Team Note of HYEA Team

hyea, teamnote, sty

Compiled on October 7, 2022

Contents

1 Have you...

1.1 tried...

- **Reading the problem once more?**
- doubting “obvious" things?
- writing obivous things?
- radical greedy approach?
- thinking in reverse direction?
- a greedy algorithm?
- network flow when your greedy algorithms stuck?
- a dynamic programming?
- checking the range of answer?
- random algorithm?
- graph modeling using states?
- inverting state only on odd indexes?
- square root decomposition?
- calculating error bound on a real number usage?

1.2 checked...

- **you have read the statement correctly?**
- typo copying the team note?
- initialization on multiple test case problem?
- additional information from the problem?
- undefined behavior?
- overflow?
- function without return value?
- real number error?
- implicit conversion?
- comparison between signed and unsigned integer?

2 Data Structure

Should be **tested**.

2.1 Bottom-up lazy segment tree

Usage: Give monoid $(S, \cdot)$ and morphism $F : S \to S$ with $f(x \cdot y) = f(x) \cdot f(y)$

- `S`: type of S, `S op(S a, S b)`: $a \cdot b$, `S e()`: identity element e
- `F`: type of F, `S mapping(F f, S s)`: $f(s)$, `F composition(F f, F g)`: $f \circ g$, `F id()`: $id(x) = x$
- `LazySeg(int n)`: $a_0 = \cdots = a_{N-1} = e$, `LazySeg(vector<S> v)`: Init array with `v`
- `void set(int p, S x)`: $a_p = x$, `void apply(int l, int r, F f)`: $a_i = f(a_i)$ for $i = l, \cdots, r-1$.
- `S prod(int l, int r)`: $a_l \cdot a_{l+1} \cdot \cdots \cdot a_{r-1}$, `S all_prod()`: $a_0 \cdot \cdots \cdot a_{N-1}$

Time Complexity: $\mathcal{O}(n)$ for constructor, $\mathcal{O}(\log n)$ for query

```
template <class S, S (*op)(S, S), S (*e)(),
  class F, S (*mapping)(F, S), F (*composition)(F, F), F (*id)()>
class LazySeg {
  int N, log;
  vector<S> d;
  vector<F> lz;

  void pull(int k) { d[k] = op(d[2 * k], d[2 * k + 1]); }
  void put(int k, F f) {
    d[k] = mapping(f, d[k]);
    if (k < N) lz[k] = composition(f, lz[k]);
  }
  void push(int k) {
    put(2 * k, lz[k]);
    put(2 * k + 1, lz[k]);
    lz[k] = id();
  }
public:
  LazySeg() : LazySeg(0) {}
  explicit LazySeg(int n) : LazySeg(vector<S>(n, e())) {}
  explicit LazySeg(const vector<S> &v) {
    log = 31 - __builtin_clz(v.size() | 1);
    N = 1 << log;
    d = vector<S>(2 * N, e());
    lz = vector<F>(N, id());
    for (int i = 0; i < (int)v.size(); i++) d[N + i] = v[i];
    for (int i = N - 1; i >= 1; i--) pull(i);
  }
  void set(int p, S x) {
    p += N;
    for (int i = log; i >= 1; i--) push(p >> i);
    d[p] = x;
    for (int i = 1; i <= log; i++) pull(p >> i);
  }
  S prod(int l, int r) {
    if (l == r) return e();
    l += N, r += N;
    for (int i = log; i >= 1; i--) {
```

```
    if (((l >> i) << i) != l) push(l >> i);
    if (((r >> i) << i) != r) push((r - 1) >> i);
  }
  S sml = e(), smr = e();
  while (l < r) {
    if (l & 1) sml = op(sml, d[l++]);
    if (r & 1) smr = op(d[--r], smr);
    l >>= 1, r >>= 1;
  }
  return op(sml, smr);
}
S all_prod() { return d[1]; }
void apply(int l, int r, F f) {
  if (l == r) return;
  l += N, r += N;
  for (int i = log; i >= 1; i--) {
    if (((l >> i) << i) != l) push(l >> i);
    if (((r >> i) << i) != r) push((r - 1) >> i);
  }
  int l2 = l, r2 = r;
  while (l < r) {
    if (l & 1) put(l++, f);
    if (r & 1) put(--r, f);
    l >>= 1, r >>= 1;
  }
  l = l2, r = r2;
  for (int i = 1; i <= log; i++) {
    if (((l >> i) << i) != l) pull(l >> i);
    if (((r >> i) << i) != r) pull((r - 1) >> i);
  }
}
};
```

Should be **tested**.

2.2 Randomized Meldable Heap

Usage: Min-heap `H` is declared as `Heap<T> H`. You can use `push`, `size`, `empty`, `top`, `pop` as `std::priority_queue`. Use `H.meld(G)` to meld contents from `G` to `H`.

Time Complexity: $\mathcal{O}(logn)$

```
mt19937 gen(0x94949);
template<typename T>
struct Node {
  Node *l, *r;
  T v;
  Node(T x): l(0), r(0), v(x){}
};
template<typename T>
Node<T>* Meld(Node<T>* A, Node<T>* B) {
  if(!A) return B; if(!B) return A;
  if(B->v < A->v) swap(A, B);
  if(gen()&1) A->l = Meld(A->l, B);
  else A->r = Meld(A->r, B);
  return A;
}
template<typename T>
struct Heap {
  Node<T> *r; int s;
  Heap(): r(0), s(0){}
  void push(T x) {
    r = Meld(new Node<T>(x), r);
    ++s;
  }
  int size(){ return s; }
  bool empty(){ return s == 0;}
  T top(){ return r->v; }
  void pop() {
    Node<T>* p = r;
    r = Meld(r->l, r->r);
    delete p;
    --s;
  }
  void Meld(Heap x) {
    s += x->s;
    r = Meld(r, x->r);
  }
};
```

2.3 Convex Hull Trick

Should be **added**.

2.4 Li-Chao Tree

Should be **added**.

2.5 Palindromic Tree

Should be **added**.

2.6 Link-Cut Tree

Should be **added**.

2.7 DSU rollback + Queue undo trick

3 Math

Should be **tested**.

3.1 Modint + Barrett Reduction

Should be **added**.

3.2 Miller Rabin Primarily Test + Pollad Rho Factorization

Usage: `is_prime` For primarily test, `factor` for factorization ($n < 2^{62}$)

Time Complexity: $\mathcal{O}(B \log n)$ $(B \sim 7)$, $\mathcal{O}(n^{1/4})$

```
long mul(long a, long b, long m) {
  return (__int128)a * b % m;
}
long ipow(long a, long b, long m) {
  long r = 1, y = a % m;
  while (b) {
    if (b & 1) r = mul(r, y, m);
    y = mul(y, y, m);
    b >>= 1;
  }
  return r;
}
bool is_prime(long n) {
  if (n <= 1) return false;
  for (int a : {2, 3, 5, 13, 19, 73, 193, 407521, 299210837})
{
    if (n == a) return true;
    if (n % a == 0) return false;
  }
  long d = n - 1;
  while (!(d & 1)) d >>= 1;
  for (int a : {2, 325, 9375, 28178, 450775, 9780504,
1795265022}) {
    long t = d, y = ipow(a, t, n);
    while (t != n - 1 && y != 1 && y != n - 1) y = mul(y, y,
n), t <<= 1;
    if (y != n - 1 && !(t & 1)) return false;
  }
  return true;
}

long pollard(long n) {
  auto f = [n](long x) { return mul(x, x, n) + 1; };
  long x = 0, y = 0, t = 0, prd = 2, i = 1, q;
  while (t++ % 40 || gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = mul(prd, max(x, y) - min(x, y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return gcd(prd, n);
}
vector<long> factor(long n)
{
  if (n == 1) return {};
  if (is_prime(n)) return {n};
  long x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), r.begin(), r.end());
  return l;
}
```

3.3 Linear Determinant

Usage: `char_poly` for $\det(xI - M)$, `det_linear` for $\det(Ax + B)$
Time Complexity: $\mathcal{O}(n^3)$

```
template <typename T>
vector<T> char_poly(vector<vector<T>> M) {
  int N = M.size();
    for (int i = 0; i < N - 2; i++) {
    int p = -1;
    for (int j = i + 1; j < N; j++)
      if (M[j][i] != T(0)) {
        p = j; break;
      }
    if (p == -1) continue;
    M[i + 1].swap(M[p]);
    for (int j = 0; j < N; j++) swap(M[j][i + 1], M[j][p]);

    T r = T(1) / M[i + 1][i];
    for (int j = i + 2; j < N; j++) {
      T c = M[j][i] * r;
      for (int k = 0; k < N; k++) M[j][k] -= M[i + 1][k] * c;
      for (int k = 0; k < N; k++) M[k][i + 1] += M[k][j] * c;
    }
  }
  vector<vector<T>> P = {{T(1)}};
  for (int i = 0; i < N; i++) {
    vector<T> f(i + 2, 0);
    for (int j = 0; j <= i; j++) f[j + 1] += P[i][j];
    for (int j = 0; j <= i; j++) f[j] -= P[i][j] * M[i][i];

    T b = 1;
    for (int j = i - 1; j >= 0; j--) {
      b *= M[j + 1][j];
      T h = -M[j][i] * b;
      for (int k = 0; k <= j; k++) f[k] += h * P[j][k];
    }
    P.push_back(f);
  }
  return P.back();
}

template <typename T>
vector<T> det_linear(vector<vector<T>> A, vector<vector<T>> B)
{
  int N = A.size(), nu = 0; T det = 1;
  for (int i = 0; i < N; i++) {
    int p = -1;
    for (int j = i; j < N; j++)
      if (A[j][i] != T(0)) {
        p = j; break;
      }
    if (p == -1) {
      if (++nu > N) return vector<T>(N + 1, 0);
      for (int j = 0; j < i; j++) {
        for (int k = 0; k < N; k++)
            B[k][i] -= B[k][j] * A[j][i];
        A[j][i] = 0;
      }
      for (int j = 0; j < N; j++) swap(A[j][i], B[j][i]);
      --i; continue;
    }
    if (p != i) A[i].swap(A[p]), B[i].swap(B[p]), det = -det;
    det *= A[i][i];

    T c = T(1) / A[i][i];
    for (int j = 0; j < N; j++) A[i][j] *= c, B[i][j] *= c;
    for (int j = 0; j < N; j++) if (j != i) {
      T c = A[j][i];
      for (int k = 0; k < N; k++)
          A[j][k] -= A[i][k] * c, B[j][k] -= B[i][k] * c;
    }
  }
  for (auto &y : B) for (T &x : y) x = -x;
  auto f = char_poly(B);
  for (T &x : f) x *= det;
  f.erase(f.begin(), f.begin() + nu);
  f.resize(N + 1);
  return f;
}
```

3.4 FFT, NTT, or, xor, and convolution

Should be **added**.

3.5 Polynomial Library

Should be **added**.

3.6 Simplex Algorithm

Should be **added**.

3.7 Berlekamp-Massey Algorithm / Kitamasa

Should be **added**.

3.8 Xudyh Sieve

Should be **added**.

4 String Algorithm

Should be **tested**.

4.1 Suffix Array with LCP

Usage: `s`: string, `upper`: max `s`; e. g. 256 for ascii `string`. `sa`: pass suffix array together
Time Complexity: $\mathcal{O}(N + upper)$ for `SA`, $\mathcal{O}(N)$ for `lcp_array`

```
vector<int> ㄴㅁ(const vector<int>& s, int upper) {
  int n=s.size();
  if (n == 0) return {};
  if (n == 1) return {0};
  if (n == 2) {
    if (s[0] < s[1]) return {0, 1};
    else return {1, 0};
  }
  vector<int> sa(n),sum_l(upper+1), sum_s(upper+1);
  vector<bool> ls(n);
  for (int i=n-2; i>=0; i--)
    ls[i]=(s[i] == s[i+1]) ? ls[i+1] : (s[i] < s[i+1]);
  for (int i = 0; i < n; i++)
    if (!ls[i]) sum_s[s[i]]++;
    else sum_l[s[i]+1]++;
  for (int i=0; i<=upper; i++) {
    sum_s[i] += sum_l[i];
    if (i < upper) sum_l[i+1] += sum_s[i];
  }
  auto induce=[&](const vector<int>& lms) {
    fill(sa.begin(), sa.end(), -1);
    vector<int> buf(upper+1);
    copy(sum_s.begin(), sum_s.end(), buf.begin());
    for (auto d : lms) {
      if (d == n) continue;
      sa[buf[s[d]]++]=d;
    }
    copy(sum_l.begin(), sum_l.end(), buf.begin());
    sa[buf[s[n-1]]++]=n-1;
    for (int i=0; i < n; i++) {
      int v=sa[i];
      if (v>=1 && !ls[v-1]) sa[buf[s[v-1]]++] = v-1;
    }
    copy(sum_l.begin(), sum_l.end(), buf.begin());
    for (int i=n-1; i>=0; i--) {
      int v=sa[i];
      if (v>=1 && ls[v-1]) sa[--buf[s[v-1]+1]] = v-1;
    }
  };
  vector<int> lms_map(n+1, -1), lms;
  int m=0;
  for (int i=1; i < n; i++) if (!ls[i-1] && ls[i]) {
    lms_map[i]=m++;
    lms.push_back(i);
  }
  induce(lms);
  if (m) {
    vector<int> sorted_lms, rec_s(m);
    for (int v : sa) if (lms_map[v] != -1)
sorted_lms.push_back(v);
    int rec_upper=0;
```

```
    rec_s[lms_map[sorted_lms[0]]]=0;
    for (int i=1; i < m; i++) {
      int l=sorted_lms[i-1], r=sorted_lms[i];
      int end_l = (lms_map[l]+1 < m) ? lms[lms_map[l]+1] : n;
      int end_r = (lms_map[r]+1 < m) ? lms[lms_map[r]+1] : n;
      bool same=true;
      if (end_l-l != end_r-r) same=false;
      else {
        while (l < end_l) {
          if (s[l] != s[r]) break;
          l++, r++;
        }
        if (l == n || s[l] != s[r]) same=false;
      }
      if (!same) rec_upper++;
      rec_s[lms_map[sorted_lms[i]]]=rec_upper;
    }
    auto rec_sa = SA(rec_s, rec_upper);
    for (int i=0; i < m; i++) sorted_lms[i] = lms[rec_sa[i]];
    induce(sorted_lms);
  }
  return sa;
}

vector<int> lcp_array(const vector<int>& s, const vector<int>&
sa) {
  int n=int(s.size());
  assert(n>=1);
  vector<int> rnk(n), lcp(n-1);
  for (int i=0; i < n; i++) rnk[sa[i]]=i;
  int h=0;
  for (int i=0; i < n; i++) {
    if (h > 0) h--;
    if (rnk[i] == 0) continue;
    int j=sa[rnk[i]-1];
    for (; j+h < n && i+h < n; h++)
      if (s[j+h] != s[i+h]) break;
    lcp[rnk[i]-1]=h;
  }
  return lcp;
}
```

4.2 Z-algorithm

Usage: i-th element is common prefix of S and $S_{i...|S|}$
Time Complexity: $\mathcal{O}(N)$

```
vector<int> Z(const vector<int>& S) {
    int N = S.size();
    vector<int> Z(N);
    int L = 0, R = 0;
    for(int i = 1; i < N; i++) {
        if(i+Z[i-L] < R) Z[i] = Z[i-L];
        else {
            L = i, R = max(R, i);
            while(R<N && S[R] == S[R-i]) ++R;
            Z[i] = R-i;
        }
    }
    return Z;
}
```

4.3 Manacher

Usage: Returns palindromic radius of S. To calculate even length palindromes, insert `$` between each character.
Time Complexity: $\mathcal{O}(N)$

```
vector<int> M(const vector<int>& S) {
    int N = S.size();
    vector<int> M(N);
    int L = 0, R = 0;
    for(int i = 0; i < N; i++) {
        if(i < R && i+M[2*L-i] < R) M[i] = M[2*L-i];
        else {
            L = i, R = max(R, i);
            while(R < N && 2*i-R >=0 && S[R] == S[2*i-R]) ++R;
            M[i] = R-i;
        }
    }
    return M;
}
```

Should be **revised**.
Support Incremental Aho-corasick

4.4 Aho-Corasick

Usage: `MAXC`: size of alphabet, `F`, `FG`: failure (parent), failure graph, `ftrans`: state transition function.

```
template <int MAXC = 26> struct AhoCorasick {
  vector<array<int, MAXC>> C;
  vector<int> F;
  vector<vector<int>> FG;
  vector<bool> E;

  int node() {
    int r = C.size();
    E.push_back(0);
    F.push_back(-1);
    C.emplace_back();
    fill(C.back().begin(), C.back().end(), -1);
    return r;
  }
  int ctrans(int n, int c) {
    if (C[n][c] == -1) C[n][c] = node();
    return C[n][c];
  }
  int ftrans(int n, int c) const {
    while (n && C[n][c] == -1) n = F[n];
    return C[n][c] != -1 ? C[n][c] : 0;
  }
  AhoCorasick(vector<vector<int>> P) {
    node();
    for (int i = 0; i < (int)P.size(); i++) {
      int n = 0;
      for (int c : P[i]) n = ctrans(n, c);
      E[n] = 1;
    }
    queue<int> Q;
    F[0] = 0;
    for (int c : C[0]) if (c != -1) Q.push(c), F[c] = 0;
    while (!Q.empty()) {
      int n = Q.front(); Q.pop();
      for (int c = 0; c < MAXC; ++c) if (C[n][c] != -1) {
        int f = F[n];
        while (f && C[f][c] == -1) f = F[f];
        F[C[n][c]] = C[f][c] != -1 ? C[f][c] : 0;
        Q.emplace(C[n][c]);
      }
    }
    FG.resize(F.size());
    for (int i = 1; i < (int)F.size(); i++) {
      FG[F[i]].push_back(i);
      if (E[i]) Q.push(i);
    }
    while (!Q.empty()) {
      int n = Q.front();
      Q.pop();
      for (int f : FG[n]) E[f] = 1, Q.push(f);
    }
  }
  bool check(vector<int> V) {
    if (E[0]) return 1;
    int n = 0;
    for (int c : V) {
      n = ftrans(n, c);
      if (E[n]) return 1;
    }
    return 0;
  }
};
```

5 Graph

5.1 Dinic Algorithm

Should be **added**.

5.2 Fast MCMF with slope clculation

Should be **added**.

5.3 Global Min-cut

Should be **added**.

5.4 Gomory-Hu Tree

Should be **added**.

5.5 Perfect Elimination Ordering

Should be **added**.

5.6 General Matching

Usage: Use `init` to init, `addEdge` to add edges, `match` to get matching, `Match` to find maximum matching. Vertices have 1-based index.
Time Complexity: $\mathcal{O}(VE)$

```
const int MAXN = 2020 + 1;
struct GM { // 1-based Vertex index
  int vis[MAXN], par[MAXN], orig[MAXN], match[MAXN],
aux[MAXN], t, N;
  vector<int> conn[MAXN];
  queue<int> Q;
  void addEdge(int u, int v) {
    conn[u].push_back(v); conn[v].push_back(u);
  }
  void init(int n) {
    N = n; t = 0;
    for(int i=0; i<=n; ++i) {
      conn[i].clear();
      match[i] = aux[i] = par[i] = 0;
    }
  }
  void augment(int u, int v) {
    int pv = v, nv;
    do {
      pv = par[v]; nv = match[pv];
      match[v] = pv; match[pv] = v;
      v = nv;
    } while(u != pv);
  }
  int lca(int v, int w) {
    ++t;
    while(true) {
      if(v) {
        if(aux[v] == t) return v; aux[v] = t;
        v = orig[par[match[v]]];
      }
      swap(v, w);
    }
  }
  void blossom(int v, int w, int a) {
    while(orig[v] != a) {
      par[v] = w; w = match[v];
      if(vis[w] == 1) Q.push(w), vis[w] = 0;
      orig[v] = orig[w] = a;
      v = par[w];
    }
  }
  bool bfs(int u) {
    fill(vis+1, vis+1+N, -1); iota(orig + 1, orig + N + 1, 1);
    Q = queue<int> (); Q.push(u); vis[u] = 0;
    while(!Q.empty()) {
      int v = Q.front(); Q.pop();
      for(int x: conn[v]) {
        if(vis[x] == -1) {
          par[x] = v; vis[x] = 1;
          if(!match[x]) return augment(u, x), true;
          Q.push(match[x]); vis[match[x]] = 0;
        }
        else if(vis[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return false;
  }
  int Match() {
    int ans = 0;
    //find random matching (not necessary, constant
improvement)
    vector<int> V(N-1); iota(V.begin(), V.end(), 1);
    shuffle(V.begin(), V.end(), mt19937(0x94949));
    for(auto x: V) if(!match[x]){
      for(auto y: conn[x]) if(!match[y]) {
        match[x] = y, match[y] = x;
        ++ans; break;
      }
    }
    for(int i=1; i<=N; ++i) if(!match[i] && bfs(i)) ++ans;
    return ans;
  }
};
```

5.7 Centroid Decomposition

Usage: Fill in the function `work`.
Time Complexity: $\mathcal{O}(N \log N)$

```
int find_centroid(const vector<vector<int>> &G, const
vector<bool> &used, int v) {
  vector<tuple<int, int, int>> sz;
  function<void(int, int)> dfs = [&](int a, int p) {
    int S = 1, mx = 0;
    for (int x : G[a]) if (x != p && !used[x]) {
          dfs(x, a);
          int c = get<1>(sz.back());
          S += c, mx = max(mx, c);
        }
    sz.emplace_back(a, S, mx);
  };
  dfs(v, -1);
  int S = get<1>(sz.back());
  for (auto [i, s, mx] : sz) if (2 * max(S - s, mx) <= S)
return i;
}
answer_type solve(const vector<vector<int>>& G) {
  vector<bool> used(size(G), 0);
  answer_type answer;
  auto work = [&](int c) {
        /* Do something on rooted tree c
           DFS with !used[x] (See above) */
  };
  queue<int> Q; Q.emplace(0);
  while (!Q.empty()){
    int x = Q.front();
    Q.pop();
    int c = find_centroid(G, used, x);
    work(c);
    used[c] = 1;
    for (int x : G[c]) if (!used[x]) Q.emplace(x);
  }
  return answer;
}
```

5.8 SCC

Should be **added**.

5.9 BCC, Cut vertex, Cut edge

Should be **added**.

6 Geometry

6.1 Smallest Enclosing Circle

Usage: Use `solve` with `vector<Point>`. It returns `Circle c`, `c.p` is center, `c.r` is radius.
Time Complexity: $\mathcal{O}(n)$

```
double eps = 1e-9;
using Point = complex<double>;
struct Circle{ Point p; double r; };
double dist(Point p, Point q){ return abs(p-q); }
double area2(Point p, Point q){ return (conj(p)*q).imag();}
bool in(const Circle& c, Point p){ return dist(c.p, p) < c.r +
eps; }
Circle INVAL = Circle{Point(0, 0), -1};
Circle mCC(Point a, Point b, Point c){
  b -= a; c -= a;
  double d = 2*(conj(b)*c).imag(); if(abs(d)<eps) return
INVAL;
  Point ans = (c*norm(b) - b*norm(c)) * Point(0, -1) / d;
  return Circle{a + ans, abs(ans)};
```

```
}
Circle solve(vector<Point> p) {
  mt19937 gen(0x94949); shuffle(p.begin(), p.end(), gen);
  Circle c = INVAL;
  for(int i=0; i<p.size(); ++i) if(c.r<0 ||!in(c, p[i])){
    c = Circle{p[i], 0};
    for(int j=0; j<=i; ++j) if(!in(c, p[j])){
      Circle ans{(p[i]+p[j])*0.5, dist(p[i], p[j])*0.5};
      if(c.r == 0) {c = ans; continue;}
      Circle l, r; l = r = INVAL;
      Point pq = p[j]-p[i];
      for(int k=0; k<=j; ++k) if(!in(ans, p[k])) {
        double a2 = area2(pq, p[k]-p[i]);
        Circle c = mCC(p[i], p[j], p[k]);
        if(c.r<0) continue;
        else if(a2 > 0 && (l.r<0||area2(pq, c.p-p[i]) >
area2(pq, l.p-p[i]))) l = c;
        else if(a2 < 0 && (r.r<0||area2(pq, c.p-p[i]) <
area2(pq, r.p-p[i]))) r = c;
      }
      if(l.r<0&&r.r<0) c = ans;
      else if(l.r<0) c = r;
      else if(r.r<0) c = l;
      else c = l.r<=r.r?l:r;
    }
  }
  return c;
}
```

6.2 Voronoi Diagram + Delaneuy Triagulation

Should be **added**.

7 Misc

Should be **revised**.
Working in progress.

7.1 (WIP) Magical Polynomial 3-SAT Algorithm

Usage: Use this to solve all problems!
Time Complexity: $\mathcal{O}(n)$

7.2 Policy-based Data structure

Should be **added**.

7.3 Fast I/O

Should be **added**.