

Headers (1)

```
.vimrc

set nu rnu hls is nosol ts=4 sw=4 ch=4 sc
filetype indent on
syntax on
ca Hash w !cpp -dD -P -fpreprocessed \\\ tr -d
⇨ '[:space:]' \\\ md5sum \\\ cut -c-6
```

.bashrc

```
c() {
  g++ $1.cpp -o $1 -std=c++20 -Wall -Wextra \
  -D_GLIBCXX_DEBUG -fsanitize=address,undefined \
  -DDEBUG -gddb3
}

tc () {
  g++ $1.cpp -o $1 -std=c++20 -Wall -Wextra \
  -D_GLIBCXX_DEBUG -fsanitize=address,undefined \
  -O3
}

sc() {
  g++ $1.cpp -o $1 -std=c++20 -Wall -Wextra \
  -O3
}
```

headers

```
#c13244, includes: <bits/stdc++.h>
solution template
```

```
using namespace std;
using LL=long long;
#define FOR(i,l,r)for(auto i=l;i<=(r);++i)
#define REP(i,n)FOR(i,0,(n)-1)
#define ITF(e,c)for(auto&e:(c))
#define ssize(x)int(x.size())
template<class A,class
⇨ B>auto&operator<<(ostream&o,pair<A,B>p){return
⇨ o<<"("<p.first<<" , "<p.second<<")";}
template<class T>auto operator<<(ostream&o,T
⇨ x)->decltype(x.end(),o){o<<"{";int i=0;for(auto
⇨ e:x)o<<(" , ")2*!i++<<e;return o<<"}";}
#ifndef DEBUG
#define debug(X...)cerr<<["#X"]:
⇨ " , "[{auto...$}]{(cerr<<$<<" , ),...}<<"\n";}(X)
#else
#define debug(...){}
#endif
```

```
int main() {
  cin.tie(0)->sync_with_stdio(0);
  return 0;
}
```

test.sh

```
for ((i=0;i++;)); do
  echo -n "Test $@ $i "
  echo "$@ $i" | ./gen > t.in
  ./main < t.in > m.out & ./brut < t.in > b.out
  wait && diff -wq m.out b.out || break
  echo "OK"
done
```

stress.sh

```
set -e
for ((i=0;i<99;i++)); do
  echo -n "Test $@ $i "
  echo "$@ $i" | ./gen | command time -f "%es %MKB"
  ⇨ ./main > /dev/null
done
```

tree-gen

```
#fd1cf5

generate perfectly random tree using Prüfer code

mt19937_64 rng;
LL rd(LL l, LL r) {
  return uniform_int_distribution<LL>(l, r)(rng);
}
vector<pair<int, int>> gen_tree(int n) {
  vector<int> code(n - 2), deg(n + 1);
  ITF (x, code) x = rd(1, n), deg[x]++;
  int ptr = 1, leaf = 1;
  while (deg[ptr]) ++ptr, ++leaf;
  vector<pair<int, int>> edges;
  ITF (p, code) {
    edges.emplace_back(leaf, p);
    if (--deg[p] == 0 && p < ptr)
      leaf = p;
    else {
      do ptr++;
      while (deg[ptr]);
      leaf = ptr;
    }
  }
  edges.emplace_back(leaf, n);
  return edges;
}
```

Wzorki (2)

2.1 Geometria

Odległość punktu (x_0, y_0) od prostej $(Ax + By + C = 0)$:

|
A

x

0

+
B

y

0

+
C
|

√

A

2

+

B

2

{\displaystyle {\frac {Ax_{0}+By_{0}+C}{\sqrt {A^{2}+B^{2}}}}}

Okrąg opisany R i wpisany r na trójj: $R = \frac{abc}{4rp}$

Okrąg wpisany w trójkąt: $r = \sqrt{\frac{(p-a)(p-b)(p-c)}{p}}$

Pole trójkąta: $S = pr = \sqrt{p(p-a)(p-b)(p-c)}, p = \frac{a+b+c}{2}$

Pole poly bez samoprzecięć o wierzchołkach całkowitych $(I$ wewnątrz, B na krawędzi): $P = I + \frac{1}{2}B - 1$

2.2 Trygonometria

$\sin(\alpha \pm \beta) = \sin \alpha \cos \beta \pm \cos \alpha \sin \beta$

$\cos(\alpha \pm \beta) = \cos \alpha \cos \beta \mp \sin \alpha \sin \beta$

$\tan(\alpha \pm \beta) = \frac{\tan \alpha \pm \tan \beta}{1 \mp \tan \alpha \tan \beta}$

2.3 Równości

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \quad W = \left(-\frac{b}{2a}, -\frac{\Delta}{4a}\right)$$

$$\begin{aligned} ax + by &= e \wedge cx + dy = f \implies \\ x &= \frac{ed - bf}{ad - bc} \wedge y = \frac{af - ec}{ad - bc} \end{aligned}$$

2.4 Liczby pierwsze

$p = 962592769$ to liczba na NTT, czyli $2^{21} \mid p - 1$. Do hashowania: 970592641 (31-bit), 31443539979727 (45-bit), 3006703054056749 (52-bit). Jest 78498 pierwszych $\leq 10^6$. Generatorów jest $\phi(\phi(p^a))$, czyli dla $p > 2$ zawsze istnieje.

2.5 Liczby antypierwsze

<i>lim</i>	10 ²	10 ³	10 ⁴	10 ⁵	10 ⁶	10 ⁷	10 ⁸
<i>n</i>	60	840	7560	83160	720720	8648640	73513440
<i>d(n)</i>	12	32	64	128	240	448	768
<i>lim</i>	10 ⁹		10 ¹²		10 ¹⁵		
<i>n</i>	735134400		963761198400		866421317361600		
<i>d(n)</i>	1344		6720		26880		
<i>lim</i>	10 ¹⁸						
<i>n</i>	897612484786617600						
<i>d(n)</i>	103680						

2.6 Dzielniki

$$\sum_{d \mid n} d = O(n \log \log n)$$

2.7 Silnia

<i>n</i>	1	2	3	4	5	6	7	8	9	10
<i>n</i> !	1	2	6	24	120	720	5040	40320	362880	3628800
<i>n</i> !	1	11	12	13	14	15	16	17		
<i>n</i> !	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
<i>n</i>	20	25	30	40	50	100	150	171		
<i>n</i> !	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

2.8 Symbol Newtona

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n^k}{k!}$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n-1}{k-1} + \binom{n-2}{k-1} + \dots + \binom{k-1}{k-1}$$

$$(x+y)^n = \sum_{k=0}^n \binom{n}{k} x^k y^{n-k}$$

$$\sum_{i=0}^k \binom{n+i}{i} = \binom{n+k+1}{k+1}$$

$$(-1)^i \binom{x}{i} = \binom{i-1-x}{i}$$

$$\sum_{i=0}^k \binom{n}{i} \binom{m}{k-i} = \binom{n+m}{k}$$

$$\binom{n}{k} \binom{k}{i} = \binom{n}{i} \binom{n-i}{k-i}$$

2.9 Fibonacci

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}} \quad F_{n-1} F_{n+1} - F_n^2 = (-1)^n,$$

$$F_{n+k} = F_k F_{n+1} + F_{k-1} F_n, \quad F_n | F_{nk},$$

$$NWD(F_m, F_n) = F_{NWD(m, n)}$$

2.10 Wzorki na pewne ciągi

2.10.1 Nieporządek

Liczba takich permutacji, że $p_i \neq i$:

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} + \frac{1}{2} \right\rfloor$$

2.10.2 Liczba podziałów

Liczba sposobów zapisania n jako sumę posortowanych liczb dodatnich:

$$p(0) = 1, \quad p(n) = \sum_{k \in \mathbb{Z}, \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

szacujemy $p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$:

<i>n</i>	0	1	2	3	4	5	6	7	8	9	20	50	100
<i>p(n)</i>	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

2.10.3 Stirling pierwszego rzędu

Liczba permutacji długości n mające k cykli:

$$c(n, k) = c(n-1, k-1) + (n-1)c(n-1, k), \quad c(0, 0) = 1,$$

$$\sum_{k=0}^n c(n, k) x^k = x(x+1) \dots (x+n-1).$$

Małe wartości:

$$c(8, k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1,$$

$$c(n, 2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots$$

2.10.4 Stirling drugiego rzędu

Liczba podziałów zbioru rozmiaru n na k bloków:

$$S(n, k) = S(n-1, k-1) + kS(n-1, k), \quad S(n, 1) = S(n, n) = 1,$$

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n.$$

2.10.5 Liczby Catalana

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1, \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n, \quad C_{n+1} = \sum C_i C_{n-i}$$

$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, \dots$$

Równoważne: ścieżki na planszy $n \times n$, nawiasowania po $n()$, liczba drzew binarnych z $n+1$ liśćmi (0 lub 2 syny), skierowanych drzew z $n+1$ wierzchołkami, triangulacje $n+2$ -kąta, permutacji $[n]$ bez 3-wyrazowego rosnącego podciągu.

2.10.6 Formuła Cayley'a

Liczba różnych drzew (z dokładnością do numerowania wierzchołków) wynosi n^{n-2} . Liczba sposobów by zespojnić k spójnych o rozmiarach s_1, s_2, \dots, s_k wynosi $s_1 \cdot s_2 \cdot \dots \cdot s_k \cdot n^{k-2}$.

2.11 Pitagoras

Trójki (a, b, c) , takie że $a^2 + b^2 = c^2$: Jest $a = k \cdot (m^2 - n^2)$, $b = k \cdot (2mn)$, $c = k \cdot (m^2 + n^2)$, gdzie $m > n > 0$, $k > 0$, $m \perp n$, oraz albo m albo n jest parzyste.

2.12 Gen. względnie pierwszych par

Dwa drzewa, zaczynając od (2, 1) (parzysta-nieparzysta) oraz (3, 1) (nieparzysta-nieparzysta), rozgałęzienia są do $(2m - n, m)$, $(2m + n, m)$ oraz $(m + 2n, n)$.

Matma (3)

bitmasks

```
#a123fc
```

bit fiddling hacks

```
__builtin_popcountll(uint64_t); // # of set bits
assert(__builtin_popcount(0b0001'0010'1100) == 4);
```

```
__builtin_clzll(uint64_t); // # of leading zeros
assert(__builtin_clz(0b0001'0010'1100) == 23);
```

```
__builtin_ctzll(uint64_t); // # of trailing zeros
assert(__builtin_ctz(0b0001'0010'1100) == 2);
```

```
__builtin_ffsll(uint64_t); // # ctz but handles 0
assert(__builtin_ffs(0b0001'0010'1100) == 3);
```

```
n&(n+1) // clears all trailing ones
// 0011 0111 → 0011 000
```

```
n | (n+1) // sets the last cleared bit
// 00110101 → 00110111
```

```
n & -n // extracts the last set bit
// 0011 0100 → 0000 0100
```

```
// iterate over submasks of mask
for (int sub=mask; sub; sub=(sub-1)&m) {
}
```

bignum

```
#56ad04
```

Podstawa wynosi 1e9. Mnożenie, dzielenie, nwd oraz modulo jest kwadratowe, wersje operatorX(Num, int) liniowe. Podstawę można zmieniać (ma zachodzić base == 10^digits_per_elem).

```
// BEGIN HASH 07b311
```

```

struct Num {
    static constexpr int digits_per_elem = 9, base =
        ⇨ int(1e9);
    int sign = 0;
    vector<int> x;
    Num& shorten() {
        while(ssize(x) and x.back() == 0)
            x.pop_back();
        for(int a : x)
            assert(0 <= a and a < base);
        if(x.empty())
            sign = 0;
        return *this;
    }
    Num(string s) {
        sign = ssize(s) and s[0] == '-' ?
            ⇨ s.erase(s.begin()), -1 : 1;
        for(int i = ssize(s); i > 0; i -= digits_per_elem)
            if(i < digits_per_elem)
                x.emplace_back(stoi(s.substr(0, i)));
            else
                x.emplace_back(stoi(s.substr(i -
                    ⇨ digits_per_elem, digits_per_elem)));
        shorten();
    }
    Num() {}
    Num(LL s) : Num(to_string(s)) {}
}; // END HASH
// BEGIN HASH 7b8dd7
string to_string(const Num& n) {
    stringstream s;
    s << (n.sign == -1 ? "-" : "") << (ssize(n.x) ?
        ⇨ n.x.back() : 0);
    for(int i = ssize(n.x) - 2; i >= 0; --i)
        s << setfill('0') << setw(n.digits_per_elem) <<
            ⇨ n.x[i];
    return s.str();
}
ostream& operator<<(ostream &o, const Num& n) {
    return o << to_string(n).c_str();
} // END HASH
// BEGIN HASH 5e0053
auto operator<=>(const Num& a, const Num& b) {
    if(a.sign != b.sign or ssize(a.x) != ssize(b.x))
        return ssize(a.x) * a.sign <=> ssize(b.x) *
            ⇨ b.sign;
    for(int i = ssize(a.x) - 1; i >= 0; --i)
        if(a.x[i] != b.x[i])
            return a.x[i] * a.sign <=> b.x[i] * b.sign;
    return strong_ordering::equal;
}
bool operator==(const Num& a, const Num& b) {
    return a.x == b.x and a.sign == b.sign;
} // END HASH
// BEGIN HASH 61131b
Num abs(Num n) { n.sign &= 1; return n; }
Num operator+(Num a, Num b) {
    int mode = a.sign * b.sign >= 0 ? a.sign |= b.sign,
        ⇨ 1 : abs(b) > abs(a) ? swap(a, b), -1 : -1, carry
        ⇨ = 0;
    for(int i = 0; i < max(ssize((mode == 1 ? a :
        ⇨ b).x), ssize(b.x)) or carry; ++i) {
        if(mode == 1 and i == ssize(a.x))
            a.x.emplace_back(0);
        a.x[i] += mode * (carry + (i < ssize(b.x) ?
            ⇨ b.x[i] : 0));
        carry = a.x[i] >= a.base or a.x[i] < 0;
        a.x[i] -= mode * carry * a.base;
    }
    return a.shorten();
} // END HASH
Num operator-(Num a) { a.sign *= -1; return a; }
Num operator-(Num a, Num b) { return a + -b; }
// BEGIN HASH e17d1a
Num operator*(Num a, int b) {
    assert(abs(b) < a.base);

```

```

    int carry = 0;
    for(int i = 0; i < ssize(a.x) or carry; ++i) {
        if(i == ssize(a.x))
            a.x.emplace_back(0);
        LL cur = a.x[i] * LL(abs(b)) + carry;
        a.x[i] = int(cur % a.base);
        carry = int(cur / a.base);
    }
    if(b < 0)
        a.sign *= -1;
    return a.shorten();
} // END HASH
// BEGIN HASH 7e782b
Num operator*(const Num& a, const Num& b) {
    Num c;
    c.x.resize(ssize(a.x) + ssize(b.x));
    REP(i, ssize(a.x))
        for(int j = 0, carry = 0; j < ssize(b.x) or
            ⇨ carry; ++j) {
            LL cur = c.x[i + j] + a.x[i] * LL(j <
                ⇨ ssize(b.x) ? b.x[j] : 0) + carry;
            c.x[i + j] = int(cur % a.base);
            carry = int(cur / a.base);
        }
    c.sign = a.sign * b.sign;
    return c.shorten();
} // END HASH
// BEGIN HASH 53d883
Num operator/(Num a, int b) {
    assert(b != 0 and abs(b) < a.base);
    int carry = 0;
    for(int i = ssize(a.x) - 1; i >= 0; --i) {
        LL cur = a.x[i] + carry * LL(a.base);
        a.x[i] = int(cur / abs(b));
        carry = int(cur % abs(b));
    }
    if(b < 0)
        a.sign *= -1;
    return a.shorten();
} // END HASH
// BEGIN HASH 150a87
// zwraca a * pow(a, base, b)
Num shift(Num a, int b) {
    vector v(b, 0);
    a.x.insert(a.x.begin(), v.begin(), v.end());
    return a.shorten();
}
Num operator/(Num a, Num b) {
    assert(ssize(b.x));
    int s = a.sign * b.sign;
    Num c;
    a = abs(a);
    b = abs(b);
    for(int i = ssize(a.x) - ssize(b.x); i >= 0; --i) {
        if(a < shift(b, i)) continue;
        int l = 0, r = a.base - 1;
        while(l < r) {
            int m = (l + r + 1) / 2;
            if(shift(b * m, i) <= a)
                l = m;
            else
                r = m - 1;
        }
        c = c + shift(l, i);
        a = a - shift(b * l, i);
    }
    c.sign = s;
    return c.shorten();
} // END HASH
// BEGIN HASH 08656c
template<typename T>
Num operator%(const Num& a, const T& b) { return a -
    ⇨ ((a / b) * b); }
Num nwd(const Num& a, const Num& b) { return b ==
    ⇨ Num() ? a : nwd(b, a % b); }
// END HASH

```

crt

#e206d9, includes: extended-gcd

$\mathcal{O}(\log n)$, crt(a , m , b , n) zwraca takie x , że $x \bmod m = a$ oraz $x \bmod n = b$, m oraz n nie muszą być względnie pierwsze, ale może nie być wtedy rozwiązania (assert wywali, ale można zmienić na return -1).

```

LL crt(LL a, LL m, LL b, LL n) {
    if(n > m) swap(a, b), swap(m, n);
    auto [d, x, y] = extended_gcd(m, n);
    assert((a - b) % d == 0);
    LL ret = (b - a) % n * x % n / d * m + a;
    return ret < 0 ? ret + m * n / d : ret;
}

```

determinant

#45753a, includes: matrix-header

$\mathcal{O}(n^3)$, wyznacznik macierzy (modulo lub double)

```

T determinant(vector<vector<T>>& a) {
    int n = ssize(a);
    T res = 1;
    REP(i, n) {
        int b = i;
        FOR(j, i + 1, n - 1)
            if(abs(a[j][i]) > abs(a[b][i]))
                b = j;
        if(i != b)
            swap(a[i], a[b]), res = sub(0, res);
        res = mul(res, a[i][i]);
        if(equal(res, 0))
            return 0;
        FOR(j, i + 1, n - 1) {
            T v = divide(a[j][i], a[i][i]);
            if(not equal(v, 0))
                FOR(k, i + 1, n - 1)
                    a[j][k] = sub(a[j][k], mul(v, a[i][k]));
        }
    }
    return res;
}

```

discrete-log

#466b80, includes: simple-modulo

$\mathcal{O}(\sqrt{m} \log n)$ czasowo, $\mathcal{O}(\sqrt{n})$ pamięciowo, dla liczby pierwszej mod oraz $a, b \nmid mod$ znajdzie e takie że $a^e \equiv b \pmod{mod}$. Jak zwróci -1 to nie istnieje.

```

int discrete_log(int a, int b) {
    int n = int(sqrt(mod)) + 1;
    int an = 1;
    REP(i, n)
        an = mul(an, a);
    unordered_map<int, int> vals;
    int cur = b;
    FOR(q, 0, n) {
        vals[cur] = q;
        cur = mul(cur, a);
    }
    cur = 1;
    FOR(p, 1, n) {
        cur = mul(cur, an);
        if(vals.count(cur)) {
            int ans = n * p - vals[cur];
            return ans;
        }
    }
    return -1;
}

```

discrete-root

#7a0737, includes: primitive-root, discrete-log

Dla pierwszego mod oraz $a \perp mod$, k znajduje b takie, że $b^k = a$ (pierwiastek k -tego stopnia z a). Jak zwróci -1 to nie istnieje.

```

int discrete_root(int a, int k) {

```

```

    int g = primitive_root();
    int y = discrete_log(powi(g, k), a);
    if(y == -1)
        return -1;
    return powi(g, y);
}

```

extended-gcd

#9c311b

$\mathcal{O}(\log(\min(a, b)))$, dla danego (a, b) znajduje takie $(gcd(a, b), x, y)$, że $ax + by = gcd(a, b)$. auto [gcd, x, y] = extended_gcd(a, b);

```

tuple<LL, LL, LL> extended_gcd(LL a, LL b) {
    if(a == 0)
        return {b, 0, 1};
    auto [gcd, x, y] = extended_gcd(b % a, a);
    return {gcd, y - x * (b / a), x};
}

```

fft-mod

#79c6e2, includes: fft

$\mathcal{O}(n \log n)$, conv_mod(a , b) zwraca iloczyn wielomianów modulo, ma większą dokładność niż zwykły fft.

```

vector<int> conv_mod(vector<int> a, vector<int> b,
    ⇨ int M) {
    if(a.empty() or b.empty()) return {};
    vector<int> res(ssize(a) + ssize(b) - 1);
    const int CUTOFF = 125;
    if(min(ssize(a), ssize(b)) <= CUTOFF) {
        if(ssize(a) > ssize(b))
            swap(a, b);
        REP(i, ssize(a))
            REP(j, ssize(b))
                res[i + j] = int((res[i + j] + LL(a[i]) *
                    ⇨ b[j]) % M);
        return res;
    }
    int B = 32 - __builtin_clz(ssize(res)), n = 1 << B;
    int cut = int(sqrt(M));
    vector<Complex> L(n), R(n), outL(n), outs(n);
    REP(i, ssize(a)) L[i] = Complex((int) a[i] / cut,
        ⇨ (int) a[i] % cut);
    REP(i, ssize(b)) R[i] = Complex((int) b[i] / cut,
        ⇨ (int) b[i] % cut);
    fft(L), fft(R);
    REP(i, n) {
        int j = -i & (n - 1);
        outL[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n)
            ⇨ / 1i;
    }
    fft(outL), fft(outs);
    REP(i, ssize(res)) {
        LL av = LL(real(outL[i]) + 0.5), cv =
            ⇨ LL(imag(outs[i]) + 0.5);
        LL bv = LL(imag(outL[i]) + 0.5) +
            ⇨ LL(real(outs[i]) + 0.5);
        res[i] = int(((av % M * cut + bv) % M * cut + cv)
            ⇨ % M);
    }
    return res;
}

```

fft

#7a313d

$\mathcal{O}(n \log n)$, conv(a , b) to iloczyn wielomianów.

```

// BEGIN HASH 81676a
using Complex = complex<double>;
void fft(vector<Complex> &a) {
    int n = ssize(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<Complex> rt(2, 1);
    for(static int k = 2; k < n; k *= 2) {

```

```
R.resize(n), rt.resize(n);
auto x = polar(1.0L, acosl(-1) / k);
FOR(i, k, 2 * k - 1)
    rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
}
vector<int> rev(n);
REP(i, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
REP(i, n) if(i < rev[i]) swap(a[i], a[rev[i]]);
for(int k = 1; k < n; k *= 2) {
    for(int i = 0; i < n; i += 2 * k) REP(j, k) {
        Complex z = rt[j + k] * a[i + j + k]; // mozna
        ↪ zoptowac rozpisujac
        a[i + j + k] = a[i + j] - z;
        a[i + j] += z;
    }
}
} // END HASH
vector<double> conv(vector<double> &a, vector<double>
↪ &b) {
    if(a.empty() || b.empty()) return {};
    vector<double> res(ssize(a) + ssize(b) - 1);
    int L = 32 - __builtin_clz(ssize(res)), n = (1 <<
↪ L);
    vector<Complex> in(n), out(n);
    copy(a.begin(), a.end(), in.begin());
    REP(i, ssize(b)) in[i].imag(b[i]);
    fft(in);
    for(auto &x : in) x *= x;
    REP(i, n) out[i] = in[i] * (n - 1) - conj(in[i]);
    fft(out);
    REP(i, ssize(res)) res[i] = imag(out[i]) / (4 * n);
    return res;
}
```

gauss

```
#d36ccd, includes: matrix-header
O(nm(n+m)), Wrzucam n vectorów {wsp_x0, wsp_x1, ...,
wsp_xm - 1, suma}, gauss wtedy zwraca liczbę rozwiązań (0,1 albo
2 (tnz. nieskończoność)) oraz jedno poprawne rozwiązanie (o ile
istnieje). Przykład gauss({2, -1, 1, 7}, {1, 1, 1, 1}, {0,
1, -1, 6.5}) zwraca (1, {6.75, 0.375, -6.125}).
```

```
pair<int, vector<T>> gauss(vector<vector<T>> a) {
    int n = ssize(a); // liczba wierszy
    int m = ssize(a[0]) - 1; // liczba zmiennych
    vector<int> where(m, -1); // w ktorym wierszu jest
    ↪ zdefiniowana iDta zmienna
    for(int col = 0, row = 0; col < m and row < n;
    ↪ ++col) {
        int sel = row;
        for(int y = row; y < n; ++y)
            if(abs(a[y][col]) > abs(a[sel][col]))
                sel = y;
        if(equal(a[sel][col], 0))
            continue;
        for(int x = col; x <= m; ++x)
            swap(a[sel][x], a[row][x]);
        // teraz sel jest nieaktualne
        where[col] = row;
        for(int y = 0; y < n; ++y)
            if(y != row) {
                T wspolczynnik = divide(a[y][col],
                ↪ a[row][col]);
                for(int x = col; x <= m; ++x)
                    a[y][x] = sub(a[y][x], mul(wspolczynnik,
                    ↪ a[row][x]));
            }
        ++row;
    }
    vector<T> answer(m);
    for(int col = 0; col < m; ++col)
        if(where[col] != -1)
            answer[col] = divide(a[where[col]][m],
            ↪ a[where[col]][col]);
    for(int row = 0; row < n; ++row) {
        T got = 0;
```

gauss integral lagrange-consecutive matrix-header matrix-inverse miller-rabin multiplicative

```
for(int col = 0; col < m; ++col)
    got = add(got, mul(answer[col], a[row][col]));
if(not equal(got, a[row][m]))
    return {0, answer};
}
for(int col = 0; col < m; ++col)
    if(where[col] == -1)
        return {2, answer};
return {1, answer};
}
```

integral

```
#fad4ef
O(idk), zwraca całkę f na [l, r].
using D = long double;
D simpson(function<D (D)> f, D l, D r) {
    return (f(l) + 4 * f((l + r) / 2) + f(r)) * (r - l)
    ↪ / 6;
}
D integrate(function<D (D)> f, D l, D r, D s, D eps) {
    D m = (l + r) / 2;
    D sl = simpson(f, l, m), sr = simpson(f, m, r), s2
    ↪ = sl + sr;
    if(abs(s2 - s) < 15 * eps or r - l < 1e-10)
        return s2 + (s2 - s) / 15;
    return integrate(f, l, m, sl, eps / 2)
    + integrate(f, m, r, sr, eps / 2);
}
D integrate(function<D (D)> f, D l, D r) {
    return integrate(f, l, r, simpson(f, l, r), 1e-8);
}
```

lagrange-consecutive

```
#06efb5, includes: simple-modulo
O(n), przyjmuje wartości wielomianu w punktach 0,1,...,n-1 i
wylicza jego wartość w x. lagrange_consecutive({2, 3, 4}, 3) ==
5
```

```
int lagrange_consecutive(vector<int> y, int x) {
    int n = ssize(y), fac = 1, pref = 1, suff = 1, ret
    ↪ = 0;
    FOR(i, 1, n) fac = mul(fac, i);
    fac = inv(fac);
    REP(i, n) {
        fac = mul(fac, n - i);
        y[i] = mul(y[i], mul(pref, fac));
        y[n - 1 - i] = mul(y[n - 1 - i], mul(suff, mul(i
        ↪ % 2 ? mod - 1 : 1, fac)));
        pref = mul(pref, sub(x, i));
        suff = mul(suff, sub(x, n - 1 - i));
    }
    REP(i, n) ret = add(ret, y[i]);
    return ret;
}
```

matrix-header

```
#a1aa3e
Funkcje pomocnicze do algorytmów macierzowych.
#ifdef 1
#ifdef CHANGABLE_MOD
int mod = 998'244'353;
#else
constexpr int mod = 998'244'353;
#endif
// BEGIN HASH 2216e3
bool equal(int a, int b) {
    return a == b;
}
int mul(int a, int b) {
    return int(a * LL(b) % mod);
}
int add(int a, int b) {
    a += b;
    return a >= mod ? a - mod : a;
```

```
}
int powi(int a, int b) {
    for(int ret = 1;; b /= 2) {
        if(b == 0)
            return ret;
        if(b & 1)
            ret = mul(ret, a);
        a = mul(a, a);
    }
}
int inv(int x) {
    return powi(x, mod - 2);
}
int divide(int a, int b) {
    return mul(a, inv(b));
}
int sub(int a, int b) {
    return add(a, mod - b);
}
using T = int;
// END HASH
#else
// BEGIN HASH a32baf
constexpr double eps = 1e-9;
bool equal(double a, double b) {
    return abs(a - b) < eps;
}
#define OP(name, op) double name(double a, double b)
↪ { return a op b; }
OP(mul, *)
OP(add, +)
OP(divide, /)
OP(sub, -)
using T = double;
// END HASH
#endif
```

matrix-inverse

```
#9f7607, includes: matrix-header
O(n^3), odwrotność macierzy (modulo lub double). Zwraca rząd
macierzy. Dla odwracalnych macierzy (rząd = n) w a znajdzie się jej
odwrotność.
```

```
int inverse(vector<vector<T>>& a) {
    int n = ssize(a);
    vector<int> col(n);
    vector h(n, vector<T>(n));
    REP(i, n)
        h[i][i] = 1, col[i] = i;
    REP(i, n) {
        int r = i, c = i;
        FOR(j, i, n - 1) FOR(k, i, n - 1)
            if(abs(a[j][k]) > abs(a[r][c]))
                r = j, c = k;
        if (equal(a[r][c], 0))
            return i;
        a[i].swap(a[r]);
        h[i].swap(h[r]);
        REP(j, n)
            swap(a[j][i], a[j][c]), swap(h[j][i], h[j][c]);
        swap(col[i], col[c]);
        T v = a[i][i];
        FOR(j, i + 1, n - 1) {
            T f = divide(a[j][i], v);
            a[j][i] = 0;
            FOR(k, i + 1, n - 1)
                a[j][k] = sub(a[j][k], mul(f, a[i][k]));
            REP(k, n)
                h[j][k] = sub(h[j][k], mul(f, h[i][k]));
        }
        FOR(j, i + 1, n - 1)
            a[i][j] = divide(a[i][j], v);
        REP(j, n)
            h[i][j] = divide(h[i][j], v);
        a[i][i] = 1;
    }
}
```

```
for(int i = n - 1; i > 0; --i) REP(j, i) {
    T v = a[j][i];
    REP(k, n)
        h[j][k] = sub(h[j][k], mul(v, h[i][k]));
}
REP(i, n)
    REP(j, n)
        a[col[i]][col[j]] = h[i][j];
return n;
}
```

miller-rabin

```
#98e3d1
O(log^2 n) test pierwszości Millera-Rabina, działa dla long
longów.
LL llmul(LL a, LL b, LL m) {
    return LL(__int128_t(a) * b % m);
}
LL llpowi(LL a, LL n, LL m) {
    for (LL ret = 1;; n /= 2) {
        if (n == 0)
            return ret;
        if (n % 2)
            ret = llmul(ret, a, m);
        a = llmul(a, a, m);
    }
}
bool miller_rabin(LL n) {
    if(n < 2) return false;
    int r = 0;
    LL d = n - 1;
    while(d % 2 == 0)
        d /= 2, r++;
    for(int a : {2, 325, 9375, 28178, 450775, 9780504,
    ↪ 1795265022}) {
        if (a % n == 0) continue;
        LL x = llpowi(a, d, n);
        if(x == 1 || x == n - 1)
            continue;
        bool composite = true;
        REP(i, r - 1) {
            x = llmul(x, x, n);
            if(x == n - 1) {
                composite = false;
                break;
            }
        }
        if(composite) return false;
    }
    return true;
}
```

multiplicative

```
#6a710c, includes: sieve
O(n), mobius(n) oblicza funkcję Möbiusa na [0..n], totient(n)
oblicza funkcję Eulera na [0..n], wartości w 0
niezdefiniowane.
// BEGIN HASH f3b0be
vector<int> mobius(int n) {
    sieve(n);
    vector<int> ans(n + 1, 0);
    if (n) ans[1] = 1;
    FOR(i, 2, n) {
        int p = prime_div[i];
        if (i / p % p) ans[i] = -ans[i / p];
    }
    return ans;
} // END HASH
// BEGIN HASH 0a67bf
vector<int> totient(int n) {
    sieve(n);
    vector<int> ans(n + 1, 1);
    FOR(i, 2, n) {
        int p = prime_div[i];
```

```

    ans[i] = ans[i / p] * (p - bool(i / p % p));
}
return ans;
} // END HASH

```

ntt

#cae153, includes: simple-modulo

$\mathcal{O}(n \log n)$ mnożenie wielomianów mod 998244353.

```

// BEGIN HASH a27376
using vi = vector<int>;
constexpr int root = 3;
void ntt(vi& a, int n, bool inverse = false) {
    assert((n & (n - 1)) == 0);
    a.resize(n);
    vi b(n);
    for(int w = n / 2; w; w /= 2, swap(a, b)) {
        int r = powi(root, (mod - 1) / n * w), m = 1;
        for(int i = 0; i < n; i += w * 2, m = mul(m, r))
            ↪ REP(j, w) {
                int u = a[i + j], v = mul(a[i + j + w], m);
                b[i / 2 + j] = add(u, v);
                b[i / 2 + j + n / 2] = sub(u, v);
            }
    }
    if(inverse) {
        reverse(a.begin() + 1, a.end());
        int invn = inv(n);
        for(int& e : a) e = mul(e, invn);
    }
} // END HASH
vi conv(vi a, vi b) {
    if(a.empty() or b.empty()) return {};
    int l = ssize(a) + ssize(b) - 1, sz = 1 << __lg(2 *
    ↪ l - 1);
    ntt(a, sz), ntt(b, sz);
    REP(i, sz) a[i] = mul(a[i], b[i]);
    ntt(a, sz, true), a.resize(l);
    return a;
}

```

pi

#5af6fc

$\mathcal{O}(n^{\frac{3}{4}})$, liczba liczb pierwszych na przedziale $[1, n]$. Pi pi(n);
pi.query(d); // musi zachodzić $d \mid n$

```

struct Pi {
    vector<LL> w, dp;
    int id(LL v) {
        if (v <= w.back() / v)
            return int(v - 1);
        return ssize(w) - int(w.back() / v);
    }
    Pi(LL n) {
        for (LL i = 1; i * i <= n; ++i) {
            w.push_back(i);
            if (n / i != i)
                w.emplace_back(n / i);
        }
        sort(w.begin(), w.end());
        for (LL i : w)
            dp.emplace_back(i - 1);
        for (LL i = 1; (i + 1) * (i + 1) <= n; ++i) {
            if (dp[i] == dp[i - 1])
                continue;
            for (int j = ssize(w) - 1; w[j] >= (i + 1) * (i
            ↪ + 1); --j)
                dp[j] -= dp[id(w[j] / (i + 1))] - dp[i - 1];
        }
    }
    LL query(LL v) {
        assert(w.back() % v == 0);
        return dp[id(v)];
    }
};

```

ntt pi primitive-root pythagorean-triples rho-pollard sieve simple-modulo tonelli-shanks xor-base

primitive-root

#8870d1, includes: simple-modulo, rho-pollard

$\mathcal{O}(\log^2(mod))$, dla pierwszego mod znajduje generator modulo mod
(z być może sporą stałą).

```

int primitive_root() {
    if(mod == 2)
        return 1;
    int q = mod - 1;
    vector<LL> v = factor(q);
    vector<int> fact;
    REP(i, ssize(v))
        if(!i or v[i] != v[i - 1])
            fact.emplace_back(v[i]);
    while(true) {
        int g = rd(2, q);
        auto is_good = [&] {
            for(auto &f : fact)
                if(powi(g, q / f) == 1)
                    return false;
            return true;
        };
        if(is_good())
            return g;
    }
}

```

pythagorean-triples

#8b8dbe

Wyznacza wszystkie trójki (a, b, c) takie, że $a^2 + b^2 = c^2$,
 $gcd(a, b, c) = 1$ oraz $c \leq$ limit. Zwraca tylko jedną z (a, b, c) oraz
 (b, a, c) .

```

vector<tuple<int, int, int>> pythagorean_triples(int
    ↪ limit) {
    vector<tuple<int, int, int>> ret;
    function<void(int, int, int)> gen = [&](int a, int
    ↪ b, int c) {
        if (c > limit)
            return;
        ret.emplace_back(a, b, c);
        REP(i, 3) {
            gen(a + 2 * b + 2 * c, 2 * a + b + 2 * c, 2 * a
            ↪ + 2 * b + 3 * c);
            a = -a;
            if (i) b = -b;
        }
    };
    gen(3, 4, 5);
    return ret;
}

```

rho-pollard

#2b0d5e, includes: miller-rab

$\mathcal{O}(n^{\frac{1}{4}})$, factor(n) zwraca vector dzielników pierwszych n ,
niekoniecznie posortowany, get_pairs(n) zwraca posortowany
vector par (dzielnik pierwszych, krotność) dla liczby n ,
all_factors(n) zwraca vector wszystkich dzielników n ,
niekoniecznie posortowany, factor(12) = {2, 2, 3},
factor(545423) = {53, 41, 251};, get_pairs(12) = {(2,
2), (3, 1)}, all_factors(12) = {1, 3, 2, 6, 4,
12}.

```

// BEGIN HASH ffa3b2
LL rho_pollard(LL n) {
    if(n % 2 == 0) return 2;
    for(LL i = 1; i++) {
        auto f = [&](LL x) { return (llmul(x, x, n) + i)
            ↪ % n; };
        LL x = 2, y = f(x), p;
        while((p = __gcd(n - x + y, n)) == 1)
            x = f(x), y = f(f(y));
        if(p != n) return p;
    }
}
vector<LL> factor(LL n) {

```

```

    if(n == 1) return {};
    if(miller_rabin(n)) return {n};
    LL x = rho_pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), r.begin(), r.end());
    return l;
} // END HASH
vector<pair<LL, int>> get_pairs(LL n) {
    auto v = factor(n);
    sort(v.begin(), v.end());
    vector<pair<LL, int>> ret;
    REP(i, ssize(v)) {
        int x = i + 1;
        while (x < ssize(v) and v[x] == v[i])
            ++x;
        ret.emplace_back(v[i], x - i);
        i = x - 1;
    }
    return ret;
}
vector<LL> all_factors(LL n) {
    auto v = get_pairs(n);
    vector<LL> ret;
    function<void(LL, int)> gen = [&](LL val, int p) {
        if (p == ssize(v)) {
            ret.emplace_back(val);
            return;
        }
        auto [x, cnt] = v[p];
        gen(val, p + 1);
        REP(i, cnt) {
            val *= x;
            gen(val, p + 1);
        }
    };
    gen(1, 0);
    return ret;
}

```

sieve

#e4c334

$\mathcal{O}(n)$, sieve(n) przetwarza liczby do n włącznie, comp[i] oznacza
czy i jest złożone, primes zawiera wszystkie liczby pierwsze \leq n ,
prime_div[i] zawiera najmniejszy dzielnik pierwszy i , na CF dla
 $n = 1e8$ działa w 1.2s.

```

vector<bool> comp;
vector<int> primes, prime_div;
void sieve(int n) {
    primes.clear();
    comp.resize(n + 1);
    prime_div.resize(n + 1);
    FOR(i, 2, n) {
        if (!comp[i]) primes.emplace_back(i),
            ↪ prime_div[i] = i;
        for (int p : primes) {
            int x = i * p;
            if (x > n) break;
            comp[x] = true;
            prime_div[x] = p;
            if (i % p == 0) break;
        }
    }
}

```

simple-modulo

#ec6f32

podstawowe operacje na modulo, pamięta o constexpr.

```

// BEGIN HASH 368fc1
#ifdef CHANGABLE_MOD
int mod = 998'244'353;
#else
constexpr int mod = 998'244'353;
#endif
int add(int a, int b) {

```

```

    a += b;
    return a >= mod ? a - mod : a;
}
int sub(int a, int b) {
    return add(a, mod - b);
}
int mul(int a, int b) {
    return int(a * LL(b) % mod);
}
int powi(int a, int b) {
    for(int ret = 1;; b /= 2) {
        if(b == 0)
            return ret;
        if(b & 1)
            ret = mul(ret, a);
        a = mul(a, a);
    }
}
int inv(int x) {
    return powi(x, mod - 2);
} // END HASH
struct BinomCoeff {
    vector<int> fac, rev;
    BinomCoeff(int n) {
        fac = rev = vector(n + 1, 1);
        FOR(i, 1, n) fac[i] = mul(fac[i - 1], i);
        rev[n] = inv(fac[n]);
        for(int i = n; i > 0; --i)
            rev[i - 1] = mul(rev[i], i);
    }
    int operator()(int n, int k) {
        return mul(fac[n], mul(rev[n - k], rev[k]));
    }
};

```

tonelli-shanks

#d6b02b

$\mathcal{O}(\log^2(p))$, dla pierwszego p oraz $0 \leq a \leq p - 1$ znajduje takie x , że
 $x^2 \equiv a \pmod{p}$ lub -1 jeżeli takie x nie istnieje, można przepisać by
działało dla LL

```

int mul(int a, int b, int p) {
    return int(a * LL(b) % p);
}
int powi(int a, int b, int p) {
    for (int ret = 1;; b /= 2) {
        if (!b) return ret;
        if (b & 1) ret = mul(ret, a, p);
        a = mul(a, a, p);
    }
}
int tonelli_shanks(int a, int p) {
    if (a == 0) return 0;
    if (p == 2) return 1;
    if (powi(a, p / 2, p) != 1) return -1;
    int q = p - 1, s = 0, z = 2;
    while (q % 2 == 0) q /= 2, ++s;
    while (powi(z, p / 2, p) == 1) ++z;
    int c = powi(z, q, p), t = powi(a, q, p);
    int r = powi(a, q / 2 + 1, p);
    while (t != 1) {
        int i = 0, x = t;
        while (x != 1) x = mul(x, x, p), ++i;
        c = powi(c, 1 << (s - i - 1), p); // !!! dla LL
        r = mul(r, c, p), c = mul(c, c, p);
        t = mul(t, c, p), s = i;
    }
    return r;
}

```

xor-base

#92d51f

$\mathcal{O}(nB + B^2)$ dla $B = bits$, dla S wyznacza minimalny zbiór B taki, że
każdy element S można zapisać jako xor jakiegoś podzbioru
 B .

```
int highest_bit(int ai) {
    return ai == 0 ? 0 : __lg(ai) + 1;
}
constexpr int bits = 30;
vector<int> xor_base(vector<int> elems) {
    vector<vector<int>> at_bit(bits + 1);
    for(int ai : elems)
        at_bit[highest_bit(ai)].emplace_back(ai);
    for(int b = bits; b >= 1; --b)
        while(ssize(at_bit[b]) > 1) {
            int ai = at_bit[b].back();
            at_bit[b].pop_back();
            ai ^= at_bit[b].back();
            at_bit[highest_bit(ai)].emplace_back(ai);
        }
    at_bit.erase(at_bit.begin());
    REP(b0, bits - 1)
        for(int a0 : at_bit[b0])
            FOR(b1, b0 + 1, bits - 1)
                for(int &a1 : at_bit[b1])
                    if((a1 >> b0) & 1)
                        a1 ^= a0;
    vector<int> ret;
    for(auto &v : at_bit) {
        assert(ssize(v) <= 1);
        for(int ai : v)
            ret.emplace_back(ai);
    }
    return ret;
}
```

Struktury danych (4)

associative-queue

#3e4a47

Kolejka wspierająca dowolną operację łączną, $\mathcal{O}(1)$ zamortyzowany. Konstruktor przyjmuje dwuargumentową funkcję oraz jej element neutralny. Dla minów jest AssocQueue<int> q[[]](int a, int b){ return min(a, b); }, numeric_limits<int>::max());

```
template<typename T>
struct AssocQueue {
    using fn = function<T(T, T)>;
    fn f;
    vector<pair<T, T>> s1, s2; // {x, f(pref)}
    AssocQueue(fn _f, T e = T()) : f(_f), s1{{e, e}}{
        ⇨ s2{{e, e}} }
    void mv() {
        if (ssize(s2) == 1)
            while (ssize(s1) > 1) {
                s2.emplace_back(s1.back().first,
                    ⇨ f(s1.back().first, s2.back().second));
                s1.pop_back();
            }
    }
    void emplace(T x) {
        s1.emplace_back(x, f(s1.back().second, x));
    }
    void pop() {
        mv();
        s2.pop_back();
    }
    T calc() {
        return f(s2.back().second, s1.back().second);
    }
    T front() {
        mv();
        return s2.back().first;
    }
    int size() {
        return ssize(s1) + ssize(s2) - 2;
    }
    void clear() {
        s1.resize(1);
    }
}
```

```
        s2.resize(1);
    }
};
```

fenwick-tree-2d

#692f3b, includes: fenwick-tree

$\mathcal{O}(\log^2 n)$, pamięć $\mathcal{O}(n \log n)$, 2D offline, wywołujemy preprocess(x, y) na pozycjach, które chcemy updateować, później init(). update(x, y, val) dodaje val do $[x, y]$, query(x, y) zwraca sumę na prostokącie $(0, 0) - (x, y)$.

```
struct Fenwick2d {
    vector<vector<int>> ys;
    vector<Fenwick> ft;
    Fenwick2d(int limx) : ys(limx) {}
    void preprocess(int x, int y) {
        for(; x < ssize(ys); x |= x + 1)
            ys[x].push_back(y);
    }
    void init() {
        for(auto &v : ys) {
            sort(v.begin(), v.end());
            ft.emplace_back(ssize(v));
        }
    }
    int ind(int x, int y) {
        auto it = lower_bound(ys[x].begin(), ys[x].end(),
            ⇨ y);
        return int(distance(ys[x].begin(), it));
    }
    void update(int x, int y, LL val) {
        for(; x < ssize(ys); x |= x + 1)
            ft[x].update(ind(x, y), val);
    }
    LL query(int x, int y) {
        LL sum = 0;
        for(x++; x > 0; x &= x - 1)
            sum += ft[x - 1].query(ind(x - 1, y + 1) - 1);
        return sum;
    }
};
```

fenwick-tree

#910494

$\mathcal{O}(\log n)$, indeksowane od 0, update(pos, val) dodaje val do elementu pos, query(pos) zwraca sumę $[0, pos]$.

```
struct Fenwick {
    vector<LL> s;
    Fenwick(int n) : s(n) {}
    void update(int pos, LL val) {
        for(; pos < ssize(s); pos |= pos + 1)
            s[pos] += val;
    }
    LL query(int pos) {
        LL ret = 0;
        for(pos++; pos > 0; pos &= pos - 1)
            ret += s[pos - 1];
        return ret;
    }
    LL query(int l, int r) {
        return query(r) - query(l - 1);
    }
};
```

find-union

#c3dcbd

$\mathcal{O}(\alpha(n))$, mniejszy do większego.

```
struct FindUnion {
    vector<int> rep;
    int size(int x) { return -rep[find(x)]; }
    int find(int x) {
        return rep[x] < 0 ? x : rep[x] = find(rep[x]);
    }
}
```

```
bool same_set(int a, int b) { return find(a) ==
    ⇨ find(b); }
bool join(int a, int b) {
    a = find(a), b = find(b);
    if(a == b)
        return false;
    if(-rep[a] < -rep[b])
        swap(a, b);
    rep[a] += rep[b];
    rep[b] = a;
    return true;
}
FindUnion(int n) : rep(n, -1) {}
};
```

hash-map

#ede6ad, includes: <ext/pb_ds/assoc_container.hpp>

$\mathcal{O}(1)$, trzeba przed includem dać undef _GLIBCXX_DEBUG.

```
using namespace __gnu_pbds;
struct chash {
    const uint64_t C = LL(2e18 * acos(-1)) + 69;
    const int RANDOM = mt19937(0)();
    size_t operator()(uint64_t x) const {
        return __builtin_bswap64((x^RANDOM) * C);
    }
};
template<class L, class R>
using hash_map = gp_hash_table<L, R, chash>;
```

line-container

#45779b

$\mathcal{O}(\log n)$ set dla funkcji liniowych, add(a, b) dodaje funkcję $y = ax + b$ query(x) zwraca największe y w punkcie x .

```
struct Line {
    mutable LL a, b, p;
    LL eval(LL a, LL b) { return a * x + b; }
    bool operator<(const Line &o) const { return a <
        ⇨ o.a; }
    bool operator<(LL x) const { return p < x; }
};
struct LineContainer : multiset<Line, less<>> {
    // jak double to inf = 1 / .0, div(a, b) = a / b
    const LL inf = LLONG_MAX;
    LL div(LL a, LL b) { return a / b - ((a ^ b) < 0 &&
        ⇨ a % b); }
    bool intersect(iterator x, iterator y) {
        if(y == end()) { x->p = inf; return false; }
        if(x->a == y->a) x->p = x->b > y->b ? inf : -inf;
        else x->p = div(y->b - x->b, x->a - y->a);
        return x->p >= y->p;
    }
    void add(LL a, LL b) {
        auto z = insert({a, b, 0}), y = z++, x = y;
        while(intersect(y, z)) z = erase(z);
        if(x != begin() && intersect(--x, y))
            intersect(x, erase(y));
        while((y = x) != begin() && (--x)->p >= y->p)
            intersect(x, erase(y));
    }
    LL query(LL x) {
        assert(!empty());
        return lower_bound(x)->eval(x);
    }
};
```

ordered-set

#0a779f, includes: <ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp>
insert(x) dodaje element x (nie ma emplace), find_by_order(i) zwraca iterator do i -tego elementu, order_of_key(x) zwraca ile jest mniejszych elementów (x nie musi być w secie). Jeśli chcemy multisetą, to używamy par (val, id).

```
using namespace __gnu_pbds;
template<class T> using ordered_set = tree<
    T,
    null_type,
    less<T>,
    rb_tree_tag,
    tree_order_statistics_node_update
>;
```

range-add

#65c934, includes: fenwick-tree

$\mathcal{O}(\log n)$ drzewo przedział-punkt (+, +), wszystko indexowane od 0, update(l, r, val) dodaje val na przedziale $[l, r]$, query(pos) zwraca wartość elementu pos .

```
struct RangeAdd {
    Fenwick f;
    RangeAdd(int n) : f(n) {}
    void update(int l, int r, LL val) {
        f.update(l, val);
        f.update(r + 1, -val);
    }
    LL query(int pos) {
        return f.query(pos);
    }
};
```

rmq

#a697d6

$\mathcal{O}(n \log n)$ czasowo i pamięciowo, Range Minimum Query z użyciem sparse table, zapytanie jest w $\mathcal{O}(1)$.

```
struct RMQ {
    vector<vector<int>> st;
    RMQ(const vector<int> &a) {
        int n = ssize(a), lg = 0;
        while((1 << lg) < n) lg++;
        st.resize(lg + 1, a);
        FOR(i, 1, lg) REP(j, n) {
            st[i][j] = st[i - 1][j];
            int q = j + (1 << (i - 1));
            if(q < n) st[i][j] = min(st[i][j], st[i -
                ⇨ 1][q]);
        }
    }
    int query(int l, int r) {
        int q = __lg(r - l + 1), x = r - (1 << q) + 1;
        return min(st[q][l], st[q][x]);
    }
};
```

treap

#f9c1bb

$\mathcal{O}(\log n)$ Implicit Treap, wszystko indexowane od 0, do Node dopisujemy jakie chcemy mieć trzymać dodatkowo dane. Jeśli chcemy robić lazy, to wykonania push należy wstawić tam gdzie oznaczono komentarzem.

```
namespace Treap {
    // BEGIN HASH
    mt19937 rng_key(0);
    struct Node {
        int prio, cnt = 1;
        Node *l = nullptr, *r = nullptr;
        Node() : prio(int(rng_key())) {}
        ~Node() { delete l; delete r; }
    };
    using pNode = Node*;
    int get_cnt(pNode t) { return t ? t->cnt : 0; }
    void update(pNode t) {
        if (!t) return;
        // push(t);
        t->cnt = get_cnt(t->l) + get_cnt(t->r) + 1;
    }
    void split(pNode t, int i, pNode &l, pNode &r) {
        if (!t) {
            l = r = nullptr;
        }
    }
}
```

```

    return;
}
// push(t);
if (i <= get_cnt(t->l))
    split(t->l, i, l, t->l), r = t;
else
    split(t->r, i - get_cnt(t->l) - 1, t->r, r), l
    ⇐ t;
update(t);
}
void merge(pNode &t, pNode l, pNode r) {
    if (!l or !r) t = l ?: r;
    else if (l->prio > r->prio) {
        // push(l);
        merge(l->r, l->r, r), t = l;
    }
    else {
        // push(r);
        merge(r->l, l, r->l), t = r;
    }
    update(t);
} // END HASH
void apply_on_interval(pNode &root, int l, int r,
⇐ function<void (pNode)> f) {
    pNode left, mid, right;
    split(root, r + 1, mid, right);
    split(mid, l, left, mid);
    assert(l <= r and mid);
    f(mid);
    merge(mid, left, mid);
    merge(root, mid, right);
}
}
}

```

Grafy (5)

2sat

#e21178

$\mathcal{O}(n + m)$, Zwraca poprawne przyporządkowanie zmiennym logicznym dla problemu 2-SAT, albo mówi, że takie nie istnieje. Konstruktor przyjmuje liczbę zmiennych, ~ oznacza negację zmiennej. Po wywołaniu solve(), values[0..n-1] zawiera wartości rozwiązania.

```

struct TwoSat {
    int n;
    vector<vector<int>> gr;
    vector<int> values;
    TwoSat(int _n = 0) : n(_n), gr(2 * n) {}
    void either(int f, int j) {
        f = max(2 * f, -1 - 2 * f);
        j = max(2 * j, -1 - 2 * j);
        gr[f].emplace_back(j ^ 1);
        gr[j].emplace_back(f ^ 1);
    }
    void set_value(int x) { either(x, x); }
    void implication(int f, int j) { either(~f, j); }
    int add_var() {
        gr.emplace_back();
        gr.emplace_back();
        return n++;
    }
    void at_most_one(vector<int>& li) {
        if(ssize(li) <= 1) return;
        int cur = ~li[0];
        FOR(i, 2, ssize(li) - 1) {
            int next = add_var();
            either(cur, ~li[i]);
            either(cur, next);
            either(~li[i], next);
            cur = ~next;
        }
        either(cur, ~li[1]);
    }
    vector<int> val, comp, z;
}

```

```

int t = 0;
int dfs(int i) {
    int low = val[i] = ++t, x;
    z.emplace_back(i);
    for(auto &e : gr[i]) if(!comp[e])
        low = min(low, val[e] ?: dfs(e));
    if(low == val[i]) do {
        x = z.back(); z.pop_back();
        comp[x] = low;
        if (values[x >> 1] == -1)
            values[x >> 1] = x & 1;
    } while (x != i);
    return val[i] = low;
}
bool solve() {
    values.assign(n, -1);
    val.assign(2 * n, 0);
    comp = val;
    REP(i, 2 * n) if(!comp[i]) dfs(i);
    REP(i, n) if(comp[2 * i] == comp[2 * i + 1])
        ⇐ return 0;
    return 1;
}
}

```

biconnected

#e53996

$\mathcal{O}(n + m)$, dwuspójne składowe, mosty oraz punkty artykulacji. po skonstruowaniu, bicon = zbiór list id krawędzi, bridges = lista id krawędzi będącymi mostami, arti_points = lista wierzchołków będącymi punktami artykulacji. Tablice są nieposortowane. Wspiera multikrawędzie i wiele spójnych, ale nie pętle.

```

struct Low {
    vector<vector<int>> graph;
    vector<int> low, pre;
    vector<pair<int, int>> edges;
    vector<vector<int>> bicon;
    vector<int> bicon_stack, arti_points, bridges;
    int gtime = 0;
    void dfs(int v, int p) {
        low[v] = pre[v] = gtime++;
        bool considered_parent = false;
        int son_count = 0;
        bool is_arti = false;
        for(int e : graph[v]) {
            int u = edges[e].first ^ edges[e].second ^ v;
            if(u == p and not considered_parent)
                considered_parent = true;
            else if(pre[u] == -1) {
                bicon_stack.emplace_back(e);
                dfs(u, v);
                low[v] = min(low[v], low[u]);
                if(low[u] >= pre[v]) {
                    bicon.emplace_back();
                    do {
                        bicon.back().emplace_back(bicon_stack.back());
                        bicon_stack.pop_back();
                    } while(bicon.back().back() != e);
                }
                ++son_count;
                if(p != -1 and low[u] >= pre[v])
                    is_arti = true;
                if(low[u] > pre[v])
                    bridges.emplace_back(e);
            }
            else if(pre[v] > pre[u]) {
                low[v] = min(low[v], pre[u]);
                bicon_stack.emplace_back(e);
            }
        }
        if(p == -1 and son_count > 1)
            is_arti = true;
        if(is_arti)
            arti_points.emplace_back(v);
    }
}

```

```

Low(int n, vector<pair<int, int>> _edges) :
    ⇐ graph(n), low(n), pre(n, -1), edges(_edges) {
    REP(i, ssize(edges)) {
        auto [v, u] = edges[i];
    #ifdef LOCAL
        assert(v != u);
    #endif
        graph[v].emplace_back(i);
        graph[u].emplace_back(i);
    }
    REP(v, n)
        if(pre[v] == -1)
            dfs(v, -1);
}
}

```

eulerian-path

#295863

$\mathcal{O}(n + m)$, ścieżka eulera. Zwraca tupla (exists, ids, vertices). W exists jest informacja czy jest ścieżka/cykl eulera, ids zawiera id kolejnych krawędzi, vertices zawiera listę wierzchołków na tej ścieżce. Dla cyklu, vertices[0] == vertices[m].

```

tuple<bool, vector<int>, vector<int>>
⇐ eulerian_path(int n, const vector<pair<int, int>>
⇐ &edges, bool directed) {
    vector<int> in(n);
    vector<vector<int>> adj(n);
    int start = 0;
    REP(i, ssize(edges)) {
        auto [a, b] = edges[i];
        start = a;
        ++in[b];
        adj[a].emplace_back(i);
        if (not directed)
            adj[b].emplace_back(i);
    }
    int cnt_in = 0, cnt_out = 0;
    REP(i, n) {
        if (directed) {
            if (abs(ssize(adj[i]) - in[i]) > 1)
                return {};
            if (in[i] < ssize(adj[i]))
                start = i, ++cnt_in;
            else
                cnt_out += in[i] > ssize(adj[i]);
        }
        else if (ssize(adj[i]) % 2)
            start = i, ++cnt_in;
    }
    vector<int> ids, vertices;
    vector<bool> used(ssize(edges));
    function<void (int)> dfs = [&](int v) {
        while (ssize(adj[v])) {
            int id = adj[v].back(), u = v ^ edges[id].first
            ⇐ ^ edges[id].second;
            adj[v].pop_back();
            if (used[id]) continue;
            used[id] = true;
            dfs(u);
            ids.emplace_back(id);
        }
    };
    dfs(start);
    if (cnt_in + cnt_out > 2 or not
    ⇐ all_of(used.begin(), used.end(), identity{}))
        return {};
    reverse(ids.begin(), ids.end());
    if (ssize(ids))
        vertices = {start};
    for (int id : ids)
        vertices.emplace_back(vertices.back() ^
        ⇐ edges[id].first ^ edges[id].second);
    return {true, ids, vertices};
}

```

hld

#013f82

$\mathcal{O}(q \log n)$ Heavy-Light Decomposition. get_vertex(v) zwraca pozycję odpowiadającą wierzchołkowi. get_path(v, u) zwraca przedziały do obsługi drzewem przedziałowym. get_path(v, u) jeśli robisz operacje na wierzchołkach. get_path(v, u, false) jeśli na krawędziach (nie zawiera lca). get_subtree(v) zwraca przedział preorder odpowiadający poddrzewu v.

```

struct HLD {
    // BEGIN HASH 32f81f
    vector<vector<int>> &adj;
    vector<int> sz, pre, pos, nxt, par;
    int t = 0;
    void init(int v, int p = -1) {
        par[v] = p;
        sz[v] = 1;
        if(ssize(adj[v]) > 1 && adj[v][0] == p)
            swap(adj[v][0], adj[v][1]);
        for(int &u : adj[v]) if(u != par[v]) {
            init(u, v);
            sz[v] += sz[u];
            if(sz[u] > sz[adj[v][0]])
                swap(u, adj[v][0]);
        }
    }
    void set_paths(int v) {
        pre[v] = t++;
        for(int &u : adj[v]) if(u != par[v]) {
            nxt[u] = (u == adj[v][0] ? nxt[v] : u);
            set_paths(u);
        }
        pos[v] = t;
    }
    HLD(int n, vector<vector<int>> &_adj)
        : adj(_adj), sz(n), pre(n), pos(n), nxt(n),
        ⇐ par(n) {
        init(0), set_paths(0);
    } // END HASH
    int lca(int v, int u) {
        while(nxt[v] != nxt[u]) {
            if(pre[v] < pre[u])
                swap(v, u);
            v = par[nxt[v]];
        }
        return (pre[v] < pre[u] ? v : u);
    }
    vector<pair<int, int>> path_up(int v, int u) {
        vector<pair<int, int>> ret;
        while(nxt[v] != nxt[u]) {
            ret.emplace_back(pre[nxt[v]], pre[v]);
            v = par[nxt[v]];
        }
        if(pre[u] != pre[v]) ret.emplace_back(pre[u] + 1,
        ⇐ pre[v]);
        return ret;
    }
    int get_vertex(int v) { return pre[v]; }
    vector<pair<int, int>> get_path(int v, int u, bool
    ⇐ add_lca = true) {
        int w = lca(v, u);
        auto ret = path_up(v, w);
        auto path_u = path_up(u, w);
        if(add_lca) ret.emplace_back(pre[w], pre[w]);
        ret.insert(ret.end(), path_u.begin(),
        ⇐ path_u.end());
        return ret;
    }
    pair<int, int> get_subtree(int v) { return {pre[v],
    ⇐ pos[v] - 1}; }
};

```

scc

#a1bad8

konstruktor $\mathcal{O}(n)$, `get_compressed` $\mathcal{O}(n \log n)$. `group[v]` to numer silnie spójnej wierzchołka v , `order` to toposort, w którym krawędzie idą w lewo (z lewej są liście), `get_compressed()` zwraca graf silnie spójnych, `get_compressed(false)` nie usuwa multikrawędzi.

```
struct SCC {
    int n;
    vector<vector<int>> &graph;
    int group_cnt = 0;
    vector<int> group;
    vector<vector<int>> rev_graph;
    vector<int> order;
    void order_dfs(int v) {
        group[v] = 1;
        for(int u : rev_graph[v])
            if(group[u] == 0)
                order_dfs(u);
        order.emplace_back(v);
    }
    void group_dfs(int v, int color) {
        group[v] = color;
        for(int u : graph[v])
            if(group[u] == -1)
                group_dfs(u, color);
    }
    SCC(vector<vector<int>> &graph) : graph(_graph) {
        n = ssize(graph);
        rev_graph.resize(n);
        REP(v, n)
            for(int u : graph[v])
                rev_graph[u].emplace_back(v);
        group.resize(n);
        REP(v, n)
            if(group[v] == 0)
                order_dfs(v);
        reverse(order.begin(), order.end());
        debug(order);
        group.assign(n, -1);
        for(int v : order)
            if(group[v] == -1)
                group_dfs(v, group_cnt++);
    }
    vector<vector<int>> get_compressed(bool delete_same
    ⇨ = true) {
        vector<vector<int>> ans(group_cnt);
        REP(v, n)
            for(int u : graph[v])
                if(group[v] != group[u])
                    ans[group[v]].emplace_back(group[u]);
        if(not delete_same)
            return ans;
        REP(v, group_cnt) {
            sort(ans[v].begin(), ans[v].end());
            ans[v].erase(unique(ans[v].begin(),
            ⇨ ans[v].end()), ans[v].end());
        }
        return ans;
    }
};
```

toposort

#9de42b
 $\mathcal{O}(n)$, `get_toposort_order(g)` zwraca listę wierzchołków takich, że krawędzie są od wierzchołków wcześniejszych w liście do późniejszych. `get_new_vertex_id_from_order(order)` zwraca odwrotność tej permutacji, tzn. dla każdego wierzchołka trzyma jego nowy numer, aby po przenumerowaniu grafu istniały krawędzie tylko do wierzchołków o większych numerach. `permute(elems, new_id)` zwraca przepermutowaną tablicę `elems` według nowych numerów wierzchołków (przydatne jak się trzyma informacje o wierzchołkach, a chce się zrobić przenumerowanie topologiczne). `renumerate_vertices(...)` zwraca nowy graf, w którym wierzchołki są przenumerowane. Nowy graf: `renumerate_vertices(graph, get_new_vertex_id_from_order(get_toposort_order(graph)))`.

toposort hopcroft-karp hungarian konig-theorem matching

```
// BEGIN HASH 6b6518
vector<int> get_toposort_order(vector<vector<int>>
    ⇨ graph) {
    int n = ssize(graph);
    vector<int> indeg(n);
    REP(v, n)
        for(int u : graph[v])
            ++indeg[u];
    vector<int> que;
    REP(v, n)
        if(indeg[v] == 0)
            que.emplace_back(v);
    vector<int> ret;
    while(not que.empty()) {
        int v = que.back();
        que.pop_back();
        ret.emplace_back(v);
        for(int u : graph[v])
            if(--indeg[u] == 0)
                que.emplace_back(u);
    }
    return ret;
} // END HASH
vector<int> get_new_vertex_id_from_order(vector<int>
    ⇨ order) {
    vector<int> ret(ssize(order), -1);
    REP(v, ssize(order))
        ret[order[v]] = v;
    return ret;
}
template<class T>
vector<T> permute(vector<T> elems, vector<int>
    ⇨ new_id) {
    vector<T> ret(ssize(elems));
    REP(v, ssize(elems))
        ret[new_id[v]] = elems[v];
    return ret;
}
vector<vector<int>>
    ⇨ renumerate_vertices(vector<vector<int>> graph,
    ⇨ vector<int> new_id) {
    int n = ssize(graph);
    vector<vector<int>> ret(n);
    REP(v, n)
        for(int u : graph[v])
            ret[new_id[v]].emplace_back(new_id[u]);
    REP(v, n)
        for(int u : ret[v])
            assert(v < u);
    return ret;
}
```

Flowy i matchingi (6)

hopcroft-karp

#6911f0

$\mathcal{O}(m\sqrt{n})$ Hopcroft-Karp do liczenia matchingu. Przydaje się głównie w aproksymacji, ponieważ po k iteracjach gwarantuje matching o rozmiarze przynajmniej $k/(k+1)$ -best matching. Wierzchołki grafu muszą być podzielone na warstwy $[0, n0)$ oraz $[n0, n0 + n1)$. Zwraca rozmiar matchingu oraz przypisanie (lub -1, gdy nie jest zmatchowane).

```
pair<int, vector<int>>
    ⇨ hopcroft_karp(vector<vector<int>> graph, int n0,
    ⇨ int n1) {
    assert(n0 + n1 == ssize(graph));
    REP(v, n0 + n1)
        for(int u : graph[v])
            assert((v < n0) != (u < n0));
    vector<int> matched_with(n0 + n1, -1), dist(n0 + 1);
    constexpr int inf = int(1e9);
    vector<int> manual_que(n0 + 1);
    auto bfs = [&] {
```

```
    int head = 0, tail = -1;
    fill(dist.begin(), dist.end(), inf);
    REP(v, n0)
        if(matched_with[v] == -1) {
            dist[1 + v] = 0;
            manual_que[++tail] = v;
        }
    while(head <= tail) {
        int v = manual_que[head++];
        if(dist[1 + v] < dist[0])
            for(int u : graph[v])
                if(dist[1 + matched_with[u]] == inf) {
                    dist[1 + matched_with[u]] = dist[1 + v] +
                    ⇨ 1;
                    manual_que[++tail] = matched_with[u];
                }
        return dist[0] != inf;
    };
    function<bool (int)> dfs = [&](int v) {
        if(v == -1)
            return true;
        for(auto u : graph[v])
            if(dist[1 + matched_with[u]] == dist[1 + v] +
            ⇨ 1) {
                if(dfs(matched_with[u])) {
                    matched_with[v] = u;
                    matched_with[u] = v;
                    return true;
                }
            }
            dist[1 + v] = inf;
            return false;
        };
        int answer = 0;
        for(int iter = 0; bfs(); ++iter)
            REP(v, n0)
                if(matched_with[v] == -1 and dfs(v))
                    ++answer;
        return {answer, matched_with};
    }
```

hungarian

#4444a8

$\mathcal{O}(n_0^2 \cdot n_1)$, dla macierzy wag (mogą być ujemne) między dwoma warstwami o rozmiarach $n0$ oraz $n1$ ($n0 \leq n1$) wyznacza minimalną sumę wag skojarzenia pełnego. Zwraca sumę wag oraz matching.

```
pair<LL, vector<int>> hungarian(vector<vector<int>>
    ⇨ a) {
    if(a.empty())
        return {0, {}};
    int n0 = ssize(a) + 1, n1 = ssize(a[0]) + 1;
    assert(n0 <= n1);
    vector<int> p(n1), ans(n0 - 1);
    vector<LL> u(n0), v(n1);
    FOR(i, 1, n0 - 1) {
        p[0] = i;
        int j0 = 0;
        vector<LL> dist(n1, numeric_limits<LL>::max());
        vector<int> pre(n1, -1);
        vector<bool> done(n1 + 1);
        do {
            done[j0] = true;
            int i0 = p[j0], j1 = -1;
            LL delta = numeric_limits<LL>::max();
            FOR(j, 1, n1 - 1)
                if(!done[j]) {
                    auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                    if(cur < dist[j])
                        dist[j] = cur, pre[j] = j0;
                    if(dist[j] < delta)
                        delta = dist[j], j1 = j;
                }
            REP(j, n1) {
```

```
                if(done[j])
                    u[p[j]] += delta, v[j] -= delta;
                else
                    dist[j] -= delta;
            }
            j0 = j1;
        } while(p[j0]);
        while(j0) {
            int j1 = pre[j0];
            p[j0] = p[j1], j0 = j1;
        }
    }
    FOR(j, 1, n1 - 1)
        if(p[j])
            ans[p[j] - 1] = j - 1;
    return {-v[0], ans};
}
```

konig-theorem

#d37a69, includes: matching

$\mathcal{O}(n + matching(n, m))$ wyznaczanie w grafie dwudzielnym kolejno minimalnego pokrycia krawędziowego (PK), maksymalnego zbioru niezależnych wierzchołków (NW), minimalnego pokrycia wierzchołkowego (PW) korzystając z maksymalnego zbioru niezależnych krawędzi (NK) (tak zwany matching). Z tw. Koniga zachodzi $|NK| = n - |PK| = n - |NW| = |PW|$.

```
// BEGIN HASH 27f048
vector<pair<int, int>>
    ⇨ get_min_edge_cover(vector<vector<int>> graph) {
    vector<int> match = Matching(graph)().second;
    vector<pair<int, int>> ret;
    REP(v, ssize(match))
        if(match[v] != -1 and v < match[v])
            ret.emplace_back(v, match[v]);
        else if(match[v] == -1 and not graph[v].empty())
            ret.emplace_back(v, graph[v].front());
    return ret;
} // END HASH
// BEGIN HASH b5f6d5
array<vector<int>, 2>
    ⇨ get_coloring(vector<vector<int>> graph) {
    int n = ssize(graph);
    vector<int> match = Matching(graph)().second;
    vector<int> color(n, -1);
    function<void (int)> dfs = [&](int v) {
        color[v] = 0;
        for(int u : graph[v])
            if(color[u] == -1) {
                color[u] = true;
                dfs(match[u]);
            }
    };
    REP(v, n)
        if(match[v] == -1)
            dfs(v);
    REP(v, n)
        if(color[v] == -1)
            dfs(v);
    array<vector<int>, 2> groups;
    REP(v, n)
        groups[color[v]].emplace_back(v);
    return groups;
}
vector<int>
    ⇨ get_max_independent_set(vector<vector<int>> graph)
    ⇨ {
    return get_coloring(graph)[0];
}
vector<int> get_min_vertex_cover(vector<vector<int>>
    ⇨ graph) {
    return get_coloring(graph)[1];
} // END HASH
```

matching

#686f47

Średnio około $\mathcal{O}(n \log n)$, najgorzej $\mathcal{O}(n^2)$. Wierzchołki grafu nie muszą być ładnie podzielone na dwa przedziały, musi być po prostu dwudzielny. Na przykład `auto [match_size, match] = Matching(graph)();`

```
struct Matching {
    vector<vector<int>> &adj;
    vector<int> mat, vis;
    int t = 0, ans = 0;
    bool mat_dfs(int v) {
        vis[v] = t;
        for(int u : adj[v])
            if(mat[u] == -1) {
                mat[u] = v;
                mat[v] = u;
                return true;
            }
        for(int u : adj[v])
            if(vis[mat[u]] != t && mat_dfs(mat[u])) {
                mat[u] = v;
                mat[v] = u;
                return true;
            }
        return false;
    }
    Matching(vector<vector<int>> &adj) : adj(_adj) {
        mat = vis = vector<int>(ssize(adj), -1);
    }
    pair<int, vector<int>> operator()() {
        int d = -1;
        while(d != 0) {
            d = 0, ++t;
            REP(v, ssize(adj))
                if(mat[v] == -1)
                    d += mat_dfs(v);
            ans += d;
        }
        return {ans, mat};
    }
};
```

Geometria (7)

advanced-complex

#bcc8b5, includes: point
Większość nie działa dla intów.

```
constexpr D pi = acos(-1);
// nachylenie k> y = kx + m
D slope(P a, P b) { return tan(arg(b - a)); }
// rzut p na ab
P project(P p, P a, P b) {
    return a + (b - a) * dot(p - a, b - a) / norm(a - b);
}
// odbicie p wzgledem ab
P reflect(P p, P a, P b) {
    return a + conj((p - a) / (b - a)) * (b - a);
}
// obrot a wzgledem p o theta radianow
P rotate(P a, P p, D theta) {
    return (a - p) * polar(1.0L, theta) + p;
}
// kat ABC, w radianach z przedzialu [0..pi]
D angle(P a, P b, P c) {
    return abs(remainder(arg(a - b) - arg(c - b), 2.0 * pi));
}
// szybkie przeciecie prostych, nie dziala dla
// rownoleglych
P intersection(P a, P b, P p, P q) {
    D c1 = cross(p - a, b - a), c2 = cross(q - a, b - a);
    return (c1 * q - c2 * p) / (c1 - c2);
}
// check czy sa rownolegle
```

```
bool is_parallel(P a, P b, P p, P q) {
    P c = (a - b) / (p - q); return equal(c, conj(c));
}
// check czy sa prostopadle
bool is_perpendicular(P a, P b, P p, P q) {
    P c = (a - b) / (p - q); return equal(c, -conj(c));
}
// zwraca takie q, ze (p, q) jest rownolegle do (a, b)
P parallel(P a, P b, P p) {
    return p + a - b;
}
// zwraca takie q, ze (p, q) jest prostopadle do (a, b)
P perpendicular(P a, P b, P p) {
    return reflect(p, a, b);
}
// przeciecie srodkowych trojkata
P centro(P a, P b, P c) {
    return (a + b + c) / 3.0L;
}
```

angle-sort

#bebd3a, includes: point

$\mathcal{O}(n \log n)$, zwraca wektory P posortowane kątowno zgodnie z ruchem wskazówek zegara od najbliższego kątowno do wektora (0, 1) włącznie. Aby posortować po argumentcie (kącie) swapujemy x, y, używamy angle-sort i ponownie swapujemy x, y. Zakłada że nie ma punktu (0, 0) na wejściu.

```
vector<P> angle_sort(vector<P> t) {
    for(P p : t) assert(not equal(p, P(0, 0)));
    auto it = partition(t.begin(), t.end(), [](P a){
        return P(0, 0) < a; });
    auto cmp = [&](P a, P b) {
        return sign(cross(a, b)) == -1;
    };
    sort(t.begin(), it, cmp);
    sort(it, t.end(), cmp);
    return t;
}
```

angle180-intervals

#50d79d, includes: angle-sort

$\mathcal{O}(n)$, ZAKŁADA że punkty są posortowane kątowno. Zwraca n par $[i, r]$, gdzie r jest maksymalnym cyklicznie indeksem, że wszystkie punkty w tym cyklicznym przedziale są ściśle „po prawej” stronie wektora (0, 0) — $in[i]$, albo są na tej półprostej.

```
vector<pair<int, int>> angle180_intervals(vector<P>
    in) {
    // in must be sorted by angle
    int n = ssize(in);
    vector<int> nxt(n);
    iota(nxt.begin(), nxt.end(), 1);
    int r = nxt[n - 1] = 0;
    vector<pair<int, int>> ret(n);
    REP(l, n) {
        if(nxt[r] == l) r = nxt[r];
        auto good = [&](int i) {
            auto c = cross(in[l], in[i]);
            if(not equal(c, 0)) return c < 0;
            if((P(0, 0) < in[l]) != (P(0, 0) < in[i]))
                return false;
            return l < i;
        };
        while(nxt[r] != l and good(nxt[r]))
            r = nxt[r];
        ret[l] = {l, r};
    }
    return ret;
}
```

area

#31c1e1, includes: point

Pole wielokąta, niekoniecznie wypukłego. W vectorze muszą być wierzchołki zgodnie z kierunkiem ruchu zegara. Jeśli D jest intem to może się psuć / 2. `area(a, b, c)` zwraca pole trójkąta o takich długościach boku.

```
D area(vector<P> pts) {
    int n = ssize(pts);
    D ans = 0;
    REP(i, n) ans += cross(pts[i], pts[(i + 1) % n]);
    return fabs1(ans / 2);
}
D area(D a, D b, D c) {
    D p = (a + b + c) / 2;
    return sqrt1(p * (p - a) * (p - b) * (p - c));
}
```

circle-intersection

#afa5cb, includes: point

Przecięcia okręgu oraz prostej $ax + by + c = 0$ oraz przecięcia okręgu oraz okręgu. Gdy `ssize(circle_circle(...)) == 3` to jest nieskończenie wiele rozwiązań.

```
// BEGIN HASH 16976c
vector<P> circle_line(D r, D a, D b, D c) {
    D len_ab = a * a + b * b,
    x0 = -a * c / len_ab,
    y0 = -b * c / len_ab,
    d = r * r - c * c / len_ab,
    mult = sqrt(d / len_ab);
    if(sign(d) < 0)
        return {};
    else if(sign(d) == 0)
        return {x0, y0};
    return {
        {x0 + b * mult, y0 - a * mult},
        {x0 - b * mult, y0 + a * mult}
    };
}
vector<P> circle_line(D x, D y, D r, D a, D b, D c) {
    return circle_line(r, a, b, c + (a * x + b * y));
}
// END HASH
// BEGIN HASH 17de82
vector<P> circle_circle(D x1, D y1, D r1, D x2, D y2,
    in D r2) {
    x2 -= x1;
    y2 -= y1;
    // now x1 = y1 = 0;
    if(sign(x2) == 0 and sign(y2) == 0) {
        if(equal(r1, r2))
            return {{0, 0}, {0, 0}, {0, 0}}; // inf points
        else
            return {};
    }
    auto vec = circle_line(r1, -2 * x2, -2 * y2,
        x2 * x2 + y2 * y2 + r1 * r1 - r2 * r2);
    for(P &p : vec)
        p += P(x1, y1);
    return vec;
}
// END HASH
```

circle-tangents

#7bf712, includes: point

$\mathcal{O}(1)$, dla dwóch okręgów zwraca dwie styczne (wewnętrzne lub zewnętrzne, zależnie od wartości inner). Zwraca $1 + \text{sign}(\text{dist}(p0, p1) - (\text{inside ? } r0 + r1 : \text{abs}(r0 - r1)))$ rozwiązań, albo 0 gdy $p1 = p2$. Działa gdy jakiś promień jest 0 – przydatne do policzenia stycznej punktu do okręgu.

```
vector<pair<P, P>> circle_tangents(P p1, D r1, P p2,
    in D r2, bool inner) {
    if(inner) r2 *= -1;
    P d = p2 - p1;
    D dr = r1 - r2, d2 = dot(d, d), h2 = d2 - dr * dr;
    if(equal(d2, 0) or sign(h2) < 0)
        return {};
    vector<pair<P, P>> ret;
```

```
for(D sign : {-1, 1}) {
    P v = (d * dr + P(-d.y(), d.x()) * sqrt(max(D(0),
        h2)) * sign) / d2;
    ret.emplace_back(p1 + v * r1, p2 + v * r2);
}
ret.resize(1 + (sign(h2) > 0));
return ret;
}
```

closest-pair

#51c5b5, includes: point

$\mathcal{O}(n \log n)$, zakłada `ssize(in) > 1`.

```
pair<P, P> closest_pair(vector<P> in) {
    set<P> s;
    sort(in.begin(), in.end(), [](P a, P b) { return
        a.y() < b.y(); });
    pair<D, pair<P, P>> ret(1e18, {P(), P()});
    int j = 0;
    for (P p : in) {
        P d(1 + sqrt(ret.first), 0);
        while (in[j].y() <= p.y() - d.x())
            s.erase(in[j++]);
        auto lo = s.lower_bound(p - d), hi =
            s.upper_bound(p + d);
        for (; lo != hi; ++lo)
            ret = min(ret, {pow(dist(*lo, p), 2), {*lo,
                s.insert(p);
            }
        });
    }
    return ret.second;
}
```

convex-gen

#d0f6d0, includes: point, angle-sort, headers/gen

Generatorka wielokątów wypukłych. Zwraca wielokąt z co najmniej n - PROC punktami w zakresie $[-\text{range}, \text{range}]$. Jeśli $n (n > 2)$ jest około $\text{range}^{\frac{2}{3}}$, to powinno chodzić $\mathcal{O}(n \log n)$. Dla większych n może nie dać rady. Ostatni punkt jest zawsze w (0, 0) - można dodać przesunięcie o wektor dla pełnej losowości.

```
vector<int> num_split(int value, int n) {
    vector<int> v(n, value);
    REP(i, n - 1)
        v[i] = rd(0, value);
    sort(v.begin(), v.end());
    adjacent_difference(v.begin(), v.end(), v.begin());
    return v;
}
vector<int> capped_zero_split(int cap, int n) {
    int m = rd(1, n - 1);
    auto lf = num_split(cap, m);
    auto rg = num_split(cap, n - m);
    for (int i : rg)
        lf.emplace_back(-i);
    return lf;
}
vector<P> gen_convex_polygon(int n, int range, bool
    strictly_convex = false) {
    assert(n > 2);
    vector<P> t;
    const double PROC = 0.9;
    do {
        t.clear();
        auto dx = capped_zero_split(range, n);
        auto dy = capped_zero_split(range, n);
        shuffle(dx.begin(), dx.end(), rng);
        REP (i, n)
            if (dx[i] || dy[i])
                t.emplace_back(dx[i], dy[i]);
        t = angle_sort(t);
        if (strictly_convex) {
            vector<P> nt(1, t[0]);
            FOR (i, 1, ssize(t) - 1) {
                if (!sign(cross(t[i], nt.back())))
                    nt.back() += t[i];
```



```

    else
        nt.emplace_back(t[i]);
    }
    while (!nt.empty() && !sign(cross(nt.back(),
    ↪ nt[0]))) {
        nt[0] += nt.back();
        nt.pop_back();
    }
    t = nt;
}
} while (ssize(t) < n * PROC);
partial_sum(t.begin(), t.end(), t.begin());
return t;
}

```

convex-hull-online

#54b0dd

$\mathcal{O}(\log n)$ na każdą operację dodania, Wyznacza górną otoczkę wypukłą online.

```

using P = pair<int, int>;
LL operator*(P l, P r) {
    return l.first * LL(r.second) - l.second *
    ↪ LL(r.first);
}
P operator-(P l, P r) {
    return {l.first - r.first, l.second - r.second};
}
int sign(LL x) {
    return x > 0 ? 1 : x < 0 ? -1 : 0;
}
int dir(P a, P b, P c) {
    return sign((b - a) * (c - b));
}
struct UpperConvexHull {
    set<P> hull;
    void add_point(P p) {
        if(hull.empty()) {
            hull = {p};
            return;
        }
        auto it = hull.lower_bound(p);
        if(*hull.begin() < p and p < *prev(hull.end())) {
            assert(it != hull.end() and it != hull.begin());
            if(dir(*prev(it), p, *it) >= 0)
                return;
        }
        it = hull.emplace(p).first;
        auto have_to_rm = [&](auto iter) {
            if(iter == hull.end() or next(iter) ==
            ↪ hull.end() or iter == hull.begin())
                return false;
            return dir(*prev(iter), *iter, *next(iter)) >=
            ↪ 0;
        };
        while(have_to_rm(next(it)))
            it = prev(hull.erase(next(it)));
        while(it != hull.begin() and have_to_rm(prev(it)))
            it = hull.erase(prev(it));
    }
};

```

convex-hull

#a838ba, includes: point

$\mathcal{O}(n \log n)$, top_bot_hull zwraca osobno górę i dół, hull zwraca punkty na otoczce clockwise gdzie pierwszy jest najbardziej lewym.

```

array<vector<P>, 2> top_bot_hull(vector<P> in) {
    sort(in.begin(), in.end());
    array<vector<P>, 2> ret;
    REP(d, 2) {
        for(auto p : in) {
            while(ssize(ret[d]) > 1 and
            ↪ dir(ret[d].end()[-2], ret[d].back(), p) >= 0)
                ret[d].pop_back();
        }
    }
}

```

convex-hull-online convex-hull furthest-pair halfplane-intersection intersect-lines is-in-hull line point

```

    ret[d].emplace_back(p);
    }
    reverse(in.begin(), in.end());
    }
    return ret;
}
vector<P> hull(vector<P> in) {
    if(ssize(in) <= 1) return in;
    auto ret = top_bot_hull(in);
    REP(d, 2) ret[d].pop_back();
    ret[0].insert(ret[0].end(), ret[1].begin(),
    ↪ ret[1].end());
    return ret[0];
}

```

furthest-pair

#d59d33, includes: convex-hull

$\mathcal{O}(n)$ po puszczeniu otoczki, zakłada $n \geq 2$.

```

pair<P, P> furthest_pair(vector<P> in) {
    in = hull(in);
    int n = ssize(in), j = 1;
    pair<0, pair<P, P>> ret;
    REP(i, j)
        for(;; j = (j + 1) % n) {
            ret = max(ret, {dist(in[i], in[j]), {in[i],
            ↪ in[j]}});
            if (sign(cross(in[(j + 1) % n] - in[j], in[i] +
            ↪ 1] - in[i])) <= 0)
                break;
        }
    return ret.second;
}

```

halfplane-intersection

#26d886, includes: point

$\mathcal{O}(n \log n)$ wyznaczenie punktów na brzegu/otoczce przecięcia podanych półpłaszczyzn. Halfplane(a, b) tworzy półpłaszczyznę wzdłuż prostej $a \rightarrow b$ z obszarem po lewej stronie wektora $a \rightarrow b$. Jeżeli zostało zwróconych mniej, niż trzy punkty, to pole przecięcia jest puste. Na przykład halfplane_intersection({Halfplane(P(2, 1), P(4, 2)), Halfplane(P(6, 3), P(2, 4)), Halfplane(P(-4, 7), P(4, 2))}) == {(4, 2), (6, 3), (0, 4.5)}. Pole przecięcia jest zawsze ograniczone, ponieważ w kodzie są dodawane cztery półpłaszczyzny o współrzędnych w +/-inf, ale nie należy na tym polegać przez eps oraz błędy precyzji (najlepiej jest zmniejszyć inf tyle, ile się da).

```

struct Halfplane {
    P p, pq;
    D angle;
    Halfplane() {}
    Halfplane(P a, P b) : p(a), pq(b - a) {
        angle = atan2(pq.imag(), pq.real());
    }
};
ostream& operator<<(ostream&o, Halfplane h) {
    return o << '(' << h.p << ", " << h.pq << ", " <<
    ↪ h.angle << ')';
}
bool is_outside(Halfplane hi, P p) {
    return sign(cross(hi.pq, p - hi.p)) == -1;
}
P inter(Halfplane s, Halfplane t) {
    D alpha = cross(t.p - s.p, t.pq) / cross(s.pq,
    ↪ t.pq);
    return s.p + s.pq * alpha;
}
vector<P> halfplane_intersection(vector<Halfplane> h)
↪ {
    for(int i = 0; i < 4; ++i) {
        constexp D inf = 1e9;
        array box = {P(-inf, -inf), P(inf, -inf), P(inf,
        ↪ inf), P(-inf, inf)};
        h.emplace_back(box[i], box[(i + 1) % 4]);
    }
}

```

```

sort(h.begin(), h.end(), [&](Halfplane l, Halfplane
↪ r) {
    return l.angle < r.angle;
});
deque<Halfplane> dq;
for(auto &hi : h) {
    while(ssize(dq) >= 2 and is_outside(hi,
    ↪ inter(dq.end()[-1], dq.end()[-2]))
        dq.pop_back();
    while(ssize(dq) >= 2 and is_outside(hi,
    ↪ inter(dq[0], dq[1]))
        dq.pop_front();
    if(ssize(dq) and sign(cross(hi.pq, dq.back().pq))
    ↪ == 0) {
        if(sign(dot(hi.pq, dq.back().pq)) < 0)
            return {};
        if(is_outside(hi, dq.back().p))
            dq.pop_back();
        else
            continue;
    }
    dq.emplace_back(hi);
}
while(ssize(dq) >= 3 and is_outside(dq[0],
↪ inter(dq.end()[-1], dq.end()[-2]))
    dq.pop_back();
while(ssize(dq) >= 3 and is_outside(dq.end()[-1],
↪ inter(dq[0], dq[1]))
    dq.pop_front();
vector<P> ret;
REP(i, ssize(dq))
    ret.emplace_back(inter(dq[i], dq[(i + 1) %
    ↪ ssize(dq)]));
ret.erase(unique(ret.begin(), ret.end(), [&](P l, P
↪ r) { return equal(l, r); }), ret.end());
if(ssize(ret) >= 2 and equal(ret.front(),
↪ ret.back()))
    ret.pop_back();
for(Halfplane hi : h)
    if(ssize(ret) <= 2 and is_outside(hi, ret[0]))
        return {};
return ret;
}

```

intersect-lines

#d88112, includes: point

$\mathcal{O}(1)$ ale intersect_segments ma sporą stałą (ale działa na wszystkich edge-case'ach). Jeżeli intersect_segments zwróci dwa punkty to wszystkie inf rozwiązań są pomiędzy.

```

// BEGIN HASH 95db50
P intersect_lines(P a, P b, P c, P d) {
    D c1 = cross(c - a, b - a), c2 = cross(d - a, b -
    ↪ a);
    // c1 == c2 => równolege
    return (c1 * d - c2 * c) / (c1 - c2);
} // END HASH
// BEGIN HASH 65e219
bool on_segment(P a, P b, P p) {
    return equal(cross(a - p, b - p), 0) and sign(dot(a
    ↪ - p, b - p)) <= 0;
} // END HASH
// BEGIN HASH 2b171b
bool is_intersection_segment(P a, P b, P c, P d) {
    auto aux = [&](D q, D w, D e, D r) {
        return sign(max(q, w) - min(e, r)) >= 0;
    };
    return aux(c.x(), d.x(), a.x(), b.x()) and
    ↪ aux(a.x(), b.x(), c.x(), d.x())
        and aux(c.y(), d.y(), a.y(), b.y()) and
        ↪ aux(a.y(), b.y(), c.y(), d.y())
        and dir(a, d, c) * dir(b, d, c) != 1
        and dir(d, b, a) * dir(c, b, a) != 1;
} // END HASH
// BEGIN HASH e5125d
vector<P> intersect_segments(P a, P b, P c, P d) {

```

```

D acd = cross(c - a, d - c), bcd = cross(c - b, d -
↪ c),
    cab = cross(a - c, b - a), dab = cross(a - d,
    ↪ b - a);
if(sign(acd) * sign(bcd) < 0 and sign(cab) *
    ↪ sign(dab) < 0)
    return {(a * bcd - b * acd) / (bcd - acd)};
set<P> s;
if(on_segment(c, d, a)) s.emplace(a);
if(on_segment(c, d, b)) s.emplace(b);
if(on_segment(a, b, c)) s.emplace(c);
if(on_segment(a, b, d)) s.emplace(d);
return {s.begin(), s.end()};
} // END HASH

```

is-in-hull

#0425ab, includes: intersect-lines

$\mathcal{O}(\log n)$, zwraca czy punkt jest wewnątrz otoczki h. Zakłada że punkty są clockwise oraz nie ma trzech współliniowych (działa na convex-hull).

```

bool is_in_hull(vector<P> h, P p, bool can_on_edge) {
    if(ssize(h) < 3) return can_on_edge and
    ↪ on_segment(h[0], h.back(), p);
    int l = 1, r = ssize(h) - 1;
    if(dir(h[0], h[l], p) >= can_on_edge or dir(h[0],
    ↪ h[r], p) <= -can_on_edge)
        return false;
    while(r - l > 1) {
        int m = (l + r) / 2;
        (dir(h[0], h[m], p) < 0 ? l : r) = m;
    }
    return dir(h[l], h[r], p) < can_on_edge;
}

```

line

#441452, includes: point

Konwersja różnych postaci prostej.

```

struct Line {
    D A, B, C;
    // postac ogolna Ax + By + C = 0
    Line(D a, D b, D c) : A(a), B(b), C(c) {}
    tuple<D, D, D> get_tuple() { return {A, B, C}; }
    // postac kierunkowa ax + b = y
    Line(D a, D b) : A(a), B(-1), C(b) {}
    pair<D, D> get_dir() { return {- A / B, - C / B}; }
    // prosta pq
    Line(P p, P q) {
        assert(not equal(p, q));
        if(not equal(p.x(), q.x())) {
            A = (q.y() - p.y()) / (p.x() - q.x());
            B = 1, C = -(A * p.x() + B * p.y());
        }
        else A = 1, B = 0, C = -p.x();
    }
    pair<P, P> get_pts() {
        if(!equal(B, 0)) return { P(0, - C / B), P(1, -
        ↪ (A + C) / B) };
        return { P(- C / A, 0), P(- C / A, 1) };
    }
    D directed_dist(P p) {
        return (A * p.x() + B * p.y() + C) / sqrt(A * A +
        ↪ B * B);
    }
    D dist(P p) {
        return abs(directed_dist(p));
    }
};

```

point

#a14c07

Wrapper na std::complex, definiy trzeba dać nad bitsami, wtedy istnieje p.x() oraz p.y(). abs długość, arg kąt $(-\pi, \pi]$ gdzie (0, 1) daje $\frac{\pi}{2}$, polar(len, angle) tworzy P. Istnieją atan2, asin, sinh.

```
// Before include bits:
// #define real x
// #define imag y
using D = long double;
using P = complex<D>;
constexpr D eps = 1e-9;
bool equal(D a, D b) { return abs(a - b) < eps; }
bool equal(P a, P b) { return equal(a.x(), b.x()) and
    ⇨ equal(a.y(), b.y()); }
int sign(D a) { return equal(a, 0) ? 0 : a > 0 ? 1 :
    ⇨ -1; }
namespace std { bool operator<(P a, P b) { return
    ⇨ sign(a.x() - b.x()) == 0 ? sign(a.y() - b.y()) < 0
    ⇨ : a.x() < b.x(); } }
// cross({1, 0}, {0, 1}) = 1
D cross(P a, P b) { return a.x() * b.y() - a.y() *
    ⇨ b.x(); }
D dot(P a, P b) { return a.x() * b.x() + a.y() *
    ⇨ b.y(); }
D dist(P a, P b) { return abs(a - b); }
int dir(P a, P b, P c) { return sign(cross(b - a, c -
    ⇨ b)); }
```

Tekstówki (8)

hashing

#364cc1

Hashowanie z małą stałą. Można zmienić bazę (jeśli serio trzeba).
openssl prime -generate -bits 60 generuje losową liczbę
pierwszą o 60 bitach ($\leq 1.15 \cdot 10^{18}$).

```
struct Hashing {
    vector<LL> ha, pw;
    static constexpr LL mod = (1ll << 61) - 1;
    LL reduce(LL x) { return x >= mod ? x - mod : x; }
    LL mul(LL a, LL b) {
        const auto c = __int128(a) * b;
        return reduce(LL(c & mod) + LL(c >> 61));
    }
    Hashing(const vector<int> &str, const int base =
    ⇨ 37) {
        int len = ssize(str);
        ha.resize(len + 1);
        pw.resize(len + 1, 1);
        REP(i, len) {
            ha[i + 1] = reduce(mul(ha[i], base) + str[i] +
            ⇨ 1);
            pw[i + 1] = mul(pw[i], base);
        }
        LL operator()(int l, int r) {
            return reduce(ha[r + 1] - mul(ha[l], pw[r - l +
            ⇨ 1]) + mod);
        }
    };
};
```

kmp

#81f31b

$\mathcal{O}(n)$, zachodzi $[0, \text{pi}[i]] = (i - \text{pi}[i], i]$.
get_kmp({0,1,0,0,1,0,0,1,0,1}) ==
{0,0,1,1,2,3,2,3,4,5},
get_borders({0,1,0,0,1,0,1,0,0,1}) ==
{2,5,10}.

```
// BEGIN HASH 3eb302
vector<int> get_kmp(vector<int> str) {
    int len = ssize(str);
    vector<int> ret(len);
    for(int i = 1; i < len; i++) {
        int pos = ret[i - 1];
        while(pos and str[i] != str[pos])
            pos = ret[pos - 1];
        ret[i] = pos + (str[i] == str[pos]);
    }
}
```

```
return ret;
} // END HASH
vector<int> get_borders(vector<int> str) {
    vector<int> kmp = get_kmp(str), ret;
    int len = ssize(str);
    while(len) {
        ret.emplace_back(len);
        len = kmp[len - 1];
    }
    return vector<int>(ret.rbegin(), ret.rend());
}
```

manacher

#ca63bf

$\mathcal{O}(n)$, radius[p][i] = rad = największy promień palindromu
parzystości p o środku i. $L = i - \text{rad} + 1$, $R = i + \text{rad}$ to palindrom.
Dla [abaababaa] daje [003000020], [0100141000].

```
array<vector<int>, 2> manacher(vector<int> &in) {
    int n = ssize(in);
    array<vector<int>, 2> radius = {{vector<int>(n -
    ⇨ 1), vector<int>(n)}};
    REP(parity, 2) {
        int z = parity ^ 1, L = 0, R = 0;
        REP(i, n - z) {
            int &rad = radius[parity][i];
            if(i <= R - z)
                rad = min(R - i, radius[parity][L + (R - i -
                ⇨ z)]);
            int l = i - rad + z, r = i + rad;
            while(0 <= l - 1 && r + 1 < n && in[l - 1] ==
            ⇨ in[r + 1])
                ++rad, ++r, --l;
            if(r > R)
                L = l, R = r;
        }
    }
    return radius;
}
```

pref

#8f8b4c

$\mathcal{O}(n)$, zwraca tablicę prefixo prefixową
 $[0, \text{pref}[i]] = [i, i + \text{pref}[i]]$.

```
vector<int> pref(vector<int> str) {
    int n = ssize(str);
    vector<int> ret(n);
    ret[0] = n;
    int i = 1, m = 0;
    while(i < n) {
        while(m + i < n and str[m + i] == str[m])
            m++;
        ret[i++] = m;
        m = max(0, m - 1);
        for(int j = 1; ret[j] < m; m--)
            ret[i++] = ret[j++];
    }
    return ret;
}
```

squares

#bed028, includes: pref

$\mathcal{O}(n \log n)$, zwraca wszystkie skompresowane trójki
(start_l, start_r, len) oznaczające, że podśłowa zaczynające się w
[start_l, start_r] o długości len są kwadratami, jest ich
 $\mathcal{O}(n \log n)$.

```
vector<tuple<int, int, int>> squares(const
    ⇨ vector<int> &s) {
    vector<tuple<int, int, int>> ans;
    vector pos(ssize(s) + 2, -1);
    FOR(mid, 1, ssize(s) - 1) {
        int part = mid & ~(mid - 1), off = mid - part;
        int end = min(mid + part, ssize(s));
        vector a(s.begin() + off, s.begin() + off + part),
            b(s.begin() + mid, s.begin() + end),
```

```
ra(a.rbegin(), a.rend());
REP(j, 2) {
    auto z1 = pref(ra), bha = b;
    bha.emplace_back(-1);
    for(int x : a) bha.emplace_back(x);
    auto z2 = pref(bha);
    for(auto *v : {&z1, &z2}) {
        v[0][0] = ssize(v[0]);
        v->emplace_back(0);
    }
    REP(c, ssize(a)) {
        int l = ssize(a) - c, x = c - min(l - 1,
            ⇨ z1[l]),
            y = c - max(l - z2[ssize(b) + c + 1], j),
            sb = (j ? end - y - l * 2 : off + x),
            se = (j ? end - x - l * 2 + 1 : off + y +
            ⇨ 1),
            &p = pos[l];
        if (x > y) continue;
        if (p != -1 && get<1>(ans[p]) + 1 == sb)
            get<1>(ans[p]) = se - 1;
        else
            p = ssize(ans), ans.emplace_back(sb, se -
            ⇨ 1, l);
    }
    a = vector(b.rbegin(), b.rend());
    b.swap(ra);
}
}
return ans;
}
```

Optymalizacje (9)

divide-and-conquer-dp

#6ceec5

$\mathcal{O}(nm \log m)$, dla funkcji $\text{cost}(k, j)$ wylicza
 $\text{dp}(i, j) = \min_{0 \leq k < j} \text{dp}(i - 1, k - 1) + \text{cost}(k, j)$. Działa tylko wtedy,
gdy $\text{opt}(i, j - 1) \leq \text{opt}(i, j)$, a jest to zawsze spełnione, gdy
 $\text{cost}(b, c) \leq \text{cost}(a, d)$ oraz
 $\text{cost}(a, c) + \text{cost}(b, d) \leq \text{cost}(a, d) + \text{cost}(b, c)$ dla
 $a \leq b \leq c \leq d$.

```
vector<LL> divide_and_conquer_optimization(int n, int
    ⇨ m, function<LL(int,int)> cost) {
    vector<LL> dp_before(m);
    auto dp_cur = dp_before;
    REP(i, m)
        dp_before[i] = cost(0, i);
    function<void(int,int,int,int)> compute = [&](int
    ⇨ l, int r, int optl, int oprt) {
        if (l > r)
            return;
        int mid = (l + r) / 2, opt;
        pair<LL, int> best = {numeric_limits<LL>::max(),
            ⇨ -1};
        FOR(k, optl, min(mid, oprt))
            best = min(best, {(k ? dp_before[k - 1] : 0) +
            ⇨ cost(k, mid), k});
        tie(dp_cur[mid], opt) = best;
        compute(l, mid - 1, optl, opt);
        compute(mid + 1, r, opt, oprt);
    };
    REP(i, n) {
        compute(0, m - 1, 0, m - 1);
        swap(dp_before, dp_cur);
    }
    return dp_before;
}
```

fio

#115ad1

FIO do wpychania kolanem. Nie należy wtedy używać
cin/cout

```
#ifndef ONLINE_JUDGE
// write this when judge is on Windows
inline int getchar_unlocked() { return
    ⇨ _getchar_nolock(); }
inline void putchar_unlocked(char c) {
    ⇨ _putchar_nolock(c); }
#endif
// BEGIN HASH 1ed0dd
int fastin() {
    int n = 0, c = getchar_unlocked();
    while(isspace(c))
        c = getchar_unlocked();
    while(!isdigit(c)) {
        n = 10 * n + (c - '0');
        c = getchar_unlocked();
    }
    return n;
} // END HASH
// BEGIN HASH 3abf5f
int fastin_negative() {
    int n = 0, negative = false, c = getchar_unlocked();
    while(isspace(c))
        c = getchar_unlocked();
    if(c == '-') {
        negative = true;
        c = getchar_unlocked();
    }
    while(isdigit(c)) {
        n = 10 * n + (c - '0');
        c = getchar_unlocked();
    }
    return negative ? -n : n;
} // END HASH
// BEGIN HASH 323fab
double fastin_double() {
    double x = 0, t = 1;
    int negative = false, c = getchar_unlocked();
    while(isspace(c))
        c = getchar_unlocked();
    if (c == '-') {
        negative = true;
        c = getchar_unlocked();
    }
    while (isdigit(c)) {
        x = x * 10 + (c - '0');
        c = getchar_unlocked();
    }
    if (c == '.') {
        c = getchar_unlocked();
        while (isdigit(c)) {
            t /= 10;
            x = x + t * (c - '0');
            c = getchar_unlocked();
        }
    }
    return negative ? -x : x;
} // END HASH
// BEGIN HASH 0b2d96
void fastout(int x) {
    if(x == 0) {
        putchar_unlocked('0');
        putchar_unlocked(' ');
        return;
    }
    if(x < 0) {
        putchar_unlocked('-');
        x *= -1;
    }
    static char t[10];
    int i = 0;
    while(x) {
        t[i++] = char('0' + (x % 10));
        x /= 10;
    }
    while(--i >= 0)
```

```

    putchar_unlocked(t[i]);
    putchar_unlocked(' ');
}
void nl() { putchar_unlocked('\n'); }
// END HASH

```

knuth

#99b095

$\mathcal{O}(n^2)$, dla tablicy $cost(i, j)$ wylicza $dp(i, j) = \min_{i \leq k < j} dp(i, k) + dp(k + 1, j) + cost(i, j)$. Działa tylko wtedy, gdy $opt(i, j - 1) \leq opt(i, j) \leq opt(i + 1, j)$, a jest to zawsze spełnione, gdy $cost(b, c) \leq cost(a, d)$ oraz $cost(a, c) + cost(b, d) \leq cost(a, d) + cost(b, c)$ dla $a \leq b \leq c \leq d$.

```

LL knuth_optimization(vector<vector<LL>> cost) {
    int n = ssize(cost);
    vector dp(n, vector<LL>(n,
        ⇨ numeric_limits<LL>::max()));
    vector opt(n, vector<int>(n));
    REP(i, n) {
        opt[i][i] = i;
        dp[i][i] = cost[i][i];
    }
    for(int i = n - 2; i >= 0; --i)
        FOR(j, i + 1, n - 1)
            FOR(k, opt[i][j] - 1, min(j - 1, opt[i + 1][j]))
                if(dp[i][j] >= dp[i][k] + dp[k + 1][j] +
                    ⇨ cost[i][j]) {
                    opt[i][j] = k;
                    dp[i][j] = dp[i][k] + dp[k + 1][j] +
                        ⇨ cost[i][j];
                }
    }
    return dp[0][n - 1];
}

```

linear-knapsack

#3d7116

$\mathcal{O}(n \cdot \max(w_i))$ zamiast typowego $\mathcal{O}(n \cdot \sum(w_i))$, pamięć $\mathcal{O}(n + \max(w_i))$, plecak zwracający największą otrzymywalną sumę ciężarów $\leq \text{bound}$.

```

LL knapsack(vector<int> w, LL bound) {
    erase_if(w, [=](int x){ return x > bound; });
    {
        LL sum = accumulate(w.begin(), w.end(), 0LL);
        if(sum <= bound)
            return sum;
    }
    LL w_init = 0;
    int b;
    for(b = 0; w_init + w[b] <= bound; ++b)
        w_init += w[b];
    int W = *max_element(w.begin(), w.end());
    vector<int> prev_s(2 * W, -1);
    auto get = [&](vector<int> &v, LL i) -> int& {
        return v[i - (bound - W + 1)];
    };
    for(LL mu = bound + 1; mu <= bound + W; ++mu)
        get(prev_s, mu) = 0;
    get(prev_s, w_init) = b;
    FOR(t, b, ssize(w) - 1) {
        vector curr_s = prev_s;
        for(LL mu = bound - W + 1; mu <= bound; ++mu)
            get(curr_s, mu + w[t]) = max(get(curr_s, mu +
                ⇨ w[t]), get(prev_s, mu));
        for(LL mu = bound + w[t]; mu >= bound + 1; --mu)
            for(int j = get(curr_s, mu) - 1; j >=
                ⇨ get(prev_s, mu); --j)
                get(curr_s, mu - w[j]) = max(get(curr_s, mu -
                    ⇨ w[j]), j);
        swap(prev_s, curr_s);
    }
    for(LL mu = bound; mu >= 0; --mu)
        if(get(prev_s, mu) != -1)
            return mu;
}

```

```

    assert(false);
}

```

pragmy

#4ad365

Pragmy do wypychania kolaniem

```

#pragma GCC optimize("Ofast")
#pragma GCC target("avx,avx2")

```

random

#bc664b

Szybsze rand.

```

uint32_t xorshf96() {
    static uint32_t x = 123456789, y = 362436069, z =
        ⇨ 521288629;
    uint32_t t;
    x ^= x << 16;
    x ^= x >> 5;
    x ^= x << 1;
    t = x;
    x = y;
    y = z;
    z = t ^ x ^ y;
    return z;
}

```

sos-dp

#a206d3

$\mathcal{O}(n2^n)$, dla tablicy $A[i]$ oblicza tablicę $F[mask] = \sum_{i \in mask} A[i]$, czyli sumę po podmaskach. Może też liczyć sumę po nadmaskach. `sos_dp(2, {4, 3, 7, 2})` zwraca {4, 7, 11, 16}, `sos_dp(2, {4, 3, 7, 2}, true)` zwraca {16, 5, 9, 2}.

```

vector<LL> sos_dp(int n, vector<LL> A, bool nad =
    ⇨ false) {
    int N = (1 << n);
    if (nad) REP(i, N / 2) swap(A[i], A[(N - 1) ^ i]);
    auto F = A;
    REP(i, n)
        REP(mask, N)
            if ((mask >> i) & 1)
                F[mask] += F[mask ^ (1 << i)];
    if (nad) REP(i, N / 2) swap(F[i], F[(N - 1) ^ i]);
    return F;
}

```

Utils (10)

dzien-probny

#56d6c0, includes: data-structures/ordered-set

Rzeczy do przetestowania w dzień próbny.

```

// alternatywne żmnoenie LL, gdyby na wypadek gdyby
⇨ nie żbyo __int128
LL l1mul(LL a, LL b, LL m) {
    return (a * b - (LL)((long double) a * b / m) * m +
        ⇨ m) % m;
}
void test_int128() {
    __int128 x = (1ll << 62);
    x *= x;
    string s;
    while(x) {
        s += char(x % 10 + '0');
        x /= 10;
    }
    assert(s ==
        ⇨ "61231558446921906466935685523974676212");
}
void test_float128() {
    __float128 x = 4.2;
    assert(abs(double(x * x) - double(4.2 * 4.2)) <
        ⇨ 1e-9);
}

```

```

}
void test_clock() {
    long seed = chrono::system_clock::now()
        .time_since_epoch().count();
    (void) seed;
    auto start = chrono::system_clock::now();
    while(true) {
        auto end = chrono::system_clock::now();
        int ms = int(
            chrono::duration_cast<chrono::milliseconds>
                (end - start).count());
        if(ms > 420)
            break;
    }
}
void test_policy() {
    ordered_set<int> s;
    s.insert(1);
    s.insert(2);
    assert(s.order_of_key(1) == 0);
    assert(*s.find_by_order(1) == 2);
}
void test_math() {
    constexpr long double pi = acos(-1);
    assert(3.14 < pi && pi < 3.15);
}
}

```

python

#ebda60

Przykładowy kod w Pythonie z różną funkcjonalnością.

```

fib_mem = [1] * 2
def fill_fib(n):
    global fib_mem
    while len(fib_mem) <= n:
        fib_mem.append(fib_mem[-2] + fib_mem[-1])
def main():
    assert list(range(3, 6)) == [3, 4, 5]
    s = set()
    s.add(5)
    for x in s:
        print(x)
    s = [2 * x for x in s]
    print(eval("s[0] + 10"))
    m = {}
    m[5] = 6
    assert 5 in m
    assert list(m) == [5] # only keys!
    line_list = list(map(int, input().split())) # gets
        ⇨ a list of integers in the line
    print(line_list)
    print(' '.join(["a", "b", str(5)]))
    while True:
        try:
            line_int = int(input())
        except Exception as e:
            break
main()

```