Team Reference

Pollard

```cpp
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;
}
```

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

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

Кол-во графов

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

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

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

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

Простые числа

1009,1013;10007,10009;100003,100019
1000003,1000033;10000019,10000079
100000007,100000037
10000000019,10000000033
1000000000039,1000000000061

100000000000031,100000000000067
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: $998244353 = 7{\cdot}17{\cdot}2^{23} + 1; w : 3^{7*17}$.

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
/** Begin fast allocation */
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!

/** End fast allocation */


#define pb push_back
#define mp make_pair
#define fst first
#define snd second
#define ll long long
#define forn(i, n) for (int i = 0; i < (int) (n); i++)
#define forlr(i, l, r) for (int i = (int) l; i <= (int) (r); i++)
#define forrl(i, r, l) for (int i = (int) r; i >= (int) (l); i--)

/** Interface */

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;
```

```cpp
    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;
```

```
struct line {
    ll k, b;
    ll at(ll x) const {
        return k * x + b;
    }
};
double intersec(line a, line b) {
    return 1.0 * (b.b - a.b) / (a.k - b.k);
}
struct convex_hull_trick {
    vector<double> x = {-1e18};
    vector<line> lines;
    void add(line l) {
        // l.k increasing
        if (lines.empty()) {
            lines.pb(l);
        } else {
            while (lines.size() > 1 && intersec(l, lines[lines.size() - 2]) <
            ↪ x.back()) {
                lines.pop_back();
                x.pop_back();
            }
            x.push_back(intersec(l, lines.back()));
            lines.push_back(l);
        }
    }
};
struct cht_forward_iterator {
    int ci = 0;
    ll promote(const convex_hull_trick& cht, ll value) {
        while (ci < (int)cht.x.size() - 1 && cht.x[ci + 1] < value)
            ci++;
        return cht.lines[ci].at(value);
    }
};
const int g = 275;
struct sqrt_dec {
    struct block {
        vector<int> bs;
        convex_hull_trick cht;
        cht_forward_iterator it;
        block(vector<int> b) {
            bs = b;
            int k = b.size();
            forn(i, k) {
                cht.add({ b[i], 1ll * b[i] * (k - i) });
            }
        }
        ll value(ll x) {
            if (cht.lines.empty())
                return 0;
```

```
                return it.promote(cht, x);
            }
        };
        void insert(block& bl, int b) {
            auto v = std::move(bl.bs);
            v.insert(lower_bound(all(v), b), b);
            bl = block(std::move(v));
        }
        vector<block> blocks;
        sqrt_dec(int n) {
            blocks.resize((n + g - 1) / g, block({}));
        }
        void add(int i, int b) {
            insert(blocks[i / g], b);
        }
        ll get_max() {
            int sm = 0;
            ll ans = 0;
            for (int i = blocks.size() - 1; i >= 0; i--) {
                ans = max(ans, blocks[i].value(sm));
                sm += blocks[i].bs.size();
            }
            return ans;
        }
};
```

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

treap v[max_n * 4];
int last = 0;

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

void add_char(char c){
    int cur = last;
    int new_treap = add_treap(v[cur].len + 1);
    last = new_treap;
    while (cur != -1){
        if (v[cur].go.count(c) == 0){
            v[cur].go[c] = new_treap;
            v[new_treap].sum_in += v[cur].sum_in;
            cur = v[cur].suff;
            if (cur == -1)
                v[new_treap].suff = 0;
        }else{
            int a = v[cur].go[c];
            if (v[a].len == v[cur].len + 1){
                v[new_treap].suff = a;
            }else{
                int b = add_treap(v[cur].len + 1);
                v[b].go = v[a].go;
                v[b].suff = v[a].suff;
                v[new_treap].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;
        }
    }
}
```

```
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);
    }
}
```

```
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;
}
```

linear sieve

```
const int N = 10000000;
int lp[N+1];
vector<int> pr;

for (int i=2; i<=N; ++i) {
	if (lp[i] == 0) {
		lp[i] = i;
		pr.push_back (i);
	}
	for (int j=0; j<(int)pr.size() && pr[j]<=lp[i] && i*pr[j]<=N; ++j)
		lp[i * pr[j]] = pr[j];
}
```

kasai

```
//p[i] -- prefix (id of first symbol) on i-th position of suff array (from 0)
for (int i = 0; i < n; i++)
    r[p[i]] = i;
int k = 0;
for (int j = 0; j < n; j++){
    int i = r[j];
    k--;
    if (k < 0 || i == n - 1)
        k = 0;
    if (i != n - 1)
        while (s[p[i] + k] == s[p[i + 1] + k])
            k++;
    lcp[i] = k;
}
for (int i = 0; i + 1 < n; i++)
    cout << " ";
```

```
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";
```

```cpp
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])
            if (d[eds[x].to] == -1 && eds[x].c - eds[x].f >= (1 << k)){
                d[eds[x].to] = d[v] + 1;
                q.push(eds[x].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]++) {
        if (d[v] + 1 != d[eds[gr[v][it[v]]].to])
            continue;
        int num = gr[v][it[v]];
        int res = dfs(eds[gr[v][it[v]]].to, min(flow, eds[gr[v][it[v]]].c -
         ↪ eds[gr[v][it[v]]].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[550];
int flows[550];

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));
```

```
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;
    }
}
```

```
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);
}
```

Sum over subsets

```
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)];
}
```

```
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;
}
```

suf array + lcp

```
char *s; // входная строка
int n; // длина строки
const int maxlen = ...; // максимальная длина строки
const int alphabet = 256; // размер алфавита, <= maxlen
//
int p[maxlen], cnt[maxlen], c[maxlen];
memset (cnt, 0, alphabet * sizeof(int));
for (int i=0; i<n; ++i)
	++cnt[s[i]];
for (int i=1; i<alphabet; ++i)
	cnt[i] += cnt[i-1];
for (int i=0; i<n; ++i)
	p[--cnt[s[i]]] = i;
c[p[0]] = 0;
int classes = 1;
for (int i=1; i<n; ++i) {
	if (s[p[i]] != s[p[i-1]])  ++classes;
	c[p[i]] = classes-1;
}
//
int pn[maxlen], cn[maxlen];
for (int h=0; (1<<h)<n; ++h) {
	for (int i=0; i<n; ++i) {
		pn[i] = p[i] - (1<<h);
		if (pn[i] < 0)  pn[i] += n;
	}
	memset (cnt, 0, classes * sizeof(int));
	for (int i=0; i<n; ++i)
		++cnt[c[pn[i]]];
	for (int i=1; i<classes; ++i)
		cnt[i] += cnt[i-1];
	for (int i=n-1; i>=0; --i)
		p[--cnt[c[pn[i]]]] = pn[i];
	cn[p[0]] = 0;
	classes = 1;
	for (int i=1; i<n; ++i) {
		int mid1 = (p[i] + (1<<h)) % n,  mid2 = (p[i-1] + (1<<h)) % n;
		if (c[p[i]] != c[p[i-1]] || c[mid1] != c[mid2])
			++classes;
		cn[p[i]] = classes-1;
	}
	memcpy (c, cn, n * sizeof(int));
}
```

```
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;
    }
}
```

```
int log = 20;
int N = 1 << log;

typedef complex<double> cd;

int rev[N];
cd root[N];

void init() {
    for (int i = 0; i != N; ++i)
        rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (maxlog - 1));

    const double pi = acos(-1);
    for (int k = 1; k < maxn; k *= 2) {
        cd tmp(pi / k);
        root[k] = {1, 0};
        for (int i = 1; i < k; i++)
            root[k + i] = (i & 1) ? root[(k + i) >> 1] * tmp : root[(k + i) >> 1];
    }
}

void fft(vector<cd>& a) {
    for (int i = 0; i != N; ++i)
        if (rev[i] < i)
            swap(res[rev[i]], res[i]);
    for (int k = 1; k < maxn; k *= 2)
        for (int i = 0; i < maxn; i += 2 * k)
            for (int j = 0; j != k; ++j) {
                cd tmp = root[k + j] * res[i + j + k];
                res[i + j + k] = res[i + j] - tmp;
                res[i + j]     = res[i + j] + tmp;
            }
}

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

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

```
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;
            }
        }
}
```

```cpp
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;
        }
    }
}
```

```
Добавим к нашему графу вершину s и рёбра из неё во все остальные вершины. Запустим алгоритм Форда-Беллмана и попросим его построить нам квадратную матрицу со следующим условием: d[i][u] - длина минимального пути от s до u ровно из i ребер. Тогда длина оптимального цикла μ* минимального среднего веса вычисляется как minumaxkd[n][u]-d[k][u]n-k.

Достаточно будет доказать это правило для μ*=0, так как для других μ* можно просто отнять эту величину от всех ребер и получить снова случай с μ*=0.

Чтобы найти цикл после построения матрицы d[k][u], запомним, при каких u и k достигается оптимальное значение μ*, и, используя d[n][u], поднимемся по указателям предков. Как только мы попадем в уже посещенную вершину - мы нашли цикл минимального среднего веса.

Этот алгоритм работает за O(VE)

func findMinCycle(Graph G)
    // вводим мнимую вершину s, от которой проведём рёбра нулевого веса в каждую вершину графа
    Node s
    Edge[] e
    insert(s)
    i = 0
    for u in G
        e[i].begin = s
        e[i].end = u
        e[i].weight = 0
        i++
    // строим матрицу кратчайших расстояний, запустив алгоритм Форда-Беллмана из вершины s
    fordBellman(s)
    // m - длина оптимального цикла
    m = minumaxk((d[n][u] - d[k][u]) / (n - k))
```

DM

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

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

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

Whitespace code FFT