Contents

Think twice, code once

Template.cpp

```cpp
#pragma GCC optimize("Ofast,unroll-loops,no-stack-protector
    ")
#include <bits/stdc++.h>
using namespace std;

#define fore(i, l, r)
    \
  for (auto i = (l) - ((l) > (r)); i != (r) - ((l) > (r));
      \
      i += 1 - 2 * ((l) > (r)))
#define sz(x) int(x.size())
#define all(x) begin(x), end(x)
#define f first
#define s second
#define pb push_back

#ifdef LOCAL
#include "debug.h"
#else
#define debug(...)
#endif

using ld = long double;
using lli = long long;
using ii = pair<int, int>;

int main() {
  cin.tie(0)->sync_with_stdio(0), cout.tie(0);
  return 0;
}
```

Debug.h

```cpp
#include <bits/stdc++.h>
using namespace std;

template <class A, class B>
ostream& operator<<(ostream& os, const pair<A, B>& p) {
  return os << "(" << p.first << ", " << p.second << ")";
}

template <class A, class B, class C>
basic_ostream<A, B>& operator<<(basic_ostream<A, B>& os,
    const C& c) {
  os << "[";
  for (const auto& x : c) os << ", " + 2 * (&x == &*begin(c
      )) << x;
  return os << "]";
}

void print(string s) { cout << endl; }

template <class H, class... T>
void print(string s, const H& h, const T&... t) {
  const static string reset = "\033[0m", blue = "\033[1;34m
      ",
                      purple = "\033[3;95m";
  bool ok = 1;
  do {
    if (s[0] == '\"')
      ok = 0;
    else
      cout << blue << s[0] << reset;
    s = s.substr(1);
  } while (s.size() && s[0] != ',');
  if (ok) cout << ": " << purple << h << reset;
  print(s, t...);
}

#define debug(...) print(#__VA_ARGS__, __VA_ARGS__)
```

Randoms

```cpp
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
```

Compilation (gedit ~/.zshenv)

```bash
compile() {
  alias flags='-Wall -Wextra -Wfatal-errors -Wshadow -w -
      mcmodel=medium'
  g++-11 --std=c++17 $2 ${flags} $1.cpp -o $1
}

go() {
  file=$1
  name="${file%.*}"
  compile ${name} $3
  ./${name} < $2
}

run() { go $1 $2 "" }
debug() { go $1 $2 -DLOCAL }
```

1 Data structures

1.1 Sparse table

```cpp
template <class T, class F = function<T(const T&, const T&)
    >>
struct Sparse {
  vector<T> sp[21]; // n <= 2^21
  F f;
  int n;

  Sparse(T* begin, T* end, const F& f) : Sparse(vector<T>(
      begin, end), f) {}

  Sparse(const vector<T>& a, const F& f) : f(f), n(sz(a)) {
    sp[0] = a;
    for (int k = 1; (1 << k) <= n; k++) {
      sp[k].resize(n - (1 << k) + 1);
      fore (l, 0, sz(sp[k])) {
        int r = l + (1 << (k - 1));
        sp[k][l] = f(sp[k - 1][l], sp[k - 1][r]);
      }
    }
  }

  T query(int l, int r) {
#warning Can give TLE D:, change it to a log table
    int k = __lg(r - l + 1);
    return f(sp[k][l], sp[k][r - (1 << k) + 1]);
  }

  T queryBits(int l, int r) {
    optional<T> ans;
    for (int len = r - l + 1; len; len -= len & -len) {
      int k = __builtin_ctz(len);
      ans = ans ? f(ans.value(), sp[k][l]) : sp[k][l];
      l += (1 << k);
    }
    return ans.value();
  }
};
```

1.2 Fenwick 2D offline

```cpp
template <class T>
struct Fenwick2D { // add, build then update, query
```

```
  vector<vector<T>> fenw;
  vector<vector<int>> ys;
  vector<int> xs;
  vector<ii> pts;

  void add(int x, int y) { pts.pb({x, y}); }

  void build() {
    sort(all(pts));
    for (auto&& [x, y] : pts) {
      if (xs.empty() || x != xs.back()) xs.pb(x);
      swap(x, y);
    }
    fenw.resize(sz(xs)), ys.resize(sz(xs));
    sort(all(pts));
    for (auto&& [x, y] : pts) {
      swap(x, y);
      int i = lower_bound(all(xs), x) - xs.begin();
      for (; i < sz(fenw); i |= i + 1)
        if (ys[i].empty() || y != ys[i].back()) ys[i].pb(y)
            ;
    }
    fore (i, 0, sz(fenw)) fenw[i].resize(sz(ys[i]), T());
  }

  void update(int x, int y, T v) {
    int i = lower_bound(all(xs), x) - xs.begin();
    for (; i < sz(fenw); i |= i + 1) {
      int j = lower_bound(all(ys[i]), y) - ys[i].begin();
      for (; j < sz(fenw[i]); j |= j + 1) fenw[i][j] += v;
    }
  }

  T query(int x, int y) {
    T v = T();
    int i = upper_bound(all(xs), x) - xs.begin() - 1;
    for (; i >= 0; i &= i + 1, --i) {
      int j = upper_bound(all(ys[i]), y) - ys[i].begin() -
          1;
      for (; j >= 0; j &= j + 1, --j) v += fenw[i][j];
    }
    return v;
  }
};
```

1.3 Persistent segtree

```
template <class T>
struct Per {
  int l, r;
  Per *left, *right;
  T val;

  Per(int l, int r) : l(l), r(r), left(0), right(0) {}

  Per* pull() {
    val = left->val + right->val;
    return this;
  }

  void build() {
    if (l == r) return;
    int m = (l + r) >> 1;
    (left = new Per(l, m))->build();
    (right = new Per(m + 1, r))->build();
    pull();
  }

  template <class... Args>
  Per* update(int p, const Args&... args) {
    if (p < l || r < p) return this;
    Per* t = new Per(l, r);
    if (l == r) {
      t->val = T(args...);
      return t;
    }
    t->left = left->update(p, args...);
    t->right = right->update(p, args...);
    return t->pull();
  }

  T query(int ll, int rr) {
    if (r < ll || rr < l) return T();
    if (ll <= l && r <= rr) return val;
    return left->query(ll, rr) + right->query(ll, rr);
  }
};
```

1.4 Li Chao

```
struct LiChao {
  struct Fun {
    lli m = 0, c = -INF;
    lli operator()(lli x) const { return m * x + c; }
  } f;

  lli l, r;
  LiChao *left, *right;
  LiChao(lli l, lli r, Fun f) : l(l), r(r), f(f), left(0),
      right(0) {}

  void add(Fun& g) {
    lli m = (l + r) >> 1;
    bool bl = g(l) > f(l), bm = g(m) > f(m);
    if (bm) swap(f, g);
    if (l == r) return;
    if (bl != bm)
      left ? left->add(g) : void(left = new LiChao(l, m, g)
          );
    else
      right ? right->add(g) : void(right = new LiChao(m + 1
          , r, g));
  }

  lli query(lli x) {
    if (l == r) return f(x);
    lli m = (l + r) >> 1;
    if (x <= m) return max(f(x), left ? left->query(x) : -
        INF);
    return max(f(x), right ? right->query(x) : -INF);
  }
};
```

1.5 Wavelet

```
struct Wav {
  int lo, hi;
  Wav *left, *right;
  vector<int> amt;

  template <class Iter>
  Wav(int lo, int hi, Iter b, Iter e) : lo(lo), hi(hi) { //
      array 1-indexed
    if (lo == hi || b == e) return;
    amt.reserve(e - b + 1);
    amt.pb(0);
    int mid = (lo + hi) >> 1;
    auto leq = [mid](auto x) { return x <= mid; };
    for (auto it = b; it != e; it++) amt.pb(amt.back() +
        leq(*it));
    auto p = stable_partition(b, e, leq);
    left = new Wav(lo, mid, b, p);
    right = new Wav(mid + 1, hi, p, e);
  }
```

```
  // kth value in [l, r]
  int kth(int l, int r, int k) {
    if (r < l) return 0;
    if (lo == hi) return lo;
    if (k <= amt[r] - amt[l - 1]) return left->kth(amt[l -
        1] + 1, amt[r], k);
    return right->kth(l - amt[l - 1], r - amt[r], k - amt[r
        ] + amt[l - 1]);
  }

  // Count all values in [l, r] that are in range [x, y]
  int count(int l, int r, int x, int y) {
    if (r < l || y < x || y < lo || hi < x) return 0;
    if (x <= lo && hi <= y) return r - l + 1;
    return left->count(amt[l - 1] + 1, amt[r], x, y) +
           right->count(l - amt[l - 1], r - amt[r], x, y);
  }
};
```

1.6 Static to dynamic

```
template <class Black, class T>
struct StaticDynamic {
  Black box[25];
  vector<T> st[25];

  void insert(T& x) {
    int p = 0;
    while (p < 25 && !st[p].empty()) p++;
    st[p].pb(x);
    fore (i, 0, p) {
      st[p].insert(st[p].end(), all(st[i]));
      box[i].clear(), st[i].clear();
    }
    for (auto y : st[p]) box[p].insert(y);
    box[p].init();
  }
};
```

1.7 Ordered tree

It's a set/map, for a multiset/multimap (? add them as pairs (a[i], i)

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template <class K, class V = null_type>
using OrderedTree =
    tree<K, V, less<K>, rb_tree_tag,
         tree_order_statistics_node_update>;
#define rank order_of_key
#define kth find_by_order
```

1.8 Treap

```
struct Treap {
  static Treap* null;
  Treap *left, *right;
  unsigned pri = rng(), sz = 0;
  int val = 0;

  void push() {
    // propagate like segtree, key-values aren't modified!!
  }

  Treap* pull() {
    sz = left->sz + right->sz + (this != null);
    // merge(left, this), merge(this, right)
    return this;
  }

  Treap() { left = right = null; }

  Treap(int val) : val(val) {
    left = right = null;
    pull();
  }

  template <class F>
  pair<Treap*, Treap*> split(const F& leq) { // {<= val, >
      val}
    if (this == null) return {null, null};
    push();
    if (leq(this)) {
      auto p = right->split(leq);
      right = p.f;
      return {pull(), p.s};
    } else {
      auto p = left->split(leq);
      left = p.s;
      return {p.f, pull()};
    }
  }

  Treap* merge(Treap* other) {
    if (this == null) return other;
    if (other == null) return this;
    push(), other->push();
    if (pri > other->pri) {
      return right = right->merge(other), pull();
    } else {
      return other->left = merge(other->left), other->pull
          ();
    }
  }

  pair<Treap*, Treap*> leftmost(int k) { // 1-indexed
    return split([&](Treap* n) {
      int sz = n->left->sz + 1;
      if (k >= sz) {
        k -= sz;
        return true;
      }
      return false;
    });
  }

  auto split(int x) {
    return split([&](Treap* n) { return n->val <= x; });
  }

  Treap* insert(int x) {
    auto&& [leq, ge] = split(x);
    // auto &&[le, eq] = split(x); // uncomment for set
    return leq->merge(new Treap(x))->merge(ge); // change
        leq for le for set
  }

  Treap* erase(int x) {
    auto&& [leq, ge] = split(x);
    auto&& [le, eq] = leq->split(x - 1);
    auto&& [kill, keep] = eq->leftmost(1); // comment for
        set
    return le->merge(keep)->merge(ge); // le->merge(ge) for
         set
  }
}* Treap::null = new Treap;
```

1.9 Persistent Treap

```
struct PerTreap {
```

```cpp
  static PerTreap* null;
  PerTreap *left, *right;
  unsigned pri = rng(), sz = 0;
  int val;

  void push() {
    // propagate like segtree, key-values aren't modified!!
  }

  PerTreap* pull() {
    sz = left->sz + right->sz + (this != null);
    // merge(left, this), merge(this, right)
    return this;
  }

  PerTreap(int val = 0) : val(val) {
    left = right = null;
    pull();
  }

  PerTreap(PerTreap* t)
      : left(t->left), right(t->right), pri(t->pri), sz(t->
          sz) {
    val = t->val;
  }

  template <class F>
  pair<PerTreap*, PerTreap*> split(const F& leq) { // {<=
       val, > val}
    if (this == null) return {null, null};
    push();
    PerTreap* t = new PerTreap(this);
    if (leq(this)) {
      auto p = t->right->split(leq);
      t->right = p.f;
      return {t->pull(), p.s};
    } else {
      auto p = t->left->split(leq);
      t->left = p.s;
      return {p.f, t->pull()};
    }
  }

  PerTreap* merge(PerTreap* other) {
    if (this == null) return new PerTreap(other);
    if (other == null) return new PerTreap(this);
    push(), other->push();
    PerTreap* t;
    if (pri > other->pri) {
      t = new PerTreap(this);
      t->right = t->right->merge(other);
    } else {
      t = new PerTreap(other);
      t->left = merge(t->left);
    }
    return t->pull();
  }

  auto leftmost(int k) { // 1-indexed
    return split([&](PerTreap* n) {
      int sz = n->left->sz + 1;
      if (k >= sz) {
        k -= sz;
        return true;
      }
      return false;
    });
  }

  auto split(int x) {
    return split([&](PerTreap* n) { return n->val <= x; });
  }
}* PerTreap::null = new PerTreap;
```

2 Dynamic programming

2.1 All submasks of a mask

```cpp
for (int B = A; B > 0; B = (B - 1) & A)
```

2.2 Broken profile $\mathcal{O}(n \cdot m \cdot 2^n)$ with $n \leq m$

Cuenta todas las maneras en las que puedes acomodar fichas de 1x2 y 2x1 en un tablero $n \cdot m$

```cpp
// Answer in dp[m][0][0]

lli dp[2][N][1 << N];

dp[0][0][0] = 1;

fore (c, 0, m) {
  fore (r, 0, n + 1)
    fore (mask, 0, 1 << n) {
      if (r == n) {
        dp[~c & 1][0][mask] += dp[c & 1][r][mask];
        continue;
      }

      if (~(mask >> r) & 1) {
        dp[c & 1][r + 1][mask | (1 << r)] += dp[c & 1][r][
            mask];

        if (~(mask >> (r + 1)) & 1)
          dp[c & 1][r + 2][mask] += dp[c & 1][r][mask];
      } else {
        dp[c & 1][r + 1][mask & ~(1 << r)] += dp[c & 1][r][
            mask];
      }
    }

  fore (r, 0, n + 1)
    fore (mask, 0, 1 << n) dp[c & 1][r][mask] = 0;
}
```

2.3 Convex hull trick $\mathcal{O}(n^2) \Rightarrow \mathcal{O}(n)$

$dp[i] = \min_{j<i}(dp[j] + b[j] * a[i])$
$dp[i][j] = \min_{k<j}(dp[i-1][k] + b[k] * a[j])$
$b[j] \geq b[j+1]$ optionally $a[i] \leq a[i+1]$

```cpp
// for doubles, use INF = 1/.0, div(a,b) = a / b
struct Line {
  mutable lli m, c, p;
  bool operator<(const Line& l) const { return m < l.m; }
  bool operator<(lli x) const { return p < x; }
  lli operator()(lli x) const { return m * x + c; }
};

template <bool MAX>
struct DynamicHull : multiset<Line, less<>> {
  lli div(lli a, lli b) { return a / b - ((a ^ b) < 0 && a
       % b); }

  bool isect(iterator i, iterator j) {
    if (j == end()) return i->p = INF, 0;
    if (i->m == j->m)
      i->p = i->c > j->c ? INF : -INF;
    else
      i->p = div(i->c - j->c, j->m - i->m);
    return i->p >= j->p;
  }
```

```cpp
  void add(lli m, lli c) {
    if (!MAX) m = -m, c = -c;
    auto k = insert({m, c, 0}), j = k++, i = j;
    while (isect(j, k)) k = erase(k);
    if (i != begin() && isect(--i, j)) isect(i, j = erase(j
        ));
    while ((j = i) != begin() && (--i)->p >= j->p) isect(i,
         erase(j));
  }

  lli query(lli x) {
    if (empty()) return 0LL;
    auto f = *lower_bound(x);
    return MAX ? f(x) : -f(x);
  }
};
```

2.4 Divide and conquer $\mathcal{O}(k \cdot n^2) \Rightarrow \mathcal{O}(k \cdot nlogn)$

Split the array of size n into k continuous groups. $k \leq n$ $cost(a,c) + cost(b,d) \leq cost(a,d) + cost(b,c)$ with $a \leq b \leq c \leq d$

```cpp
lli dp[2][N];

void solve(int cut, int l, int r, int optl, int optr) {
  if (r < l) return;
  int mid = (l + r) / 2;
  pair<lli, int> best = {INF, -1};
  fore (p, optl, min(mid, optr) + 1)
    best = min(best, {dp[~cut & 1][p - 1] + cost(p, mid), p
        });
  dp[cut & 1][mid] = best.f;
  solve(cut, l, mid - 1, optl, best.s);
  solve(cut, mid + 1, r, best.s, optr);
}

fore (i, 1, n + 1) dp[1][i] = cost(1, i);
fore (cut, 2, k + 1) solve(cut, cut, n, cut, n);
```

2.5 Knuth $\mathcal{O}(n^3) \Rightarrow \mathcal{O}(n^2)$

$dp[l][r] = \min_{l \leq k \leq r} \{dp[l][k] + dp[k][r]\} + cost(l,r)$

```cpp
lli dp[N][N];
int opt[N][N];

fore (len, 1, n + 1)
  fore (l, 0, n) {
    int r = l + len - 1;
    if (r > n - 1) break;
    if (len <= 2) {
      dp[l][r] = 0;
      opt[l][r] = l;
      continue;
    }
    dp[l][r] = INF;
    fore (k, opt[l][r - 1], opt[l + 1][r] + 1) {
      lli cur = dp[l][k] + dp[k][r] + cost(l, r);
      if (cur < dp[l][r]) {
        dp[l][r] = cur;
        opt[l][r] = k;
      }
    }
  }
```

2.6 Matrix exponentiation $\mathcal{O}(n^3 \cdot logn)$

If TLE change `Mat` to `array<array<T, N>, N>`

```cpp
template <class T>
struct Mat : vector<vector<T>> {
  int n, m;

  Mat(int n, int m) : vector<vector<T>>(n, vector<T>(m)), n
      (n), m(m) {}

  Mat<T> operator*(const Mat<T>& other) {
    assert(m == other.n);
    Mat<T> ans(n, other.m);
    fore (k, 0, m)
      fore (i, 0, n)
        fore (j, 0, other.m) ans[i][j] += (*this)[i][k] *
            other[k][j];
    return ans;
  }

  Mat<T> pow(lli k) {
    assert(n == m);
    Mat<T> ans(n, n);
    fore (i, 0, n) ans[i][i] = 1;
    for (; k > 0; k >>= 1) {
      if (k & 1) ans = ans * *this;
      *this = *this * *this;
    }
    return ans;
  }
};
```

2.7 SOS dp

```cpp
// N = amount of bits
// dp[mask] = Sum of all dp[x] such that 'x' is a submask
    of 'mask'
fore (i, 0, N)
  fore (mask, 0, 1 << N)
    if (mask >> i & 1) { dp[mask] += dp[mask ^ (1 << i)]; }
```

2.8 Inverse SOS dp

```cpp
// dp[mask] = Sum of all dp[x] such that 'mask' is a
    submask of 'x'
fore (i, 0, N) {
  for (int mask = (1 << N) - 1; mask >= 0; mask--)
    if (mask >> i & 1) { dp[mask ^ (1 << i)] += dp[mask]; }
```

2.9 Steiner

```cpp
// Connect special nodes by a minimum spanning tree
// special nodes [0, k)
fore (u, k, n)
  fore (a, 0, k) umin(dp[u][1 << a], dist[u][a]);
fore (A, 0, (1 << k))
  fore (u, k, n) {
    for (int B = A; B > 0; B = (B - 1) & A)
      umin(dp[u][A], dp[u][B] + dp[u][A ^ B]);
    fore (v, k, n) umin(dp[v][A], dp[u][A] + dist[u][v]);
  }
```

3 Geometry

3.1 Geometry

```cpp
const ld EPS = 1e-20;
const ld INF = 1e18;
const ld PI = acos(-1.0);
enum { ON = -1, OUT, IN, OVERLAP };

#define eq(a, b) (abs((a) - (b)) <= +EPS)
#define neq(a, b) (!eq(a, b))
#define geq(a, b) ((a) - (b) >= -EPS)
```

```
#define leq(a, b) ((a) - (b) <= +EPS)
#define ge(a, b) ((a) - (b) > +EPS)
#define le(a, b) ((a) - (b) < -EPS)

int sgn(ld a) { return (a > EPS) - (a < -EPS); }
```

3.2 Radial order

```
struct Radial {
  Pt c;
  Radial(Pt c) : c(c) {}

  int cuad(Pt p) const {
    if (p.x > 0 && p.y >= 0) return 0;
    if (p.x <= 0 && p.y > 0) return 1;
    if (p.x < 0 && p.y <= 0) return 2;
    if (p.x >= 0 && p.y < 0) return 3;
    return -1;
  }

  bool operator()(Pt a, Pt b) const {
    Pt p = a - c, q = b - c;
    if (cuad(p) == cuad(q)) return p.y * q.x < p.x * q.y;
    return cuad(p) < cuad(q);
  }
};
```

3.3 Sort along line

```
void sortAlongLine(vector<Pt>& pts, Line l) {
  sort(all(pts), [&](Pt a, Pt b) { return a.dot(l.v) < b.
      dot(l.v); });
}
```

4 Point

4.1 Point

```
struct Pt {
  ld x, y;
  explicit Pt(ld x = 0, ld y = 0) : x(x), y(y) {}

  Pt operator+(Pt p) const { return Pt(x + p.x, y + p.y); }

  Pt operator-(Pt p) const { return Pt(x - p.x, y - p.y); }

  Pt operator*(ld k) const { return Pt(x * k, y * k); }

  Pt operator/(ld k) const { return Pt(x / k, y / k); }

  ld dot(Pt p) const {
    // 0 if vectors are orthogonal
    // - if vectors are pointing in opposite directions
    // + if vectors are pointing in the same direction
    return x * p.x + y * p.y;
  }

  ld cross(Pt p) const {
    // 0 if collinear
    // - if p is to the right of a
    // + if p is to the left of a
    // gives you 2 * area
    return x * p.y - y * p.x;
  }

  ld norm() const { return x * x + y * y; }

  ld length() const { return sqrtl(norm()); }

  Pt unit() const { return (*this) / length(); }

  ld angle() const {
    ld ang = atan2(y, x);
    return ang + (ang < 0 ? 2 * acos(-1) : 0);
  }

  Pt perp() const { return Pt(-y, x); }

  Pt rotate(ld angle) const {
    // counter-clockwise rotation in radians
    // degree = radian * 180 / pi
    return Pt(x * cos(angle) - y * sin(angle), x * sin(
        angle) + y * cos(angle));
  }

  int dir(Pt a, Pt b) const {
    // where am I on the directed line ab
    return sgn((a - *this).cross(b - *this));
  }

  bool operator<(Pt p) const { return eq(x, p.x) ? le(y, p.
      y) : le(x, p.x); }

  bool operator==(Pt p) const { return eq(x, p.x) && eq(y,
      p.y); }

  bool operator!=(Pt p) const { return !(*this == p); }

  friend ostream& operator<<(ostream& os, const Pt& p) {
    return os << "(" << p.x << ", " << p.y << ")";
  }

  friend istream& operator>>(istream& is, Pt& p) { return
      is >> p.x >> p.y; }
};
```

4.2 Angle between vectors

```
ld angleBetween(Pt a, Pt b) {
  ld x = a.dot(b) / a.length() / b.length();
  return acosl(max(-1.0, min(1.0, x)));
}
```

4.3 Closest pair of points $\mathcal{O}(n \cdot logn)$

```
pair<Pt, Pt> closestPairOfPoints(vector<Pt>& pts) {
  sort(all(pts), [&](Pt a, Pt b) { return le(a.y, b.y); });
  set<Pt> st;
  ld ans = INF;
  Pt p, q;
  int pos = 0;
  fore (i, 0, sz(pts)) {
    while (pos < i && geq(pts[i].y - pts[pos].y, ans)) st.
        erase(pts[pos++]);
    auto lo = st.lower_bound(Pt(pts[i].x - ans - EPS, -INF)
        );
    auto hi = st.upper_bound(Pt(pts[i].x + ans + EPS, -INF)
        );
    for (auto it = lo; it != hi; ++it) {
      ld d = (pts[i] - *it).length();
      if (le(d, ans)) ans = d, p = pts[i], q = *it;
    }
    st.insert(pts[i]);
  }
  return {p, q};
}
```

4.4 KD Tree

Returns nearest point, to avoid self-nearest add an id to the point

```
struct Pt {
  // Geometry point mostly
  ld operator[](int i) const { return i == 0 ? x : y; }
};

struct KDTree {
```

```
  Pt p;
  int k;
  KDTree *left, *right;

  template <class Iter>
  KDTree(Iter l, Iter r, int k = 0) : k(k), left(0), right(
      0) {
    int n = r - l;
    if (n == 1) {
      p = *l;
      return;
    }
    nth_element(l, l + n / 2, r, [&](Pt a, Pt b) { return a
        [k] < b[k]; });
    p = *(l + n / 2);
    left = new KDTree(l, l + n / 2, k ^ 1);
    right = new KDTree(l + n / 2, r, k ^ 1);
  }

  pair<ld, Pt> nearest(Pt x) {
    if (!left && !right) return {(p - x).norm(), p};
    vector<KDTree*> go = {left, right};
    auto delta = x[k] - p[k];
    if (delta > 0) swap(go[0], go[1]);
    auto best = go[0]->nearest(x);
    if (best.f > delta * delta) best = min(best, go[1]->
        nearest(x));
    return best;
  }
};
```

5 Lines and segments

5.1 Line

```
struct Line {
  Pt a, b, v;

  Line() {}
  Line(Pt a, Pt b) : a(a), b(b), v((b - a).unit()) {}

  bool contains(Pt p) { return eq((p - a).cross(b - a), 0);
      }

  int intersects(Line l) {
    if (eq(v.cross(l.v), 0)) return eq((l.a - a).cross(v),
        0) ? 1e9 : 0;
    return 1;
  }

  int intersects(Seg s) {
    if (eq(v.cross(s.v), 0)) return eq((s.a - a).cross(v),
        0) ? 1e9 : 0;
    return a.dir(b, s.a) != a.dir(b, s.b);
  }

  template <class Line>
  Pt intersection(Line l) { // can be a segment too
    return a + v * ((l.a - a).cross(l.v) / v.cross(l.v));
  }

  Pt projection(Pt p) { return a + v * proj(p - a, v); }

  Pt reflection(Pt p) { return a * 2 - p + v * 2 * proj(p -
      a, v); }
};
```

5.2 Segment

```
struct Seg {
  Pt a, b, v;

  Seg() {}
  Seg(Pt a, Pt b) : a(a), b(b), v(b - a) {}

  bool contains(Pt p) {
    return eq(v.cross(p - a), 0) && leq((a - p).dot(b - p),
        0);
  }

  int intersects(Seg s) {
    int d1 = a.dir(b, s.a), d2 = a.dir(b, s.b);
    if (d1 != d2) return s.a.dir(s.b, a) != s.a.dir(s.b, b)
        ;
    return d1 == 0 && (contains(s.a) || contains(s.b) || s.
        contains(a) ||
                       s.contains(b))
               ? 1e9
               : 0;
  }

  template <class Seg>
  Pt intersection(Seg s) { // can be a line too
    return a + v * ((s.a - a).cross(s.v) / v.cross(s.v));
  }
};
```

5.3 Projection

```
ld proj(Pt a, Pt b) { return a.dot(b) / b.length(); }
```

5.4 Distance point line

```
ld distance(Pt p, Line l) {
  Pt q = l.projection(p);
  return (p - q).length();
}
```

5.5 Distance point segment

```
ld distance(Pt p, Seg s) {
  if (le((p - s.a).dot(s.b - s.a), 0)) return (p - s.a).
      length();
  if (le((p - s.b).dot(s.a - s.b), 0)) return (p - s.b).
      length();
  return abs((s.a - p).cross(s.b - p) / (s.b - s.a).length
      ());
}
```

5.6 Distance segment segment

```
ld distance(Seg a, Seg b) {
  if (a.intersects(b)) return 0.L;
  return min(
      {distance(a.a, b), distance(a.b, b), distance(b.a, a)
          , distance(b.b, a)});
}
```

6 Circle

6.1 Circle

```
struct Cir : Pt {
  ld r;
  Cir() {}
  Cir(ld x, ld y, ld r) : Pt(x, y), r(r) {}
  Cir(Pt p, ld r) : Pt(p), r(r) {}

  int inside(Cir c) {
    ld l = c.r - r - (*this - c).length();
    return ge(l, 0) ? IN : eq(l, 0) ? ON : OVERLAP;
  }

  int outside(Cir c) {
    ld l = (*this - c).length() - r - c.r;
    return ge(l, 0) ? OUT : eq(l, 0) ? ON : OVERLAP;
  }
```

```
  int contains(Pt p) {
    ld l = (p - *this).length() - r;
    return le(l, 0) ? IN : eq(l, 0) ? ON : OUT;
  }

  Pt projection(Pt p) { return *this + (p - *this).unit() *
       r; }

  vector<Pt> tangency(Pt p) {
    // point outside the circle
    Pt v = (p - *this).unit() * r;
    ld d2 = (p - *this).norm(), d = sqrt(d2);
    if (leq(d, 0)) return {}; // on circle, no tangent
    Pt v1 = v * (r / d), v2 = v.perp() * (sqrt(d2 - r * r)
         / d);
    return {*this + v1 - v2, *this + v1 + v2};
  }

  vector<Pt> intersection(Cir c) {
    ld d = (c - *this).length();
    if (eq(d, 0) || ge(d, r + c.r) || le(d, abs(r - c.r)))
      return {}; // circles don't intersect
    Pt v = (c - *this).unit();
    ld a = (r * r + d * d - c.r * c.r) / (2 * d);
    Pt p = *this + v * a;
    if (eq(d, r + c.r) || eq(d, abs(r - c.r)))
      return {p}; // circles touch at one point
    ld h = sqrt(r * r - a * a);
    Pt q = v.perp() * h;
    return {p - q, p + q}; // circles intersects twice
  }

  template <class Line>
  vector<Pt> intersection(Line l) {
    // for a segment you need to check that the point lies
         on the segment
    ld h2 =
        r * r - l.v.cross(*this - l.a) * l.v.cross(*this -
            l.a) / l.v.norm();
    Pt p = l.a + l.v * l.v.dot(*this - l.a) / l.v.norm();
    if (eq(h2, 0)) return {p}; // line tangent to circle
    if (le(h2, 0)) return {}; // no intersection
    Pt q = l.v.unit() * sqrt(h2);
    return {p - q, p + q}; // two points of intersection (
         chord)
  }

  Cir(Pt a, Pt b, Pt c) {
    // find circle that passes through points a, b, c
    Pt mab = (a + b) / 2, mcb = (b + c) / 2;
    Seg ab(mab, mab + (b - a).perp());
    Seg cb(mcb, mcb + (b - c).perp());
    Pt o = ab.intersection(cb);
    *this = Cir(o, (o - a).length());
  }
};
```

6.2 Distance point circle

```
ld distance(Pt p, Cir c) { return max(0.L, (p - c).length()
     - c.r); }
```

6.3 Common area circle circle

```
ld commonArea(Cir a, Cir b) {
  if (le(a.r, b.r)) swap(a, b);
  ld d = (a - b).length();
  if (leq(d + b.r, a.r)) return b.r * b.r * PI;
  if (geq(d, a.r + b.r)) return 0.0;
  auto angle = [&](ld x, ld y, ld z) {
    return acos((x * x + y * y - z * z) / (2 * x * y));
  };
  auto cut = [&](ld x, ld r) { return (x - sin(x)) * r * r
       / 2; };
  ld a1 = angle(d, a.r, b.r), a2 = angle(d, b.r, a.r);
  return cut(a1 * 2, a.r) + cut(a2 * 2, b.r);
}
```

6.4 Minimum enclosing circle $\mathcal{O}(n)$ wow!!

```
Cir minEnclosing(vector<Pt>& pts) { // a bunch of points
  shuffle(all(pts), rng);
  Cir c(0, 0, 0);
  fore (i, 0, sz(pts))
    if (!c.contains(pts[i])) {
      c = Cir(pts[i], 0);
      fore (j, 0, i)
        if (!c.contains(pts[j])) {
          c = Cir((pts[i] + pts[j]) / 2, (pts[i] - pts[j]).
               length() / 2);
          fore (k, 0, j)
            if (!c.contains(pts[k])) c = Cir(pts[i], pts[j
                 ], pts[k]);
        }
    }
  return c;
}
```

7 Polygon

7.1 Area polygon

```
ld area(const vector<Pt>& pts) {
  ld sum = 0;
  fore (i, 0, sz(pts)) sum += pts[i].cross(pts[(i + 1) % sz
       (pts)]);
  return abs(sum / 2);
}
```

7.2 Perimeter

```
ld perimeter(const vector<Pt>& pts) {
  ld sum = 0;
  fore (i, 0, sz(pts)) sum += (pts[(i + 1) % sz(pts)] - pts
       [i]).length();
  return sum;
}
```

7.3 Cut polygon line

```
vector<Pt> cut(const vector<Pt>& pts, Line l) {
  vector<Pt> ans;
  int n = sz(pts);
  fore (i, 0, n) {
    int j = (i + 1) % n;
    if (geq(l.v.cross(pts[i] - l.a), 0)) // left
      ans.pb(pts[i]);
    Seg s(pts[i], pts[j]);
    if (l.intersects(s) == 1) {
      Pt p = l.intersection(s);
      if (p != pts[i] && p != pts[j]) ans.pb(p);
    }
  }
  return ans;
}
```

7.4 Common area circle polygon $\mathcal{O}(n)$

```
ld commonArea(Cir c, const vector<Pt>& poly) {
  auto arg = [&](Pt p, Pt q) { return atan2(p.cross(q), p.
       dot(q)); };
  auto tri = [&](Pt p, Pt q) {
    Pt d = q - p;
    ld a = d.dot(p) / d.norm(), b = (p.norm() - c.r * c.r)
         / d.norm();
    ld det = a * a - b;
    if (leq(det, 0)) return arg(p, q) * c.r * c.r;
    ld s = max(0.L, -a - sqrt(det)), t = min(1.L, -a + sqrt
         (det));
    if (t < 0 || 1 <= s) return arg(p, q) * c.r * c.r;
```

```
    Pt u = p + d * s, v = p + d * t;
    return u.cross(v) + (arg(p, u) + arg(v, q)) * c.r * c.r
        ;
  };
  ld sum = 0;
  fore (i, 0, sz(poly)) sum += tri(poly[i] - c, poly[(i + 1
      ) % sz(poly)] - c);
  return abs(sum / 2);
}
```

7.5 Point in polygon

```
int contains(const vector<Pt>& pts, Pt p) {
  int rays = 0, n = sz(pts);
  fore (i, 0, n) {
    Pt a = pts[i], b = pts[(i + 1) % n];
    if (ge(a.y, b.y)) swap(a, b);
    if (Seg(a, b).contains(p)) return ON;
    rays ^= (leq(a.y, p.y) && le(p.y, b.y) && p.dir(a, b) >
        0);
  }
  return rays & 1 ? IN : OUT;
}
```

7.6 Convex hull $\mathcal{O}(nlogn)$

```
vector<Pt> convexHull(vector<Pt> pts) {
  vector<Pt> hull;
  sort(all(pts),
       [&](Pt a, Pt b) { return a.x == b.x ? a.y < b.y : a.
           x < b.x; });
  pts.erase(unique(all(pts)), pts.end());
  fore (i, 0, sz(pts)) {
    while (sz(hull) >= 2 && hull.back().dir(pts[i], hull[sz
        (hull) - 2]) < 0)
      hull.pop_back();
    hull.pb(pts[i]);
  }
  hull.pop_back();
  int k = sz(hull);
  fore (i, sz(pts), 0) {
    while (sz(hull) >= k + 2 && hull.back().dir(pts[i],
        hull[sz(hull) - 2]) < 0)
      hull.pop_back();
    hull.pb(pts[i]);
  }
  hull.pop_back();
  return hull;
}
```

7.7 Is convex

```
bool isConvex(const vector<Pt>& pts) {
  int n = sz(pts);
  bool pos = 0, neg = 0;
  fore (i, 0, n) {
    Pt a = pts[(i + 1) % n] - pts[i];
    Pt b = pts[(i + 2) % n] - pts[(i + 1) % n];
    int dir = sgn(a.cross(b));
    if (dir > 0) pos = 1;
    if (dir < 0) neg = 1;
  }
  return !(pos && neg);
}
```

7.8 Point in convex polygon $\mathcal{O}(logn)$

```
bool contains(const vector<Pt>& a, Pt p) {
  int lo = 1, hi = sz(a) - 1;
  if (a[0].dir(a[lo], a[hi]) > 0) swap(lo, hi);
  if (p.dir(a[0], a[lo]) >= 0 || p.dir(a[0], a[hi]) <= 0)
    return false;
  while (abs(lo - hi) > 1) {
    int mid = (lo + hi) >> 1;
    (p.dir(a[0], a[mid]) > 0 ? hi : lo) = mid;
  }
  return p.dir(a[lo], a[hi]) < 0;
}
```

8 Graphs

8.1 Cutpoints and bridges

```
int tin[N], fup[N], timer = 0;

void weakness(int u, int p = -1) {
  tin[u] = fup[u] = ++timer;
  int children = 0;
  for (int v : graph[u])
    if (v != p) {
      if (!tin[v]) {
        ++children;
        weakness(v, u);
        fup[u] = min(fup[u], fup[v]);
        if (fup[v] >= tin[u] && !(p == -1 && children < 2))
             // u is a cutpoint
          if (fup[v] > tin[u]) // bridge u -> v
      }
      fup[u] = min(fup[u], tin[v]);
    }
}
```

8.2 Tarjan

```
int tin[N], fup[N];
bitset<N> still;
stack<int> stk;
int timer = 0;

void tarjan(int u) {
  tin[u] = fup[u] = ++timer;
  still[u] = true;
  stk.push(u);
  for (auto& v : graph[u]) {
    if (!tin[v]) tarjan(v);
    if (still[v]) fup[u] = min(fup[u], fup[v]);
  }
  if (fup[u] == tin[u]) {
    int v;
    do {
      v = stk.top();
      stk.pop();
      still[v] = false;
      // u and v are in the same scc
    } while (v != u);
  }
}
```

8.3 Two sat $\mathcal{O}(2 \cdot n)$

v: true, ~v: false

`implies(a, b)`: if a then b

a	b	a => b
F	F	T
T	T	T
F	T	T
T	F	F

`setVal(a)`: set a = true
`setVal(~a)`: set a = false

```
struct TwoSat {
  int n;
  vector<vector<int>> imp;

  TwoSat(int k) : n(k + 1), imp(2 * n) {} // 1-indexed

  void either(int a, int b) { // a || b
    a = max(2 * a, -1 - 2 * a);
```

```
    b = max(2 * b, -1 - 2 * b);
    imp[a ^ 1].pb(b);
    imp[b ^ 1].pb(a);
  }

  void implies(int a, int b) { either(~a, b); }

  void setVal(int a) { either(a, a); }

  optional<vector<int>> solve() {
    int k = sz(imp);
    vector<int> s, b, id(sz(imp));
    function<void(int)> dfs = [&](int u) {
      b.pb(id[u] = sz(s)), s.pb(u);
      for (int v : imp[u]) {
        if (!id[v])
          dfs(v);
        else
          while (id[v] < b.back()) b.pop_back();
      }
      if (id[u] == b.back())
        for (b.pop_back(), ++k; id[u] < sz(s); s.pop_back()
            ) id[s.back()] = k;
    };
    vector<int> val(n);
    fore (u, 0, sz(imp))
      if (!id[u]) dfs(u);
    fore (u, 0, n) {
      int x = 2 * u;
      if (id[x] == id[x ^ 1]) return nullopt;
      val[u] = id[x] < id[x ^ 1];
    }
    return optional(val);
  }
};
```

8.4 LCA

```
const int LogN = 1 + __lg(N);
int par[LogN][N], depth[N];

void dfs(int u, int par[]) {
  for (auto& v : graph[u])
    if (v != par[u]) {
      par[v] = u;
      depth[v] = depth[u] + 1;
      dfs(v, par);
    }
}

int lca(int u, int v) {
  if (depth[u] > depth[v]) swap(u, v);
  fore (k, LogN, 0)
    if (depth[v] - depth[u] >= (1 << k)) v = par[k][v];
  if (u == v) return u;
  fore (k, LogN, 0)
    if (par[k][v] != par[k][u]) u = par[k][u], v = par[k][v
        ];
  return par[0][u];
}

int dist(int u, int v) { return depth[u] + depth[v] - 2 *
    depth[lca(u, v)]; }

void init(int r) {
  dfs(r, par[0]);
  fore (k, 1, LogN)
    fore (u, 1, n + 1) par[k][u] = par[k - 1][par[k - 1][u
        ]];
}
```

8.5 Virtual tree $\mathcal{O}(n \cdot logn)$ "lca tree"

```
vector<int> virt[N];

int virtualTree(vector<int>& ver) {
  auto byDfs = [&](int u, int v) { return tin[u] < tin[v];
      };
  sort(all(ver), byDfs);
  fore (i, sz(ver), 1) ver.pb(lca(ver[i - 1], ver[i]));
  sort(all(ver), byDfs);
  ver.erase(unique(all(ver)), ver.end());
  for (int u : ver) virt[u].clear();
  fore (i, 1, sz(ver)) virt[lca(ver[i - 1], ver[i])].pb(ver
      [i]);
  return ver[0];
}
```

8.6 Dynamic connectivity

```
struct DynamicConnectivity {
  struct Query {
    int op, u, v, at;
  };

  Dsu dsu; // with rollback
  vector<Query> queries;
  map<ii, int> mp;
  int timer = -1;

  DynamicConnectivity(int n = 0) : dsu(n) {}

  void add(int u, int v) {
    mp[minmax(u, v)] = ++timer;
    queries.pb({'+', u, v, INT_MAX});
  }

  void rem(int u, int v) {
    int in = mp[minmax(u, v)];
    queries.pb({'-', u, v, in});
    queries[in].at = ++timer;
    mp.erase(minmax(u, v));
  }

  void query() { queries.push_back({'?', -1, -1, ++timer});
       }

  void solve(int l, int r) {
    if (l == r) {
      if (queries[l].op == '?') // solve the query here
        return;
    }
    int m = (l + r) >> 1;
    int before = sz(dsu.mem);
    for (int i = m + 1; i <= r; i++) {
      Query& q = queries[i];
      if (q.op == '-' && q.at < l) dsu.unite(q.u, q.v);
    }
    solve(l, m);
    while (sz(dsu.mem) > before) dsu.rollback();
    for (int i = l; i <= m; i++) {
      Query& q = queries[i];
      if (q.op == '+' && q.at > r) dsu.unite(q.u, q.v);
    }
    solve(m + 1, r);
    while (sz(dsu.mem) > before) dsu.rollback();
  }
};
```

8.7 Euler-tour + HLD + LCA $\mathcal{O}(n \cdot logn)$

Solves subtrees and paths problems

```
int par[N], nxt[N], depth[N], sz[N];
int tin[N], tout[N], who[N], timer = 0;
```

```
int dfs(int u) {
  sz[u] = 1;
  for (auto& v : graph[u])
    if (v != par[u]) {
      par[v] = u;
      depth[v] = depth[u] + 1;
      sz[u] += dfs(v);
      if (graph[u][0] == par[u] || sz[v] > sz[graph[u][0]])
        swap(v, graph[u][0]);
    }
  return sz[u];
}

void hld(int u) {
  tin[u] = ++timer, who[timer] = u;
  for (auto& v : graph[u])
    if (v != par[u]) {
      nxt[v] = (v == graph[u][0] ? nxt[u] : v);
      hld(v);
    }
  tout[u] = timer;
}

template <bool OverEdges = 0, class F>
void processPath(int u, int v, F f) {
  for (; nxt[u] != nxt[v]; u = par[nxt[u]]) {
    if (depth[nxt[u]] < depth[nxt[v]]) swap(u, v);
    f(tin[nxt[u]], tin[u]);
  }
  if (depth[u] < depth[v]) swap(u, v);
  f(tin[v] + OverEdges, tin[u]);
}

int lca(int u, int v) {
  int last = -1;
  processPath(u, v, [&](int l, int r) { last = who[l]; });
  return last;
}

void updatePath(int u, int v, lli z) {
  processPath(u, v, [&](int l, int r) { tree->update(l, r,
       z); });
}

void updateSubtree(int u, lli z) { tree->update(tin[u],
    tout[u], z); }

lli queryPath(int u, int v) {
  lli sum = 0;
  processPath(u, v, [&](int l, int r) { sum += tree->query(
       l, r); });
  return sum;
}

lli queryPathWithOrder(int u, int v, int x) {
  int _lca = lca(u, v);
  assert(_lca != -1);

  vector<pair<int, int>> firstHalf, secondHalf, ranges;
  processPath(
      u, _lca, [&](int l, int r) { firstHalf.push_back(
          make_pair(r, l)); });

  processPath(_lca, v, [&](int l, int r) {
    l += tin[_lca] == l;
    if (l <= r) { secondHalf.push_back(make_pair(l, r)); }
  });
  reverse(all(secondHalf));

  ranges = firstHalf;
  ranges.insert(end(ranges), begin(secondHalf), end(
       secondHalf));

  int who = -1;
  for (auto [begin, end] : ranges) {
    // if begin <= end: left to right, aka. normal
    // if begin > end: right to left,
    // e.g. begin = 3, end = 1
    // order must go 3, 2, 1

    // e.g. first node in the path(u, v) with value less
         than or equal to x
    if ((who = tree->solve(begin, end, x)) != -1) { break;
         }
  }

  return who;
}

lli querySubtree(int u) { return tree->query(tin[u], tout[u
    ]); }
```

8.8 Centroid $\mathcal{O}(n \cdot logn)$

Solves "all pairs of nodes" problems

```
int cdp[N], sz[N];
bitset<N> rem;

int dfsz(int u, int p = -1) {
  sz[u] = 1;
  for (int v : graph[u])
    if (v != p && !rem[v]) sz[u] += dfsz(v, u);
  return sz[u];
}

int centroid(int u, int size, int p = -1) {
  for (int v : graph[u])
    if (v != p && !rem[v] && 2 * sz[v] > size) return
         centroid(v, size, u);
  return u;
}

void solve(int u, int p = -1) {
  cdp[u = centroid(u, dfsz(u))] = p;
  rem[u] = true;
  for (int v : graph[u])
    if (!rem[v]) solve(v, u);
}
```

8.9 Guni $\mathcal{O}(n \cdot logn)$

Solve subtrees problems

```
int cnt[C], color[N];
int sz[N];

int guni(int u, int p = -1) {
  sz[u] = 1;
  for (auto& v : graph[u])
    if (v != p) {
      sz[u] += guni(v, u);
      if (sz[v] > sz[graph[u][0]] || p == graph[u][0]) swap
           (v, graph[u][0]);
    }
  return sz[u];
}

void update(int u, int p, int add, bool skip) {
```

```
    cnt[color[u]] += add;
    fore (i, skip, sz(graph[u]))
      if (graph[u][i] != p) update(graph[u][i], u, add, 0);
  }

  void solve(int u, int p = -1, bool keep = 0) {
    fore (i, sz(graph[u]), 0)
      if (graph[u][i] != p) solve(graph[u][i], u, !i);
    update(u, p, +1, 1); // add
    // now cnt[i] has how many times the color i appears in
        the subtree of u
    if (!keep) update(u, p, -1, 0); // remove
  }
```

8.10 Link-Cut tree $\mathcal{O}(n \cdot logn)$

Solves dynamic trees problems, can handle subtrees and paths maybe with a high constant

```
struct LinkCut {
  struct Node {
    Node *left{0}, *right{0}, *par{0};
    bool rev = 0;
    int sz = 1;
    int sub = 0, vsub = 0; // subtree
    lli path = 0; // path
    lli self = 0; // node info

    void push() {
      if (rev) {
        swap(left, right);
        if (left) left->rev ^= 1;
        if (right) right->rev ^= 1;
        rev = 0;
      }
    }

    void pull() {
      sz = 1;
      sub = vsub + self;
      path = self;
      if (left) {
        sz += left->sz;
        sub += left->sub;
        path += left->path;
      }
      if (right) {
        sz += right->sz;
        sub += right->sub;
        path += right->path;
      }
    }

    void addVsub(Node* v, lli add) {
      if (v) vsub += 1LL * add * v->sub;
    }
  };

  vector<Node> a;

  LinkCut(int n = 1) : a(n) {}

  void splay(Node* u) {
    auto assign = [&](Node* u, Node* v, int d) {
      if (v) v->par = u;
      if (d >= 0) (d == 0 ? u->left : u->right) = v;
    };
    auto dir = [&](Node* u) {
      if (!u->par) return -1;
      return u->par->left == u ? 0 : (u->par->right == u ?
          1 : -1);
    };
    auto rotate = [&](Node* u) {
      Node *p = u->par, *g = p->par;
      int d = dir(u);
      assign(p, d ? u->left : u->right, d);
      assign(g, u, dir(p));
      assign(u, p, !d);
      p->pull(), u->pull();
    };
    while (~dir(u)) {
      Node *p = u->par, *g = p->par;
      if (~dir(p)) g->push();
      p->push(), u->push();
      if (~dir(p)) rotate(dir(p) == dir(u) ? p : u);
      rotate(u);
    }
    u->push(), u->pull();
  }

  void access(int u) {
    Node* last = NULL;
    for (Node* x = &a[u]; x; last = x, x = x->par) {
      splay(x);
      x->addVsub(x->right, +1);
      x->right = last;
      x->addVsub(x->right, -1);
      x->pull();
    }
    splay(&a[u]);
  }

  void reroot(int u) {
    access(u);
    a[u].rev ^= 1;
  }

  void link(int u, int v) {
    reroot(v), access(u);
    a[u].addVsub(v, +1);
    a[v].par = &a[u];
    a[u].pull();
  }

  void cut(int u, int v) {
    reroot(v), access(u);
    a[u].left = a[v].par = NULL;
    a[u].pull();
  }

  int lca(int u, int v) {
    if (u == v) return u;
    access(u), access(v);
    if (!a[u].par) return -1;
    return splay(&a[u]), a[u].par ? -1 : u;
  }

  int depth(int u) {
    access(u);
    return a[u].left ? a[u].left->sz : 0;
  }

  // get k-th parent on path to root
  int ancestor(int u, int k) {
    k = depth(u) - k;
    assert(k >= 0);
    for (;; a[u].push()) {
      int sz = a[u].left->sz;
      if (sz == k) return access(u), u;
```

```
      if (sz < k)
        k -= sz + 1, u = u->ch[1];
      else
        u = u->ch[0];
    }
    assert(0);
  }

  lli queryPath(int u, int v) {
    reroot(u), access(v);
    return a[v].path;
  }

  lli querySubtree(int u, int x) {
    // query subtree of u, x is outside
    reroot(x), access(u);
    return a[u].vsub + a[u].self;
  }

  void update(int u, lli val) {
    access(u);
    a[u].self = val;
    a[u].pull();
  }

  Node& operator[](int u) { return a[u]; }
};
```

9 Flows

9.1 Hopcroft Karp $\mathcal{O}(e\sqrt{v})$

```
struct HopcroftKarp {
  int n, m;
  vector<vector<int>> graph;
  vector<int> dist, match;

  HopcroftKarp(int k)
      : n(k + 1), graph(n), dist(n), match(n, 0) {} // 1-
         indexed!!

  void add(int u, int v) { graph[u].pb(v), graph[v].pb(u);
     }

  bool bfs() {
    queue<int> qu;
    fill(all(dist), -1);
    fore (u, 1, n)
      if (!match[u]) dist[u] = 0, qu.push(u);
    while (!qu.empty()) {
      int u = qu.front();
      qu.pop();
      for (int v : graph[u])
        if (dist[match[v]] == -1) {
          dist[match[v]] = dist[u] + 1;
          if (match[v]) qu.push(match[v]);
        }
    }
    return dist[0] != -1;
  }

  bool dfs(int u) {
    for (int v : graph[u])
      if (!match[v] || (dist[u] + 1 == dist[match[v]] &&
           dfs(match[v]))) {
        match[u] = v, match[v] = u;
        return 1;
      }
    dist[u] = 1 << 30;
    return 0;
  }

  int maxMatching() {
    int tot = 0;
    while (bfs())
      fore (u, 1, n) tot += match[u] ? 0 : dfs(u);
    return tot;
  }
};
```

9.2 Hungarian $\mathcal{O}(n^2 \cdot m)$

n jobs, m people for max assignment

```
template <class C>
pair<C, vector<int>> Hungarian(vector<vector<C>>& a) { //
     max assignment
  int n = sz(a), m = sz(a[0]), p, q, j, k; // n <= m
  vector<C> fx(n, numeric_limits<C>::min()), fy(m, 0);
  vector<int> x(n, -1), y(m, -1);
  fore (i, 0, n)
    fore (j, 0, m) fx[i] = max(fx[i], a[i][j]);
  fore (i, 0, n) {
    vector<int> t(m, -1), s(n + 1, i);
    for (p = q = 0; p <= q && x[i] < 0; p++)
      for (k = s[p], j = 0; j < m && x[i] < 0; j++)
        if (abs(fx[k] + fy[j] - a[k][j]) < EPS && t[j] < 0)
             {
          s[++q] = y[j], t[j] = k;
          if (s[q] < 0)
            for (p = j; p >= 0; j = p) y[j] = k = t[j], p =
                 x[k], x[k] = j;
        }
    if (x[i] < 0) {
      C d = numeric_limits<C>::max();
      fore (k, 0, q + 1)
        fore (j, 0, m)
          if (t[j] < 0) d = min(d, fx[s[k]] + fy[j] - a[s[k
               ]][j]);
      fore (j, 0, m) fy[j] += (t[j] < 0 ? 0 : d);
      fore (k, 0, q + 1) fx[s[k]] -= d;
      i--;
    }
  }
  C cost = 0;
  fore (i, 0, n) cost += a[i][x[i]];
  return make_pair(cost, x);
}
```

9.3 Dinic $\mathcal{O}(\min(e \cdot flow, v^2 \cdot e))$

```
template <class F>
struct Dinic {
  struct Edge {
    int v, inv;
    F cap, flow;
    Edge(int v, F cap, int inv) : v(v), cap(cap), flow(0),
         inv(inv) {}
  };

  F EPS = (F)1e-9;
  int s, t, n;
  vector<vector<Edge>> graph;
  vector<int> dist, ptr;

  Dinic(int n) : n(n), graph(n), dist(n), ptr(n), s(n - 2),
       t(n - 1) {}

  void add(int u, int v, F cap) {
    graph[u].pb(Edge(v, cap, sz(graph[v])));
    graph[v].pb(Edge(u, 0, sz(graph[u]) - 1));
```

```
  }

  bool bfs() {
    fill(all(dist), -1);
    queue<int> qu({s});
    dist[s] = 0;
    while (sz(qu) && dist[t] == -1) {
      int u = qu.front();
      qu.pop();
      for (Edge& e : graph[u])
        if (dist[e.v] == -1)
          if (e.cap - e.flow > EPS) {
            dist[e.v] = dist[u] + 1;
            qu.push(e.v);
          }
    }
    return dist[t] != -1;
  }

  F dfs(int u, F flow = numeric_limits<F>::max()) {
    if (flow <= EPS || u == t) return max<F>(0, flow);
    for (int& i = ptr[u]; i < sz(graph[u]); i++) {
      Edge& e = graph[u][i];
      if (e.cap - e.flow > EPS && dist[u] + 1 == dist[e.v])
          {
        F pushed = dfs(e.v, min<F>(flow, e.cap - e.flow));
        if (pushed > EPS) {
          e.flow += pushed;
          graph[e.v][e.inv].flow -= pushed;
          return pushed;
        }
      }
    }
    return 0;
  }

  F maxFlow() {
    F flow = 0;
    while (bfs()) {
      fill(all(ptr), 0);
      while (F pushed = dfs(s)) flow += pushed;
    }
    return flow;
  }

  bool leftSide(int u) {
    // left side comes from sink
    return dist[u] != -1;
  }
};
```

9.4 Min-Cost flow $\mathcal{O}(\min(e \cdot flow, v^2 \cdot e))$

```
template <class C, class F>
struct Mcmf {
  struct Edge {
    int u, v, inv;
    F cap, flow;
    C cost;
    Edge(int u, int v, C cost, F cap, int inv)
        : u(u), v(v), cost(cost), cap(cap), flow(0), inv(
            inv) {}
  };

  F EPS = (F)1e-9;
  int s, t, n;
  vector<vector<Edge>> graph;
  vector<Edge*> prev;
  vector<C> cost;
  vector<int> state;

  Mcmf(int n)
      : n(n), graph(n), cost(n), state(n), prev(n), s(n - 2
          ), t(n - 1) {}

  void add(int u, int v, C cost, F cap) {
    graph[u].pb(Edge(u, v, cost, cap, sz(graph[v])));
    graph[v].pb(Edge(v, u, -cost, 0, sz(graph[u]) - 1));
  }

  bool bfs() {
    fill(all(state), 0);
    fill(all(cost), numeric_limits<C>::max());
    deque<int> qu;
    qu.push_back(s);
    state[s] = 1, cost[s] = 0;
    while (sz(qu)) {
      int u = qu.front();
      qu.pop_front();
      state[u] = 2;
      for (Edge& e : graph[u])
        if (e.cap - e.flow > EPS)
          if (cost[u] + e.cost < cost[e.v]) {
            cost[e.v] = cost[u] + e.cost;
            prev[e.v] = &e;
            if (state[e.v] == 2 || (sz(qu) && cost[qu.front
                ()] > cost[e.v]))
              qu.push_front(e.v);
            else if (state[e.v] == 0)
              qu.push_back(e.v);
            state[e.v] = 1;
          }
    }
    return cost[t] != numeric_limits<C>::max();
  }

  pair<C, F> minCostFlow() {
    C cost = 0;
    F flow = 0;
    while (bfs()) {
      F pushed = numeric_limits<F>::max();
      for (Edge* e = prev[t]; e != nullptr; e = prev[e->u])
        pushed = min(pushed, e->cap - e->flow);
      for (Edge* e = prev[t]; e != nullptr; e = prev[e->u])
          {
        e->flow += pushed;
        graph[e->v][e->inv].flow -= pushed;
        cost += e->cost * pushed;
      }
      flow += pushed;
    }
    return make_pair(cost, flow);
  }
};
```

10 Game theory

10.1 Grundy numbers

If the moves are consecutive $S = \{1, 2, 3, ..., x\}$ the game can be solved like $stackSize \pmod{x + 1} \neq 0$

```
int mem[N];

int mex(set<int>& st) {
  int x = 0;
  while (st.count(x)) x++;
  return x;
}

int grundy(int n) {
  if (n < 0) return INF;
```

```
  if (n == 0) return 0;
  int& g = mem[n];
  if (g == -1) {
    set<int> st;
    for (int x : {a, b}) st.insert(grundy(n - x));
    g = mex(st);
  }
  return g;
}
```

11 Math

11.1 Bits

Bits++	
Operations on *int*	Function
`x & -x`	Least significant bit in x
`__lg(x)`	Most significant bit in x
`c = x&-x, r = x+c; (((r^x) » 2)/c) | r`	Next number after x with same number of bits set
__builtin_	Function
popcount(x)	Amount of 1's in x
clz(x)	0's to the **left** of biggest bit
ctz(x)	0's to the **right** of smallest bit

11.2 Bitset

Bitset<Size>	
Operation	Function
_Find_first()	Least significant bit
_Find_next(idx)	First set bit after index *idx*
any(), none(), all()	Just what the expression says
set(), reset(), flip()	Just what the expression says x2
to_string('.', 'A')	Print 011010 like .AA.A.

11.3 Probability

Conditional

The event A happens and the event B has already happened

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

If **independent** events

$$P(A|B) = P(A), P(B|A) = P(B)$$

Bayes theorem

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Binomial

$$B = \binom{n}{x} \cdot p^x \cdot (1-p)^{n-x}$$

n = number of trials
x = number of **success** from n trials
p = probability of **success** on a single trial

Geometric

Probability of success at the nth-event after failing the others

$$G = (1-p)^{n-1} \cdot p$$

n = number of trials
p = probability of *success* on a single trial

Poisson

$$Po = \frac{\lambda^k \cdot e^{-\lambda}}{k!}$$

λ = number of times an event is expected (occurs / time)
k = number of occurring events in the limited period of time

Example: The event happens 4 times per minute and we want k events to happen in 10 minutes, then $\lambda = 4 \cdot 10 = 40$

Expected value

$$E_x = \sum_{\forall x} x \cdot p(x)$$

11.4 Gauss jordan $\mathcal{O}(n^2 \cdot m)$

```
template <class T>
pair<int, vector<T>> gauss(vector<vector<T>> a, vector<T> b
    ) {
  const double EPS = 1e-6;
  int n = a.size(), m = a[0].size();
  for (int i = 0; i < n; i++) a[i].push_back(b[i]);
  vector<int> where(m, -1);
  for (int col = 0, row = 0; col < m and row < n; col++) {
    int sel = row;
    for (int i = row; i < n; ++i)
      if (abs(a[i][col]) > abs(a[sel][col])) sel = i;
    if (abs(a[sel][col]) < EPS) continue;
    for (int i = col; i <= m; i++) swap(a[sel][i], a[row][i
        ]);
    where[col] = row;

    for (int i = 0; i < n; i++)
      if (i != row) {
        T c = a[i][col] / a[row][col];
        for (int j = col; j <= m; j++) a[i][j] -= a[row][j]
             * c;
      }
    row++;
  }
  vector<T> ans(m, 0);
  for (int i = 0; i < m; i++)
    if (where[i] != -1) ans[i] = a[where[i]][m] / a[where[i
        ]][i];
  for (int i = 0; i < n; i++) {
    T sum = 0;
    for (int j = 0; j < m; j++) sum += ans[j] * a[i][j];
    if (abs(sum - a[i][m]) > EPS) return pair(0, vector<T
        >());
  }
  for (int i = 0; i < m; i++)
    if (where[i] == -1) return pair(INF, ans);
  return pair(1, ans);
}
```

11.5 Xor basis

```
template <int D>
struct XorBasis {
  using Num = bitset<D>;
  array<Num, D> basis, keep;
  vector<int> from;
  int n = 0, id = -1;

  XorBasis() : from(D, -1) { basis.fill(0); }

  bool insert(Num x) {
    ++id;
    Num k;
    fore (i, D, 0)
      if (x[i]) {
```

```
      if (!basis[i].any()) {
        k[i] = 1, from[i] = id, keep[i] = k;
        basis[i] = x, n++;
        return 1;
      }
      x ^= basis[i], k ^= keep[i];
    }
  return 0;
}

optional<Num> find(Num x) {
  // is x in xor-basis set?
  // v ^ (v ^ x) = x
  Num v;
  fore (i, D, 0)
    if (x[i]) {
      if (!basis[i].any()) return nullopt;
      x ^= basis[i];
      v[i] = 1;
    }
  return optional(v);
}

optional<vector<int>> recover(Num x) {
  auto v = find(x);
  if (!v) return nullopt;
  Num tmp;
  fore (i, D, 0)
    if (v.value()[i]) tmp ^= keep[i];
  vector<int> ans;
  for (int i = tmp._Find_first(); i < D; i = tmp.
      _Find_next(i))
    ans.pb(from[i]);
  return ans;
}

optional<Num> operator[](lli k) {
  lli tot = (1LL << n);
  if (k > tot) return nullopt;
  Num v = 0;
  fore (i, D, 0)
    if (basis[i]) {
      lli low = tot / 2;
      if ((low < k && v[i] == 0) || (low >= k && v[i])) v
          ^= basis[i];
      if (low < k) k -= low;
      tot /= 2;
    }
  return optional(v);
}
};
```

12 Combinatorics

12.1 Factorial

```
fac[0] = 1LL;
fore (i, 1, N) fac[i] = lli(i) * fac[i - 1] % MOD;
ifac[N - 1] = fpow(fac[N - 1], MOD - 2, MOD);
for (int i = N - 2; i >= 0; i--) ifac[i] = lli(i + 1) *
    ifac[i + 1] % MOD;
```

12.2 Factorial mod small prime

```
lli facMod(lli n, int p) {
  lli r = 1LL;
  for (; n > 1; n /= p) {
    r = (r * ((n / p) % 2 ? p - 1 : 1)) % p;
    fore (i, 2, n % p + 1) r = r * i % p;
  }
  return r % p;
}
```

12.3 Choose

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

$$\binom{n}{k_1, k_2, ..., k_m} = \frac{n!}{k_1! * k_2! * ... * k_m!}$$

```
lli choose(int n, int k) {
  if (n < 0 || k < 0 || n < k) return 0LL;
  return fac[n] * ifac[k] % MOD * ifac[n - k] % MOD;
}

lli choose(int n, int k) {
  lli r = 1;
  int to = min(k, n - k);
  if (to < 0) return 0;
  fore (i, 0, to) r = r * (n - i) / (i + 1);
  return r;
}
```

12.4 Pascal

```
fore (i, 0, N) {
  choose[i][0] = choose[i][i] = 1;
  for (int j = 1; j <= i; j++)
    choose[i][j] = choose[i - 1][j - 1] + choose[i - 1][j];
}
```

12.5 Stars and bars

Enclosing n objects in k boxes

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

12.6 Lucas

Changes $\binom{n}{k}$ mod p, with $n \geq 2e6, k \geq 2e6$ and $p \leq 1e7$

$$\binom{n}{k} \equiv \prod_{i=0}^{n} \binom{n_i}{k_i} \bmod p$$

```
lli lucas(lli n, lli k) {
  if (k == 0) return 1LL;
  return lucas(n / MOD, k / MOD) * choose(n % MOD, k % MOD)
      % MOD;
}
```

12.7 Burnside lemma

Burnside's lemma is a result in group theory that can help when counting objects with symmetry taken into account. It gives a formula to count objects, where two objects that are related by a symmetry (rotation or reflection, for example) are not to be counted as distinct.

let G be a finite group. For each g in G let $f(g)$ denote the set of elements that are fixed by g.

$$|classes| = \frac{1}{|G|} \cdot \sum_{g \in G} f(g)$$

12.8 Catalan

Number of ways to insert n pairs of parentheses in a word of $n + 1$ letters.

Consider all the $\binom{2n}{n}$ paths on squared paper that start at (0, 0), end at (n, n) and at each step, either make a (+1,+1) step or a (+1,-1) step. Then the number of such paths that never go below the x-axis.

Number of ordered rooted trees with n nodes, not including the root.

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

$C_n = \binom{2n}{n} - \binom{2n}{n+1}$

i	0/1	2	3	4	5	6	7	8	9	10	11
C_i	1	2	5	14	42	132	429	1430	4862	16796	

```
catalan[0] = 1LL;
fore (i, 0, N) {
  catalan[i + 1] =
      catalan[i] * lli(4 * i + 2) % MOD * fpow(i + 2, MOD -
          2) % MOD;
}
```

12.9 Bell numbers

The number of ways a set of n elements can be partitioned into **nonempty** subsets

$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} \cdot B_k$

i	5	6	7	8	9	10	11
B_i	52	203	877	4140	21147	115975	678570

12.10 Stirling numbers

Count the number of permutations of n elements with k disjoint cycles Signed way, $k > 0$

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

The unsigned way doesn't have sign $|-(n-1)|$

The sum of products of the $\binom{n}{k}$ subsets of size k of $\{0, 1, ...n-1\}$ is $s(n, n-k)$

12.11 Stirling numbers 2

How many ways are of dividing a set of n **different** objects into k **nonempty** subsets. $\left\{{n \atop k}\right\}$

$s2(0,0) = 1$, $s2(n,0) = s2(0,n) = 0$
$s2(n,k) = s2(n-1,k-1) + k \cdot s2(n-1,k)$

$s2(n,k) = \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \cdot \binom{k}{i} \cdot (k-i)^n$

```
Mint stirling2(int n, int k) {
  Mint sum = 0;
  fore (i, 0, k + 1)
    sum += fpow<Mint>(-1, i) * choose(k, i) * fpow<Mint>(k
        - i, n);
  return sum * ifac(k);
};
```

13 Number theory

13.1 Amount of divisors $\mathcal{O}(n^{1/3})$

```
ull amountOfDivisors(ull n) {
  ull cnt = 1;
  for (auto p : primes) {
    if (1LL * p * p * p > n) break;
    if (n % p == 0) {
      ull k = 0;
      while (n > 1 && n % p == 0) n /= p, ++k;
      cnt *= (k + 1);
    }
  }
  ull sq = mysqrt(n); // the last x * x <= n
  if (miller(n))
    cnt *= 2;
  else if (sq * sq == n && miller(sq))
    cnt *= 3;
  else if (n > 1)
    cnt *= 4;
  return cnt;
}
```

13.2 Chinese remainder theorem

- $x \equiv 3 \pmod 4$
- $x \equiv 5 \pmod 6$
- $x \equiv 2 \pmod 5$

$x \equiv 47 \pmod{60}$

```
pair<lli, lli> crt(pair<lli, lli> a, pair<lli, lli> b) {
  if (a.s < b.s) swap(a, b);
  auto p = euclid(a.s, b.s);
  lli g = a.s * p.f + b.s * p.s, l = a.s / g * b.s;
  if ((b.f - a.f) % g != 0) return {-1, -1}; // no solution
  p.f = a.f + (b.f - a.f) % b.s * p.f % b.s / g * a.s;
  return {p.f + (p.f < 0) * l, l};
}
```

13.3 Euclid $\mathcal{O}(log(a \cdot b))$

```
pair<lli, lli> euclid(lli a, lli b) {
  if (b == 0) return {1, 0};
  auto p = euclid(b, a % b);
  return {p.s, p.f - a / b * p.s};
}
```

13.4 Inverse

```
lli inv(lli a, lli m) {
  a %= m;
  assert(a);
  return a == 1 ? 1 : m - 1LL * inv(m, a) * m / a;
}
```

13.5 Phi $\mathcal{O}(\sqrt{n})$

```
lli phi(lli n) {
  if (n == 1) return 0;
  lli r = n;
  for (lli i = 2; i * i <= n; i++)
    if (n % i == 0) {
      while (n % i == 0) n /= i;
      r -= r / i;
    }
  if (n > 1) r -= r / n;
  return r;
}
```

13.6 Miller rabin $\mathcal{O}(Witnesses \cdot (logn)^3)$

```
ull mul(ull x, ull y, ull MOD) {
  lli ans = x * y - MOD * ull(1.L / MOD * x * y);
  return ans + MOD * (ans < 0) - MOD * (ans >= lli(MOD));
}

// use mul(x, y, mod) inside fpow
bool miller(ull n) {
```

```
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  ull k = __builtin_ctzll(n - 1), d = n >> k;
  for (ull p : {2, 325, 9375, 28178, 450775, 9780504, 17952
      65022}) {
    ull x = fpow(p % n, d, n), i = k;
    while (x != 1 && x != n - 1 && p % n && i--) x = mul(x,
         x, n);
    if (x != n - 1 && i != k) return 0;
  }
  return 1;
}
```

13.7 Pollard Rho $\mathcal{O}(n^{1/4})$

```
ull rho(ull n) {
  auto f = [n](ull x) { return mul(x, x, n) + 1; };
  ull x = 0, y = 0, t = 30, 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);
}

// if used multiple times, try memorization!!
// try factoring small numbers with sieve
void pollard(ull n, map<ull, int>& fac) {
  if (n == 1) return;
  if (miller(n)) {
    fac[n]++;
  } else {
    ull x = rho(n);
    pollard(x, fac);
    pollard(n / x, fac);
  }
}
```

14 Polynomials

14.1 Berlekamp Massey

For a linear recurrence of length n you need to feed at least $2n$ terms into Berlekamp-Massey to guarantee getting the same or equivalent recurrence.

```
template <class T>
struct BerlekampMassey {
  int n;
  vector<T> s, t, pw[20];

  vector<T> combine(vector<T> a, vector<T> b) {
    vector<T> ans(sz(t) * 2 + 1);
    for (int i = 0; i <= sz(t); i++)
      for (int j = 0; j <= sz(t); j++) ans[i + j] += a[i] *
          b[j];
    for (int i = 2 * sz(t); i > sz(t); --i)
      for (int j = 0; j < sz(t); j++) ans[i - 1 - j] += ans
          [i] * t[j];
    ans.resize(sz(t) + 1);
    return ans;
  }

  BerlekampMassey(const vector<T>& s) : n(sz(s)), t(n), s(s
      ) {
    vector<T> x(n), tmp;
    t[0] = x[0] = 1;
    T b = 1;
    int len = 0, m = 0;
    fore (i, 0, n) {
      ++m;
      T d = s[i];
      for (int j = 1; j <= len; j++) d += t[j] * s[i - j];
      if (d == 0) continue;
      tmp = t;
      T coef = d / b;
      for (int j = m; j < n; j++) t[j] -= coef * x[j - m];
      if (2 * len > i) continue;
      len = i + 1 - len;
      x = tmp;
      b = d;
      m = 0;
    }
    t.resize(len + 1);
    t.erase(t.begin());
    for (auto& x : t) x = -x;
    pw[0] = vector<T>(sz(t) + 1), pw[0][1] = 1;
    fore (i, 1, 20) pw[i] = combine(pw[i - 1], pw[i - 1]);
  }

  T operator[](lli k) {
    vector<T> ans(sz(t) + 1);
    ans[0] = 1;
    fore (i, 0, 20)
      if (k & (1LL << i)) ans = combine(ans, pw[i]);
    T val = 0;
    fore (i, 0, sz(t)) val += ans[i + 1] * s[i];
    return val;
  }
};
```

14.2 Lagrange $\mathcal{O}(n)$

Calculate the extrapolation of $f(k)$, given all the sequence $f(0), f(1), f(2), ..., f(n)$
$\sum_{i=1}^{10} i^5 = 220825$

```
template <class T>
struct Lagrange {
  int n;
  vector<T> y, suf, fac;

  Lagrange(vector<T>& y) : n(sz(y)), y(y), suf(n + 1, 1),
      fac(n, 1) {
    fore (i, 1, n) fac[i] = fac[i - 1] * i;
  }

  T operator[](lli k) {
    for (int i = n - 1; i >= 0; i--) suf[i] = suf[i + 1] *
        (k - i);

    T pref = 1, val = 0;
    fore (i, 0, n) {
      T num = pref * suf[i + 1];
      T den = fac[i] * fac[n - 1 - i];
      if ((n - 1 - i) % 2) den *= -1;
      val += y[i] * num / den;
      pref *= (k - i);
    }
    return val;
  }
};
```

14.3 FFT

```
template <class Complex>
void FFT(vector<Complex>& a, bool inv = false) {
  const static double PI = acos(-1.0);
  static vector<Complex> root = {0, 1};
  int n = sz(a);
  for (int i = 1, j = 0; i < n - 1; i++) {
    for (int k = n >> 1; (j ^= k) < k; k >>= 1);
```

```
    if (i < j) swap(a[i], a[j]);
  }
  int k = sz(root);
  if (k < n)
    for (root.resize(n); k < n; k <<= 1) {
      Complex z(cos(PI / k), sin(PI / k));
      fore (i, k >> 1, k) {
        root[i << 1] = root[i];
        root[i << 1 | 1] = root[i] * z;
      }
    }
  for (int k = 1; k < n; k <<= 1)
    for (int i = 0; i < n; i += k << 1)
      fore (j, 0, k) {
        Complex t = a[i + j + k] * root[j + k];
        a[i + j + k] = a[i + j] - t;
        a[i + j] = a[i + j] + t;
      }
  if (inv) {
    reverse(1 + all(a));
    for (auto& x : a) x /= n;
  }
}

template <class T>
vector<T> convolution(const vector<T>& a, const vector<T>&
    b) {
  if (a.empty() || b.empty()) return {};

  int n = sz(a) + sz(b) - 1, m = n;
  while (n != (n & -n)) ++n;

  vector<complex<double>> fa(all(a)), fb(all(b));
  fa.resize(n), fb.resize(n);
  FFT(fa, false), FFT(fb, false);
  fore (i, 0, n) fa[i] *= fb[i];
  FFT(fa, true);

  vector<T> ans(m);
  fore (i, 0, m) ans[i] = round(real(fa[i]));
  return ans;
}

template <class T>
vector<T> convolutionTrick(const vector<T>& a,
                           const vector<T>& b) { // 2 FFT's
                               instead of 3!!
  if (a.empty() || b.empty()) return {};

  int n = sz(a) + sz(b) - 1, m = n;
  while (n != (n & -n)) ++n;

  vector<complex<double>> in(n), out(n);
  fore (i, 0, sz(a)) in[i].real(a[i]);
  fore (i, 0, sz(b)) in[i].imag(b[i]);

  FFT(in, false);
  for (auto& x : in) x *= x;
  fore (i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
  FFT(out, false);

  vector<T> ans(m);
  fore (i, 0, m) ans[i] = round(imag(out[i]) / (4 * n));
  return ans;
}
```

14.4 Primitive root

```
int primitive(int p) {
  auto fpow = [&](lli x, int n) {
    lli r = 1;
    for (; n > 0; n >>= 1) {
      if (n & 1) r = r * x % p;
      x = x * x % p;
    }
    return r;
  };

  for (int g = 2; g < p; g++) {
    bool can = true;
    for (int i = 2; i * i < p; i++)
      if ((p - 1) % i == 0) {
        if (fpow(g, i) == 1) can = false;
        if (fpow(g, (p - 1) / i) == 1) can = false;
      }
    if (can) return g;
  }
  return -1;
}
```

14.5 NTT

```
template <const int G, const int M>
void NTT(vector<Modular<M>>& a, bool inv = false) {
  static vector<Modular<M>> root = {0, 1};
  static Modular<M> primitive(G);
  int n = sz(a);
  for (int i = 1, j = 0; i < n - 1; i++) {
    for (int k = n >> 1; (j ^= k) < k; k >>= 1);
    if (i < j) swap(a[i], a[j]);
  }
  int k = sz(root);
  if (k < n)
    for (root.resize(n); k < n; k <<= 1) {
      auto z = primitive.pow((M - 1) / (k << 1));
      fore (i, k >> 1, k) {
        root[i << 1] = root[i];
        root[i << 1 | 1] = root[i] * z;
      }
    }
  for (int k = 1; k < n; k <<= 1)
    for (int i = 0; i < n; i += k << 1)
      fore (j, 0, k) {
        auto t = a[i + j + k] * root[j + k];
        a[i + j + k] = a[i + j] - t;
        a[i + j] = a[i + j] + t;
      }
  if (inv) {
    reverse(1 + all(a));
    auto invN = Modular<M>(1) / n;
    for (auto& x : a) x = x * invN;
  }
}

template <int G = 3, const int M = 998244353>
vector<Modular<M>> convolution(vector<Modular<M>> a, vector
    <Modular<M>> b) {
  // find G using primitive(M)
  // Common NTT couple (3, 998244353)
  if (a.empty() || b.empty()) return {};

  int n = sz(a) + sz(b) - 1, m = n;
  while (n != (n & -n)) ++n;
  a.resize(n, 0), b.resize(n, 0);

  NTT<G, M>(a), NTT<G, M>(b);
  fore (i, 0, n) a[i] = a[i] * b[i];
  NTT<G, M>(a, true);

  return a;
}
```

15 Strings

15.1 KMP $\mathcal{O}(n)$

- aaabaab - $[0, 1, 2, 0, 1, 2, 0]$
- abacaba - $[0, 0, 1, 0, 1, 2, 3]$

```
template <class T>
vector<int> lps(T s) {
  vector<int> p(sz(s), 0);
  for (int j = 0, i = 1; i < sz(s); i++) {
    while (j && (j == sz(s) || s[i] != s[j])) j = p[j - 1];
    if (j < sz(s) && s[i] == s[j]) j++;
    p[i] = j;
  }
  return p;
}

// positions where t is on s
template <class T>
vector<int> kmp(T& s, T& t) {
  vector<int> p = lps(t), pos;
  debug(lps(t), sz(s));
  for (int j = 0, i = 0; i < sz(s); i++) {
    while (j && (j == sz(t) || s[i] != t[j])) j = p[j - 1];
    if (j < sz(t) && s[i] == t[j]) j++;
    if (j == sz(t)) pos.pb(i - sz(t) + 1);
  }
  return pos;
}
```

15.2 KMP automaton $\mathcal{O}(Alphabet * n)$

```
template <class T, int ALPHA = 26>
struct KmpAutomaton : vector<vector<int>> {
  KmpAutomaton() {}
  KmpAutomaton(T s) : vector<vector<int>>(sz(s) + 1, vector
      <int>(ALPHA)) {
    s.pb(0);
    vector<int> p = lps(s);
    auto& nxt = *this;
    nxt[0][s[0] - 'a'] = 1;
    fore (i, 1, sz(s))
      fore (c, 0, ALPHA)
        nxt[i][c] = (s[i] - 'a' == c ? i + 1 : nxt[p[i - 1
            ]][c]);
  }
};
```

15.3 Manacher $\mathcal{O}(n)$

- aaabaab - $[[0, 1, 1, 0, 0, 2, 0], [0, 1, 0, 2, 0, 0, 0]]$
- abacaba - $[[0, 0, 0, 0, 0, 0, 0], [0, 1, 0, 3, 0, 1, 0]]$

```
template <class T>
vector<vector<int>> manacher(T& s) {
  vector<vector<int>> pal(2, vector<int>(sz(s), 0));
  fore (k, 0, 2) {
    int l = 0, r = 0;
    fore (i, 0, sz(s)) {
      int t = r - i + !k;
      if (i < r) pal[k][i] = min(t, pal[k][l + t]);
      int p = i - pal[k][i], q = i + pal[k][i] - !k;
      while (p >= 1 && q + 1 < sz(s) && s[p - 1] == s[q + 1
          ])
        ++pal[k][i], --p, ++q;
      if (q > r) l = p, r = q;
    }
  }
  return pal;
}
```

15.4 Hash

$bases = [1777771, 10006793, 10101283, 10101823, 10136359, 10157387, 10166249]$

$mods = [999727999, 1000000123, 1000002193, 1000008223, 1000009999, 1000027163, 1070777777]$

```
struct Hash : array<int, 2> {
  static constexpr array<int, 2> mod = {1070777777, 1070777
      777};
#define oper(op)                                              \
  friend Hash operator op(Hash a, Hash b) {                   \
    fore (i, 0, sz(a)) a[i] = (1LL * a[i] op b[i] + mod[i])
        % mod[i]; \
    return a;
                                    \
  }
  oper(+) oper(-) oper(*)
} pw[N], ipw[N];

struct Hashing {
  vector<Hash> h;

  static void init() {
#warning "Ensure all base[i] > alphabet"
    pw[0] = ipw[0] = {1, 1};
    Hash base = {12367453, 14567893};
    Hash inv = {::inv(base[0], base.mod[0]), ::inv(base[1],
        base.mod[1])};
    fore (i, 1, N) {
      pw[i] = pw[i - 1] * base;
      ipw[i] = ipw[i - 1] * inv;
    }
  }

  Hashing(string& s) : h(sz(s) + 1) {
    fore (i, 0, sz(s)) {
      int x = s[i] - 'a' + 1;
      h[i + 1] = h[i] + pw[i] * Hash{x, x};
    }
  }

  Hash query(int l, int r) { return (h[r + 1] - h[l]) * ipw
      [l]; }

  lli queryVal(int l, int r) {
    Hash hash = query(l, r);
    return (1LL * hash[0] << 32) | hash[1];
  }
};

// // Save len in the struct and when you do a cut
// Hash merge(vector<Hash>& cuts) {
//   Hash f = {0, 0};
//   fore (i, sz(cuts), 0) {
//     Hash g = cuts[i];
//     f = g + f * pw[g.len];
//   }
//   return f;
// }
```

15.5 Min rotation $\mathcal{O}(n)$

- baabaaa - 4
- abacaba - 6

```
template <class T>
int minRotation(T& s) {
```

```cpp
  int n = sz(s), i = 0, j = 1;
  while (i < n && j < n) {
    int k = 0;
    while (k < n && s[(i + k) % n] == s[(j + k) % n]) k++;
    (s[(i + k) % n] <= s[(j + k) % n] ? j : i) += k + 1;
    j += i == j;
  }
  return i < n ? i : j;
}
```

15.6 Suffix array $\mathcal{O}(nlogn)$

- Duplicates $\sum_{i=1}^{n} lcp[i]$
- Longest Common Substring of various strings
  Add $notUsed$ characters between strings, i.e. $a + \$ + b + \# + c$
  Use two-pointers to find a range $[l, r]$ such that all $notUsed$ characters are present, then $query(lcp[l + 1], .., lcp[r])$ for that window is the common length.

```cpp
template <class T>
struct SuffixArray {
  int n;
  T s;
  vector<int> sa, pos, sp[25];

  SuffixArray(const T& x) : n(sz(x) + 1), s(x), sa(n), pos(
      n) {
    s.pb(0);
    fore (i, 0, n) sa[i] = i, pos[i] = s[i];
    vector<int> nsa(sa), npos(n), cnt(max(260, n), 0);
    for (int k = 0; k < n; k ? k *= 2 : k++) {
      fill(all(cnt), 0);
      fore (i, 0, n) nsa[i] = (sa[i] - k + n) % n, cnt[pos[
          i]]++;
      partial_sum(all(cnt), cnt.begin());
      for (int i = n - 1; i >= 0; i--) sa[--cnt[pos[nsa[i
          ]]]] = nsa[i];
      for (int i = 1, cur = 0; i < n; i++) {
        cur += (pos[sa[i]] != pos[sa[i - 1]] ||
                pos[(sa[i] + k) % n] != pos[(sa[i - 1] + k)
                    % n]);
        npos[sa[i]] = cur;
      }
      pos = npos;
      if (pos[sa[n - 1]] >= n - 1) break;
    }
    sp[0].assign(n, 0);
    for (int i = 0, j = pos[0], k = 0; i < n - 1; ++i, ++k)
        {
      while (k >= 0 && s[i] != s[sa[j - 1] + k])
        sp[0][j] = k--, j = pos[sa[j] + 1];
    }
    for (int k = 1, pw = 1; pw < n; k++, pw <<= 1) {
      sp[k].assign(n, 0);
      for (int l = 0; l + pw < n; l++)
        sp[k][l] = min(sp[k - 1][l], sp[k - 1][l + pw]);
    }
  }

  int lcp(int l, int r) {
    if (l == r) return n - l;
    tie(l, r) = minmax(pos[l], pos[r]);
    int k = __lg(r - l);
    return min(sp[k][l + 1], sp[k][r - (1 << k) + 1]);
  }

  auto at(int i, int j) { return sa[i] + j < n ? s[sa[i] +
      j] : 'z' + 1; }

  int count(T& t) {
    int l = 0, r = n - 1;
    fore (i, 0, sz(t)) {
      int p = l, q = r;
      for (int k = n; k > 0; k >>= 1) {
        while (p + k < r && at(p + k, i) < t[i]) p += k;
        while (q - k > l && t[i] < at(q - k, i)) q -= k;
      }
      l = (at(p, i) == t[i] ? p : p + 1);
      r = (at(q, i) == t[i] ? q : q - 1);
      if (at(l, i) != t[i] && at(r, i) != t[i] || l > r)
        return 0;
    }
    return r - l + 1;
  }

  bool compare(ii a, ii b) {
    // s[a.f ... a.s] < s[b.f ... b.s]
    int common = lcp(a.f, b.f);
    int szA = a.s - a.f + 1, szB = b.s - b.f + 1;
    if (common >= min(szA, szB)) return tie(szA, a) < tie(
        szB, b);
    return s[a.f + common] < s[b.f + common];
  }
};
```

15.7 Aho Corasick $\mathcal{O}(\sum s_i)$

```cpp
struct AhoCorasick {
  struct Node : map<char, int> {
    int link = 0, up = 0;
    int cnt = 0, isWord = 0;
  };

  vector<Node> trie;

  AhoCorasick(int n = 1) { trie.reserve(n), newNode(); }

  int newNode() {
    trie.pb({});
    return sz(trie) - 1;
  }

  void insert(string& s, int u = 0) {
    for (char c : s) {
      if (!trie[u][c]) trie[u][c] = newNode();
      u = trie[u][c];
    }
    trie[u].cnt++, trie[u].isWord = 1;
  }

  int next(int u, char c) {
    while (u && !trie[u].count(c)) u = trie[u].link;
    return trie[u][c];
  }

  void pushLinks() {
    queue<int> qu;
    qu.push(0);
    while (!qu.empty()) {
      int u = qu.front();
      qu.pop();
      for (auto& [c, v] : trie[u]) {
        int l = (trie[v].link = u ? next(trie[u].link, c) :
            0);
        trie[v].cnt += trie[l].cnt;
        trie[v].up = trie[l].isWord ? l : trie[l].up;
        qu.push(v);
```

```
      }
    }
  }

  template <class F>
  void goUp(int u, F f) {
    for (; u != 0; u = trie[u].up) f(u);
  }

  int match(string& s, int u = 0) {
    int ans = 0;
    for (char c : s) {
      u = next(u, c);
      ans += trie[u].cnt;
    }
    return ans;
  }

  Node& operator[](int u) { return trie[u]; }
};
```

15.8 Eertree $\mathcal{O}(\sum s_i)$

```
struct Eertree {
  struct Node : map<char, int> {
    int link = 0, len = 0;
  };

  vector<Node> trie;
  string s = "$";
  int last;

  Eertree(int n = 1) {
    trie.reserve(n), last = newNode(), newNode();
    trie[0].link = 1, trie[1].len = -1;
  }

  int newNode() {
    trie.pb({});
    return sz(trie) - 1;
  }

  int next(int u) {
    while (s[sz(s) - trie[u].len - 2] != s.back()) u = trie
        [u].link;
    return u;
  }

  void extend(char c) {
    s.push_back(c);
    last = next(last);
    if (!trie[last][c]) {
      int v = newNode();
      trie[v].len = trie[last].len + 2;
      trie[v].link = trie[next(trie[last].link)][c];
      trie[last][c] = v;
    }
    last = trie[last][c];
  }

  Node& operator[](int u) { return trie[u]; }

  void substringOccurrences() {
    fore (u, sz(s), 0) trie[trie[u].link].occ += trie[u].
        occ;
  }

  lli occurences(string& s, int u = 0) {
    for (char c : s) {
      if (!trie[u].count(c)) return 0;
      u = trie[u][c];
    }
    return trie[u].occ;
  }
};
```

15.9 Suffix automaton $\mathcal{O}(\sum s_i)$

- $sam[u].len - sam[sam[u].link].len =$ distinct strings
- Number of different substrings (dp) $\mathcal{O}(\sum s_i)$

$$diff(u) = 1 + \sum_{v \in trie[u]} diff(v)$$

- Total length of all different substrings (2 x dp)

$$totLen(u) = \sum_{v \in trie[u]} diff(v) + totLen(v)$$

- Leftmost occurrence $\mathcal{O}(|s|)$ $trie[u].pos = trie[u].len - 1$ if it is **clone** then $trie[clone].pos = trie[q].pos$
- All occurrence positions
- Smallest cyclic shift $\mathcal{O}(|2 * s|)$ Construct sam of $s + s$, find the lexicographically smallest path of $sz(s)$
- Shortest non-appearing string $\mathcal{O}(|s|)$

$$nonAppearing(u) = \min_{v \in trie[u]} nonAppearing(v) + 1$$

```
struct SuffixAutomaton {
  struct Node : map<char, int> {
    int link = -1, len = 0;
  };

  vector<Node> trie;
  int last;

  SuffixAutomaton(int n = 1) { trie.reserve(2 * n), last =
      newNode(); }

  int newNode() {
    trie.pb({});
    return sz(trie) - 1;
  }

  void extend(char c) {
    int u = newNode();
    trie[u].len = trie[last].len + 1;
    int p = last;
    while (p != -1 && !trie[p].count(c)) {
      trie[p][c] = u;
      p = trie[p].link;
    }
    if (p == -1)
      trie[u].link = 0;
    else {
      int q = trie[p][c];
      if (trie[p].len + 1 == trie[q].len)
        trie[u].link = q;
      else {
        int clone = newNode();
        trie[clone] = trie[q];
        trie[clone].len = trie[p].len + 1;
        while (p != -1 && trie[p][c] == q) {
          trie[p][c] = clone;
          p = trie[p].link;
        }
        trie[q].link = trie[u].link = clone;
      }
    }
    last = u;
  }
```

```
  string kthSubstring(lli kth, int u = 0) {
    // number of different substrings (dp)
    string s = "";
    while (kth > 0)
      for (auto& [c, v] : trie[u]) {
        if (kth <= diff(v)) {
          s.pb(c), kth--, u = v;
          break;
        }
        kth -= diff(v);
      }
    return s;
  }

  void substringOccurrences() {
    // trie[u].occ = 1, trie[clone].occ = 0
    vector<int> who(sz(trie) - 1);
    iota(all(who), 1);
    sort(all(who), [&](int u, int v) { return trie[u].len >
          trie[v].len; });
    for (int u : who) {
      int l = trie[u].link;
      trie[l].occ += trie[u].occ;
    }
  }

  lli occurences(string& s, int u = 0) {
    for (char c : s) {
      if (!trie[u].count(c)) return 0;
      u = trie[u][c];
    }
    return trie[u].occ;
  }

  int longestCommonSubstring(string& s, int u = 0) {
    int mx = 0, len = 0;
    for (char c : s) {
      while (u && !trie[u].count(c)) {
        u = trie[u].link;
        len = trie[u].len;
      }
      if (trie[u].count(c)) u = trie[u][c], len++;
      mx = max(mx, len);
    }
    return mx;
  }

  string smallestCyclicShift(int n, int u = 0) {
    string s = "";
    fore (i, 0, n) {
      char c = trie[u].begin()->f;
      s += c;
      u = trie[u][c];
    }
    return s;
  }

  int leftmost(string& s, int u = 0) {
    for (char c : s) {
      if (!trie[u].count(c)) return -1;
      u = trie[u][c];
    }
    return trie[u].pos - sz(s) + 1;
  }

  Node& operator[](int u) { return trie[u]; }
};
```