



Uniwersytet Warszawski

UW2

Tomasz Nowak, Michał Staniewski, Arkadiusz Czarkowski

2021-11-01

1

Utils

2

Podejścia

3

Wzorki

4

Matma

5

Struktury danych

6

Grafy

7

Geometria

8

Tekstówki

9

Optymalizacje

10

Randomowe rzeczy

Utils (1)

```
headers
Opis: Nagłówki używane w każdym kodzie. Działa na każdy kontener i pary
Użycie: debug(a, b, c); wypisze [a, b, c]:  a; b; c;
<bits/stdc++.h>                                     3a8221, 16 lines

using namespace std;
using LL = long long;
#define FOR(i, l, r) for(int i = (l); i <= (r); ++i)
#define REP(i, n) FOR(i, 0, (n) - 1)
#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 << ')';
}
#ifdef DEBUG
#define debug(x...) cerr << "[" #x "]: ", [](auto... $) {((cerr
    << $ << "; ", ...); }(x), cerr << '\n'
#else
#define debug(...) {}
#endif
```

```
headers/.bashrc                                     11 lines

c() {
    clang++ -std=c++17 -Wall -Wextra -Wshadow \
        -Wconversion -Wno-sign-conversion -Wfloat-equal \
        -D_GLIBCXX_DEBUG -fsanitize=address,undefined -ggdb3 \
        -DDEBUG -DLOCAL $1.cpp -o $1
}
nc() {
    clang++ -DLOCAL -O3 -std=c++17 -static $1.cpp -o $1 # -m32
}
alias cp='cp -i'
alias mv='mv -i'
```

```
headers/.vimrc                                     3 lines

set nu rnu hls is nosol ts=4 sw=4 ch=2 sc
filetype indent plugin on
syntax on

headers/sprawdzaczka.sh                             13 lines

#!/bin/bash
for ((i=0; i<1000000; i++)); do
    ./gen < g.in > t.in
    ./main < t.in > m.out
    ./brute < t.in > b.out

    if diff -w m.out b.out > /dev/null; then
        printf "OK $i\r"
    else
        echo "WA"
        exit 0
    fi
done
```

Podejścia (2)

- Czytanie ze zrozumieniem
- dynamik, zachłan
- dziel i zwyciężaj - matematyka dyskretna, $opt(i) \leq opt(i + 1)$
- sposób "liczba dobrych obiektów = liczba wszystkich obiektów - liczba złych obiektów"
- czy warunek konieczny = warunek wystarczający?
- odpowiednie przekształcenie równania; uniezależnienie funkcji od jakiejś zmiennej, zauważenie wypukłości
- zastanowić się nad łatwiejszym problemem, bez jakiegoś elementu z treści
- sprowadzić problem do innego, łatwiejszego/mniejszego problemu
- sprowadzić problem 2D do problemu 1D (zamiatanie; niezależność wyniku dla współrzędnych X od współrzędnych Y)
- konstrukcja grafu
- określenie struktury grafu
- optymalizacja bruta do wzorcówki
- czy można poprawić (może zachłannie) rozwiązanie nieoptymalne?
- czy są ciekawe fakty w rozwiązaniach optymalnych? (może się do tego przydać brute)
- sprawdzić czy w zadaniu czegoś jest "mało" (np. czy wynik jest mały, albo jakaś zmienna, może się do tego przydać brute)
- odpowiednio "wzbogacić" jakiś algorytm
- cokolwiek poniżej 10^9 operacji ma szansę wejść

- co można wykonać offline? czy jest coś, czego kolejność nie ma znaczenia?
- co można posortować? czy jest zawsze jakaś pewna optymalna kolejność?
- narysować dużo swoich własnych przykładów i coś z nich wywnioskować
- skupienie się na pozycji jakiegoś specjalnego elementu, np. najmniejszego
- szacowanie wyniku - czy wynik jest mały? czy umiem skonstruować algorytm który zawsze znajdzie upper bound na wynik?
- sklepać brute który sprawdza obserwacje, zawsze jeśli potrzebujemy zoptymalizować dp, wypisać wartości na małym przykładzie
- pierwiastki - elementy $> i < \sqrt{N}$ osobno, rebuild co \sqrt{N} operacji, jeśli suma wartości $= N$, jest \sqrt{N} różnych wartości
- rozwiązania probabilistyczne, paradoks urodzeń
- meet in the middle, backtrack
- sprowadzić stan do jednoznacznej postaci na podstawie podanych operacji, co pozwala sprawdzić czy z jednego stanu da się otrzymać drugi

Wzorki (3)

3.1 Równości

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Wierzchołek paraboli $= (-\frac{b}{2a}, -\frac{\Delta}{4a})$.

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

3.2 Pitagoras

Trójki (a, b, c) , takie że $a^2 + b^2 = c^2$:

$$a = k \cdot (m^2 - n^2), \quad b = k \cdot (2mn), \quad c = k \cdot (m^2 + n^2),$$

gdzie $m > n > 0$, $k > 0$, $m \perp n$, oraz albo m albo n jest parzyste.

3.3 Generowanie względnie pierwszych par

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

3.4 Liczby pierwsze

p = 962592769 to liczba na NTT, czyli 2²¹ | *p* − 1, which may be useful. Do hashowania: 970592641 (31-bit), 31443539979727 (45-bit), 3006703054056749 (52-bit).

Jest 78498 pierwszych ≤ 1 000 000.

Generatorów jest *ϕ*(*ϕ*(*p^a*)), czyli dla *p* > 2 zawsze istnieje.

3.5 Dzielniki

∑_{*d* | *n*} *d* = *O*(*n* log log *n*).

Liczba dzielników *n* jest co najwyżej 100 dla *n* < 5*e*4, 500 dla *n* < 1*e*7, 2000 dla *n* < 1*e*10, 200 000 dla *n* < 1*e*19.

3.6 Lemat Burnside’a

Liczba takich samych obiektów z dokładnością do symetrii wynosi

$$\frac{1}{|G|} \sum_{g \in G} |X^g|,$$

Gdzie *G* to zbiór symetrii (ruchów) oraz *X^g* to punkty (obiekty) stałe symetrii *g*.

3.7 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>	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		

3.8 Symbol Newtona

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

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

3.9 Wzorki na pewne ciągi

3.9.1 Nieporządek

Liczba takich permutacji, że *p_i* ≠ *i* (żadna liczba nie wraca na tą samą pozycję).

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

3.9.2 Liczba podziałów

Liczba sposobów zapisania *n* jako sumę posortowanych liczb dodatnich.

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

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

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

3.9.3 Liczby Eulera pierwszego rzędu

Liczba permutacji *π* ∈ *S_n* gdzie *k* elementów jest większych niż poprzedni: *k* razy *π*(*j*) > *π*(*j* + 1), *k* + 1 razy *π*(*j*) ≥ *j*, *k* razy *π*(*j*) > *j*.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$

$$E(n,0) = E(n,n-1) = 1$$

$$E(n,k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

3.9.4 Stirling pierwszego rzędu

Liczba permutacji długości *n* mające *k* cykli.

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

$$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1 \\ c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$$

3.9.5 Stirling drugiego rzędu

Liczba permutacji długości *n* mające *k* spójnych.

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

$$S(n,1) = S(n,n) = 1$$

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

3.9.6 Liczby Catalana

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

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

$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$$

- ścieżki na planszy *n* × *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?

3.9.7 Formuła Cayley’a

Liczba różnych drzew (z dokładnością do numerowania wierzchołków) wynosi *nⁿ⁻²*. Liczba sposobów by zespójnić *k* spójnych o rozmiarach *s₁*, *s₂*, ..., *s_k* wynosi *s₁* · *s₂* · ... · *s_k* · *n^{k-2}*.

3.10 Funkcje multiplikatywne

- id*(*n*) = *n*, 1 * *ϕ* = *id*
- 1(*n*) = 1
- τ*(*n*) = liczba dzielników dodatnich, 1 * 1 = *τ*
- σ*(*n*) = suma dzielników dodatnich, *id* * 1 = *σ*
- ϕ*(*n*) = liczba liczb względnie pierwszych z *n* większych równych 1, *id* * *μ* = *ϕ*
- μ*(*n*) = 1 dla *n* = 1, 0 gdy istnieje *p*, że *p*² | *n*, oraz (−1)^{*k*} jak *n* jest iloczynem *k* parami różnych liczb pierwszych
- ϵ*(*n*) = 1 dla *n* = 1 oraz 0 dla *n* > 1, *f* * *ϵ* = *f*, 1 * *μ* = *ϵ*
- (*f* * *g*)(*n*) = ∑_{*d* | *n*} *f*(*d*)*g*(*n*/*d*)

- $f * g = g * f$
- $f * (g * h) = (f * g) * h$
- $f * (g + h) = f * g + f * h$
- jak dwie z trzech funkcji $f * g = h$ są multiplikatywne, to trzecia też
- $f * g = \epsilon \Rightarrow g(n) = -\frac{\sum_{d|n, d>1} f(d)g(\frac{n}{d})}{f(1)}$
- równoważne:
 - $g(n) = \sum_{d|n} f(d)$
 - $f(n) = \sum_{d|n} g(d)\mu(\frac{n}{d})$
 - $\sum_{k=1}^n g(k) = \sum_{d=1}^n \lfloor \frac{n}{d} \rfloor f(d)$
- $\varphi(p^k) = p^k - p^{k-1} = p^{k-1}(p - 1)$
- $\varphi(n) = n \cdot (1 - \frac{1}{p_1}) \cdot (1 - \frac{1}{p_2}) \dots (1 - \frac{1}{p_k})$

3.11 Zasada włączeń i wyłączeń

$$|\bigcup_{i=1}^n A_i| = \sum_{\emptyset \neq J \subseteq \{1, \dots, n\}} (-1)^{|J|+1} |\bigcap_{j \in J} A_j|$$

3.12 Fibonacci

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

$F_{n-1}F_{n+1} - F_n^2 = (-1)^n, F_{n+k} = F_kF_{n+1} + F_{k-1}F_n,$
 $F_n | F_{nk}, NWD(F_m, F_n) = F_{NWD(m, n)}$

Matma (4)

extended-gcd
Opis: Dla danego (a, b) znajduje takie $(gcd(a, b), x, y)$, że $ax + by = gcd(a, b)$
Czas: $\mathcal{O}(\log(\max(a, b)))$
Użycie: LL gcd, x, y; tie(gcd, x, y) = extended_gcd(a, b);

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

crt
Opis: Chińskie Twierdzenie o Resztach
Czas: $\mathcal{O}(\log n)$ Pamięć : $\mathcal{O}(1)$
Użycie: crt(a, m, b, n) zwraca takie x, że x mod m = a i x mod n = b
m i n nie muszą być wzlędnie pierwsze, ale może nie być wtedy rozwiązania
uwali się wtedy assertcik, można zmienić na return -1

```
LL crt(LL a, LL m, LL b, LL n) {
    if(n > m) swap(a, b), swap(m, n);
    LL d, x, y;
    tie(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;
}
```

floor-sum
Opis: Liczy $\sum_{i=0}^{n-1} \lfloor \frac{a \cdot i + b}{c} \rfloor$
Czas: $\mathcal{O}(\log(c))$
Użycie: floor_sum(n, a, b, c)
Działa dla $0 \leq a, b < c$ oraz $1 \leq c, n \leq 10^9$.
Dla innych n, a, b, c trzeba uważać lub użyć __int128.

```
LL floor_sum(LL n, LL a, LL b, LL c) {
    LL ans = 0;
    if(a >= c) {
        ans += (n - 1) * n * (a / c) / 2;
        a %= c;
    }
    if(b >= c) {
        ans += n * (b / c);
        b %= c;
    }
    LL d = (a * (n - 1) + b) / c;
    if(d == 0) return ans;
    ans += d * (n - 1) - floor_sum(d, c, c - b - 1, a);
    return ans;
}
```

berlekamp-massey
Opis: Zgadywanie rekurencji liniowej
Czas: $\mathcal{O}(n^2 \log k)$ Pamięć : $\mathcal{O}(n)$
Użycie: Berlekamp_Massey<mod> bm(x) zgaduje rekurencję ciągu x
bm.get(k) zwraca k-ty wyraz ciągu x (index 0)

```
template<int mod>
struct BerlekampMassey {
    int mul(int a, int b) {
        return (LL) a * b % mod;
    }
    int add(int a, int b) {
        return a + b < mod ? a + b : a + b - mod;
    }
    int qpow(int a, int b) {
        if(b == 0) return 1;
        if(b % 2 == 1) return mul(qpow(a, b - 1), a);
        return qpow(mul(a, a), b / 2);
    }

    int n;
    vector<int> x, C;
    BerlekampMassey(vector<int> &_x) : x(_x) {
        vector<int> B; B = C = {1};
        int b = 1, m = 0;
        REP(i, ssize(x)) {
            m++; int d = x[i];
            FOR(j, 1, ssize(C) - 1)
                d = add(d, mul(C[j], x[i - j]));
            if(d == 0) continue;
            auto _B = C;
            C.resize(max(ssize(C), m + ssize(B)));
            int coef = mul(d, qpow(b, mod - 2));
            FOR(j, m, m + ssize(B) - 1)
                C[j] = (C[j] - mul(coef, B[j - m]) + mod) % mod;
            if(ssize(_B) < m + ssize(B)) { B = _B; b = d; m = 0; }
        }
        C.erase(C.begin());
        for(int &t : C) t = add(mod, -t);
        n = ssize(C);
    }
}
```

```
vector<int> combine(vector<int> a, vector<int> b) {
    vector<int> ret(n * 2 + 1);
    REP(i, n + 1) REP(j, n + 1)
        ret[i + j] = add(ret[i + j], mul(a[i], b[j]));
    for(int i = 2 * n; i > n; i--) REP(j, n)
        ret[i - j - 1] = add(ret[i - j - 1], mul(ret[i], C[j]));
    return ret;
}

int get(LL k) {
    vector<int> r(n + 1), pw(n + 1);
    r[0] = pw[1] = 1;
    for(k++; k; k /= 2) {
        if(k % 2) r = combine(r, pw);
        pw = combine(pw, pw);
    }
    LL ret = 0;
    REP(i, n) ret = add(ret, mul(r[i + 1], x[i]));
    return ret;
}
};
```

miller-rabin
Opis: Test pierwszości Millera-Rabina
Czas: $\mathcal{O}(\log^2 n)$ Pamięć : $\mathcal{O}(1)$
Użycie: miller_rabin(n) zwraca czy n jest pierwsze
działa dla long longów

```
LL mul(LL a, LL b, LL mod) {
    return (a * b - (LL)((long double) a * b / mod) * mod + mod) % mod;
}

LL qpow(LL a, LL n, LL mod) {
    if(n == 0) return 1;
    if(n % 2 == 1) return mul(qpow(a, n - 1, mod), a, mod);
    return qpow(mul(a, a, mod), n / 2, mod);
}
```

```
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, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if(n == a) return true;
        LL x = qpow(a, d, n);
        if(x == 1 || x == n - 1)
            continue;
        bool composite = true;
        REP(i, r - 1) {
            x = mul(x, x, n);
            if(x == n - 1) {
                composite = false;
                break;
            }
        }
        if(composite) return false;
    }
    return true;
}
```

rho-pollard
Opis: Rozkład na czynniki Rho Pollarda
Czas: $\mathcal{O}\left(n^{\frac{1}{4}}\right)$

Użycie: factor(n) zwraca vector dzielników pierwszych n, niekoniecznie posortowany
factor(12) = {2, 2, 3}, factor(545423) = {53, 41, 251};
"../miller-rabin/main.cpp" 9ebbcf, 19 lines

```
LL rho_pollard(LL n) {
    if(n % 2 == 0) return 2;
    for(LL i = 1;; i++) {
        auto f = [&](LL x) { return (mul(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;
}
```

fft
Opis: Mnożenie wielomianów
Czas: $\mathcal{O}(n \log n)$
Użycie: conv(a, b) zwraca iloczyn wielomianów a i b a39251, 38 lines

```
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, M_PI / 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]; // można zoptymować
            rozpisując
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
    }
}
```

```
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;
}
```

fft-mod
Opis: Mnożenie wielomianów
Czas: $\mathcal{O}(n \log n)$
Użycie: conv_mod(a, b) zwraca iloczyn wielomianów a i b modulo, ma większą dokładność niż zwykłe fft
"../fft/main.cpp" 6fe8fa, 22 lines

```
vector<LL> conv_mod(vector<LL> &a, vector<LL> &b, int M) {
    if(a.empty() || b.empty()) return {};
    vector<LL> res(ssize(a) + ssize(b) - 1);
    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] = ((av % M * cut + bv) % M * cut + cv) % M;
    }
    return res;
}
```

fwht
Opis: FWHT
Czas: $\mathcal{O}(n \log n)$ Pamigł: $\mathcal{O}(1)$
Użycie: n musi być potęgą dwójki.

```
fwht_or(a)[i] = suma(j będące podmaską i) a[j].
ifwht_or(fwht_or(a)) == a.
convolution_or(a, b)[i] = suma(j | k == i) a[j] * b[k].

fwht_and(a)[i] = suma(j będące nadmaską i) a[j].
ifwht_and(fwht_and(a)) == a.
convolution_and(a, b)[i] = suma(j & k == i) a[j] * b[k].

fwht_xor(a)[i] = suma(j oraz i mają parzyście wspólnie
zapalonych bitów) a[j] - suma(j oraz i mają nieparzyście)
a[j].
ifwht_xor(fwht_xor(a)) == a.
convolution_xor(a, b)[i] = suma(j ^ k == i) a[j] * b[k].
```

b9f7b7, 89 lines

```
vector<int> fwht_or(vector<int> a) {
    int n = ssize(a);
    assert((n & (n - 1)) == 0);
    for(int s = 1; 2 * s <= n; s *= 2)
        for(int l = 0; l < n; l += 2 * s)
            for(int i = l; i < l + s; ++i)
                a[i + s] += a[i];
    return a;
}

vector<int> ifwht_or(vector<int> a) {
    int n = ssize(a);
    assert((n & (n - 1)) == 0);
    for(int s = n / 2; s >= 1; s /= 2)
        for(int l = 0; l < n; l += 2 * s)
            for(int i = l; i < l + s; ++i)
                a[i + s] -= a[i];
    return a;
}
```

```
vector<int> convolution_or(vector<int> a, vector<int> b) {
    int n = ssize(a);
    assert((n & (n - 1)) == 0 and ssize(b) == n);
    a = fwht_or(a);
    b = fwht_or(b);
    REP(i, n)
        a[i] *= b[i];
    return ifwht_or(a);
}
```

```
vector<int> fwht_and(vector<int> a) {
    int n = ssize(a);
    assert((n & (n - 1)) == 0);
    for(int s = 1; 2 * s <= n; s *= 2)
        for(int l = 0; l < n; l += 2 * s)
            for(int i = l; i < l + s; ++i)
                a[i] += a[i + s];
    return a;
}

vector<int> ifwht_and(vector<int> a) {
    int n = ssize(a);
    assert((n & (n - 1)) == 0);
    for(int s = n / 2; s >= 1; s /= 2)
        for(int l = 0; l < n; l += 2 * s)
            for(int i = l; i < l + s; ++i)
                a[i] -= a[i + s];
    return a;
}

vector<int> convolution_and(vector<int> a, vector<int> b) {
    int n = ssize(a);
    assert((n & (n - 1)) == 0 and ssize(b) == n);
    a = fwht_and(a);
    b = fwht_and(b);
    REP(i, n)
        a[i] *= b[i];
    return ifwht_and(a);
}
```

```
vector<int> fwht_xor(vector<int> a) {
    int n = ssize(a);
    assert((n & (n - 1)) == 0);
    for(int s = 1; 2 * s <= n; s *= 2)
        for(int l = 0; l < n; l += 2 * s)
            for(int i = l; i < l + s; ++i) {
                int t = a[i + s];
                a[i + s] = a[i] - t;
                a[i] += t;
            }
    return a;
}

vector<int> ifwht_xor(vector<int> a) {
    int n = ssize(a);
    assert((n & (n - 1)) == 0);
    for(int s = n / 2; s >= 1; s /= 2)
        for(int l = 0; l < n; l += 2 * s)
            for(int i = l; i < l + s; ++i) {
                int t = a[i + s];
                a[i + s] = (a[i] - t) / 2;
                a[i] = (a[i] + t) / 2;
            }
    return a;
}

vector<int> convolution_xor(vector<int> a, vector<int> b) {
    int n = ssize(a);
    assert((n & (n - 1)) == 0 and ssize(b) == n);
    a = fwht_xor(a);
    b = fwht_xor(b);
    REP(i, n)
        a[i] *= b[i];
}
```

```
    return ifwht_xor(a);
}
```

integral
Opis: Wzór na całkę z zasady Simpsona - zwraca całkę na przedziale [a, b]
Czas: $\mathcal{O}(n)$
Użycie: integral([](T x) { return 3 * x * x - 8 * x + 3; }, a, b)
Daj asserta na błąd, ewentualnie zwiększ n (im większe n, tym mniejszy błąd)

```
using T = double;
T integral(function<T(T)> f, T a, T b) {
    const int n = 1000;
    T delta = (b - a) / n, sum = f(a) + f(b);
    FOR(i, 1, n - 1)
        sum += f(a + i * delta) * (i & 1 ? 4 : 2);
    return sum * delta / 3;
}
```

primitive-root
Opis: Dla pierwszego p znajduje generator modulo p
Czas: $\mathcal{O}(\log^2(p))$ (ale spora stała, zależy)
"../rho-pollard/main.cpp", "../../random-stuff/rd/main.cpp"

```
LL exp(LL a, LL b, int m) {
    if(b == 0) return 1;
    if(b & 1) return a * exp(a, b - 1, m) % m;
    return exp(a * a % m, b / 2, m);
}
int primitive_root(int p) {
    int q = p - 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(1) {
        int g = my_rd(2, q); bool good = 1;
        for(auto &f : fact)
            if(exp(g, q / f, p) == 1) {
                good = 0; break;
            }
        if(good) return g;
    }
}
```

discrete-log
Opis: Dla liczby pierwszej p oraz $a, b \nmid p$ znajdzie e takie że $a^e \equiv b \pmod p$
Czas: $\mathcal{O}(\sqrt{n} \log n)$
Pamięć: $\mathcal{O}(\sqrt{n})$

```
int discrete_log(int a, int b, int p) {
    map<int, int> sl;
    LL mult = 1, sq = sqrt(p);
    REP(i, sq) {
        sl[mult] = i; mult = mult * a % p;
    }
    int t = 1;
    debug(sl, t);
    REP(i, sq + 2) {
        int inv = b * exp(t, p - 2, p) % p;
        if(sl.count(inv)) return i * sq + sl[inv];
        t = t * mult % p;
    }
    return -1;
}
```

Struktury danych (5)

find-union
Opis: Find and union z mniejszy do wiekszego
Czas: $\mathcal{O}(\alpha(n))$ oraz $\mathcal{O}(n)$ pamięciowo

```
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) {}
};
```

fenwick-tree
Opis: Drzewo potęgowe
Czas: $\mathcal{O}(\log n)$
Użycie: wszystko indexowane od 0
update(pos, val) dodaje val do elementu pos
query(pos) zwraca sumę na przedziale [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;
    }
};
```

fenwick-tree-2d
Opis: Drzewo potęgowe 2d offline
Czas: $\mathcal{O}(\log^2 n)$ Pamięć $\mathcal{O}(n \log n)$
Użycie: wywołujemy preprocess(x, y) na pozycjach, które chcemy updateować, później init()
update(x, y, val) dodaje val do a[x, y], query(x, y) zwraca sumę na prostokącie (0, 0) - (x, y)
"../fenwick-tree/main.cpp"

```
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) + 1);
        }
    }
};
```

```
int ind(int x, int y) {
    auto it = lower_bound(ys[x].begin(), ys[x].end(), y);
    return 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;
}
};
```

lazy-segment-tree
Opis: Drzewo przedział-przedział
Czas: $\mathcal{O}(\log n)$ Pamięć : $\mathcal{O}(n)$
Użycie: add(l, r, val) dodaje na przedziale
quert(l, r) bierze maxa z przedziału
Zmieniając z maxa na co innego trzeba edytować funkcje add_val i f

```
using T = int;
struct Node {
    T val, lazy;
    int sz = 1;
};
```

```
struct Tree {
    vector<Node> tree;
    int sz = 1;

    void add_val(int v, T val) {
        tree[v].val += val;
        tree[v].lazy += val;
    }
}
```

```
T f(T a, T b) { return max(a, b); }
```

```
Tree(int n) {
    while(sz < n) sz *= 2;
    tree.resize(sz * 2);
    for(int i = sz - 1; i >= 1; i--)
        tree[i].sz = tree[i * 2].sz * 2;
}
```

```
void propagate(int v) {
    REP(i, 2)
        add_val(v * 2 + i, tree[v].lazy);
    tree[v].lazy = 0;
}
```

```
T query(int l, int r, int v = 1) {
    if(l == 0 && r == tree[v].sz - 1)
        return tree[v].val;
    propagate(v);
    int m = tree[v].sz / 2;
    if(r < m)
        return query(l, r, v * 2);
    else if(m <= l)
        return query(l - m, r - m, v * 2 + 1);
    else
        return f(query(l, m - 1, v * 2), query(0, r - m, v * 2 + 1));
}
```

```
void add(int l, int r, T val, int v = 1) {
```

```
if(l == 0 && r == tree[v].sz - 1) {
    add_val(v, val);
    return;
}
propagate(v);
int m = tree[v].sz / 2;
if(r < m)
    add(l, r, val, v * 2);
else if(m <= 1)
    add(l - m, r - m, val, v * 2 + 1);
else
    add(l, m - 1, val, v * 2), add(0, r - m, val, v * 2 + 1);

tree[v].val = f(tree[v * 2].val, tree[v * 2 + 1].val);
}
};
```

rmq
Opis: Range Minimum Query z użyciem sparse table
Czas: $\mathcal{O}(n \log n)$
Pamięć: $\mathcal{O}(n \log n)$
Użycie: RMQ(vec) tworzy sparse table na ciągu vec
query(l, r) odpowiada na RMQ w $\mathcal{O}(1)$

```
struct RMQ {
    vector<vector<int>> st;
    vector<int> pre;
    RMQ(vector<int> &a) {
        int n = ssize(a), lg = 0;
        while((1 << lg) < n) lg++;
        st.resize(lg + 1, vector<int>(a));
        st[0] = 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]);
        }
        pre.resize(n + 1);
        FOR(i, 2, n) pre[i] = pre[i / 2] + 1;
    }

    int query(int l, int r) {
        int q = pre[r - 1 + 1], x = r - (1 << q) + 1;
        return min(st[q][l], st[q][x]);
    }
};
```

ordered-set
Opis: set z dodatkowymi funkcjami
Użycie: 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 multiset, to używamy par {val, id}.

```
Przed includem trzeba dać undef _GLIBCXX_DEBUG
<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp>
0a779f, 9 lines

using namespace __gnu_pbds;

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

```
hash-map
Opis: szybsza mapa
Czas:  $\mathcal{O}(1)$ 
Użycie: np hash_map<int, int>
trzeba przed includem dać undef _GLIBCXX_DEBUG
<ext/pb_ds/assoc_container.hpp>
c0ab57, 11 lines

using namespace __gnu_pbds;

struct chash {
    const uint64_t C = LL(2e18 * M_PI) + 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
Opis: Set dla funkcji liniowych
Czas:  $\mathcal{O}(\log n)$ 
Użycie: add(a, b) dodaje funkcję y = ax + b
query(x) zwraca największe y w punkcie x, x < inf
45779b, 30 lines

struct Line {
    mutable LL a, b, p;
    LL eval(LL x) const { 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);
    }
};
```

lichao-tree
Opis: Dla funkcji, których pary przecinają się co najwyżej raz, oblicza maximum w punkcie x. Podany kod jest dla funkcji liniowych
6440db, 51 lines

```
constexpr LL inf = LL(1e9);
struct Function {
    int a, b;
    LL operator()(int x) {
        return x * LL(a) + b;
    }
    Function(int p = 0, int q = inf) : a(p), b(q) {}
};
ostream& operator<<(ostream &os, Function f) {
```

```
return os << make_pair(f.a, f.b);
}

struct LiChaoTree {
    int size = 1;
    vector<Function> tree;

    LiChaoTree(int n) {
        while(size < n)
            size *= 2;
        tree.resize(size << 1);
    }

    LL get_min(int x) {
        int v = x + size;
        LL ans = inf;
        while(v) {
            ans = min(ans, tree[v](x));
            v >>= 1;
        }
        return ans;
    }

    void add_func(Function new_func, int v, int l, int r) {
        int m = (l + r) / 2;
        bool domin_l = tree[v](l) > new_func(l),
            domin_m = tree[v](m) > new_func(m);
        if(domin_m)
            swap(tree[v], new_func);

        if(l == r)
            return;
        else if(domin_l == domin_m)
            add_func(new_func, v << 1 | 1, m + 1, r);
        else
            add_func(new_func, v << 1, l, m);
    }

    void add_func(Function new_func) {
        add_func(new_func, 1, 0, size - 1);
    }
};
```

treap
Opis: Implicit Treap
Czas: wszystko w $\mathcal{O}(\log n)$
Użycie: wszystko indexowane od 0
insert(key, val) insertuję na pozycję key
treap[i] zwraca i-tą wartość
907bf8, 42 lines

```
mt19937 rng_key(0);

struct Treap {
    struct Node {
        int prio, val, cnt;
        Node *l = nullptr, *r = nullptr;
        Node(int _val) : prio(rng_key()), val(_val) {}
    };
    using pNode = Node*;
    pNode root = nullptr;

    int cnt(pNode t) { return t ? t->cnt : 0; }
    void update(pNode t) {
        if(!t) return;
        t->cnt = cnt(t->l) + cnt(t->r) + 1;
    }

    void split(pNode t, int key, pNode &l, pNode &r) {
        if(!t) l = r = nullptr;
```

```
    else if(key <= cnt(t->l))
        split(t->l, key, l, t->l), r = t;
    else
        split(t->r, key - cnt(t->l) - 1, t->r, r), l = t;
    update(t);
}

void merge(pNode &t, pNode l, pNode r) {
    if(!l || !r) t = (l ? l : r);
    else if(l->prio > r->prio)
        merge(l->r, l->r, r), t = l;
    else
        merge(r->l, l, r->l), t = r;
    update(t);
}

void insert(int key, int val) {
    pNode t;
    split(root, key, root, t);
    merge(root, root, new Node(val));
    merge(root, root, t);
}
};
```

persistent-treap
Opis: Implicit Persistent Treap
Czas: wszystko w $\mathcal{O}(\log n)$
Użycie: wszystko indexowane od 0
insert(key, val) insertuję na pozycję key
kopiowanie struktury działa w $\mathcal{O}(1)$
robimy sobie vector<Treap>, żeby obsługiwać trwałość

mt19937 rng_key(0);

```
struct Treap {
    struct Node {
        int val, prio, sub = 1;
        Node *l = nullptr, *r = nullptr;
        Node(int _val) : val(_val), prio(rng_key()) {}
    };
    using pNode = Node*;
    pNode root = nullptr;

    int get_sub(pNode n) { return n ? n->sub : 0; }
    void update(pNode n) {
        if(!n) return;
        n->sub = get_sub(n->l) + get_sub(n->r) + 1;
    }

    void split(pNode t, int key, pNode &l, pNode &r) {
        if(!t) l = r = nullptr;
        else {
            t = new Node(*t);
            if(key <= get_sub(t->l))
                split(t->l, key, l, t->l), r = t;
            else
                split(t->r, key - get_sub(t->l) - 1, t->r, r), l = t;
        }
        update(t);
    }

    void merge(pNode &t, pNode l, pNode r) {
        if(!l || !r) t = (l ? l : r);
        else if(l->prio > r->prio) {
            l = new Node(*l);
            merge(l->r, l->r, r), t = l;
        }
        else {
            r = new Node(*r);
            merge(r->l, l, r->l), t = r;
        }
        update(t);
    }
};
```

persistent-treap eulerian-path jump-ptr scc

```
    merge(r->l, l, r->l), t = r;
}
update(t);
}

void insert(pNode &t, int key, pNode it) {
    if(!t) t = it;
    else if(it->prio > t->prio)
        split(t, key, it->l, it->r), t = it;
    else {
        t = new Node(*t);
        if(key <= get_sub(t->l))
            insert(t->l, key, it);
        else
            insert(t->r, key - get_sub(t->l) - 1, it);
    }
    update(t);
}

void insert(int key, int val) {
    insert(root, key, new Node(val));
}

void erase(pNode &t, int key) {
    if(get_sub(t->l) == key)
        merge(t, t->l, t->r);
    else {
        t = new Node(*t);
        if(key <= get_sub(t->l))
            erase(t->l, key);
        else
            erase(t->r, key - get_sub(t->l) - 1);
    }
    update(t);
}

void erase(int key) {
    assert(key < get_sub(root));
    erase(root, key);
}
};
```

Grafy (6)

eulerian-path
Opis: Ścieżka eulera
Czas: $\mathcal{O}(n)$
Użycie: Krawędzie to pary (to, id) gdzie id dla grafu nieskierowanego jest takie samo dla (u, v) i (v, u)
Graf musi być spójny, po zainicjalizowaniu w .path jest ścieżka/cykl eulera, vector o długości m + 1 kolejnych wierzchołków
Jeśli nie ma ścieżki/cyklu, path jest puste. Dla cyklu, path[0] == path[m]

```
using PII = pair<int, int>;
struct EulerianPath {
    vector<vector<PII>> adj;
    vector<bool> used;
    vector<int> path;
    void dfs(int v) {
        while(!adj[v].empty()) {
            int u, id; tie(u, id) = adj[v].back();
            adj[v].pop_back();
            if(used[id]) continue;
            used[id] = true;
            dfs(u);
        }
        path.emplace_back(v);
    }
};
```

```
EulerianPath(int m, vector<vector<PII>> _adj) : adj(_adj) {
    used.resize(m); dfs(0);
    if(ssize(path) != m + 1) path.clear();
    reverse(path.begin(), path.end());
}
};
```

jump-ptr
Opis: Jump Pointery
Czas: $\mathcal{O}(n \log n + q \log n)$
Użycie: konstruktor - JumpPtr(graph)
można ustawić roota
jump_up(v, k) zwraca wierzchołek o k wyższy niż v
jeśli nie istnieje, zwraca -1
lca(a, b) zwraca lca wierzchołków

```
struct JumpPtr {
    int LOG = 20;
    vector<vector<int>> &graph, jump;
    vector<int> par, dep;
    void par_dfs(int v) {
        for(int u : graph[v]) {
            if(u != par[v]) {
                par[u] = v;
                dep[u] = dep[v] + 1;
                par_dfs(u);
            }
        }
    }
    JumpPtr(vector<vector<int>> &_graph, int root = 0) : graph(_graph) {
        int n = ssize(graph);
        par.resize(n, -1);
        dep.resize(n);
        par_dfs(root);
        jump.resize(LOG, vector<int>(n));
        jump[0] = par;
        FOR(i, 1, LOG - 1) REP(j, n)
            jump[i][j] = jump[i - 1][j] == -1 ? -1 : jump[i - 1][jump[i - 1][j]];
    }
    int jump_up(int v, int k) {
        for(int i = LOG - 1; i >= 0; i--)
            if(k & (1 << i))
                v = jump[i][v];
        return v;
    }
    int lca(int a, int b) {
        if(dep[a] < dep[b]) swap(a, b);
        int delta = dep[a] - dep[b];
        a = jump_up(a, delta);
        if(a == b) return a;

        for(int i = LOG - 1; i >= 0; i--) {
            if(jump[i][a] != jump[i][b]) {
                a = jump[i][a];
                b = jump[i][b];
            }
        }
        return par[a];
    }
};
```

SCC
Opis: Silnie Spójnie Składowe
Czas: $\mathcal{O}(\log n)$

Użycie: konstruktor - SCC(graph)

group[v] to numer silnie spójnej wierzchołka v
get_compressed() zwraca graf silnie spójnych
get_compressed(false) nie usuwa multikrawędzi

a1bad8, 61 lines

```
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;
    }
};
```

biconnected

Opis: Dwuspójne składowe

Czas: $\mathcal{O}(n)$

Użycie: add_edge(u, v) dodaje krawędź (u, v), u != v, bo get() nie działa
po wywołaniu init() w .bicon mamy dwuspójne(vector ideków krawędzi na każdą), w .edges mamy krawędzie

15f4ec, 45 lines

```
struct BiconComps {
    using PII = pair<int, int>;
    vector<vector<int>> graph, bicon;
    vector<int> low, pre, s;
    vector<array<int, 2>> edges;
    BiconComps(int n) : graph(n), low(n), pre(n, -1) {}
    void add_edge(int u, int v) {
        int q = ssize(edges);
        graph[u].emplace_back(q);
        graph[v].emplace_back(q);
        edges.push_back({u, v});
    }

    int get(int v, int id) {
        for(int r : edges[id])
            if(r != v) return r;
    }

    int t = 0;
    void dfs(int v, int p) {
        low[v] = pre[v] = t++;
        bool par = false;
        for(int e : graph[v]) {
            int u = get(v, e);
            if(u == p && !par) {
                par = true;
                continue;
            }
            else if(pre[u] == -1) {
                s.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(s.back());
                        s.pop_back();
                    } while(bicon.back().back() != e);
                }
            }
            else if(pre[v] > pre[u]) {
                low[v] = min(low[v], pre[u]);
                s.emplace_back(e);
            }
        }
    }

    void init() { dfs(0, -1); }
};
```

2sat

Opis: Zwraca poprawne przyporządkowanie zmiennym logicznym dla problemu 2-SAT, albo mówi, że takie nie istnieje

Czas: $\mathcal{O}(n + m)$, gdzie n to ilość zmiennych, i m to ilość przyporządkowań.
Użycie: TwoSat ts(ilość zmiennych);

ôznacza negację
ts.either(0, ~3); // var 0 is true or var 3 is false
ts.set_value(2); // var 2 is true
ts.at_most_one({0,~1,2}); // co najwyżej jedna z var 0, ~1 i 2 to prawda
ts.solve(); // rozwiązuje i zwraca true jeśli rozwiązanie istnieje
ts.values[0..N-1] // to wartości rozwiązania

304dcc, 59 lines

```
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); }
```

```
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;
}

};
```

hld

Opis: Heavy-Light Decomposition

Czas: $\mathcal{O}(q \log n)$

Użycie: konstruktor - HLD(n, adj)
lca(v, u) zwraca lca
get_vertex(v) zwraca pozycję odpowiadającą wierzchołkowi
get_path(v, u) zwraca przedziały do obsługiwanego 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 odpowiadający poddrzewu v

```
struct HLD {
    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);
    }
    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}; }
};
```

centro-decomp
Opis: template do Centroid Decomposition
Czas: $\mathcal{O}(n \log n)$
Użycie: konstruktor - HLD(n, graf)
swój kod wrzucamy do funkcji decomp

```
struct CentroDecomp {
    vector<vector<int>> &adj;
    vector<bool> done;
    vector<int> sub, par;
    CentroDecomp(int n, vector<vector<int>> &_adj) : adj(_adj), done(n), sub(n), par(n) {}

    void dfs(int v) {
        sub[v] = 1;
        for(int u : adj[v]) {
            if(!done[u] && u != par[v]) {
                par[u] = v; dfs(u);
                sub[v] += sub[u];
            }
        }
    }
    int centro(int v) {
        par[v] = -1; dfs(v);
        for(int sz = sub[v];; ) {
            pair<int, int> mx = {0, 0};
            for(int u : adj[v])
                if(!done[u] && u != par[v])
                    mx = max(mx, {sub[u], u});
            if(mx.first * 2 <= sz) return v;
            v = mx.second;
        }
    }
    void decomp(int v) {
        done[v = centro(v)] = true;
        // kodzik idzie tutaj
        for(int u : adj[v])
            if(!done[u])
                decomp(u);
    }
};
```

matching
Opis: Turbo Matching
Czas: Średnio około $\mathcal{O}(n \log n)$, najgorzej $\mathcal{O}(n^2)$
Użycie: wierzchołki grafu nie muszą być ładnie podzielone na dwa przedziały, musi być po prostu dwudzielny.

```
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);
}
int get() {
    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;
};
```

flow
Opis: Dinic bez skalowania
Czas: $\mathcal{O}(V^2 E)$
Użycie: Dinic flow(2); flow.add_edge(0, 1, 5); cout << flow(0, 1); // 5
funkcja get_flowng() zwraca dla każdej oryginalnej krawędzi, ile przez nią leci

```
struct Dinic {
    using T = int;
    struct Edge {
        int v, u;
        T flow, cap;
    };
    int n;
    vector<vector<int>> graph;
    vector<Edge> edges;

    Dinic(int N) : n(N), graph(n) {}

    void add_edge(int v, int u, T cap) {
        debug(v, u, cap);
        int e = ssize(edges);
        graph[v].emplace_back(e);
        graph[u].emplace_back(e + 1);
        edges.emplace_back(Edge{v, u, 0, cap});
        edges.emplace_back(Edge{u, v, 0, 0});
    }

    vector<int> dist;
    bool bfs(int source, int sink) {
        dist.assign(n, 0);
        dist[source] = 1;
        deque<int> que = {source};
        while(ssize(que) & dist[sink] == 0) {
            int v = que.front();
            que.pop_front();
            for(int e : graph[v])
                if(edges[e].flow != edges[e].cap & dist[edges[e].u] == 0) {
                    dist[edges[e].u] = dist[v] + 1;
                    que.emplace_back(edges[e].u);
                }
        }
        return dist[sink] != 0;
    }

    vector<int> ended_at;
    T dfs(int v, int sink, T flow = numeric_limits<T>::max()) {
        if(flow == 0 or v == sink)
            return flow;
        for(; ended_at[v] != ssize(graph[v]); ++ended_at[v]) {
            Edge &e = edges[graph[v][ended_at[v]]];
            if(dist[v] + 1 == dist[e.u])
```

```

    if(T pushed = dfs(e.u, sink, min(flow, e.cap - e.flow))
        ) {
        e.flow += pushed;
        edges[graph[v][ended_at[v]] ^ 1].flow -= pushed;
        return pushed;
    }
}
return 0;
}

T operator()(int source, int sink) {
    T answer = 0;
    while(true) {
        if(not bfs(source, sink))
            break;
        ended_at.assign(n, 0);
        while(T pushed = dfs(source, sink))
            answer += pushed;
    }
    return answer;
}

map<pair<int, int>, T> get_flowng() {
    map<pair<int, int>, T> ret;
    REP(v, n)
        for(int i : graph[v]) {
            if(i % 2) // considering only original edges
                continue;
            Edge &e = edges[i];
            ret[make_pair(v, e.u)] = e.flow;
        }
    return ret;
}
};

```

mcmf

Opis: Min-cost max-flow z SPFA

Czas: kto wie

Użycie: MCMF flow(2); flow.add_edge(0, 1, 5, 3); cout << flow(0, 1); // 15

można przepisać funkcję get_flowng() z Dinic'a

f08e56, 79 lines

```

struct MCMF {
    struct Edge {
        int v, u, flow, cap;
        LL cost;
        friend ostream& operator<<(ostream &os, Edge &e) {
            return os << vector<LL>{e.v, e.u, e.flow, e.cap, e.cost};
        }
    };
};

```

```

int n;
const LL inf_LL = 1e18;
const int inf_int = 1e9;
vector<vector<int>> graph;
vector<Edge> edges;

```

```
MCMF(int N) : n(N), graph(n) {}
```

```

void add_edge(int v, int u, int cap, LL cost) {
    int e = ssize(edges);
    graph[v].emplace_back(e);
    graph[u].emplace_back(e + 1);
    edges.emplace_back(Edge{v, u, 0, cap, cost});
    edges.emplace_back(Edge{u, v, 0, 0, -cost});
}

```

```

pair<int, LL> augment(int source, int sink) {
    vector<LL> dist(n, inf_LL);
}

```

```

vector<int> from(n, -1);
dist[source] = 0;
deque<int> que = {source};
vector<bool> inside(n);
inside[source] = true;

while(ssize(que)) {
    int v = que.front();
    inside[v] = false;
    que.pop_front();

    for(int i : graph[v]) {
        Edge &e = edges[i];
        if(e.flow != e.cap and dist[e.u] > dist[v] + e.cost) {
            dist[e.u] = dist[v] + e.cost;
            from[e.u] = i;
            if(not inside[e.u]) {
                inside[e.u] = true;
                que.emplace_back(e.u);
            }
        }
    }
}

if(from[sink] == -1)
    return {0, 0};

int flow = inf_int, e = from[sink];
while(e != -1) {
    flow = min(flow, edges[e].cap - edges[e].flow);
    e = from[edges[e].v];
}
e = from[sink];
while(e != -1) {
    edges[e].flow += flow;
    edges[e ^ 1].flow -= flow;
    e = from[edges[e].v];
}
return {flow, flow * dist[sink]};
}

```

```

pair<int, LL> operator()(int source, int sink) {
    int flow = 0;
    LL cost = 0;
    pair<int, LL> got;
    do {
        got = augment(source, sink);
        flow += got.first;
        cost += got.second;
    } while(got.first);
    return {flow, cost};
}
};

```

Geometria (7)

point

Opis: Double może być LL, ale nie int. p.x oraz p.y nie można zmieniać (to kopie). Nie tworzyć zmiennych o nazwie "x" lub "y".

Użycie: P p = {5, 6}; abs(p) = length; arg(p) = kąt; polar(len, angle); exp(angle)

fda436, 33 lines

```

using Double = long double;
using P = complex<Double>;
#define x real()
#define y imag()

```

```

constexpr Double eps = 1e-9;
bool equal(Double a, Double b) {

```

```

    return abs(a - b) <= eps;
}

int sign(Double a) {
    return equal(a, 0) ? 0 : a > 0 ? 1 : -1;
}

struct Sortx {
    bool operator()(const P &a, const P &b) const {
        return make_pair(a.x, a.y) < make_pair(b.x, b.y);
    }
};

istream& operator>>(istream &is, P &p) {
    Double a, b;
    is >> a >> b;
    p = P(a, b);
    return is;
}

bool operator==(P a, P b) {
    return equal(a.x, b.x) && equal(a.y, b.y);
}

// cross({1, 0}, {0, 1}) = 1
Double cross(P a, P b) { return a.x * b.y - a.y * b.x; }
Double dot(P a, P b) { return a.x * b.x + a.y * b.y; }
Double sq_dist(P a, P b) { return dot(a - b, a - b); }
Double dist(P a, P b) { return abs(a - b); }

```

advanced-complex

Opis: Randomowe przydatne wzorki, większość nie działa dla intów

../point/main.cpp daaa0f, 43 lines

```

// nachylenie k -> y = kx + m
Double 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, Double theta) {
    return (a - p) * polar(1.0L, theta) + p;
}
// kat ABC w radianach, zawsze zwraca mniejszy kat
Double angle(P a, P b, P c) {
    return abs(remainder(arg(a - b) - arg(c - b), 2.0 * M_PI));
}
// szybkie przeciecie prostych, nie dziala dla rownoległych
P intersection(P a, P b, P p, P q) {
    Double 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 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 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;
}
```

line

Opis: konwersja różnych postaci prostej

```
"/point/main.cpp" dd1432, 23 lines

struct Line {
    using D = Double;
    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_sta() { 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.x, q.x) or not equal(p.y, q.y));
        if(!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) };
    }
};
```

intersect-lines

Opis: Przecięcie prostych lub odcinków

Użycie: intersection(a, b, c, d) zwraca przecięcie prostych ab oraz cd

v = intersect(a, b, c, d, s) zwraca przecięcie (s ? odcinków : prostych) ab oraz cd

if ssize(v) == 0: nie ma przecięć

if ssize(v) == 1: v[0] jest przecięciem

if ssize(v) == 2 and s: (v[0], v[1]) to odcinek, w którym są wszystkie inf rozwiązań

if ssize(v) == 2 and s == false: v to niezdefiniowane punkty (inf rozwiązań)

```
"/point/main.cpp" 3a1213, 26 lines

P intersection(P a, P b, P c, P d) {
    Double c1 = cross(c - a, b - a), c2 = cross(d - a, b - a);
    assert(c1 != c2); // proste nie moga byc rownolegle
    return (c1 * d - c2 * c) / (c1 - c2);
}

bool on_segment(P a, P b, P p) {
    return equal(cross(a - p, b - p), 0) and dot(a - p, b - p) <= 0;
}

vector<P> intersect(P a, P b, P c, P d, bool segments) {
    Double 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((segments and sign(acd) * sign(bcd) < 0 and sign(cab) * sign(dab) < 0)
        or (not segments and not equal(bcd, acd)))
        return {(a * bcd - b * acd) / (bcd - acd)};
    if(not segments)
        return {a, a};
}
```

line intersect-lines area convex-hull circles hashing

```
// skip for not segments
set<P, Sortx> 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()};
}
```

area

Opis: Pole wielokąta, niekoniecznie wypukłego

Użycie: w vectorze muszą być wierzchołki zgodnie z kierunkiem ruchu zegara. Jeśli Double jest intem to może się psuć / 2. area(a, b, c) zwraca pole trójkąta o takich długościach boku

```
"/point/main.cpp" bba541, 10 lines

Double area(vector<P> pts) {
    int n = size(pts);
    Double ans = 0;
    REP(i, n) ans += cross(pts[i], pts[(i + 1) % n]);
    return ans / 2;
}

Double area(Double a, Double b, Double c) {
    Double p = (a + b + c) / 2;
    return sqrt(p * (p - a) * (p - b) * (p - c));
}
```

convex-hull

Opis: Otoczka wypukła, osobno góra i dół

Czas: $O(n \log n)$

Użycie: top_bot_hull zwraca osobno górę i dół po id

hull_id zwraca całą otoczkę po id

hull zwraca punkty na otoczce

```
"/point/main.cpp" 6eb7f2, 38 lines

Double cross(P a, P b, P c) { return sign(cross(b - a, c - a)); }

pair<vector<int>, vector<int>> top_bot_hull(vector<P> &pts) {
    int n = ssize(pts);
    vector<int> ord(n);
    REP(i, n) ord[i] = i;
    sort(ord.begin(), ord.end(), [&](int i, int j) {
        P &a = pts[i], &b = pts[j];
        return make_pair(a.x, a.y) < make_pair(b.x, b.y);
    });

    vector<int> top, bot;
    REP(dir, 2) {
        vector<int> &hull = (dir ? bot : top);
        auto l = [&](int i) { return pts[hull[ssize(hull) - i]]; };
        for(int i : ord) {
            while(ssize(hull) > 1 && cross(l(2), l(1), pts[i]) >= 0)
                hull.pop_back();
            hull.emplace_back(i);
        }
        reverse(ord.begin(), ord.end());
    }
    return {top, bot};
}
```

```
vector<int> hull_id(vector<P> &pts) {
    vector<int> top, bot;
    tie(top, bot) = top_bot_hull(pts);
    top.pop_back(), bot.pop_back();
    top.insert(top.end(), bot.begin(), bot.end());
    return top;
}

vector<P> hull(vector<P> &pts) {
```

```
vector<P> ret;
for(int i : hull_id(pts))
    ret.emplace_back(pts[i]);
return ret;
}
```

circles

Opis: Przecięcia okręgu oraz prostej $ax+by+c=0$ oraz przecięcia okręgu oraz okręgu.

Użycie: ssize(circle_circle(...)) == 3 to jest nieskończenie wiele rozwiązań

```
"/point/main.cpp" a9d88d, 36 lines

using D = Double;

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));
}

vector<P> circle_circle(D x1, D y1, D r1, D x2, D y2, 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;
}
```

Tekstówki (8)

hashing

Czas: $O(1)$

Użycie: Hashing hsh(str);

hsh(l, r) zwraca hasza [l, r] włącznie można zmienić modulo i bazę

```
"/random-stuff/rd/main.cpp" 299a85, 28 lines

struct Hashing {
    vector<int> ha, pw;
    int mod = 1e9 + 696969;
    int base;

    Hashing(string &str, int b) {
        base = b;
        int len = ssize(str);
        ha.resize(len + 1);
    }
}
```

```
pw.resize(len + 1, 1);
REP(i, len) {
    ha[i + 1] = int(((LL) ha[i] * base + str[i] - 'a' + 1) % mod);
    pw[i + 1] = int(((LL) pw[i] * base) % mod);
}

int operator()(int l, int r) {
    return int(((ha[r + 1] - (LL) ha[l] * pw[r - l + 1]) % mod + mod) % mod);
};

struct DoubleHashing {
    Hashing h1, h2;
    DoubleHashing(string &str) : h1(str, 31), h2(str, 33) {} //
        change to rd on codeforces
    LL operator()(int l, int r) {
        return h1(l, r) * LL(h2.mod) + h2(l, r);
    }
};
```

kmp
Opis: KMP(str) zwraca tablicę pi. [0, pi[i]] = (i - pi[i], i)
Czas: $\mathcal{O}(n)$

bc0e11, 11 lines

```
vector<int> KMP(string &str) {
    int len = ssize(str);
    vector<int> ret(len);
    for(int i = 1; i < len; i++)
    {
        int pos = ret[i - 1];
        while(pos && str[i] != str[pos]) pos = ret[pos - 1];
        ret[i] = pos + (str[i] == str[pos]);
    }
    return ret;
}
```

pref
Opis: pref(str) zwraca tablicę prefixo prefixowż [0, pref[i]] = [i, i + pref[i]]
Czas: $\mathcal{O}(n)$

6c98b2, 13 lines

```
vector<int> pref(string &str) {
    int len = ssize(str);
    vector<int> ret(len);
    ret[0] = len;
    int i = 1, m = 0;
    while(i < len) {
        while(m + i < len && str[m + i] == str[m]) m++;
        ret[i++] = m;
        m = (m != 0 ? m - 1 : 0);
        for(int j = 1; ret[j] < m; m--) ret[i++] = ret[j++];
    }
    return ret;
}
```

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

ca63bf, 18 lines

```
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;
}
```

trie
Opis: Trie
Czas: $\mathcal{O}(n \log \alpha)$
Użycie: Trie trie; trie.add(str);

dcd05a, 15 lines

```
struct Trie {
    vector<unordered_map<char, int>> child = {{{}};
    int get_child(int v, char a) {
        if(child[v].find(a) == child[v].end()) {
            child[v][a] = ssize(child);
            child.emplace_back();
        }
        return child[v][a];
    }
    void add(string word) {
        int v = 0;
        for(char c : word)
            v = get_child(v, c);
    }
};
```

suffix-automaton
Opis: buduje suffix automaton. Wystąpienia wzorca, liczba różnych pod-słów, sumaryczna długość wszystkich pod-słów, leksykograficznie k-te pod-słowo, najmniejsze przesunięcie cykliczne, liczba wystąpień pod-słowa, pierwsze wystąpienie, najkrótsze niewystępujące pod-słowo, longest common sub-string dwóch słów, LCS wielu słów
Czas: $\mathcal{O}(n\alpha)$ (szybsze, ale więcej pamięci) albo $\mathcal{O}(n \log \alpha)$ (mapa)

0d06ff, 54 lines

```
struct SuffixAutomaton {
    static constexpr int sigma = 26;
    using Node = array<int, sigma>; // map<int, int>
    Node new_node;

    vector<Node> edges;
    vector<int> link = {-1}, length = {0};
    int last = 0;

    SuffixAutomaton() {
        new_node.fill(-1); // -1 - stan nieistniejący
        edges = {new_node}; // dodajemy stan startowy, który reprezentuje puste słowo
    }

    void add_letter(int c) {
        edges.emplace_back(new_node);
        length.emplace_back(length[last] + 1);
        link.emplace_back(0);

        int r = ssize(edges) - 1, p = last;
        while(p != -1 && edges[p][c] == -1) {
            edges[p][c] = r;
            p = link[p];
        }
        if(p != -1) {
```

```
int q = edges[p][c];
if(length[p] + 1 == length[q])
    link[r] = q;
else {
    edges.emplace_back(edges[q]);
    length.emplace_back(length[p] + 1);
    link.emplace_back(link[q]);
    int q_prim = ssize(edges) - 1;

    link[q] = link[r] = q_prim;
    while(p != -1 && edges[p][c] == q) {
        edges[p][c] = q_prim;
        p = link[p];
    }
}
last = r;
}

bool is_inside(vector<int> &s) {
    int q = 0;
    for(int c : s) {
        if(edges[q][c] == -1)
            return false;
        q = edges[q][c];
    }
    return true;
};
```

suffix-array
Opis: Tablica suffixowa
Czas: $\mathcal{O}(n \log n)$
Użycie: SuffixArray t(s, lim) - lim to rozmiar alfabetu sa zawiera posortowane suffixy, zawiera pusty suffix lcp[i] to lcp suffixu sa[i - 1] i sa[i]
Dla s = "aabaaa" sa = {6, 3, 0, 4, 1, 5, 2}, lcp = {0, 0, 3, 1, 2, 0, 1}

d9039e, 29 lines

```
struct SuffixArray {
    vector<int> sa, lcp;
    SuffixArray(string& s, int lim = 256) { // lub basic_string<int>
        int n = ssize(s) + 1, k = 0, a, b;
        vector<int> x(s.begin(), s.end() + 1);
        vector<int> y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y;
        iota(sa.begin(), sa.end(), 0);

        for(int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j;
            iota(y.begin(), y.end(), n - j);
            REP(i, n) if(sa[i] >= j)
                y[p++] = sa[i] - j;
            fill(ws.begin(), ws.end(), 0);
            REP(i, n) ws[x[i]]++;
            FOR(i, 1, lim - 1) ws[i] += ws[i - 1];
            for(int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y);
            p = 1, x[sa[0]] = 0;
            FOR(i, 1, n - 1) a = sa[i - 1], b = sa[i], x[b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
        }
        FOR(i, 1, n - 1) rank[sa[i]] = i;
        for(int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
            for(k && k--, j = sa[rank[i] - 1];
                s[i + k] == s[j + k]; k++);
    }
};
```

Optymalizacje (9)

pragmy

Opis: Pragmy do wypychania kolanem

61c4f7, 2 lines

#pragma GCC optimize("Ofast")

#pragma GCC target("avx,avx2")

fio

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

8eb22d, 52 lines

#ifdef WIN32

inline int getchar_unlocked() { return _getchar_nolock(); }

inline void putchar_unlocked(char c) { return _putchar_nolock(c); }

#endif

int fastin() {

int n = 0, c = getchar_unlocked();

while(c < '0' or '9' < c)

c = getchar_unlocked();

while('0' <= c and c <= '9') {

n = 10 * n + (c - '0');

c = getchar_unlocked();

}

return n;

}

int fastin_negative() {

int n = 0, negative = false, c = getchar_unlocked();

while(c != '-' and (c < '0' or '9' < c))

c = getchar_unlocked();

if(c == '-') {

negative = true;

c = getchar_unlocked();

}

while('0' <= c and c <= '9') {

n = 10 * n + (c - '0');

c = getchar_unlocked();

}

return negative ? -n : n;

}

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++] = '0' + (x % 10);

x /= 10;

}

while(--i >= 0)

putchar_unlocked(t[i]);

putchar_unlocked(' ');

}

void nl() { putchar_unlocked('\n'); }

Randomowe rzeczy (10)

math-constants

Opis: Jeśli np M_PI się nie kompiluje, dodaj ten define w pierwszym wierszu

ac1260, 1 lines

#define _USE_MATH_DEFINES

dzien-probny

Opis: Rzeczy do przetestowania w dzień próbny

"../data-structures/ordered-set/main.cpp"

3439f3, 51 lines

void test_int128() {

__int128 x = (1llu << 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 seeed = chrono::system_clock::now().time_since_epoch().count();

(void) seeed;

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_rd() {

// czy jest sens to testowac?

mt19937_64 my_rng(0);

auto rd = [&](int l, int r) {

return uniform_int_distribution<int>(l, r)(my_rng);

};

assert(rd(0, 0) == 0);

}

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() {

assert(3.14 < M_PI && M_PI < 3.15);

assert(3.14 < M_PI1 && M_PI1 < 3.15);

}

10.1 Troubleshoot

Przed submitem:

- Narysuj parę przykładów i przetestuj kod
- Czy limity czasu są ostre? Wygeneruj maxtest.

- Czy zużycie pamięci jest spoko?
- Czy gdzieś mogą być overflowy?
- Upewnij się, żeby submitnąć dobry plik.

Wrong Answer:

- Wydrukuj kod i debug output
- Czy czyścisz struktury pomiędzy test case'ami?
- Czy wczytujesz całe wejście?
- Czy twój kod obsługuje cały zasięg wejścia?
- Przeczytaj jeszcze raz treść.
- Czy zrozumiałeś dobrze zadanie?
- Czy obsługujesz dobrze wszystkie przypadki brzegowe?
- Niezainicjalizowane zmienne?
- Overflowy?
- Mylisz n z m lub i z j, itp?
- Czy format wyjścia jest na pewno dobry?
- Czy jesteś pewien, że twój algorytm działa?
- Czy są specjalne przypadki, o których nie pomyślałeś?
- Dodaj asserty, może submitnij jeszcze raz z nimi.
- Stwórz/Wygeneruj przykłady.
- Wytłumacz algorytm komuś innemu.
- Poproś kogoś, żeby spojrział na twój kod.
- Przejdź się, np do toalety.
- Przepisz kod od nowa, lub niech ktoś inny to zrobi.
- Przeleć przez tą listę jeszcze raz.

Runtime Error:

- Czy przetestowałeś lokalnie wszystkie przypadki brzegowe?
- Niezainicjalizowane zmienne?
- Czy odwołujesz się poza zasięg vectora?
- Czy jakieś asserty mogły się odpalić?
- Dzielenie przez 0? mod 0?
- Nieskończona rekurencja?
- Unieważnione iteratory, wskaźniki, referencje?
- Czy używasz za dużo pamięci?

Time Limit Exceeded:

- Czy mogą być gdzieś nieskończone pętle?
- Jaka jest złożoność algorytmu?
- Czy nie kopiujesz dużo niepotrzebnych danych? (referencje)
- Pamiętaj o linijkach do iostreama
- Zastąp vectory i mapy w kodzie (odpowiednio array i unordered_map)
- Co inni myślą o twoim algorytmie?

Memory Limit Exceeded:

- Jaka jest maksymalna ilość pamięci twój algorytm potrzebuje?
- Czy czyścisz struktury pomiędzy test case'ami?