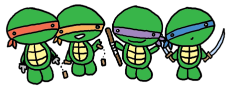

XIII LO Szczecin

Wojownicze Żółwie Ninja

Tomasz Nowak, Michał Staniewski, Justyna Jaworska

2019-10-12

Utils (1)

headers

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

<bits/stdc++.h> 1084fb, 39 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...);
}
#ifdef DEBUG
  const int seed = 1;
  struct Nl{~Nl(){cerr << '\n';}};
# define debug(x...) cerr << (strcmp(#x, "") ? #x ":  " : ""),
    dump(x), Nl(), cerr << ""
#else
  const int seed = chrono::system_clock::now().time_since_epoch
      ().count();
# define debug(...) 0 && cerr
#endif
mt19937_64 rng(seed);
int rd(int l, int r) {
  return uniform_int_distribution<int>(l, r)(rng);
}

// int main() {
//     ios_base::sync_with_stdio(0);
//     cin.tie(0);
//
// }
```

headers/bazshrc.sh
10 lines

```
c() {
  clang++ -O3 -std=c++11 -Wall -Wextra -Wshadow \
    -Wconversion -Wno-sign-conversion -Wfloat-equal \
    -D_GLIBCXX_DEBUB -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 Pitagolas

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

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

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

3.7 Wzorki na pewne ciągi

3.7.1 Derangements

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 Partition function

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 $n+1$ liśćiami (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) = extendedGcd(a, b);`

402db5, 7 lines

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

Struktury danych (5)

find-union

Opis: Find 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 pare

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: indexowanie od 0
Użycie: `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: lepszy set. Jeśli chcemy multiseta, to używamy par {val, id}. Nie dziala z -D_GLIBCXX_DEBUG
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

<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);
  }
};
```

Grafy (6)

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()};
}
```

Stringsy (8)

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;
}
```

Optymizacje (9)

Randomowe rzeczy (10)