



Justyna To Chlep

Wojownicze Żółwie Ninja

Tomasz Nowak, Michał Staniewski, Justyna Jaworska

2019-10-12

- 1 Utils
- 2 Math
- 3 Data structures
- 4 Graphs
- 5 Geometry
- 6 Strings
- 7 Optimizations
- 8 Random stuff

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>
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);
}
```

```
headers/bazshrc.sh
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
set nu rnu hls is nosol ts=4 sw=4 ch=2 sc
filetype indent plugin on
syntax on
example-code
Opis:    jakiś tam opis, można walnąć latexa:    2 + 2    =    5.
          ęóąśłżźćńĖŌĄŚŁŻŻĆŃ
Czas:     $\mathcal{O}(n\sqrt{n}\log^2 n)$ , gdzie  $n$  to jakaś fajna zmienna,  $\mathcal{O}(n\log n)$ 
          pamięciowo
Użycie: int rd = getRandomValue(0, 5);
          int rd01 = ExampleStruct().get();
          ęóąśłżźćńĖŌĄŚŁŻŻĆŃ
mt19937_64 rng(chrono::system_clock::now().time_since_epoch().
  count());
int getRandomValue(int l, int r) {
  return uniform_int_distribution<int>(l, r)(rng);
}
struct ExampleStruct {
  int random_variable;
  constexpr int left = 0, right = 1;
  ExampleStruct() {
    random_variable = getRandomValue(left, right);
    if(random_variable == 0) {
      // some random bulls**t to show the style
      ++random_variable;
    }
    else
      --random_variable;
  }
  int& get_value() {
    return random_variable;
  }
};
Math (2)
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);
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};
}
```

Data structures (3)

```
find-union
Opis: Find 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) {}
};
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
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 <= 1)
        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 <= 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);

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

segment-tree

Opis: Drzewo punkt-przedział
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

```
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

```
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 działa 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>
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 przecinają się co najwyżej raz, oblicza maximum w punkcie x. Podany kod jest dla funkcji liniowych

```
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);
}
};
```

Graphs (4)

Geometry (5)

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)

```
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()};
}
```

Strings (6)

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

Optimizations (7)

Random stuff (8)