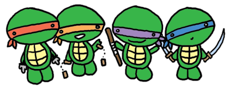

XIII LO Szczecin

# Wojownicze Żółwie Ninja

Tomasz Nowak, Michał Staniewski, Justyna Jaworska

2019-11-27

# Utils (1)

headers
**Opis:** Nagłówki używane w każdym kodzie. Działa na każdy kontener i pary
**Użycie:** `debug(a, b, c) << d << e;` wypisze `a, b, c: a; b; c; de`

<bits/stdc++.h> 46f465, 33 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)
template<class T> int size(T &&x) {
  return int(x.size());
}
template<class A, class B> ostream& operator<<(ostream &out,
    const pair<A, B> &p) {
  return out << '(' << p.first << ", " << p.second << ')';
}
template<class T> auto operator<<(ostream &out, T &&x) ->
    decltype(x.begin(), out) {
  out << '{';
  for(auto it = x.begin(); it != x.end(); ++it)
    out << *it << (it == prev(x.end()) ? "" : ", ");
  return out << '}';
}
void dump() {}
template<class T, class... Args> void dump(T &&x, Args... args)
    {
  cerr << x << ";  ";
  dump(args...);
}
struct Nl{~Nl(){cerr << '\n';}};
#ifdef DEBUG
# define debug(x...) cerr << (strcmp(#x, "") ? #x ":  " : ""),
    dump(x), Nl(), cerr << ""
#else
# define debug(...) 0 && cerr
#endif

const int seed = chrono::system_clock::now().time_since_epoch()
    .count();
mt19937_64 rng(seed);
int rd(int l, int r) {
  return uniform_int_distribution<int>(l, r)(rng);
}
```

headers/bazshrc.sh 10 lines

```
c() {
  clang++ -O3 -std=c++11 -Wall -Wextra -Wshadow \
    -Wconversion -Wno-sign-conversion -Wfloat-equal \
    -D_GLIBCXX_DEBUG -fsanitize=address,undefined -ggdb3 \
    -DDEBUG $1.cpp -o $1
}

nc() {
  clang++ -O3 -std=c++11 -static $1.cpp -o $1 # -m32
}
```

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 < conf.txt > gen.txt
  ./main < gen.txt > main.txt
  ./brute < gen.txt > brute.txt

  if diff -w main.txt brute.txt > /dev/null; then
    echo "OK $i"
  else
    echo "WA"
    exit 0
  fi
done
```

# Podejścia (2)

- dynamik, zachłan
- sposób "liczba dobrych obiektów = liczba wszystkich obiektów - liczba złych obiektow"
- czy warunek konieczny = warunek wystarczający?
- odpowiednie przekształcenie równania
- zastanowić się nad łatwiejszym problemem, bez jakiegoś elementu z treści
- sprowadzić problem do innego, łatwiejszego/mniejszego problemu
- sprowadzić problem 2D do problemu 1D (szczególny przypadek: zamiatanie; częsty przypadek: 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? Coś można posortować? Coś można shuffle'ować?
- narysować dużo swoich własnych przykładów i coś z nich wywnioskować

# 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 $(2m - n, m)$, $(2m + n, m)$ oraz $(m + 2n, n)$.

## 3.4 Liczby pierwsze

$p = 962592769$ to liczba na NTT, czyli $2^{21} \mid p - 1$, which may be useful. Do hashowania: 970592641 (31-bit), 31443539979727 (45-bit), 3006703054056749 (52-bit).

Jest 78498 pierwszych $\leq 1\,000\,000$.

Generatorów jest $\phi(\phi(p^a))$, czyli dla $p > 2$ zawsze istnieje.

## 3.5 Dzielniki

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

Liczba dzielników $n$ jest co najwyżej 100 dla $n < 5e4$, 500 dla $n < 1e7$, 2000 dla $n < 1e10$, 200 000 dla $n < 1e19$.

### 3.5.1 Lemat Burnside'a

Liczba takich samych obiektów z dokładnością do symetrii wynosi Given a group $G$ of symmetries and a set $X$, the number of elements of $X$ *up to symmetry* equals

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

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

## 3.6 Silnia

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

### 3.6.1 Symbol Newtona

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

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

## 3.7 Wzorki na pewne ciągi

### 3.7.1 Nieporządek

Liczba takich permutacji, że $p_i \neq i$ (żadna liczba nie wraca na tą samą pozycję).

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

### 3.7.2 Liczba podziałów

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

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

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

| $n$ | 0 1 2 3 4 5 | 6 | 7 | 8 | 9 | 20 | 50 | 100 |
|---|---|---|---|---|---|---|---|---|
| $p(n)$ | 1 1 2 3 5 7 | 11 | 15 | 22 | 30 | 627 | ∼2e5 | ∼2e8 |

### 3.7.3 Liczby Eulera pierwszego rzędu

Liczba permutacji $\pi \in S_n$ gdzie $k$ elementów jest większych niż poprzedni: $k$ razy $\pi(j) > \pi(j+1)$, $k+1$ razy $\pi(j) \geq j$, $k$ razy $\pi(j) > j$.

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

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

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

### 3.7.4 Stirling pierwszego rzędu

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

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

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

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

### 3.7.5 Stirling drugiego rzędu

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

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

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

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

### 3.7.6 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 $n \times n$.
- nawiasowania po $n$ ().
- liczba drzew binarnych z $n+1$ liściami (0 lub 2 syny).
- skierowanych drzew z $n+1$ wierzchołkami.
- triangulacje $n+2$-kąta.
- permutacji $[n]$ bez 3-wyrazowego rosnącego podciągu?

### 3.7.7 Formuła Cayley'a

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

## 3.8 Funkcje multiplikatywne

- $id(n) = n$, $\mathbb{1} * \varphi = id$
- $\mathbb{1}(n) = 1$
- $\tau(n)$ = liczba dzielników dodatnich, $\mathbb{1} * \mathbb{1} = \tau$
- $\sigma(n)$ = suma dzielników dodatnich, $id * \mathbb{1} = \sigma$
- $\varphi(n)$ = liczba liczb względnie pierwszych z $n$ większych równych 1, $id * \mu = \varphi$
- $\mu(n) = 1$ dla $n = 1$, 0 gdy istnieje $p$, że $p^2|n$, oraz $(-1)^k$ jak $n$ jest iloczynem $k$ parami różnych liczb pierwszych
- $\epsilon(n) = 1$ dla $n = 1$ oraz 0 dla $n > 1$, $f * \epsilon = f$, $\mathbb{1} * \varphi = \epsilon$
- $(f * g)(n) = \sum_{d|n} f(d)g(\frac{n}{d})$
- $f * g = g * f$
- $f * (g * h) = (f * g) * h$
- $f * (g + h) = f * g + f * h$
- 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:
  - $g(n) = \sum_{d|n} f(d)$
  - $f(n) = \sum_{d|n} g(d)\mu(\frac{n}{d})$
  - $\sum_{k=1}^{n} g(k) = \sum_{d=1}^{n}\lfloor\frac{n}{d}\rfloor f(d)$
- $\varphi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1)$
- $\varphi(n) = n \cdot (1 - \frac{1}{p_1}) \cdot (1 - \frac{1}{p_2}) \ldots (1 - \frac{1}{p_k})$

## 3.9 Zasada włączeń i wyłączeń

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

## 3.10 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);

deaf46, 7 lines

```
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 względnie pierwsze, ale może nie być wtedy rozwiązania
uwali się wtedy assercik, można zmienić na return -1

"../extended-gcd/main.cpp" 269203, 9 lines

```
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;
}
```

## berlekamp-massey

**Opis:** Zgadywanie rekurencji
**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)

606849, 58 lines

```
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 n) {
    if(n == 0) return 1;
    if(n % 2 == 1) return mul(qpow(a, n - 1), a);
    return qpow(mul(a, a), n / 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, size(x)) {
      m++;
      int d = x[i];
      FOR(j, 1, size(C) - 1)
        d = add(d, mul(C[j], x[i - j]));
      if(d == 0) continue;
      auto _B = C;
      C.resize(max(size(C), m + size(B)));
      int coef = mul(d, qpow(b, mod - 2));
      FOR(j, m, m + size(B) - 1)
        C[j] = (C[j] - mul(coef, B[j - m]) + mod) % mod;
      if(size(_B) < m + size(B)) { B = _B; b = d; m = 0; }
    }
    C.erase(C.begin());
    for(int &t : C) t = add(mod, -t);
    n = size(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:** $\mathcal{O}\left(\log^2_n\right)$ Pamięć : $\mathcal{O}(1)$
**Użycie:** miller_rabin(n) zwraca czy n jest pierwsze
dziala dla long longów

623bb2, 33 lines

```
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}) {
    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;
}
```

## fft

**Opis:** Mnożenie wielomianów
**Czas:** $\mathcal{O}(n \log n)$
**Użycie:** conv(a, b) zwraca iloczyn wielomianów a i b
conv_mod(a, b, M) zwraca iloczyn modulo, większa dokładność

0f1926, 62 lines

```
using Complex = complex<double>;
void fft(vector<Complex> &a) {
  int n = size(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_PIl / 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) {
      auto x = (double *) &rt[j + k], y = (double *) &a[i + j +
          k];
      Complex z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1] *
          y[0]);
      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(size(a) + size(b) - 1);
  int L = 32 - __builtin_clz(size(res)), n = (1 << L);
  vector<Complex> in(n), out(n);
  copy(a.begin(), a.end(), in.begin());
  REP(i, size(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, size(res)) res[i] = imag(out[i]) / (4 * n);
  return res;
}

vector<LL> conv_mod(vector<LL> &a, vector<LL> &b, int M) {
  if(a.empty() || b.empty()) return {};
  vector<LL> res(size(a) + size(b) - 1);
  int B = 32 - __builtin_clz(size(res)), n = 1 << B;
  int cut = int(sqrt(M));
  vector<Complex> L(n), R(n), outl(n), outs(n);
  REP(i, size(a)) L[i] = Complex((int) a[i] / cut, (int) a[i] %
      cut);
  REP(i, size(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, size(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;
}
```

### rho-pollard

**Opis:** Rozklad na czynniki Rho Pollarda
**Czas:** $\mathcal{O}\left(n^{\frac{1}{4}}\right)$
"../miller-rabin/main.cpp" 719c9e, 19 lines

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

### integral

**Opis:** Wzór na całkę z zasady Simpsona. Daj asserta na błąd, zwiększ n
6a3b16, 8 lines

```
using T = double;
T intergral(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, 2 * n - 1)
    sum += f(a + i * delta) * (i & 1 ? 4 : 2);
  return sum * dif / 3;
}
```

## Struktury danych (5)

### find-union

**Opis:** Find and union z mniejszy do wiekszego
**Czas:** $\mathcal{O}(\alpha(n))$ oraz $\mathcal{O}(n)$ pamięciowo
c3dcbd, 19 lines

```
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) {}
};
```

### lazy-segment-tree

**Opis:** Drzewo przedzial-przedzial
**Czas:** $\mathcal{O}(\log n)$ Pamięć : $\mathcal{O}(n)$
**Użycie:** `add(l, r, val)` dodaje na przedziale
`quert(l, r)` bierze maxa z przedzialu
`Zmieniając z maxa na co innego trzeba edytować`
`funkcje add_val i f`
a98ace, 59 lines

```
struct Node {
  int val, lazy;
  int size = 1;
};

struct Tree {
  vector<Node> nodes;
  int size = 1;

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

  int f(int a, int b) { return max(a, b); }

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

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

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

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

### segment-tree

**Opis:** Drzewo punkt-przedzial
**Czas:** $\mathcal{O}(\log n)$ Pamięć : $\mathcal{O}(n)$
**Użycie:** `Tree(n, val = 0) tworzy drzewo o n liściach, o`
`wartościach val`
`update(pos, val) zmienia element pos na val`
`query(l, r) zwraca f na przedziale`
`edytujesz funkcję f, można T ustawić na long longa albo parę`
92195e, 30 lines

```
struct Tree {
  using T = int;
  T f(T a, T b) { return a + b; }
  vector<T> nodes;
  int size = 1;

  Tree(int n, T val = 0) {
    while(size < n) size *= 2;
    nodes.resize(size * 2, val);
  }

  void update(int pos, T val) {
    nodes[pos += size] = val;
    while(pos /= 2)
      nodes[pos] = f(nodes[pos * 2], nodes[pos * 2 + 1]);
  }

  T query(int l, int r) {
    l += size, r += size;
    T ret = (l != r ? f(nodes[l], nodes[r]) : nodes[l]);
    while(l + 1 < r) {
      if(l % 2 == 0)
        ret = f(ret, nodes[l + 1]);
      if(r % 2 == 1)
        ret = f(ret, nodes[r - 1]);
      l /= 2, r /= 2;
    }
    return ret;
  }
};
```

### 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 pierwszych pos elementów`
`lower_bound(val) zwraca pos, że suma [0, pos] <= val`
78e5fe, 26 lines

```
struct Fenwick {
  vector<LL> s;
  Fenwick(int n) : s(n) {}

  void update(int pos, LL val) {
    for(; pos < size(s); pos |= pos + 1)
      s[pos] += val;
  }

  LL query(int pos) {
    LL ret = 0;
    for(; pos > 0; pos &= pos - 1)
      ret += s[pos - 1];
    return ret;
  }

  int lower_bound(LL val) {
    if(val <= 0) return -1;
    int pos = 0;
    for(int pw = 1 << 25; pw; pw /= 2) {
      if(pos + pw <= size(s) && s[pos + pw - 1] < sum)
```

```
      pos += pw, sum -= s[pos - 1];
    }
    return pos;
  }
};
```

## ordered-set

**Opis:** Ordered Set
**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 multiseta, to używamy par {val, id}.
Nie dziala z -D_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
>;
```

## lichao-tree

**Opis:** Dla funkcji, których pary przecinaja sie co najwyżej raz, oblicza maximum w punkcie x. Podany kod jest dla funkcji liniowych a7f64a, 50 lines

```
struct Function {
  int a, b;
  L operator()(int x) {
    return x * L(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);
  }

  L get_min(int x) {
    int v = x + size;
    L 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);
  }
};
```

## line-container

**Opis:** Set dla funkcji liniowych
**Czas:** $\mathcal{O}(\log n)$ 61ff3a, 33 lines

```
struct Line {
  mutable LL a, b, p;
  LL eval(LL x) { return a * x + b; }
  bool operator<(const Line & o) const { return a < o.a; }
  bool operator<(LL x) const { return p < x; }
};

constexpr LL inf = 1e18 + 7;
LL divide(LL a, LL b) { return a / b - ((a ^ b) < 0 && a % b);
    }
LL better(const Line &x, const Line &y) {
  if(x.a == y.a) return x.b >= y.b ? inf : -inf;
  return divide(y.b - x.b, x.a - y.a);
}

struct LineContainer : multiset<Line, less<>> {
  bool intersect(iterator x, iterator y) {
    if(y == end()) { x->p = inf; return 0; }
    x->p = better(*x, *y);
    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, y =
        erase(y));
    while((y = x) != begin() && (--x)->p >= y->p)
      intersect(x, erase(y));
  }
  LL query(LL x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.eval(x);
  }
};
```

# Grafy (6)

## hld

**Opis:** Heavy-Light Decomposition
**Czas:** $\mathcal{O}(\log n)$
**Użycie:** kontruktor - HLD(n, graph)
lca(v, u) zwraca lca
get_vertex(v) zwraca pozycję odpowiadającą wierzcholkowi
get_path(v, u) zwraca przedzialy do obslugiwania drzewem przedzialowym
get_path(v, u) jeśli robisz operacje na wierzcholkach
get_path(v, u, false) jeśli na krawędziach
get_subtree(v) zwraca przedzial odpowiadający podrzewu v 0ef9dc, 66 lines

```
struct HLD {
  vector<vector<int>> graph;
  vector<int> size, pre, pos, nxt, par;
  int t = 0;

  void init(int v, int p = -1) {
    par[v] = p;
    size[v] = 1;
    for(int &u : graph[v]) if(u != par[v]) {
      init(u, v);
      size[v] += size[u];
      if(size[u] > size[graph[v][0]])
        swap(u, graph[v][0]);
    }
  }

  void set_paths(int v) {
    pre[v] = t++;
    for(int &u : graph[v]) if(u != par[v]) {
      nxt[u] = (u == graph[v][0] ? nxt[v] : u);
      set_paths(u);
    }
    pos[v] = t;
  }

  HLD(int n, vector<vector<int>> graph, int root = 0)
    : graph(graph), size(n), pre(n), pos(n), nxt(n), par(n) {
    init(root);
    set_paths(root);
  }

  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]);
    while(!path_u.empty()) {
      ret.emplace_back(path_u.back());
      path_u.pop_back();
    }
    return ret;
  }

  pair<int, int> get_subtree(int v) { return {pre[v], pos[v] -
      1}; }
};
```

## scc

**Opis:** Silnie Spójnie Skladowe
**Czas:** $\mathcal{O}(\log n)$
**Użycie:** kontruktor - SCC(graph)
group[v] to numer silnie spójnej wierzcholka v
get_compressed() zwraca graf siline spójnych
get_compressed(false) nie usuwa multikrawędzi

112027, 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 = size(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;
  }
};
```

## 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)
konstruktor - EulerianPath(m, graph)
graf musi być spójny
get_path() zwraca ścieżkę eulera
get_cycle() zwraca cykl eulera
jeśli nie ma, obie funkcję zwrócą pusty vector

202e02, 40 lines

```
struct EulerianPath {
  vector<vector<pair<int, int>>> graph;
  vector<bool> used;
  vector<int> in, out;
  vector<int> path, cycle

  void dfs(int v = 0) {
    in[v]++;
    while(!graph[v].empty()) {
      auto edge = graph[v].back();
      graph[v].pop_back();
      int u = edge.first;
      int id = edge.second;
      if(used[id]) continue;
      used[id] = true;
      out[v]++;
      dfs(u);
    }
    path.emplace_back(v);
  }

  EulerianPath(int m, vector<vector<pair<int, int>>> &graph) :
      graph(graph) {
    int n = size(graph);
    used.resize(m);
    in.resize(n);
    out.resize(n);

    dfs();
    in[0]--;
    debug(path, in, out);
    cycle = path;
    REP(i, n) if(in[i] != out[i]) cycle.clear();
    if(path.size() != 0) in[path.back()]++, out[path[0]]++;
    REP(i, n) if(in[i] != out[i]) path.clear();
    reverse(path.begin(), path.end());
  }

  vector<int> get_path() { return path; }
  vector<int> get_cycle() { return cycle; }
};
```

## 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 wierzcholek o k wyższy niż v
jeśli nie istnieje, zwraca -1
lca(a, b) zwraca lca wierzcholków

d7a477, 49 lines

```
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 = size(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];
  }
};
```

## 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_flowing() zwraca dla każdej oryginalnej krawędzi, ile przez nią leci

fed904, 78 lines

```
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() << "adding edge " << make_pair(v, u) << " with cap
        " << cap;
```

```
    int e = size(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(size(que) and dist[sink] == 0) {
      int v = que.front();
      que.pop_front();
      for(int e : graph[v])
        if(edges[e].flow != edges[e].cap and 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] != size(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_flowing() {
    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_flowing()` z Dinic'a

2baac2, 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 = size(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(size(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};
  }
};
```

## matching

**Opis:** Turbo Matching
**Czas:** Mniej więcej $\mathcal{O}(n \log n)$, najgorzej $\mathcal{O}(n^2)$
**Użycie:** wierzcholki grafu nie muszą być ladnie podzielone na dwia przedzialy, musi być po prostu dwudzielny. Funkcja match() dziala w o(n), nieważne jak male zmiany się wprowadzi do aktualnego matchingu.

0290f0, 41 lines

```
vector<vector<int>> graph;
vector<int> match, vis;
int t = 0;

bool match_dfs(int v) {
  vis[v] = t;
  for(int u : graph[v])
    if(match[u] == -1) {
      match[u] = v;
      match[v] = u;
      return true;
    }

  for(int u : graph[v])
    if(vis[match[u]] != t && match_dfs(match[u])) {
      match[u] = v;
      match[v] = u;
      return true;
    }
  return false;
}

int match() {
  int n = int(graph.size());
  match.resize(n, -1);
  vis.resize(n);

  int d = -1;
  while(d != 0) {
    d = 0;
    ++t;
    for(int v = 0; v < n; ++v)
      if(match[v] == -1)
        d += match_dfs(v);
  }
  int ans = 0;
  for(int v = 0; v < n; ++v)
    if(match[v] != -1)
      ++ans;
  return ans / 2;
}
```

## floyd-warshall

**Opis:** Floyd-Warshall
**Czas:** $\mathcal{O}\left(n^3\right)$
**Użycie:** FloydWarshall(graph) zwraca macież odległości graph to macierz sąsiedztwa z wagami

7457d0, 6 lines

```
vector<vector<LL>> FloydWarshall(vector<vector<int>> graph) {
  int n = size(graph);
  vector<vector<LL>> dist(n, vector<LL>(n, 1e18));
  REP(k, n) REP(i, n) REP(j, n)
    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
}
```

## 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);
oznacza 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

841cb2, 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(size(li) <= 1) return;
    int cur = ~li[0];
    FOR(i, 2, size(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;
  }
};
```

## matching

**Opis:** Turbo Matching
**Czas:** Mniej więcej $\mathcal{O}(n\log n)$, najgorzej $\mathcal{O}\left(n^2\right)$
**Użycie:** wierzcholki grafu nie muszą być ladnie podzielone na dwia przedzialy, musi być po prostu dwudzielny. Funkcja match() dziala w o(n), nieważne jak male zmiany się wprowadzi do aktualnego matchingu.

0290f0, 41 lines

```
vector<vector<int>> graph;
vector<int> match, vis;
int t = 0;

bool match_dfs(int v) {
  vis[v] = t;
  for(int u : graph[v])
    if(match[u] == -1) {
      match[u] = v;
      match[v] = u;
      return true;
    }

  for(int u : graph[v])
    if(vis[match[u]] != t && match_dfs(match[u])) {
      match[u] = v;
      match[v] = u;
      return true;
    }
  return false;
}

int match() {
  int n = int(graph.size());
  match.resize(n, -1);
  vis.resize(n);

  int d = -1;
  while(d != 0) {
    d = 0;
    ++t;
    for(int v = 0; v < n; ++v)
      if(match[v] == -1)
        d += match_dfs(v);
  }
  int ans = 0;
  for(int v = 0; v < n; ++v)
    if(match[v] != -1)
      ++ans;
  return ans / 2;
}
```

## flow

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

fed904, 78 lines

```
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() << "adding edge " << make_pair(v, u) << " with cap
        " << cap;
    int e = size(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(size(que) and dist[sink] == 0) {
      int v = que.front();
      que.pop_front();
      for(int e : graph[v])
        if(edges[e].flow != edges[e].cap and 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] != size(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_flowing() {
  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_flowing() z Dinic'a
2baac2, 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 = size(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(size(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)
0e17a7, 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;
}
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;
}
P rotate(P x, P center, Double radians) {
  return (x - center) * exp(P(0, radians)) + center;
}
```

## intersect-lines

**Opis:** Przecięcie prostych lub odcinków
**Użycie:** v = intersect(a, b, c, d, s) zwraca przecięcie (s ? odcinków : prostych) ab oraz cd
if size(v) == 0: nie ma przecięć
if size(v) == 1: v[0] jest przecięciem
if size(v) == 2 and s: (v[0], v[1]) to odcinek, w którym są wszystkie inf rozwiązań
if size(v) == 2 and s == false: v to niezdefiniowane punkty (inf rozwiązań)
"../point/main.cpp" cfa1cd, 20 lines

```
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};
  // 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()};
}
```

# Tekstówki (8)

## hashing

**Czas:** $\mathcal{O}(1)$
**Użycie:** get_hash(l, r) zwraca hasza [l, r] włącznie
można zmienić modulo i bazę
4767a1, 20 lines

```
struct Hashing {
  vector<LL> ha, pw;
  LL mod = 1000696969;
  int base;

  Hashing(string &str) {
    base = rd(30, 50);
    int len = size(str);
    ha.resize(len + 1);
    pw.resize(len + 1, 1);
    REP(i, len) {
      ha[i + 1] = (ha[i] * base + str[i] - 'a' + 1) % mod;
      pw[i + 1] = (pw[i] * base) % mod;
```

```
    }
  }

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

## kmp
**Opis:** KMP(str) zwraca tablicę pi
**Czas:** $\mathcal{O}(n)$

65f132, 11 lines

```
vector<int> KMP(string &str) {
  int len = size(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 prefixową [0, pref[i]) = [i, i + pref[i])
**Czas:** $\mathcal{O}(n)$

68517d, 13 lines

```
vector<int> pref(string &str) {
  int len = size(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. $L = i - rad + !p$, $R = i + rad$ to palindrom. Dla [abaababaab] daje [003000020], [0100141000].
**Czas:** $\mathcal{O}(n)$

be40a9, 18 lines

```
array<vector<int>, 2> manacher(vector<int> &in) {
  int n = size(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);`

9aa8f1, 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] = size(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 podslów, sumaryczna długość wszystkich podslów, leksykograficznie k-te podslowo, najmniejsze przesunięcie cykliczne, liczba wystąpień podslowa, pierwsze wystąpienie, najkrótsze niewystępujące podslowo, longest common substring dwóch slów, LCS wielu slów
**Czas:** $\mathcal{O}(n\alpha)$ (szybsze, ale więcej pamięci) albo $\mathcal{O}(n \log \alpha)$ (mapa)

641790, 53 lines

```
struct SuffixAutomaton { 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 nieistniejacy
        edges = {new_node};  // dodajemy stan startowy, ktory
            reprezentuje puste slowo
    }

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

        int r = size(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 = size(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 = "aabaaab" sa = {6, 3, 0, 4, 1, 5, 2}, lcp = {0, 0, 3, 1, 2, 0, 1}`

5923f8, 29 lines

```
struct SuffixArray {
  vector<int> sa, lcp;
  SuffixArray(string& s, int lim = 256) { // lub basic_string<
      int>
    int n = size(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")
```

# Randomowe rzeczy (10)