> 1)] = name - 1;
    }
    for (i = n - 1, j = n - 1; i >= n1; i--)
        if (SA[i] >= 0)
            SA[j--] = SA[i];
    int *s1 = SA + n - n1;
    if (name < n1)
        SA_IS(s1, SA, n1, name - 1);
    else
        forn(i, n1)
            SA[s1[i]] = i;
    getBuckets<1>(s, bkt, n, K);
    for (i = 1, j = 0; i < n; i++)
        if (isLMS(i))
            s1[j++] = i;
    forn(i, n1)
        SA[i] = s1[SA[i]];
    fill(SA + n1, SA + n, -1);
    for (i = n1 - 1; i >= 0; i--) {
        j = SA[i], SA[i] = -1;
        SA[--bkt[s[j]]] = j;
    }
    induceSA1(t, SA, s, bkt, n, K);
    induceSAs(t, SA, s, bkt, n, K);
}
// suffix array O(n) ends

// suffix array O(n log n) begins
string str;
int N, m, SA [MAX_N], LCP [MAX_N];
int x [MAX_N], y [MAX_N], w [MAX_N], c [MAX_N];

inline bool cmp (const int a, const int b, const
↪ int l) { return (y [a] == y [b] && y [a + l]
↪ == y [b + l]); }

void Sort () {
    for (int i = 0; i < m; ++i) w[i] = 0;
    for (int i = 0; i < N; ++i) ++w[x[y[i]]];
    for (int i = 0; i < m - 1; ++i) w[i + 1] +=
    ↪ w[i];
    for (int i = N - 1; i >= 0; --i)
    ↪ SA[--w[x[y[i]]]] = y[i];
}

void DA () {
    for (int i = 0; i < N; ++i) x[i] = str[i],
    ↪ y[i] = i;
    Sort ();
    for (int i, j = 1, p = 1; p < N; j <<= 1, m =
    ↪ p) {
        for (p = 0, i = N - j; i < N; i++) y[p++]
        ↪ = i;
        for (int k = 0; k < N; ++k) if (SA[k] >=
        ↪ j) y[p++] = SA[k] - j;
        Sort();
        for (swap (x, y), p = 1, x[SA[0]] = 0, i
        ↪ = 1; i < N; ++i) x[SA [i]] = cmp
        ↪ (SA[i - 1], SA[i], j) ? p - 1 : p++;
    }
}

// common for all algorithms
void kasaiLCP () {
    for (int i = 0; i < N; i++) c[SA[i]] = i;
    for (int i = 0, j, k = 0; i < N; LCP [c[i++]]
    ↪ = k)
        if (c [i] > 0) for (k ? k-- : 0, j =
        ↪ SA[c[i] - 1]; str[i + k] == str[j +
        ↪ k]; k++);
        else k = 0;
}

void suffixArray () { // require last symbol is
↪ char(0)
    m = 256;
    N = str.size();
    DA ();
    kasaiLCP ();
}
// suffix array O(n log n) ends

// bad suffix automaton begins

struct node{
    map<char, int> go;
    int len, suff;
    long long sum_in;
    node(){}
};

node v[max_n * 4];

int add_node(int max_len){
    //v[number].sum_in = 0;
    v[number].len = max_len;
    v[number].suff = -1;
    number++;
    return number - 1;
}

int last = add_node(0);

void add_char(char c) {
    int cur = last;
    int new_node = add_node(v[cur].len + 1);
    last = new_node;
```

```
    while (cur != -1){
        if (v[cur].go.count(c) == 0){
            v[cur].go[c] = new_node;
            //v[new_node].sum_in +=
            ↪ v[cur].sum_in;
            cur = v[cur].suff;
            if (cur == -1)
                v[new_node].suff = 0;
        }else{
            int a = v[cur].go[c];
            if (v[a].len == v[cur].len + 1){
                v[new_node].suff = a;
            }else{
                int b = add_node(v[cur].len + 1);
                v[b].go = v[a].go;
                v[b].suff = v[a].suff;
                v[new_node].suff = b;
                while (cur != -1 &&
                ↪ v[cur].go.count(c) != 0 &&
                ↪ v[cur].go[c] == a){
                    v[cur].go[c] = b;
                    //v[a].sum_in -=
                    ↪ v[cur].sum_in;
                    //v[b].sum_in +=
                    ↪ v[cur].sum_in;
                    cur = v[cur].suff;
                }
                v[a].suff = b;
            }
            return;
        }
    }
}

// bad suffix automaton ends
```

```
// pollard begins

const int max_step = 4e5;

unsigned long long gcd(unsigned long long a,
↪ unsigned long long b){
    if (!a) return 1;
    while (a) swap(a, b%=a);
    return b;
}

unsigned long long get(unsigned long long a,
↪ unsigned long long b){
    if (a > b)
        return a-b;
    else
        return b-a;
}

unsigned long long pollard(unsigned long long n){
    unsigned long long x = (rand() + 1) % n, y =
    ↪ 1, g;
    int stage = 2, i = 0;
    g = gcd(get(x, y), n);
    while (g == 1) {
        if (i == max_step)
            break;
        if (i == stage) {
            y = x;
            stage <<= 1;
        }
        x = (x * (__int128)x + 1) % n;
        i++;
        g = gcd(get(x, y), n);
    }
    return g;
}

// pollard ends
```

```
// linear sieve begins

const int N = 1000000;

int pr[N + 1], sz = 0;
/* minimal prime, mobius function, euler function
↪ */
int lp[N + 1], mu[N + 1], phi[N + 1];

lp[1] = mu[1] = phi[1] = 1;
for (int i = 2; i <= N; ++i) {
        if (lp[i] == 0)
                lp[i] = pr[sz++] = i;
        for (int j = 0; j < sz && pr[j] <= lp[i]
        ↪ && i * pr[j] <= N; ++j)
                lp[i * pr[j]] = pr[j];

    mu[i] = lp[i] == lp[i / lp[i]] ? 0 : -1 *
    ↪ mu[i / lp[i]];
    phi[i] = phi[i / lp[i]] * (lp[i] == lp[i /
    ↪ lp[i]] ? lp[i] : lp[i] - 1);
}

// linear sieve ends
```

```
// discrete log in sqrt(p) begins

int k = sqrt((double)p) + 2;

for (int i = k; i >= 1; i--)
    mp[bin(b, (i * 1ll * k) % (p-1), p)] = i;

bool answered = false;
int ans = INT32_MAX;
for (int i = 0; i <= k; i++){
    int sum = (n * 1ll * bin(b, i, p)) % p;
    if (mp.count(sum) != 0){
        int an = mp[sum] * 1ll * k - i;
        if (an < p)
            ans = min(an, ans);
    }
}

// discrete log in sqrt(p) ends
```

```
// prime roots mod n begins

int num = 0;
```

```
long long phi = n, nn = n;
for (long long x:primes){
    if (x*x>nn)
        break;
    if (nn % x == 0){
        while (nn % x == 0)
            nn /= x;
        phi -= phi/x;
        num++;
    }
}
if (nn != 1){
    phi -= phi/nn;
    num++;
}
if (!((num == 1 && n % 2 != 0) || n == 4 || n == 2 || (num == 2 && n % 2 == 0 && n % 4 != 0))){
    cout << ''-1\n'';
    continue;
}
vector<long long> v;
long long pp = phi;
for (long long x:primes){
    if (x*x>pp)
        break;
    if (pp % x == 0){
        while (pp % x == 0)
            pp /= x;
        v.push_back(x);
    }
}
if (pp != 1){
    v.push_back(pp);
}
while (true){
    long long a = primes[rand()%5000]%n;
    if (gcd(a, n) != 1)
        continue;
    bool bb = false;
    for (long long x:v)
        if (pow(a, phi/x) == 1){
            bb = true;
            break;
        }
    if (!bb){
        cout << a << ''\n'';
        break;
    }
}

// prime roots mod n ends
```

```
// simplex begins

const double EPS = 1e-9;

typedef vector<double> vdbl;

// n variables, m inequalities
// Ax <= b, c*x -> max, x >= 0
double simplex( int n, int m, const vector<vdbl> &a0, const vdbl &b, const vdbl &c, vdbl &x ) {
        // Ax + Ez = b, A[m]*x -> max
        // x = 0, z = b, x >= 0, z >= 0
        vector<vdbl> a(m + 2, vdbl(n + m + 2));
        vector<int> p(m);
        forn(i, n)
                a[m + 1][i] = c[i];
        forn(i, m) {
                forn(j, n)
                        a[i][j] = a0[i][j];
                a[i][n + i] = 1;
                a[i][m + n] = -1;
                a[i][m + n + 1] = b[i];
                p[i] = n + i;
        }

        // basis: enter ”j”, leave ”ind+n”
        auto pivot = [&]( int j, int ind ) {
                double coef = a[ind][j];
                assert(fabs(coef) > EPS);
                forn(col, n + m + 2)
                        a[ind][col] /= coef;
                forn(row, m + 2)
                        if (row != ind &&
                        fabs(a[row][j]) >
                        EPS) {
                                coef = a[row][j];
                                forn(col, n + m +
                                2)
                                        a[row][col]
                                        -=
                                        a[ind][col]
                                        *
                                        coef;
                                a[row][j] = 0; //
                                reduce
                                precision
                                error
                        }
                p[ind] = j;
        };

        // the Simplex itself
        auto iterate = [&]( int nn ) {
                for (int run = 1; run; ) {
                        run = 0;
                        forn(j, nn) {
                                if (a[m][j] >
                                EPS) { //
                                strictly
                                positive
                                        run = 1;
                                        double mi
                                        =
                                        INFINITY,
                                        t;
                                        int ind =
                                        -1;
```

```
forn(i, m)
if (a[i][j] > EPS && (t = a[i][n + m + 1]) < 0)
if (ind == -1)
pivot(j, ind);
return false;
int mi = min_element(b.begin(), b.end()) - b.begin();
if (b[mi] < -EPS) {
a[m][n + m] = -1;
pivot(n + m, mi);
assert(iterate(n + m + 1));
if (a[m][m + n + 1] > EPS) // optimal value is positive
return NAN;
forn(i, m)
if (p[i] == m + n) {
int j = 0;
while (find(p.begin(), p.end(), j) != p.end() || fabs(a[i][j]) < EPS)
j++, assert(j < m + n);
pivot(j, i);
}
if (a[i][j] > 0)
swap(a[m], a[m + 1]);
if (!iterate(n + m))
return INFINITY;
x = vdbl(n, 0);
forn(i, m)
if (p[i] < n)
x[p[i]] = a[i][n + m + 1];
return -a[m][n + m + 1];
}
// simplex usage:
vdbl x(n);
double result = simplex(n, m, a, b, c, x);
if(isinf(result))
puts("Unbounded");
else if (isnan(result))
puts("No solution");
else {
printf("%.9f :", result);
forn(i, n)
printf(" %.9f", x[i]);
puts("");
}

// simplex ends
return
// sum over subsets begins
// fast subset convolution O(n 2^n)
for(int i = 0; i < (1<<N); ++i)
F[i] = A[i];
for(int i = 0; i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
if(mask & (1<<i))
F[mask] += F[mask^(1<<i)];
}
// sum over subsets ends

// algebra begins

Pick
B + Г / 2 - 1,
где B - количество целочисленных точек внутри многоугольника, а Г - количество целочисленных точек на границе многоугольника.

Newton
x_{i+1}=x_i-f(x_i)/f'(x_i)

Catalan
C_n=\sum\limits_{k=0}^{n-1} C_{k} C_{n-1-k}
C_i=\frac 1 {n + 1} \binom {2n} {n}

G_N:=2^{n(n-1)/2}
Количество связных помеченных графов
Conn_N = G_N - \frac 1 N \sum\limits_{K=1}^{N-1} K \binom N K Conn_K G_{N-K}
```

```
Количество помеченных графов с K компонентами связности
D[N][K]=\sum\limits_{S=1}^N \binom {N-1} {S-1} Conn_S D[N-S][K-1]

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


Интегрирование по формуле Симпсона
\int_a^b f(x)dx - ?
x_i := a+ih, i=0\ldots 2n
h = \frac {b-a} {2n}

\int = (f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+2f(x_4)+\ldots+4f(x_{2n-1})+f(x_{2n}))\frac h 3
Погрешность имеет порядок уменьшения как O(n^4).

// algebra ends
```

```
// wavelet tree begins

struct wavelet_tree{
        int lo, hi;
        wavelet_tree *l, *r;
        vi b;

        //nos are in range [x,y]
        //array indices are [from, to)
        wavelet_tree(int *from, int *to, int x, int y){
                lo = x, hi = y;
                if(lo == hi or from >= to) return;

                int mid = (lo+hi)/2;
                auto f = [mid](int x){
                        return x <= mid;
                };
                b.reserve(to-from+1);
                b.pb(0);
                //b[i] = no of elements from first "i" elements that go to left node
                for(auto it = from; it != to; it++)
                        b.pb(b.back() + f(*it));

                //see how lambda function is used here
                auto pivot = stable_partition(from, to, f);
                l = new wavelet_tree(from, pivot, lo, mid);
                r = new wavelet_tree(pivot, to, mid+1, hi);
        }

        //kth smallest element in [l, r]
        int kth(int l, int r, int k){
                if(l > r) return 0;
                if(lo == hi) return lo;
                //how many nos are there in left node from [l, r]
                int inleft = b[r] - b[l-1];
                int lb = b[l-1]; //amt of nos from first (l-1) nos that go in left
                int rb = b[r]; //amt of nos from first (r) nos that go in left
                //so [lb+1, rb] represents nos from [l, r] that go to left
                if(k <= inleft) return this->l->kth(lb+1, rb , k);
                //(l-1-lb) is amt of nos from first (l-1) nos that go to right
                //(r-rb) is amt of nos from first (r) nos that go to right
                //so [l-lb, r-rb] represents nos from [l, r] that go to right
                return this->r->kth(l-lb, r-rb, k-inleft);
        }
};

// wavelet tree ends
```

```
// berlecamp-massey begins

const int SZ = MAXN;

ll qp(ll a, ll b) {
    ll x = 1;
    a %= MOD;
    while (b) {
        if (b & 1) x = x * a % MOD;
        a = a * a % MOD;
        b >>= 1;
    }
    return x;
}

namespace linear_seq {
    inline vector<int> BM(vector<int> x) {
        vector<int> ls, cur;
        int lf, ld;
        for (int i = 0; i < int(x.size()); ++i) {
            ll t = 0;
            for (int j = 0; j < int(cur.size()); ++j)
                t = (t + x[i - j - 1] * (ll) cur[j]) % MOD;
```

```
            if ((t - x[i]) % MOD == 0) continue;
            if (!cur.size()) {
                cur.resize(i + 1);
                lf = i;
                ld = (t - x[i]) % MOD;
                continue;
            }
            ll k = -(x[i] - t) * qp(ld, MOD - 2)
            ↪ % MOD;
            vector<int> c(i - lf - 1);
            c.pb(k);
            for (int j = 0; j < int(ls.size());
            ↪ ++j)
                c.pb(-ls[j] * k % MOD);
            if (c.size() < cur.size())
            ↪ c.resize(cur.size());
            for (int j = 0; j < int(cur.size());
            ↪ ++j)
                c[j] = (c[j] + cur[j]) % MOD;
            if (i - lf + (int) ls.size() >= (int)
            ↪ cur.size())
                ls = cur, lf = i, ld = (t - x[i])
                ↪ % MOD;
            cur = c;
        }
        for (int i = 0; i < int(cur.size()); ++i)
            cur[i] = (cur[i] % MOD + MOD) % MOD;
        return cur;
    }

    int m;
    ll a[SZ], h[SZ], t_[SZ], s[SZ], t[SZ];

    inline void mull(ll* p, ll* q) {
        for (int i = 0; i < m + m; ++i) t_[i] =
        ↪ 0;
        for (int i = 0; i < m; ++i)
            if (p[i])
                for (int j = 0; j < m; ++j)
                    t_[i + j] = (t_[i + j] + p[i]
                    ↪ * q[j]) % MOD;
        for (int i = m + m - 1; i >= m; --i)
            if (t_[i])
                for (int j = m - 1; ~j; --j)
                    t_[i - j - 1] = (t_[i - j -
                    ↪ 1] + t_[i] * h[j]) % MOD;
        for (int i = 0; i < m; ++i) p[i] = t_[i];
    }

    inline ll calc(ll K) {
        for (int i = m; ~i; --i)
            s[i] = t[i] = 0;
        s[0] = 1;
        if (m != 1) t[1] = 1; else t[0] = h[0];
        while (K) {
            if (K & 1) mull(s, t);
            mull(t, t);
            K >>= 1;
        }
        ll su = 0;
        for (int i = 0; i < m; ++i) su = (su +
        ↪ s[i] * a[i]) % MOD;
        return (su % MOD + MOD) % MOD;
    }

    inline int work(vector<int> x, ll n) {
        if (n < int(x.size())) return x[n];
        vector<int> v = BM(x);
        m = v.size();
        if (!m) return 0;
        for (int i = 0; i < m; ++i) h[i] = v[i],
        ↪ a[i] = x[i];
        return calc(n);
    }

    //b=a0/(1-p)
    inline void calc_generating_function(const
    ↪ vector<int>& b, vector<int>& p,
    ↪ vector<int>& a0) {
        p = BM(b);
        a0.resize(p.size());
        for (int i = 0; i < a0.size(); ++i) {
            a0[i] = b[i];
            for (int j = 0; j < i; ++j) {
                a0[i] += MOD - (p[j] * 1ll * b[i
                ↪ - j - 1]) % MOD;
                if (a0[i] > MOD) {
                    a0[i] -= MOD;
                }
            }
        }
    }
}

// berlecamp-massey ends


// AND-FFT begins

void fast_fourier(vector<int>& a) { // AND-FFT.
    for (int k = 1; k < SZ(a); k *= 2)
        for (int start = 0; start < (1 << K);
        ↪ start += 2 * k) {
            for (int off = 0; off < k; ++off) {
                int a_val = a[start + off];
                int b_val = a[start + k + off];

                a[start + off] = b_val;
                a[start + k + off] = add(a_val,
                ↪ b_val);
            }
        }
}

void inverse_fast_fourier(vector<int>& a) {
    for (int k = 1; k < SZ(a); k *= 2)
        for (int start = 0; start < (1 << K);
        ↪ start += 2 * k) {
            for (int off = 0; off < k; ++off) {
                int a_val = a[start + off];
                int b_val = a[start + k + off];

```

```
                a[start + off] = sub(b_val, a_val);
                a[start + k + off] = a_val;
            }
        }
}

// AND-FFT ends

// 2-chinese begins

template <typename Info>
class DSU {
  public:
    DSU ( int n ) : jump (new int[n]), rank (new int [n]), info (new Info [n]) {
      for (int i = 0; i < n; i++) {
        jump[i] = i;
        rank[i] = 0;
      }
    }
    Info& operator [] ( int x ) {
      return info[get (x)];
    }
    void merge ( int a, int b, const Info &comment ) {
      a = get (a);
      b = get (b);
      if (rank[a] <= rank[b]) {
        jump[a] = b;
        rank[b] += rank[a] == rank[b];
        info[b] = comment;
      } else {
        jump[b] = a;
        info[a] = comment;
      }
    }
  private:
    int *jump, *rank;
    Info *info;

    int get ( int x ) {
      return jump[x] == x ? x : (jump[x] = get (jump[x]));
    }
};


struct Treap {
  int value, add;
  int source, target, height;
  int min_value, min_path;

  Treap *left, *right;

  Treap ( int _source, int _target, int _value ) : value (_value), add (0), source (_source), target (_target) {
    height = rand ();
    min_value = value, min_path = 0;
    left = right = 0;
  }

  Treap& operator += ( int sub ) {
    add += sub;
    return *this;
  }

  void push () {
    if (!add)
      return;
    if (left) {
      left->add += add;
    }
    if (right) {
      right->add += add;
    }
    value += add;
    min_value += add;
    add = 0;
  }

  void recalc () {
    min_value = value;
    min_path = 0;
    if (left && left->min_value + left->add < min_value) {
      min_value = left->min_value + left->add;
      min_path = -1;
    }
    if (right && right->min_value + right->add < min_value) {
      min_value = right->min_value + right->add;
      min_path = +1;
    }
  }
};

Treap* treap_merge ( Treap *x, Treap *y ) {
  if (!x)
    return y;
  if (!y)
    return x;
  if (x->height < y->height) {
    x->push ();
    x->right = treap_merge (x->right, y);
    x->recalc ();
    return x;
  } else {
    y->push ();
    y->left = treap_merge (x, y->left);
    y->recalc ();
    return y;
  }
}

Treap* treap_getmin ( Treap *x, int &source, int &target, int &value ) {
  assert (x);
  x->push ();
  if (x->min_path == 0) {
    // memory leak, sorry
    source = x->source;
```

```
    target = x->target;
    value = x->value + x->add;
    return treap_merge (x->left, x->right);
  } else if (x->min_path == -1) {
    x->left = treap_getmin (x->left, source,
    ↪  target, value);
    value += x->add;
    x->recalc ();
    return x;
  } else if (x->min_path == +1) {
    x->right = treap_getmin (x->right, source,
    ↪  target, value);
    value += x->add;
    x->recalc ();
    return x;
  } else
    assert (0);
}

Treap* treap_add ( Treap *x, int add ) {
  if (!x)
    return 0;
  return &((*x) += add);
}


int main () {
  int n, m;
  while (scanf (''%d%d'', &n, &m) == 2) {
    Treap * g[n + 1];
    for (int i = 0; i <= n; i++)
      g[i] = 0;
    for (int i = 1; i <= n; i++) {
      int a;
      assert (scanf (''%d'', &a) == 1);
      g[i] = treap_merge (g[i], new Treap (i, 0,
      ↪  a));
    }
    n++;
    for (int i = 0; i < m; i++) {
      int a, b, c;
      assert (scanf (''%d%d%d'', &a, &b, &c) == 3);
      g[b] = treap_merge (g[b], new Treap (b, a,
      ↪  c));
    }
    DSU <pair <int, Treap*> > dsu (n + 1);
    for (int i = 0; i < n; i++) {
      dsu[i] = make_pair (i, g[i]);
    }

    int ans = 0, k = n;
    int jump[2 * n], jump_from[2 * n], parent[2 *
    ↪  n], c[n];
    vector <int> children[2 * n];
    memset (c, 0, sizeof (c[0]) * n);
    memset (parent, -1, sizeof (parent[0]) * 2 *
    ↪  n);
    vector <int> finish;
    for (int i = 0; i < n; i++) {
      if (dsu[i].first == 0)
        continue;
      int u = i;
      c[u] = 1;
      while (true) {
        int source, target, value;
        dsu[u].second = treap_getmin
        ↪  (dsu[u].second, source, target,
        ↪  value);
        if (dsu[target] == dsu[u])
          continue;
        treap_add (dsu[u].second, -value);
        ans += value;
        jump_from[dsu[u].first] = source;
        jump[dsu[u].first] = target;
        if (dsu[target].first == 0)
          break;
        if (!c[target]) {
          c[target] = 1;
          u = target;
          continue;
        }
        assert (k < 2 * n);
        int node = k++, t = target;
        parent[dsu[u].first] = node;
        children[node].push_back (dsu[u].first);
        dsu[u].first = node;
        Treap *v = dsu[u].second;
        while (dsu[t].first != node) {
          int next = jump[dsu[t].first];
          parent[dsu[t].first] = node;
          children[node].push_back
          ↪  (dsu[t].first);
          v = treap_merge (v, dsu[t].second);
          dsu.merge (u, t, make_pair (node, v));
          t = next;
        }
      }
      u = i;
      while (dsu[u].first) {
        int next = jump[dsu[u].first];
        finish.push_back (dsu[u].first);
        dsu.merge (u, 0, make_pair (0, (Treap
        ↪  *)0));
        u = next;
      }
    }
    bool ok[k];
    int res[n];
    memset (ok, 0, sizeof (ok[0]) * k);
    memset (res, -1, sizeof (res[0]) * n);
    function <void (int, int)> add_edge = [&ok,
    ↪  &parent, &res, &n] ( int a, int b ) {
      assert (0 <= a && a < n);
      assert (0 <= b && b < n);
      assert (res[a] == -1);
      res[a] = b;
      while (a != -1 && !ok[a]) {
        ok[a] = true;
        a = parent[a];
      }
    };
```

```
    function <void (int)> reach = [&ok, &reach, &children, &jump, &jump_from, &add_edge](int u ) {
      if (!ok[u])
        add_edge (jump_from[u], jump[u]);
      for (auto x : children[u])
        reach (x);
    };
    for (auto x : finish)
       reach (x);
    printf ("%d\n", ans);
    for (int i = 1; i < n; i++)
      printf ("%d%c", res[i] ? res[i] : -1, "\n"[i < n - 1]);
  }
  return 0;
}

// 2-chinese ends

// slow min circulation begins

struct Edge {
    int a;
    int b;
    int cost;
};

vector<int> negative_cycle(int n, vector<Edge> &edges) {
    // O(nm), return ids of edges in negative cycle

    vector<int> d(n);
    vector<int> p(n, -1); // last edge ids

    const int inf = 1e9;

    int x = -1;
    for (int i = 0; i < n; i++) {
        x = -1;
        for (int j = 0; j < edges.size(); j++) {
            Edge &e = edges[j];

            if (d[e.b] > d[e.a] + e.cost) {
                d[e.b] = max(-inf, d[e.a] + e.cost);
                p[e.b] = j;
                x = e.b;
            }
        }
    }

    if (x == -1)
        return vector<int>(); // no negative cycle

    for (int i = 0; i < n; i++)
        x = edges[p[x]].a;

    vector<int> result;
    for (int cur = x; ; cur = edges[p[cur]].a) {
        if (cur == x && result.size() > 0) break;
        result.push_back(p[cur]);
    }
    reverse(result.begin(), result.end());

    return result;
}

vector<int> min_avg_cycle(int n, vector<Edge> &edges) {
    const int inf = 1e3;

    for (auto &e : edges)
        e.cost *= n * n;

    int l = -inf;
    int r = inf;
    while (l + 1 < r) {
        int m = (l + r) / 2;
        for (auto &e : edges)
            e.cost -= m;

        if (negative_cycle(n, edges).empty())
            l = m;
        else
            r = m;

        for (auto &e : edges)
            e.cost += m;
    }

    if (r >= 0) // if only negative needed
        return vector<int>();

    for (auto &e : edges)
        e.cost -= r;

    vector<int> result = negative_cycle(n, edges);


    for (auto &e : edges)
        e.cost += r;

    for (auto &e : edges)
        e.cost /= n * n;

    return result;
}

struct edge {
    int from, to;
    int c, f, cost;
};

const int max_n = 200;

vector<int> gr[max_n];
vector<edge> edges;

void add(int fr, int to, int c, int cost) {
```

```
    gr[fr].push_back(edges.size());
    edges.push_back({fr, to, c, 0, cost});
    gr[to].push_back(edges.size());
    edges.push_back({to, fr, 0, 0, -cost}); // single
}

void calc_min_circulation(int n) {
    while (true) {
        vector<Edge> eds;
        vector<int> origin;

        for (int i = 0; i < edges.size(); i++) {
            edge &e = edges[i];
            if (e.c - e.f > 0) {
                eds.push_back({e.from, e.to, e.cost});
                origin.push_back(i);
            }
        }

        vector<int> cycle = negative_cycle(n, eds);

        if (cycle.empty())
            break;

        for (auto id : cycle) {
            int x = origin[id];
            edges[x].f += 1;
            edges[x ^ 1].f -= 1;
        }
    }
}

// slow min circulation ends
```

DM

Кол-во корневых деревьев:
$t(G) = \frac{1}{n}\lambda_2 \dots \lambda_n \ (\lambda_1 = 0)$

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

```
// fast hashtable begins

#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
gp_hash_table<int, int> table;

const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash {
    int operator()(int x) { return hash<int>{}(x ^ RANDOM); }
};
gp_hash_table<key, int, chash> table;

// fast hashtable ends
```

Наличие совершенного паросочетания:
T – матрица с нулями на диагонали. Если есть ребро (i, j), то $a_{i,j} := x_{i,j}$, $a_{j,i} = -x_{i,j}$
$\det(T) = 0 \Leftrightarrow$ нет совершенного паросочетания.

Whitespace code FFT