Contents

1 Basic

1.1 ∼/.vimrc

```
set nocp nu rnu cul ai ci cin si sta
set sc si ts=4 sw=4 sts=4 bs=2 et
set hls sm is ic scs bg=dark
set ru stal=2 ls=2 so=5 wrap lbr
filetype plugin indent on
syntax enable
colo delek
no ; :
no <C-l> :nohl<CR>
au filetype c,cpp ino <F9> <ESC>:w<CR>:!~/r.sh '%'<CR>
au filetype c,cpp  no <F9> <ESC>:w<CR>:!~/r.sh '%'<CR>
let leader = '\'
function! Tg()
    s,^\(\s*\)\?,\1// ,e
    s,^\(\s*\)\(// \?\)\{2},\1,e
endfunc
au filetype c,cpp no <leader><leader> :call Tg()<CR>
```

1.2 ∼/r.sh

```
#!/bin/bash
f=${1?"fn"}
o=.${f%.*}.
s=""
if [ $# = 1 ] || [ $2 = 1 ]; then
  ARGS="-DDEBUG -I$HOME/include_debug"
  s="$s.d"
else
  ARGS="-I$HOME/include"
fi
s="$s.$(md5sum $f | awk '{ print $1 }')"
if [ -e $o$s ]; then
  time >&2 echo cached
else
  rm $o* || true
  set -eux
  time g++ -std=c++17 -Wall -Wextra -Wshadow \
    -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC \
    -Wconversion $ARGS $f -o $o$s
  # -fsanitize=address -fsanitize=undefined
fi
time ./$o$s
```

1.3 preompile.sh

```
cp -r `dirname $(dirname $(g++ df.cpp -H 2>&1 |
  head -n 1 | awk '{ print $2 }'))` ~/include
g++ -std=c++17 stdc++.h -I$HOME/include
```

1.4 Default Code

```
#pragma GCC optimize("Ofast,unroll-loops,fast-math")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
// #pragma GCC ivdep // before loop
#include<bits/stdc++.h>
using namespace std;
#ifdef DEBUG
#define fast
#else
#define fast cin.tie(0)->sync_with_stdio(0)
#define endl '\n'
#define cerr if(1); else cerr
#endif
#define _ <<' '<<
#define ALL(v) v.begin(),v.end()
#define ft first
#define sd second
using ll = long long;
using ld = long double;
using pii = pair<int,int>;
```

1.5 readchar

```
inline char readchar() {
  static const int size = 65536;
  static char buf[size];
  static char *p = buf, *end = buf;
  if (p == end) end = buf +
    fread_unlocked(buf, 1, size, stdin), p = buf;
  return *p++; }
```

1.6 Black Magic

```
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/assoc_container.hpp> //rb_tree
using namespace __gnu_pbds;
using heap = __gnu_pbds::priority_queue<int>;
// less_equal: multi set
template<typename T, typename U = null_type>
using rkt = tree<T, U, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
mt19937 rng((int)chrono::steady_clock::now().
    time_since_epoch().count());
// [0,n), [l,r]
template<typename T> T randint(T l, T r) { return
    uniform_int_distribution<T>(l,r)(rng); }
auto randint(auto n) { return randint(0,n-1); }
// comparator overload
auto cmp = [](seg a, seg b){ return a.func() < b.func()
    ; };
set<seg, decltype(cmp)> s(cmp);
map<seg, int, decltype(cmp)> mp(cmp);
priority_queue<seg, vector<seg>, decltype(cmp)> pq(cmp)
    ; // max heap
struct hasher { // hash func overload
  size_t operator()(const pii &p) const {
    return p.ft * 2 + p.sd * 3; }
}; // T = pii, operator==
unordered_map<pii, int, hasher> hsh;
int main() {
  heap h1, h2; h1.push(1), h1.push(3);
  h2.push(2), h2.push(4); h1.join(h2);
  cerr _ h1.size() _ h2.size() _ h1.top(); //4 0 4
  rkt<int> st; for (int x : {0, 2, 3, 4}) st.insert(x);
  cerr _ *st.find_by_order(2) _ st.order_of_key(1);//31
  // shuffle(ALL(v),rng);
} // __int128_t,__float128_t
```

2 Data Structure

2.1 BIT

```
template<typename S>
struct BIT { // 0-based
#define lb(x) (x&-x)
  int sz; vector<S> ary;
  BIT(int _sz): sz(_sz), ary(_sz) {}
  void update(int x, S v) {
    for(x++; x <= sz; x += lb(x))
      ary[x-1] += v;
  }
  S query(int x) { // [0,x]
    S r;
    if (x >= sz) x = sz;
    for(x++; x > 0; x -= lb(x))
      r += ary[x-1];
    return r;
  }
  S query(int l, int r) { // [l,r]
    if (l > r) return S{};
    return query(r) - query(l-1);
  }
#undef lb
};
struct S {
  int v;
  S(int _v = 0): v(_v) {}
  void operator+=(S o) { v += o.v; }
};
```

2.2 sparse table

```
template<typename T = int, typename CMP = greater<T>>
struct SparseTable {
  int n;
  T st[__lg(MAXN) + 1][MAXN];
  CMP cmp;
  inline T max(T a, T b) { return cmp(a,b) ? a : b; }
  void init(int _n, auto data) {
    n = _n;
    for (int i = 0; i < n; ++i) st[0][i] = data[i];
    for (int i = 1, t = 2; t < n; t <<= 1, i++)
      for (int j = 0; j + t <= n; j++)
        st[i][j] = max(st[i-1][j], st[i-1][j + t/2]);
  }
  T query(int a, int b) { // [a,b]
    int t = __lg(b - a + 1);
    return max(st[t][a], st[t][b - (1 << t) + 1]);
  }
};
```

2.3 Segment Tree

```
struct SegmentTree { // Node, V
#define MYZ int m = l + (r - l) / 2, \
  y = o + 1, z = o + (r - l) / 2 * 2
  int n; vector<Node> ary;
  SegmentTree(int _n, auto& init) { build(_n, init); }
  void build(int _n, const auto& init) {
    n = _n; ary.resize(2*n); build(0, 0, n, init); }
  void modify(int ql, int qr, auto v) {
    modify(0, 0, n, ql, qr, v); }
  auto query(int ql, int qr) {
    return query(0, 0, n, ql, qr); }
  void build(int o, int l, int r, const auto& init) {
    if (l == r-1) {
      ary[o] = Node(init[l]); // TODO
    } else {
      MYZ;
      build(y, l, m, init);
      build(z, m, r, init);
      pull(o, l, r);
    }
  }
  inline void tag(int o, int l, int r, int v) { /**/ }
  inline void push(int o, int l, int r) { MYZ; /**/ }
  inline void pull(int o, int l, int r) { MYZ; /**/ }
  void modify(int o,int l,int r,int ql,int qr, V v) {
    if (r <= ql or qr <= l) return;
    if (ql <= l and r <= qr) {
      tag(o, l, r, v); // TODO
      return;
    }
    MYZ; push(o, l, r);
    modify(y, l, m, ql, qr, v);
    modify(z, m, r, ql, qr, v);
    pull(o, l, r);
  }
  Node query(int o, int l, int r, int ql, int qr) {
    if (r <= ql or qr <= l) return Node{};
    if (ql <= l and r <= qr) return ary[o]; // TODO
    MYZ; push(o, l, r);
    return query(y, l, m, ql, qr)
         + query(z, m, r, ql, qr);
  }
#undef MYZ
};
```

2.4 ZKW Segment Tree

```
const int N = 200000;
struct segtree {
  int n;
  Node tr[N*2], tag[N];
  void upd(int p, Node d, int h) {
    tr[p] += d<<h;
    if(p < n) tag[p] += d;
  }
  void push(int p) {
    for(int h = __lg(n); h >= 0; h--) {
      int i = p>>h;
      if(!tag[i/2]) continue;
      upd(i,tag[i/2],h);
      upd(i^1,tag[i/2],h);
      tag[i/2] = 0;
    }
  }
  Node query(int l, int r) {
    Node resl=0, resr=0; // initialized as identity
    push(l+n), push(r-1+n);
    for(l+=n,r+=n; l<r; l>>=1,r>>=1) {
      if(l&1) resl = resl + tr[l++];
      if(r&1) resr = tr[--r] + resr;
    }
    return resl + resr;
  }
  void pull(int p) {
    for(int h=1; p>1; p>>=1, h++)
    tr[p>>1] = tr[p^1]+tr[p] + (tag[p>>1]<<h);
  }

  void add(int l,int r,Node k) {
    int tl = l, tr = r, h = 0;
    push(l+n), push(r-1+n);
    for(l+=n, r+=n; l<r; l/=2, r/=2, h++) {
      if(l&1) upd(l++,k,h);
      if(r&1) upd(--r,k,h);
    }
    pull(tl+n), pull(tr-1+n);
  }
  void init(ll v[],int _n) {
    n = _n;
    for(int i = 0; i < n; i++) tr[i+n] = v[i];
    for(int i = n-1; i > 0; i--)
      tr[i] = tr[i*2]+tr[i*2|1];
  }
} sgt;
```

2.5 treap

```
struct Treap;
using TreapP = Treap*;
struct Treap {
  int sz, data;
  TreapP l, r;
  Treap(int k): sz(1), data(k), l(0), r(0) {}
};
inline int sz(TreapP o) { return o ? o->sz : 0; }
void pull(TreapP o) { o->sz = sz(o->l)+sz(o->r)+1; }
void push(TreapP o) {}
TreapP merge(TreapP a, TreapP b) {
  if (!a or !b) return a ? a : b;
  TreapP r; // new{ r <- ab }
  if (randint(sz(a)+sz(b)) < sz(a))
       r = a, push(r), r->r = merge(a->r, b);
  else r = b, push(r), r->l = merge(a, b->l);
  return pull(r), r;
}
void split(TreapP o, TreapP &a, TreapP &b, int k) {
  if (!o) return a = b = 0, void();
  push(o);
  if (o->data <= k) // new { ab <- o }
       a = o, split(o->r, a->r, b, k), pull(a);
  else b = o, split(o->l, a, b->l, k), pull(b);
}
void split2(TreapP o, TreapP &a, TreapP &b, int k) {
  if (sz(o) <= k) return a = o, b = 0, void();
  push(o);
  if (sz(o->l) + 1 <= k) // new { ab <- o }
    a = o, split2(o->r, a->r, b, k - sz(o->l) - 1);
  else b = o, split2(o->l, a, b->l, k);
  pull(o); // a b
}
TreapP kth(TreapP o, int k) {
  if (k <= sz(o->l)) return kth(o->l, k);
  if (k == sz(o->l) + 1) return o;
  return kth(o->r, k - sz(o->l) - 1);
}
int Rank(TreapP o, int key) {
  if (o->data < key)
    return sz(o->l) + 1 + Rank(o->r, key);
  else return Rank(o->l, key);
}
bool erase(TreapP &o, int k) {
  if (!o) return 0;
  if (o->data == k) {
    TreapP t = o;
    push(o), o = merge(o->l, o->r);
    delete t;
    return 1;
  }
  TreapP &t = k < o->data ? o->l : o->r;
  return erase(t, k) ? pull(o), 1 : 0;
}
void insert(TreapP &o, int k) {
  TreapP a, b;
  split(o, a, b, k);
  o = merge(a, merge(new Treap(k), b));
}
void interval(TreapP &o, int l, int r) {
  TreapP a, b, c;
  split2(o, a, b, l - 1), split2(b, b, c, r);
  // operate
  o = merge(a, merge(b, c));
}
```

2.6 link cut tree

```
struct Splay;
using SplayP = Splay*;
struct Splay { // xor-sum
  static Splay nil;
  int val, sum, rev, size;
  SplayP ch[2], f;
  Splay(int _val = 0): val(_val), sum(_val),
    rev(0), size(1), ch{ &nil, &nil }, f(&nil) {}
  bool isr() {
    return f->ch[0] != this && f->ch[1] != this;
  }
  int dir() { return f->ch[0] == this ? 0 : 1; }
  void setCh(SplayP c, int d) {
    ch[d] = c;
    if (c != &nil) c->f = this;
    pull();
  }
  void push() {
    if (!rev) return;
    swap(ch[0], ch[1]);
    if (ch[0] != &nil) ch[0]->rev ^= 1;
    if (ch[1] != &nil) ch[1]->rev ^= 1;
    rev = 0;
  }
  void pull() {
    // take care of the nil!
    size = ch[0]->size + ch[1]->size + 1;
    sum = ch[0]->sum ^ ch[1]->sum ^ val;
    if (ch[0] != &nil) ch[0]->f = this;
    if (ch[1] != &nil) ch[1]->f = this;
  }
} Splay::nil; auto nil = &Splay::nil;
void rotate(SplayP x) {
  SplayP p = x->f;
  int d = x->dir();
  if (!p->isr()) p->f->setCh(x, p->dir());
  else x->f = p->f;
  p->setCh(x->ch[!d], d);
  x->setCh(p, !d);
  p->pull(), x->pull();
}
void splay(SplayP x) {
  vector<SplayP > splayVec;
  for (SplayP q = x;; q = q->f) {
    splayVec.eb(q);
    if (q->isr()) break;
  }
  reverse(ALL(splayVec));
  for (auto it : splayVec) it->push();
  while (!x->isr()) {
    if (x->f->isr()) rotate(x);
    else if (x->dir() == x->f->dir())
      rotate(x->f), rotate(x);
    else rotate(x), rotate(x);
  }
}
SplayP access(SplayP x) {
  SplayP q = nil;
  for (; x != nil; x = x->f)
    splay(x), x->setCh(q, 1), q = x;
  return q;
}
void root_path(SplayP x) { access(x), splay(x); }
void chroot(SplayP x) {
  root_path(x), x->rev ^= 1;
  x->push(), x->pull();
}
void split(SplayP x, SplayP y) {
  chroot(x), root_path(y);
}
void link(SplayP x, SplayP y) {
  root_path(x), chroot(y);
  x->setCh(y, 1);
}
void cut(SplayP x, SplayP y) {
  split(x, y);
  if (y->size != 5) return;
  y->push();
  y->ch[0] = y->ch[0]->f = nil;
}
SplayP get_root(SplayP x) {
  for (root_path(x); x->ch[0] != nil; x = x->ch[0])
    x->push();
  splay(x);
  return x;
}
bool conn(SplayP x, SplayP y) {
  return get_root(x) == get_root(y);
}
SplayP lca(SplayP x, SplayP y) {
  access(x), root_path(y);
  if (y->f == nil) return y;
  return y->f;
}
void change(SplayP x, int val) {
  splay(x), x->val = val, x->pull();
}
int query(SplayP x, SplayP y) {
  split(x, y);
  return y->sum;
}
```

3 Graph

3.1 LCA

```
int lgN = __lg(n)+1; // dfs: dep[i], pars[i][0]
vector<vector<int>> pars(n,vector<int>(lgN));
for(int d = 1; d < lgN; d++) // pars[0][0] = 0
    for(int i = 0; i < n; i++)
        pars[i][d] = pars[pars[i][d-1]][d-1];
auto gopar = [&](int x, int w) {
    for (int d = lgN-1; d >= 0; d--)
        if (w >= (1<<d)) x = pars[x][d], w -= (1<<d);
    return x; };
auto lca = [&](int u, int v) {
    if (dep[u] < dep[v]) swap(u,v);
    u = gopar(u, dep[u] - dep[v]);
    if (u == v) return u;
    for(int d = lgN-1; d >= 0; d--)
        if (pars[u][d] != pars[v][d])
            u = pars[u][d], v = pars[v][d];
    return pars[u][0]; };
```

3.2 BCC(vertex)

```
vector<int> G[MAXN]; // 1-base
vector<int> nG[MAXN], bcc[MAXN];
int low[MAXN], dfn[MAXN], Time;
int bcc_id[MAXN], bcc_cnt; // 1-base
bool is_cut[MAXN]; // whether is av
bool cir[MAXN];
int st[MAXN], top;
void dfs(int u, int pa = -1) {
  int child = 0;
  low[u] = dfn[u] = ++Time;
  st[top++] = u;
  for (int v : G[u])
    if (!dfn[v]) {
      dfs(v, u), ++child;
      low[u] = min(low[u], low[v]);
      if (dfn[u] <= low[v]) {
        is_cut[u] = 1;
        bcc[++bcc_cnt].clear();
        int t;
        do {
          bcc_id[t = st[--top]] = bcc_cnt;
          bcc[bcc_cnt].eb(t);
        } while (t != v);
        bcc_id[u] = bcc_cnt;
        bcc[bcc_cnt].eb(u);
      }
    } else if (dfn[v] < dfn[u] and v != pa)
      low[u] = min(low[u], dfn[v]);
  if (pa == -1 and child < 2) is_cut[u] = 0;
}
void bcc_init(int n) {
  Time = bcc_cnt = top = 0;
  for (int i = 1; i <= n; ++i)
    G[i].clear(), dfn[i] = bcc_id[i] = is_cut[i] = 0;
}
void bcc_solve(int n) {
  for (int i = 1; i <= n; ++i)
    if (!dfn[i]) dfs(i);
  // circle-square tree
  for (int i = 1; i <= n; ++i)
    if (is_cut[i])
      bcc_id[i] = ++bcc_cnt, cir[bcc_cnt] = 1;
  for (int i = 1; i <= bcc_cnt and !cir[i]; ++i)
    for (int j : bcc[i]) if (is_cut[j])
      nG[i].eb(bcc_id[j]), nG[bcc_id[j]].eb(i);
}
```

3.3 BCC(bridge)

```
// if there are multi-edges, then they are not bridges
void dfs(int c, int p) {
  tin[c] = low[c] = ++t;
  st.push(c);
  for (auto [x,i]: G[c]) if (x != p) {
      if (tin[x]) {
        low[c] = min(low[c], tin[x]);
        continue;
      }
      dfs(x, c);
      low[c] = min(low[c], low[x]);
      if (low[x] == tin[x]) br[i] = true;
    }
  if (tin[c] == low[c]) {
    ++sz;
    while (st.size()) {
      int u = st.top(); st.pop();
      bcc[u] = sz;
      if (u == c) break;
    }
  }
}
```

3.4 2SAT (SCC)

```
struct SAT { // 0-base
  int low[MAXN], dfn[MAXN], bln[MAXN], n, Time, nScc;
  bool instack[MAXN], istrue[MAXN];
  stack<int> st;
  vector<int> G[MAXN], SCC[MAXN];
  void init(int _n) {
    n = _n; // assert(n * 2 <= MAXN);
    for (int i = 0; i < n*2; ++i) G[i].clear();
  }
  void add_edge(int a, int b) { G[a].eb(b); }
  int rv(int a) { return a >= n ? a - n : a + n; }
  void add_clause(int a, int b) {
    add_edge(rv(a), b), add_edge(rv(b), a); }
  void dfs(int u) {
    dfn[u] = low[u] = ++Time;
    instack[u] = 1, st.push(u);
    for (int i : G[u])
      if (!dfn[i])
        dfs(i), low[u] = min(low[i], low[u]);
      else if (instack[i] and dfn[i] < dfn[u])
        low[u] = min(low[u], dfn[i]);
    if (low[u] == dfn[u]) {
      for (int x = -1; x != u;)
        x = st.top(), st.pop(),
        instack[x] = 0, bln[x] = nScc;
      ++nScc;
    }
  }
  bool solve() {
    Time = nScc = 0;
    for (int i = 0; i < n*2; ++i)
      SCC[i].clear(), low[i] = dfn[i] = bln[i] = 0;
    for (int i = 0; i < n*2; ++i)
      if (!dfn[i]) dfs(i);
    for (int i = 0; i < n*2; ++i) SCC[bln[i]].eb(i);
    for (int i = 0; i < n; ++i) {
      if (bln[i] == bln[i+n]) return false;
      istrue[i] = bln[i] < bln[i+n];
      istrue[i+n] = !istrue[i];
    }
    return true;
  }
};
```

3.5 二分圖匹配

```
array<int,SZ> mp;
array<bool,SZ> vis;
bool dfs(int now) {
    if(vis[now]) return false;
    vis[now] = true;
    for(int i = 0; i < n; i++) {
        if(!G[now][i]) continue;
        if(mp[i] == -1 or dfs(mp[i]))
            return mp[i] = now , true;
    }
    return false;
}
int solve() {
    mp.fill(-1);
    int r = 0;
    for(int i = 0; i < n; i++) {
        vis.fill(false);
        if(dfs(i)) r++;
    }
    return r;
}
```

3.6 Virtual Tree

```
vector<int> vG[MAXN];
int top, st[MAXN];
void insert(int u) {
  if (top == -1) return st[++top] = u, void();
  int p = LCA(st[top], u);
  if (p == st[top]) return st[++top] = u, void();
  while (top >= 1 and dep[st[top-1]] >= dep[p])
    vG[st[top-1]].eb(st[top]), --top;
  if (st[top] != p)
    vG[p].eb(st[top]), --top, st[++top] = p;
  st[++top] = u;
}
void reset(int u) {
  for (int i : vG[u]) reset(i);
  vG[u].clear();
}
void solve(vector<int> &v) {
  top = -1;
  sort(ALL(v),[&](int a,int b){return dfn[a]<dfn[b];});
  for (int i : v) insert(i);
  while (top > 0) vG[st[top-1]].eb(st[top]), --top;
  // do something
  reset(v[0]);
}
```

3.7 Heavy Light Decomposition

```
struct Heavy_light_Decomposition { // 1-base
  int n, t, et, ulink[MAXN], deep[MAXN],
    mxson[MAXN], w[MAXN], pa[MAXN], pl[MAXN],
    data[MAXN], dt[MAXN], bln[MAXN], edge[MAXN];
  vector<pii> G[MAXN];
  void init(int _n) {
    n = _n, t = 0, et = 1;
    for (int i = 1; i <= n; ++i)
      G[i].clear(), mxson[i] = 0;
  }
  void add_edge(int a, int b, int w) {
    G[a].eb(b, et), G[b].eb(a, et), edge[et++] = w;
  }
  void dfs(int u, int f, int d) {
    w[u] = 1, pa[u] = f, deep[u] = d++;
    for (auto &i : G[u])
      if (i.X != f) {
        dfs(i.X, u, d), w[u] += w[i.X];
        if (w[mxson[u]] < w[i.X]) mxson[u] = i.X;
      } else bln[i.Y] = u, dt[u] = edge[i.Y];
  }
  void cut(int u, int link) {
    data[pl[u] = t++] = dt[u], ulink[u] = link;
    if (!mxson[u]) return;
    cut(mxson[u], link);
    for (auto i : G[u])
      if (i.X != pa[u] and i.X != mxson[u])
        cut(i.X, i.X);
  }
  void build() { dfs(1, 1, 1), cut(1, 1), /*build*/; }
  int query(int a, int b) {
    int ta = ulink[a], tb = ulink[b], re = 0;
    while (ta != tb)
      if (deep[ta] < deep[tb])
        /*query*/, tb = ulink[b = pa[tb]];
      else /*query*/, ta = ulink[a = pa[ta]];
    if (a == b) return re;
    if (pl[a] > pl[b]) swap(a, b);
    /*query*/
    return re;
  }
};
```

3.8 Centroid Decomposition

```
struct Cent_Dec { // 1-base
  vector<pll> G[N];
  pll info[N]; // store info. of itself
  pll upinfo[N]; // store info. of climbing up
  int n, pa[N], layer[N], sz[N], done[N];
  ll dis[__lg(N) + 1][N];
  void init(int _n) {
    n = _n, layer[0] = -1;
    fill_n(pa + 1, n, 0), fill_n(done + 1, n, 0);
    for (int i = 1; i <= n; ++i) G[i].clear();
  }
  void add_edge(int a, int b, int w) {
    G[a].pb(pll(b, w)), G[b].pb(pll(a, w));
  }
  void get_cent(
    int u, int f, int &mx, int &c, int num) {
    int mxsz = 0;
    sz[u] = 1;
    for (pll e : G[u])
      if (!done[e.X] && e.X != f) {
        get_cent(e.X, u, mx, c, num);
        sz[u] += sz[e.X], mxsz = max(mxsz, sz[e.X]);
      }
    if (mx > max(mxsz, num - sz[u]))
      mx = max(mxsz, num - sz[u]), c = u;
  }
  void dfs(int u, int f, ll d, int org) {
    // if required, add self info or climbing info
    dis[layer[org]][u] = d;
    for (pll e : G[u])
      if (!done[e.X] && e.X != f)
        dfs(e.X, u, d + e.Y, org);
  }
  int cut(int u, int f, int num) {
    int mx = 1e9, c = 0, lc;
    get_cent(u, f, mx, c, num);
    done[c] = 1, pa[c] = f, layer[c] = layer[f] + 1;
    for (pll e : G[c])
      if (!done[e.X]) {
        if (sz[e.X] > sz[c])
```

```
          lc = cut(e.X, c, num - sz[c]);
        else lc = cut(e.X, c, sz[e.X]);
        upinfo[lc] = pll(), dfs(e.X, c, e.Y, c);
      }
    return done[c] = 0, c;
  }
  void build() { cut(1, 0, n); }
  void modify(int u) {
    for (int a = u, ly = layer[a]; a;
         a = pa[a], --ly) {
      info[a].X += dis[ly][u], ++info[a].Y;
      if (pa[a])
        upinfo[a].X += dis[ly - 1][u], ++upinfo[a].Y;
    }
  }
  ll query(int u) {
    ll rt = 0;
    for (int a = u, ly = layer[a]; a;
         a = pa[a], --ly) {
      rt += info[a].X + info[a].Y * dis[ly][u];
      if (pa[a])
        rt -=
          upinfo[a].X + upinfo[a].Y * dis[ly - 1][u];
    }
    return rt;
  }
};
```

4 Flow

4.1 Flow Model

- Maximum/Minimum flow with lower bound / Circulation problem
  1. Construct super source S and sink T.
  2. For each edge (x, y, l, u), connect $x \to y$ with capacity $u - l$.
  3. For each vertex v, denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  4. If $in(v) > 0$, connect $S \to v$ with capacity $in(v)$, otherwise, connect $v \to T$ with capacity $-in(v)$.
     - To maximize, connect $t \to s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T. If $f \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
     - To minimize, let f be the maximum flow from S to T. Connect $t \to s$ with capacity ∞ and let the flow from S to T be f'. If $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, f' is the answer.
  5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
  1. Redirect every edge: $y \to x$ if $(x, y) \in M$, $x \to y$ otherwise.
  2. DFS from unmatched vertices in X.
  3. $x \in X$ is chosen iff x is unvisited.
  4. $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
  1. Consruct super source S and sink T
  2. For each edge (x, y, c), connect $x \to y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \to x$ with $(cost, cap) = (-c, 1)$
  3. For each edge with $c < 0$, sum these cost as K, then increase $d(y)$ by 1, decrease $d(x)$ by 1
  4. For each vertex v with $d(v) > 0$, connect $S \to v$ with $(cost, cap) = (0, d(v))$
  5. For each vertex v with $d(v) < 0$, connect $v \to T$ with $(cost, cap) = (0, -d(v))$
  6. Flow from S to T, the answer is the cost of the flow $C + K$
- Maximum density induced subgraph
  1. Binary search on answer, suppose we're checking answer T
  2. Construct a max flow model, let K be the sum of all weights
  3. Connect source $s \to v$, $v \in G$ with capacity K
  4. For each edge (u, v, w) in G, connect $u \to v$ and $v \to u$ with capacity w
  5. For $v \in G$, connect it with sink $v \to t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
  6. T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
  1. For each $v \in V$ create a copy v', and connect $u' \to v'$ with weight $w(u, v)$.
  2. Connect $v \to v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v.
  3. Find the minimum weight perfect matching on G'.
- Project selection problem
  1. If $p_v > 0$, create edge (s, v) with capacity p_v; otherwise, create edge (v, t) with capacity $-p_v$.
  2. Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v.
  3. The mincut is equivalent to the maximum profit of a subset of projects.
- 0/1 quadratic programming

  $$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x\bar{y} + \sum_{xyx'y'} c_{xyx'y'}(x\bar{y} + x'\bar{y'})$$

  can be minimized by the mincut of the following graph:

  1. Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y.
  2. Create edge (x, y) with capacity c_{xy}.
  3. Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

4.2 Dinic

```
template<int MAXV, typename T = int, T INF = INT_MAX>
struct Dinic { // 0-base
  struct edge {
    int to; size_t rev; T cap, flow;
  };
  vector<edge> G[MAXV];
  int n, s, t, dis[MAXV]; size_t cur[MAXV];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; i++) G[i].clear();
  }
  void reset() {
    for (int i = 0; i < n; i++)
      for (auto &j : G[i]) j.flow = 0;
  }
  void add_edge(int u, int v, T cap) {
    G[u].eb(edge{ v, G[v].size(), cap, 0 });
    G[v].eb(edge{ u, G[u].size()-1, 0, 0 });
  }
  T dfs(int u, T cap) {
    if (u == t or !cap) return cap;
    for (auto &i = cur[u]; i < G[u].size(); i++) {
      edge &e = G[u][i];
      if (dis[e.to] == dis[u]+1 and e.flow != e.cap) {
        T df = dfs(e.to, min(e.cap - e.flow, cap));
        if (df) {
          e.flow += df;
          G[e.to][e.rev].flow -= df;
          return df;
        }
      }
    }
    dis[u] = -1;
    return 0;
  }
  bool bfs() {
    fill_n(dis, n, -1);
    queue<int> q;
    q.push(s), dis[s] = 0;
    while (q.size()) {
      int x = q.front(); q.pop();
      for (auto &u : G[x])
        if (dis[u.to] == -1 and u.flow != u.cap)
          q.push(u.to), dis[u.to] = dis[x] + 1;
    }
    return dis[t] != -1;
  }
  T maxflow(int _s, int _t) {
    s = _s, t = _t;
    T flow = 0, df;
    while (bfs()) {
      fill_n(cur, n, -1);
      while ((df = dfs(s, INF))) flow += df;
    }
    return flow;
  }
};
```

4.3 BoundedFlow

```
template<int MAXV, typename T = int, T INF = INT_MAX>
struct BoundedFlow { // 0-base
  struct edge {
    int to; size_t rev; T cap, flow;
  };
  vector<edge> G[MAXV];
  int n, s, t, dis[MAXV]; size_t cur[MAXV];T cnt[MAXV];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n + 2; ++i)
      G[i].clear(), cnt[i] = 0;
  }
  void add_edge(int u, int v, T lcap, T rcap) {
    cnt[u] -= lcap, cnt[v] += lcap;
    G[u].eb(edge{ v, G[v].size(), rcap, lcap });
    G[v].eb(edge{ u, G[u].size()-1, 0, 0 });
  }
  void add_edge(int u, int v, T cap) {
    add_edge(u, v, 0, cap); }
  T dfs(int u, T cap) {
    if (u == t or !cap) return cap;
    for (auto &i = cur[u]; i < G[u].size(); i++) {
      edge &e = G[u][i];
      if (dis[e.to] == dis[u]+1 and e.cap != e.flow) {
        T df = dfs(e.to, min(e.cap - e.flow, cap));
        if (df) {
          e.flow += df, G[e.to][e.rev].flow -= df;
          return df;
        }
      }
    }
    dis[u] = -1;
    return 0;
  }
  bool bfs() {
    fill_n(dis, n + 3, -1);
    queue<int> q;
    q.push(s), dis[s] = 0;
    while (q.size()) {
      int u = q.front(); q.pop();
      for (auto &e : G[u])
        if (dis[e.to] == -1 and e.flow != e.cap)
          q.push(e.to), dis[e.to] = dis[u] + 1;
    }
    return dis[t] != -1;
  }
  T maxflow(int _s, int _t) {
    s = _s, t = _t;
    T flow = 0, df;
    while (bfs()) {
      fill_n(cur, n + 3, 0);
      while ((df = dfs(s, INF))) flow += df;
    }
    return flow;
  }
  bool solve() {
    T sum = 0;
    for (int i = 0; i < n; ++i)
      if (cnt[i] > 0)
        add_edge(n + 1, i, cnt[i]), sum += cnt[i];
      else if (cnt[i] < 0) add_edge(i, n + 2, -cnt[i]);
    if (sum != maxflow(n + 1, n + 2)) sum = -1;
    for (int i = 0; i < n; ++i)
      if (cnt[i] > 0)
        G[n + 1].pop_back(), G[i].pop_back();
      else if (cnt[i] < 0)
        G[i].pop_back(), G[n + 2].pop_back();
    return sum != -1;
  }
  T solve(int _s, int _t) {
    add_edge(_t, _s, INF);
    if (!solve()) return -1; // invalid flow
    T x = G[_t].back().flow;
    return G[_t].pop_back(), G[_s].pop_back(), x;
  }
};
```

4.4 Min Cost Max Flow

```
template<int MAXV, typename T = ll, T INF = LLONG_MAX>
struct MCMF { // 0-base
  struct edge {
    int from, to, rev;
    T cap, flow, cost;
  } * past[MAXV];
  vector<edge> G[MAXV];
  bitset<MAXV> inq;
  int s, t, n;
  T mx, flow, cost, dis[MAXV], up[MAXV];
  bool BellmanFord() {
    fill_n(dis, n, INF);
    queue<int> q;
    q.push(s), inq.reset(), inq[s] = 1;
    up[s] = mx - flow, past[s] = 0, dis[s] = 0;
    while (!q.empty()) {
      int u = q.front(); q.pop(), inq[u] = 0;
      if (!up[u]) continue;
      for (auto &e : G[u])
        if (e.flow != e.cap and
          dis[e.to] > dis[u] + e.cost) {
          dis[e.to] = dis[u] + e.cost, past[e.to] = &e;
          up[e.to] = min(up[u], e.cap - e.flow);
          if (!inq[e.to]) inq[e.to] = 1, q.push(e.to);
        }
    }
    if (dis[t] == INF) return 0;
    flow += up[t], cost += up[t] * dis[t];
    for (int i = t; past[i]; i = past[i]->from) {
      auto &e = *past[i];
      e.flow += up[t], G[e.to][e.rev].flow -= up[t];
    }
    return 1;
  }
  auto solve(int _s, int _t) {
    s = _s, t = _t, cost = 0, flow = 0;
    while (BellmanFord()) ;
    return pair{ flow, cost };
  }
  void init(int _n, T _mx = INF) {
    n = _n, mx = _mx;
    for (int i = 0; i < n; ++i) G[i].clear();
  }
  void add_edge(int a, int b, T cap, T c) {
    G[a].eb(edge{ a, b, G[b].size(), cap, 0, c });
    G[b].eb(edge{ b, a, G[a].size()-1, 0, 0, -c });
  }
};
```

4.5 ZKW Min Cost Max Flow

```
template<int MAXV, typename T = int, T INF = INT_MAX>
struct ZKW_MCMF {
  struct Edge {
    int u, v, nxt; T cap, cost;
  } edge[MAXV * MAXV];
  int add, head[MAXV];
  int cur[MAXV]; T dis[MAXV];
  bitset<MAXV> vis;
  int s, t, n;
  T min_cost, max_flow;
  void init(int _n) {
    n = _n, add = 0;
    fill_n(head, n, -1);
  }
  void add_edge(int u, int v, T cp, T ct) {
    edge[add] = Edge{ u, v, head[u], cp, ct };
    head[u] = add++;
    edge[add] = Edge{ v, u, head[v], 0, -ct };
    head[v] = add++;
  }
  T aug(int u, T flow) {
    if (u == t) return flow;
    vis[u] = true;
    for (int &i = cur[u]; i != -1; i = edge[i].nxt) {
      int v = edge[i].v;
      if (edge[i].cap and !vis[v] and
          dis[u] == dis[v] + edge[i].cost) {
        T tmp = aug(v, min(flow, edge[i].cap));
        edge[i].cap -= tmp;
        edge[i ^ 1].cap += tmp;
        if (tmp) return tmp;
      }
    }
    return 0;
  }
  bool modify_label() {
    T d = INF;
    for (int u = 0; u < n; u++) if (vis[u])
      for (int i = head[u]; i != -1; i = edge[i].nxt) {
        int v = edge[i].v;
        if (edge[i].cap and !vis[v])
          d = min(d, dis[v] + edge[i].cost - dis[u]);
      }
    if (d == INF) return false;
    for (int i = 0; i < n; ++i) if (vis[i]) {
      vis[i] = false;
      dis[i] += d;
    }
    return true;
  }
  auto solve(int _s, int _t) {
    s = _s, t = _t;
    min_cost = max_flow = 0;
    fill_n(dis, n, 0);
    while (true) {
      copy_n(head, n, cur);
      while (true) {
        vis.reset();
        T tmp = aug(s, INF);
        if (tmp == 0) break;
        max_flow += tmp;
        min_cost += tmp * dis[s];
      }
      if (!modify_label()) break;
    }
    return pair{ min_cost, max_flow };
  }
};
```

4.6 Global min cut

```
// global min cut
struct SW { // O(V^3)
  static const int MXN = 514;
  int n, vst[MXN], del[MXN];
  int edge[MXN][MXN], wei[MXN];
  void init(int _n) {
    n = _n, MEM(edge, 0), MEM(del, 0);
  }
  void addEdge(int u, int v, int w) {
    edge[u][v] += w, edge[v][u] += w;
  }
  void search(int &s, int &t) {
    MEM(vst, 0), MEM(wei, 0), s = t = -1;
    while (1) {
      int mx = -1, cur = 0;
      for (int i = 0; i < n; ++i)
        if (!del[i] && !vst[i] && mx < wei[i])
          cur = i, mx = wei[i];
      if (mx == -1) break;
      vst[cur] = 1, s = t, t = cur;
      for (int i = 0; i < n; ++i)
        if (!vst[i] && !del[i]) wei[i] += edge[cur][i];
    }
  }
  int solve() {
    int res = INF;
    for (int i = 0, x, y; i < n - 1; ++i) {
      search(x, y), res = min(res, wei[y]), del[y] = 1;
      for (int j = 0; j < n; ++j)
        edge[x][j] = (edge[j][x] += edge[y][j]);
    }
    return res;
  }
};
```

4.7 Kuhn Munkres

```
struct KM { // 0-base
  int w[MAXN][MAXN], hl[MAXN], hr[MAXN], slk[MAXN], n;
  int fl[MAXN], fr[MAXN], pre[MAXN], qu[MAXN], ql, qr;
  bool vl[MAXN], vr[MAXN];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j) w[i][j] = 0;// TODO
  }
  void add_edge(int a, int b, int wei) {
    w[a][b] = wei;
  }
  bool check(int x) {
    if (vl[x] = 1, ~fl[x])
      return vr[qu[qr++] = fl[x]] = 1;
    while (~x) swap(x, fr[fl[x] = pre[x]]);
    return 0;
  }
  void bfs(int s) {
    fill(slk, slk + n, INF);
    fill(vl, vl + n, 0), fill(vr, vr + n, 0);
    ql = qr = 0, qu[qr++] = s, vr[s] = 1;
    while (1) {
      int d;
      while (ql < qr)
        for (int x = 0, y = qu[ql++]; x < n; ++x)
          if (!vl[x] &&
            slk[x] >= (d = hl[x] + hr[y] - w[x][y]))
            if (pre[x] = y, d) slk[x] = d;
            else if (!check(x)) return;
      d = INF;
      for (int x = 0; x < n; ++x)
        if (!vl[x] && d > slk[x]) d = slk[x];
      for (int x = 0; x < n; ++x) {
        if (vl[x]) hl[x] += d;
        else slk[x] -= d;
        if (vr[x]) hr[x] -= d;
      }
      for (int x = 0; x < n; ++x)
        if (!vl[x] && !slk[x] && !check(x)) return;
    }
  }
  int solve() {
    fill(fl,fl+n,-1),fill(fr,fr+n,-1),fill(hr,hr+n,0);
    for (int i = 0; i < n; ++i)
      hl[i] = *max_element(w[i], w[i] + n);
    for (int i = 0; i < n; ++i) bfs(i);
    int res = 0;
    for (int i = 0; i < n; ++i) res += w[i][fl[i]];
    return res;
  }
};
```

5 String

5.1 KMP

```
int F[MAXN];
vector<int> match(auto A, auto B) {
  const int Asz = A.size(), Bsz = B.size();
  vector<int> ans{};
  F[0] = -1, F[1] = 0;
  for (int i = 1, j = 0; i < Bsz; F[++i] = ++j) {
    if (B[i] == B[j]) F[i] = F[j]; // optimize
    while (j != -1 and B[i] != B[j]) j = F[j];
  }
  for (int i = 0, j = 0; i < Asz; ++i) {
    while (j != -1 and A[i] != B[j]) j = F[j];
    if (++j == Bsz) ans.emplace_back(i-j), j = F[j];
  }
  return ans;
}
```

5.2 Z-value

```
int z[MAXN];
void make_z(string s) {
  int l = 0, r = 0;
  for (int i = 1, sz = s.size(); i < sz; i++) {
    z[i] = max(0, min(r - i + 1, z[i - l]));
    while (i + z[i] < sz and s[i + z[i]] == s[z[i]])
         z[i]++;
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
}
```

5.3 Manacher

```
int z[MAXN*2+1];
int Manacher(string tmp) {
  string s = "&";
  int l = 0, r = 0, x, ans;
  for (char c : tmp) s += c, s += '%';
  ans = 0, x = 0;
  const int sz = s.size();
  for (int i = 1; i < sz; i++) {
    z[i] = r > i ? min(z[2 * l - i], r - i) : 1;
    while (s[i + z[i]] == s[i - z[i]]) ++z[i];
    if (z[i] + i > r) r = z[i] + i, l = i;
  }
  for (int i = 1; i < sz; i++)
    if (s[i] == '%') x = max(x, z[i]);
  ans = x / 2 * 2, x = 0;
  for (int i = 1; i < sz; i++)
    if (s[i] != '%') x = max(x, z[i]);
  return max(ans, (x - 1) / 2 * 2 + 1);
}
```

5.4 Suffix Array

```
#define FILL(a,v) memset(a,v,sizeof(a))
struct suffix_array {
  int m, box[MAXN], tp[MAXN];
  int sa[MAXN], ra[MAXN], he[MAXN];
  bool not_equ(int a, int b, int k, int n) {
    return ra[a] != ra[b] or a + k >= n or
      b + k >= n or ra[a + k] != ra[b + k];
  }
  void radix(int *key, int *it, int *ot, int n) {
    fill_n(box, m, 0);
    for (int i = 0; i < n; i++) ++box[key[i]];
    partial_sum(box, box + m, box);
    for (int i = n - 1; i >= 0; --i)
      ot[--box[key[it[i]]]] = it[i];
  }
  void make_sa(string s, int n) {
    int k = 1;
    for (int i = 0; i < n; i++) ra[i] = s[i];
    do {
      iota(tp, tp + k, n - k), iota(sa + k, sa + n, 0);
      radix(ra + k, sa + k, tp + k, n - k);
      radix(ra, tp, sa, n);
      tp[sa[0]] = 0, m = 1;
      for (int i = 1; i < n; i++) {
        m += not_equ(sa[i], sa[i - 1], k, n);
        tp[sa[i]] = m - 1;
      }
      copy_n(tp, n, ra);
      k *= 2;
    } while (k < n && m != n);
  }
  void make_he(string s, int n) {
    for (int j = 0, k = 0; j < n; j++) {
      if (ra[j])
        while (s[j + k] == s[sa[ra[j] - 1] + k]) ++k;
      he[ra[j]] = k, k = max(0, k - 1);
    }
  }
  void build(string s) {
    FILL(sa, 0), FILL(ra, 0), FILL(he, 0);
    FILL(box, 0), FILL(tp, 0), m = 256;
    make_sa(s, (int)s.size());
    make_he(s, (int)s.size());
  }
} SA;
```

5.5 SAIS

```
class SAIS {
public:
  int *SA, *H;
  // zero based, string content MUST > 0
  // result height H[i] is LCP(SA[i - 1], SA[i])
  // string, length, |sigma|
  void build(int *s, int n, int m = 128) {
    copy_n(s, n, _s);
    _h[0] = _s[n++] = 0;
    sais(_s, _sa, _p, _q, _t, _c, n, m);
    mkhei(n);
    SA = _sa + 1;
    H = _h + 1;
  }

private:
  bool _t[N * 2];
  int _s[N * 2], _c[N * 2], x[N], _p[N], _q[N * 2],
    r[N], _sa[N * 2], _h[N];
  void mkhei(int n) {
    for (int i = 0; i < n; i++) r[_sa[i]] = i;
    for (int i = 0; i < n; i++)
      if (r[i]) {
        int ans = i > 0 ? max(_h[r[i - 1]] - 1, 0) : 0;
        while (_s[i + ans] == _s[_sa[r[i] - 1] + ans])
          ans++;
        _h[r[i]] = ans;
      }
  }
  void sais(int *s, int *sa, int *p, int *q, bool *t,
    int *c, int n, int z) {
    bool uniq = t[n - 1] = 1, neq;
    int m = 0, nmxz = -1, *nsa = sa + n, *ns = s + n,
        lst = -1;

#define MAGIC(XD)                                \
  fill_n(sa, n, 0);                              \
  copy_n(c, z, x);                               \
  XD;                                            \
  copy_n(c, z - 1, x + 1);                       \
  for (int i = 0; i < n; i++)                    \
    if (sa[i] and !t[sa[i] - 1])                 \
      sa[x[s[sa[i] - 1]]++] = sa[i] - 1;         \
  copy_n(c, z, x);                               \
  for (int i = n - 1; i >= 0; i--)               \
    if (sa[i] and t[sa[i] - 1])                  \
      sa[--x[s[sa[i] - 1]]] = sa[i] - 1;

    fill_n(c, z, 0);
    for (int i = 0; i < n; i++) uniq &= ++c[s[i]] < 2;
    partial_sum(c, c + z, c);
    if (uniq) {
      for (int i = 0; i < n; i++) sa[--c[s[i]]] = i;
      return;
    }
    for (int i = n-2; i >= 0; i--)
      t[i] = (s[i] == s[i+1] ? t[i+1] : s[i] < s[i+1]);
    MAGIC( for (int i = 1; i <= n-1; i++)
      if (t[i] and !t[i-1])
        sa[--x[s[i]]] = p[q[i] = m++] = i );
    for (int i = 0; i < n; i++)
      if (sa[i] and t[sa[i]] and !t[sa[i]-1]) {
        auto st = s + lst;
        auto sz = p[q[sa[i]] + 1] - sa[i];
        neq = (lst < 0) or !equal(st, st+sz, s+sa[i]);
        ns[q[lst = sa[i]]] = nmxz += neq;
      }
    sais(ns,nsa, p + m, q + n, t + n, c + z, m,nmxz+1);
    MAGIC(for (int i = m - 1; i >= 0; i--)
            sa[--x[s[p[nsa[i]]]]] = p[nsa[i]]);
  }
} sa;
```

5.6 Aho-Corasick Automatan

```
const int len = 400000, sigma = 26;
struct AC_Automatan {
  int nx[len][sigma], fl[len], cnt[len], pri[len], top;
  int newnode() {
    fill(nx[top], nx[top] + sigma, -1);
    return top++;
  }
  void init() { top = 1, newnode(); }
  int input(
    string &s) { // return the end_node of string
    int X = 1;
    for (char c : s) {
      if (!~nx[X][c - 'a']) nx[X][c - 'a'] = newnode();
      X = nx[X][c - 'a'];
    }
    return X;
  }
  void make_fl() {
    queue<int> q;
    q.push(1), fl[1] = 0;
    for (int t = 0; !q.empty();) {
      int R = q.front();
      q.pop(), pri[t++] = R;
      for (int i = 0; i < sigma; ++i)
        if (~nx[R][i]) {
          int X = nx[R][i], Z = fl[R];
          for (; Z && !~nx[Z][i];) Z = fl[Z];
          fl[X] = Z ? nx[Z][i] : 1, q.push(X);
        }
    }
  }
  void get_v(string &s) {
    int X = 1;
    fill(cnt, cnt + top, 0);
    for (char c : s) {
      while (X && !~nx[X][c - 'a']) X = fl[X];
      X = X ? nx[X][c - 'a'] : 1, ++cnt[X];
    }
    for (int i = top - 2; i > 0; --i)
      cnt[fl[pri[i]]] += cnt[pri[i]];
  }
};
```

6 Geometry

6.1 Theorem

$V - E + F = 1 + C$

(Vertex, Edge, Field, Components)

$A = i + \frac{b}{2} - 1$

(A: 面積, i: 內部網格點數, b: 邊上網格點數)

6.2 Default Code

```
using Dt = ld;
using Pt = pair<Dt,Dt>;
using Vt = Pt;
using Line = pair<Pt,Pt>;
const double eps = 1e-9;
bool isZ(Dt x) { return -eps < x and x < eps; }
Pt operator+(Pt a,Pt b){return {a.ft+b.ft,a.sd+b.sd};}
Pt operator-(Pt a,Pt b){return {a.ft-b.ft,a.sd-b.sd};}
Pt operator*(Pt a,Dt k){return { a.ft*k, a.sd*k }; }
Pt operator/(Pt a,Dt k){return { a.ft/k, a.sd/k }; }
Dt dot(Vt a, Vt b)   { return a.ft*b.ft + a.sd*b.sd; }
Dt cross(Vt a, Vt b) { return a.ft*b.sd - a.sd*b.ft; }
Dt abs2(Vt a) { return dot(a,a); }
ld abs(Vt a) { return sqrt(dot(a,a));}
int sign(Dt x) { return isZ(x) ? 0 : x > 0 ? 1 : -1; }
int ori(Pt p1, Pt p2, Pt p3) {
  return sign(cross(p2 - p1, p3 - p2));}
bool collinearity(Pt p1, Pt p2, Pt p3) {
  return isZ(cross(p1 - p3, p2 - p3)); }
bool btw(Pt p1, Pt p2, Pt p3) {
  if(!collinearity(p1, p2, p3)) return 0;
  return sign(dot(p1 - p3, p2 - p3)) <= 0;
}
bool seg_intersect(Pt p1, Pt p2, Pt p3, Pt p4) {
  int a123 = ori(p1, p2, p3);
  int a124 = ori(p1, p2, p4);
  int a341 = ori(p3, p4, p1);
  int a342 = ori(p3, p4, p2);
  if(a123 == 0 && a124 == 0)
    return btw(p1, p2, p3) or btw(p1, p2, p4)
        or btw(p3, p4, p1) or btw(p3, p4, p2);
  return a123 * a124 <= 0 && a341 * a342 <= 0;
}
Pt intersect(Pt p1, Pt p2, Pt p3, Pt p4) {
  Dt a123 = cross(p2 - p1, p3 - p1);
  Dt a124 = cross(p2 - p1, p4 - p1);
  return (p4 * a123 - p3 * a124) / (a123 - a124);
}
Vt perp(Vt a) { return Vt{ -a.sd, a.ft }; }
Pt projection(Pt a, Pt b, Pt p) {
    return (b - a) * dot(p - a, b - a) / abs2(b - a); }
```

6.3 Convex Hull

```
void convex_hull(vector<Pt> &dots) {
  sort(ALL(dots));
  vector<Pt> A(1, dots[0]);
  const int sz = dots.size();
  for(int c = 0; c < 2; reverse(ALL(dots)), c++)
    for(int i = 1, t = A.size(); i < sz;
            A.emplace_back(dots[i++]))
      while (A.size() > t and
          ori(A[A.size()-2], A.back(), dots[i]) <= 0)
        A.pop_back();
  A.pop_back(), A.swap(dots);
} // dots.size() changed !!!
```

6.4 Polar Angle Sort

```
Pt center; //sort base
int Quadrant(Pt a) {
  if(a.ft > 0 && a.sd >= 0) return 1;
  if(a.ft <= 0 && a.sd > 0) return 2;
  if(a.ft < 0 && a.sd <= 0) return 3;
  if(a.ft >= 0 && a.sd < 0) return 4;
}
bool cmp(Pt a, Pt b) { // integer
  a = a - center, b = b - center;
  if (Quadrant(a) != Quadrant(b))
    return Quadrant(a) < Quadrant(b);
  if (cross(b, a) == 0) return abs2(a) < abs2(b);
  return cross(a, b) > 0;
}
bool cmp(Pt a, Pt b) { // float
  a = a - center, b = b - center;
  if(isZ(atan2(a.sd, a.ft) - atan2(b.sd, b.ft)))
    return abs(a) < abs(b);
  return atan2(a.sd, a.ft) < atan2(b.sd, b.ft);
}
```

6.5 Closest Pair (最近點對)

```
struct cmp_y { bool operator()(const Pt &a,
        const Pt &b) const { return a.sd < b.sd; } };
ld solve(vector<Pt> &v) {
  multiset<Pt, cmp_y> s{};
  ld ans = 1e20;
  auto upd_ans = [&](Pt a, Pt b) {
    ld dist = abs(a-b);
    if (ans > dist) ans = dist; };
  s.clear();
  sort(ALL(v), [](Pt a, Pt b) { return a.ft < b.ft
          or (a.ft == b.ft and a.sd < b.sd); });
  for (int i = 0, l = 0, n = v.size(); i < n; i++) {
    while (l < i && v[i].ft - v[l].ft >= ans)
        s.erase(s.find(v[l++]));
    auto it = s.lower_bound(Pt{v[i].ft,v[i].sd - ans});
    while (it != s.end() and it->sd - v[i].sd < ans)
      upd_ans(*it++, v[i]);
    s.insert(v[i]);
  }
  return ans;
}
```

6.6 Circle Intersect

```
struct Cir{ Pt O; Dt R; }; // Dt = ld
bool CCinter(Cir a, Cir b, Pt &p1, Pt &p2) {
  auto [o1,r1] = a; auto [o2,r2] = b;
  auto d2 = abs2(o1 - o2), d = sqrt(d2);
  if(d < max(r1, r2) - min(r1, r2) or d > r1 + r2)
      return 0;     Vt u = (o1+o2)*0.5
          + (o1-o2) * ((r2*r2 - r1*r1) / (2*d2));
  ld A = sqrt((r1 + r2 + d) * (r1 - r2 + d) *
          (r1 + r2 - d) * (-r1 + r2 + d));
  Vt v = perp(o2 - o1) * A / (2 * d2);
  p1 = u + v, p2 = u - v;
  return 1;
}
```

6.7 Circle Cover

```
const int N = 1021;
struct CircleCover {
  int C;
  Cir c[N];
  bool g[N][N], overlap[N][N];
  // Area[i] : area covered by at least i circles
  Dt Area[ N ];
  void init(int _C){ C = _C; }
  struct Teve {
    Pt p; Dt ang; int add;
    Teve() {}
    Teve(Pt _a, Dt _b, int _c): p(_a),ang(_b),add(_c){}
    bool operator<(const Teve &a) const
        { return ang < a.ang; }
  } eve[N * 2];
  // strict: x = 0, otherwise x = -1
  bool disjuct(Cir &a, Cir &b, int x)
      { return sign(abs(a.O - b.O) - a.R - b.R) > x; }
  bool contain(Cir &a, Cir &b, int x)
      { return sign(a.R - b.R - abs(a.O - b.O)) > x; }
  bool contain(int i, int j) {
    /* c[j] is non-strictly in c[i]. */
    auto sij = sign(c[i].R - c[j].R);
    return (sij > 0 or (sij == 0 and i < j))
        and contain(c[i], c[j], -1);
  }
  void solve(){
    fill_n(Area, C + 2, 0);
    for(int i = 0; i < C; ++i)
      for(int j = 0; j < C; ++j)
        overlap[i][j] = contain(i, j);
    for(int i = 0; i < C; ++i)
      for(int j = 0; j < C; ++j)
        g[i][j] = !(overlap[i][j] or overlap[j][i] or
            disjuct(c[i], c[j], -1));
    for(int i = 0; i < C; ++i){
      int E = 0, cnt = 1;
      for(int j = 0; j < C; ++j)
        if(j != i and overlap[j][i])
          ++cnt;
      for(int j = 0; j < C; ++j)
        if(i != j and g[i][j]) {
          Pt aa, bb;
          CCinter(c[i], c[j], aa, bb);
    Dt A = atan2(aa.sd - c[i].O.sd, aa.ft - c[i].O.ft);
    Dt B = atan2(bb.sd - c[i].O.sd, bb.ft - c[i].O.ft);
    eve[E++] = Teve(bb,B,1), eve[E++] = Teve(aa,A,-1);
          if(B > A) ++cnt;
        }
      if(E == 0) Area[cnt] += PI * c[i].R * c[i].R;
      else{
        sort(eve, eve + E);
        eve[E] = eve[0];
        for(int j = 0; j < E; ++j){
          cnt += eve[j].add;
          Area[cnt] += cross(eve[j].p, eve[j+1].p) *.5;
          Dt ang = eve[j + 1].ang - eve[j].ang;
          if (ang < 0) ang += 2. * PI;
          Area[cnt] += (ang-sin(ang))*c[i].R*c[i].R*.5;
        }
      }
    }
  }
};
```

6.8 Minimum Enclosing Circle

```
Pt ccCenter(const Pt &A, const Pt &B, const Pt &C) {
  Vt b = C - A, c = B - A;
  return A + perp(b * abs2(c) - c * abs2(b))
      / cross(b, c) / 2;
}
pair<Pt, ld> mec(vector<Pt> v) {
  shuffle(ALL(v), mt19937(time(0)));
  Pt o = v[0]; int sz = v.size();
  Dt r2 = 0, EPS = 1 + 1e-8; // ld
  for (int i = 0; i < sz; i++)
    if (abs2(v[i] - o) > r2 * EPS) {
      o = v[i], r2 = 0;
        for (int j = 0; j < i; j++)
          if (abs2(v[j] - o) > r2 * EPS) {
            o = (v[i] + v[j]) / 2;
            r2 = abs2(v[i] - o);
            for(int k = 0; k < j; ++k)
              if (abs2(v[k] - o) > r2 * EPS) {
                o = ccCenter(v[i], v[j], v[k]);
                r2 = abs2(v[i] - o);
  } } }
  return { o, sqrt(r2) };
}
```

6.9 Half Plane Intersection

```
bool isin( Line l0, Line l1, Line l2 ) {
  // Check inter(l1, l2) in l0
  Pt p = intersect(l1.ft, l1.sd, l2.ft, l2.sd);
  return cross(l0.sd - l0.ft, p - l0.ft) > eps;
}
/* If no solution, check: 1. ret.size() < 3
 * Or more precisely, 2. interPnt(ret[0], ret[1])
 * in all the lines. (use (l.sd - l.ft) ^ (p - l.ft) >0
 */
/* --^-- Line.ft --^-- Line.sd --^-- */
vector<Line> halfPlaneInter(vector<Line> lines) {
  int sz = lines.size();
  vector<ld> ata(sz), ord(sz);
  for(int i = 0; i < sz; ++i) {
    ord[i] = i;
    Vt d = lines[i].sd - lines[i].ft;
    ata[i] = atan2(d.sd, d.ft);
  }
  sort(ord.begin(), ord.end(), [&](int i, int j) {
      if ( isZ(ata[i] - ata[j]) )
          return cross(lines[i].sd - lines[i].ft,
            lines[j].sd - lines[i].ft) < 0;
      return ata[i] < ata[j];
  });
  vector<Line> fin;
  for (int i = 0; i < sz; ++i)
    if (!i or !isZ(ata[ord[i]] - ata[ord[i-1]]))
      fin.emplace_back(lines[ord[i]]);
  deque<Line> dq;
  for (int i = 0; i < fin.size(); i++){
    while(dq.size() >= 2 and !isin(fin[i],
        dq[dq.size()-2], dq.back()))  dq.pop_back();
    while(dq.size() >= 2 and !isin(fin[i],
        dq[0], dq[1]))  dq.pop_front();
    dq.push_back(fin[i]);
  }
  while(dq.size() >= 3 and !isin(dq[0],
        dq[dq.size()-2], dq.back()))  dq.pop_back();
  while(dq.size() >= 3 and !isin(dq.back(),
        dq[0], dq[1]))  dq.pop_front();
  vector<Line> res(ALL(dq));
  return res;
}
```

7 Math

7.1 extgcd

```
tuple<ll,ll,ll> extgcd(ll a, ll b) {
  ll s = 1, t = 0, u = 0, v = 1;
  while (b) {
    ll q = a / b;
    swap(a -= q * b, b);
    swap(s -= q * t, t);
    swap(u -= q * v, v);
  }
  return { s, u, a }; }
```

7.2 floor / ceil

```
int floor(int a,int b){return a/b - (a%b and a<0^b<0);}
int ceil (int a,int b){return a/b + (a%b and a<0^b>0);}
```

7.3 modmul

```
ull modmul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
```

7.4 Fast GCD

```
ll fast_gcd(ll x, ll y) {
    ll g = 1;
    while (x && y) {
        const int c = __builtin_ctzll(x | y);
        g <<= c; x >>= c; y >>= c;
        x >>= __builtin_ctzll(x);
        y >>= __builtin_ctzll(y);
        if (x < y) swap(x, y);
        x -= y;
    }
    return g * (x + y);
}
```

7.5 Modular

```
template <typename T> struct M {
  static T MOD; // change to constexpr if already known
  T v;
  M(T x = 0) {
    v = (-MOD <= x && x < MOD) ? x : x % MOD;
    if (v < 0) v += MOD;
  }
  explicit operator T() const { return v; }
  bool operator==(const M &b) const { return v == b.v;
      }
  bool operator!=(const M &b) const { return v != b.v;
      }
  M operator-() { return M(-v); }
  M operator+(M b) { return M(v + b.v); }
  M operator-(M b) { return M(v - b.v); }
  M operator*(M b) { return M((__int128)v * b.v % MOD);
       }
  // change implementation to extgcd if MOD is not
      prime
  M operator/(M b) { return *this * (b ^ (MOD - 2)); }
  friend M operator^(M a, ll b) {
    M r(1);
    for (; b; b >>= 1, a *= a)
      if (b & 1) r *= a;
    return r;
  }
  M operator+=(const M &b) {
    if ((v += b.v) >= MOD) v -= MOD;
    return *this;
  }
  M operator-=(const M &b) {
    if ((v -= b.v) < 0) v += MOD;
    return *this;
  }
  friend M &operator*=(M &a, M b) { return a = a * b; }
  friend M &operator/=(M &a, M b) { return a = a / b; }
};
using Mod = M<int>;
template <> int Mod::MOD = 1'000'000'007;
int &MOD = Mod::MOD;
```

7.6 Fraction

```
/* py: from fractions import Decimal, Fraction */
struct fraction {
  ll n, d;
  fraction(ll _n = 0, ll _d = 1): n(_n), d(_d) {
    ll g = __gcd(n,d);
    n /= g; d /= g;
    if(d < 0) n *= -1, d *= -1; }
  fraction operator-() { return fraction(-n,d); }
  fraction operator+(fraction &b) {
    return fraction(n*b.d+b.n*d, d*b.d); }
  fraction operator-(fraction &b){
    return fraction(n*b.d-b.n*d, d*b.d); }
  fraction operator*(fraction &b) {
    return fraction(n*b.n, d*b.d); }
  fraction operator/(fraction &b) {
    return fraction(n*b.d, d*b.n); }
};
```

7.7 Linear Sieve

```
bool *isnP;
vector<int> prime{};
void build(int n) {
    isnP = new bool[n+1]();
    isnP[0] = isnP[1] = 1;
    for(int i = 2; i <= n; i++) {
        if (!isnP[i]) prime.emplace_back(i);
        for(int p: prime) {
            if (i*p > n) break;
            isnP[i*p] = 1;
            if (i%p == 0) break;
        }
    }
}
```

7.8 Factor

```
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (is_prime(n)) return { n };
  ull x = pollard_rho(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), ALL(r));
  return l;
}
```

7.9 miller-rabin

```
// n < 4,759,123,141       3 : 2, 7, 61
// n < 1,122,004,669,633   4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383   6 : pirmes <= 13
// n < 2^64                7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
bool is_prime(ll n) { // ll tns[]
    if (n < 2 or n%2 == 0) return n == 2;
    int s = __builtin_ctzll(n-1);
    for(auto a: tns) {
        auto x = fpow(a, n >> s, n);
        int i = 0;
        while (i < s and (x+1)%n > 2)
            x = ll(LL(x) * x % n), i++;
        if (i and x != n-1) return false;
    }
    return true;
}
```

7.10 Pollard rho

```
ull pollard(ull n) {
  auto f = [n](ull x){ return modmul(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 = modmul(prd, max(x,y) - min(x,y), n)))
          prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
```

7.11 Chinese Remainder Theorem

```
ll solve_crt(ll x1, ll m1, ll x2, ll m2) {
  ll g = __gcd(m1, m2);
  if((x2 - x1) % g) return -1; // no sol
  m1 /= g; m2 /= g;
  auto [pf,ps] = extgcd(m1, m2);
  ll lcm = m1 / g * m2;
  ll res = pf * (x2 - x1) * m1 + x1;
  return (res % lcm + lcm) % lcm;
}
```

7.12 Primes

```
/* 12721 13331 14341 75577 123457 222557 556679 999983
    1097774749 1076767633 100102021 999997771
    1001010013 1000512343 987654361 999991231 999888733
     98789101 987777733 999991921 1010101333 1010102101
     1e12+39 1e15+37 2305843009213693951
    4611686018427387847 9223372036854775783
    18446744073709551557 */
```

7.13 約瑟夫問題

```
int calc(int n, int m) {
    int id = 0;
    for(int i = 2; i <= n; i++)
        id = (id+m) % i;
    return id;
}
```

7.14 Lucas

- $C_m^n = C_{m \bmod p}^{n \bmod p} \cdot C_{\lfloor m/p \rfloor}^{\lfloor n/p \rfloor} \bmod p$
- $C_m^n = 0$ if $m \le n$

7.15 FFT & NTT

```
template <typename T>
void fft_(int n,vector<T> &a,vector<T> &rt,bool inv) {
  vector<int> br(n);
  for (int i = 1; i < n; i++) {
    br[i] = (i&1) ? br[i - 1] + n / 2 : br[i / 2] / 2;
    if (br[i] > i) swap(a[i], a[br[i]]);
  }
  for (int len = 2; len <= n; len *= 2)
    for (int i = 0; i < n; i += len)
      for (int j = 0; j < len / 2; j++) {
        int pos = n / len * (inv ? len - j : j);
        T u = a[i + j], v = a[i + j + len/2] * rt[pos];
        a[i + j] = u + v, a[i + j + len/2] = u - v;
      }
  if (T minv = T(1) / T(n); inv)
    for (T &x : a) x *= minv;
}
void fft(vector<complex<double>> &a, bool inv) {
  int n = a.size();
  vector<complex<double>> rt(n + 1);
  double arg = acos(-1) * 2 / n;
  for (int i = 0; i <= n; i++)
    rt[i] = {cos(arg * i), sin(arg * i)};
  fft_(n, a, rt, inv);
}
// (2^16)+1, 65537, 3
// 7*17*(2^23)+1, 998244353, 3
// 1255*(2^20)+1, 1315962881, 3
// 51*(2^25)+1, 1711276033, 29
void ntt(vector<Mod> &a, bool inv, Mod primitive_root){
  int n = a.size();
  Mod root = primitive_root ^ (MOD - 1) / n;
  vector<Mod> rt(n + 1, 1);
  for (int i = 0; i < n; i++) rt[i + 1] = rt[i] * root;
  fft_(n, a, rt, inv);
}
```

7.16 simplex

```
const int MAXN = 11000, MAXM = 405;
const double eps = 1E-10;
double a[MAXN][MAXM], b[MAXN], c[MAXM];
double d[MAXN][MAXM], x[MAXM];
int ix[MAXN + MAXM]; // !!! array all indexed from 0
// max{cx} subject to {Ax<=b,x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
// usage :
// value = simplex(a, b, c, N, M);
double simplex(int n, int m){
  ++m;
  fill_n(d[n], m + 1, 0);
  fill_n(d[n + 1], m + 1, 0);
  iota(ix, ix + n + m, 0);
  int r = n, s = m - 1;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
    d[i][m - 1] = 1;
    d[i][m] = b[i];
    if (d[r][m] > d[i][m]) r = i;
  }
  copy_n(c, m - 1, d[n]);
  d[n + 1][m - 1] = -1;
  for (double dd;; ) {
    if (r < n) {
      swap(ix[s], ix[r + m]);
      d[r][s] = 1.0 / d[r][s];
      for (int j = 0; j <= m; ++j)
        if (j != s) d[r][j] *= -d[r][s];
      for (int i = 0; i <= n + 1; ++i) if (i != r) {
        for (int j = 0; j <= m; ++j) if (j != s)
          d[i][j] += d[r][j] * d[i][s];
        d[i][s] *= d[r][s];
      }
    }
    r = s = -1;
    for (int j = 0; j < m; ++j)
      if (s < 0 or ix[s] > ix[j]) {
        if (d[n + 1][j] > eps or
            (d[n + 1][j] > -eps and d[n][j] > eps))
          s = j;
      }
    if (s < 0) break;
    for (int i = 0; i < n; ++i) if (d[i][s] < -eps) {
      if (r < 0 or
          (dd = d[r][m] / d[r][s] - d[i][m] / d[i][s])
              < -eps or
          (dd < eps and ix[r + m] > ix[i + m]))
        r = i;
    }
    if (r < 0) return -1; // not bounded
  }
  if (d[n + 1][m] < -eps) return -1; // not executable
  double ans = 0;
  fill_n(x, m, 0);
  for (int i = m; i < n + m; ++i) { // the missing
      enumerated x[i] = 0
    if (ix[i] < m - 1){
      ans += d[i - m][m] * c[ix[i]];
      x[ix[i]] = d[i-m][m];
    }
  }
  return ans;
}
```

8 Else

8.1 Bit Hacks

```
// next permutation of x as a bit sequence
ull next_bits_permutation(ull x) {
  ull c = __builtin_ctzll(x), r = x + (1ULL << c);
  return (r ^ x) >> (c + 2) | r;
}
// iterate over all (proper) subsets of bitset s
void subsets(ull s) {
  for (ull x = s; x;) { --x &= s; /* do stuff */ }
}
```

8.2 Float Binary Search

```
union di {
  double d;
  ull i;
};
bool check(double);
// binary search in [L, R) with relative error 2^-eps
double binary_search(double L, double R, int eps) {
  di l = {L}, r = {R}, m;
  while (r.i - l.i > 1LL << (52 - eps)) {
    m.i = (l.i + r.i) >> 1;
    if (check(m.d)) r = m;
    else l = m;
  }
  return l.d;
}
```

8.3 splitmix64

```
using ull = unsigned long long;
inline ull splitmix64(ull x) {
  // change to `static ull x = SEED;` for DRBG
  ull z = (x += 0x9E3779B97F4A7C15);
  z = (z ^ (z >> 30)) * 0xBF58476D1CE4E5B9;
  z = (z ^ (z >> 27)) * 0x94D049BB133111EB;
  return z ^ (z >> 31);
}
```

8.4 Stack Size

```
constexpr size_t size = 200 << 20; // 200MiB
int main() {
  register long rsp asm("rsp");
  char *buf = new char[size];
  asm("movq %0, %%rsp\n" ::"r"(buf + size));
  // do stuff
  asm("movq %0, %%rsp\n" ::"r"(rsp));
  delete[] buf;
}
```

8.5 Dynamic Convex Trick

```
// only works for integer coordinates!!
struct Line {
    mutable ll a, b, p;
    bool operator<(const Line &rhs) const { return a <
        rhs.a; }
    bool operator<(ll x) const { return p < x; }
};
struct DynamicHull : multiset<Line, less<>> {
    static const ll kInf = 1e18;
    ll Div(ll a, ll b) { return a / b - ((a ^ b) < 0 &&
         a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) { x -> p = kInf; return 0; }
        if (x -> a == y -> a) x -> p = x -> b > y -> b
            ? kInf : -kInf;
        else x -> p = Div(y -> b - x -> b, x -> a - y
            -> a);
        return x -> p >= y -> p;
    }
    void addline(ll a, ll b) {
        auto z = insert({a, b, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y =
             erase(y));
        while ((y = x) != begin() && (--x) -> p >= y ->
             p) isect(x, erase(y));
    }
    ll query(ll x) {
        auto l = *lower_bound(x);
        return l.a * x + l.b;
    }
};
```

8.6 Mos Alogrithm with Modification

```
struct Query { // BLOCK = N^{2/3}
  int id, L, R, Li, Ri, T;
  Query(int i, int l, int r, int t):
    id(i), L(l), R(r), Li(l/BLOCK), Ri(r/BLOCK), T(t){}
  bool operator<(const Query &b) const {
    return tuple{ Li,Ri,T } < tuple{ b.Li,b.Ri,b.T }; }
};
vector<Query> query;
int cur_ans, arr[MAXN], ans[MAXQ];
void solve() {
  sort(ALL(query));
  int L = 0, R = -1, T = -1;
  for(auto q: query) {
    while(T < q.T) addTime(L,R,++T); // TODO
    while(T > q.T) subTime(L,R,T--); // TODO
    while(R < q.R) add(arr[++R]); // TODO
    while(L > q.L) add(arr[--L]); // TODO
    while(R > q.R) sub(arr[R--]); // TODO
    while(L < q.L) sub(arr[L++]); // TODO
    ans[q.id] = cur_ans;
  }
}
```

8.7 Time Segment Tree

```
vector<Event> tr[MAXT  << 1];
#define MYZ int m = l + (r - l) / 2, \
    y = o + 1, z = o + (r - l) / 2 * 2
void insert_event(int o, int l, int r, int ql, int qr,
     Event e) {
    if (r <= ql or  qr <= l) return;
    if (ql  <= l and r  <= qr)
        return tr[o].push_back(e), void();
    MYZ;
    insert_event(y, l, m, ql, qr, e);
    insert_event(z, m, r, ql, qr, e);
}
void traversal(int o, int l, int r) {
    int cnt = 0;
    for (auto e : tr[o])
        if (do_things(e))
            cnt++;
    if (l == r-1) // record ans
    else {
        MYZ;
        traversal(y, l, m);
        traversal(z, m, r);
    }
    while (cnt--) undo();
}
```

8.8 PBDS Custom Policy

```
struct Meta {
    static Meta Null;
    size_t rank;
    ll sum[2];
    Meta(size_t _r = 0): rank(_r), sum{ 0, 0 } {}
} Meta::Null;

#define getMeta(it) ({ \
    auto _it = (it); \
    (_it == end) ? Meta::Null : _it.get_metadata(); \
})
#define PB_DS_CLASS_T_DEC \
    template<typename Node_CItr, typename Node_Itr,
        typename Cmp_Fn, typename _Alloc>
#define PB_DS_CLASS_C_DEC \
    node_update<Node_CItr, Node_Itr, Cmp_Fn, _Alloc>

template<typename Node_CItr, typename Node_Itr,
    typename Cmp_Fn, typename _Alloc>
class node_update {
public:
    using metadata_type = Meta;
    inline ll sum(int i) const;
private:
    virtual Node_CItr node_begin() const = 0;
    virtual Node_CItr node_end() const = 0;
protected:
    node_update() {}
    inline void operator()(Node_Itr it, Node_CItr end)
        const {
        const auto l_meta = getMeta(it.get_l_child());
        const auto r_meta = getMeta(it.get_r_child());
        auto& meta = const_cast<Meta&>(it.get_metadata
            ());
        auto val = *(*it);
        meta.rank = 1 + l_meta.rank + r_meta.rank;
        meta.sum[0] = l_meta.sum[0] + val;
        meta.sum[1] = r_meta.sum[0] + val;
    }
};

PB_DS_CLASS_T_DEC inline ll PB_DS_CLASS_C_DEC::
sum(int i) const {
    auto it = node_begin();
    auto end = node_end();
    if (it == end) return 0;
    return it.get_metadata().sum[i];
}

template<typename T, typename U = null_type>
using rkt = tree<T, U, less<T>, rb_tree_tag,
    node_update>;
```

8.9 Java

```
import java.io.*;
import java.util.*;
import java.math.*;

public class main {
    Scanner sc;
    PrintWriter out;

    void run() throws Exception {
        sc = new Scanner(System.in);
        out = new PrintWriter(System.out);

        int n = sc.nextInt();
        sc.nextLine();
        String s = sc.nextLine();
        ArrayList<Character> v = new ArrayList<
            Character>();
        BigInteger c = BigInteger.valueOf(v.get(n));
        c.isProbablePrime(10); // 1 - 0.5 ^ 10
        c.nextProbablePrime();
        out.println(c);

        out.flush();
    }

    public static void main(String[] args) throws
        Exception {
        new main().run();
    }
}
```

8.10 Kotlin

```
import java.util.*
import kotlin.math.*
private class Scanner {
    val lines = java.io.InputStreamReader(System.`in`).
        readLines()
    var curLine = 0
    var st = StringTokenizer(lines[0])
    fun next(): String {
        while(!st.hasMoreTokens())
            st = StringTokenizer(lines[++curLine])
        return st.nextToken()
    }
    fun nextInt() = next().toInt()
    fun nextLong() = next().toLong()
}

fun main() {
    val sc = Scanner()
    val buf = StringBuilder()

    val mp = Array(5) { Array(5) { -1 } }
    val dx = intArrayOf( 1, 0 )
    val dy = intArrayOf( 0, 1 )
    val v = ArrayList<Int>()

    fun dfs(x: Int, y: Int, s: Int = 0) {
        for((dx,dy) in dx zip dy)
            dfs(x+dx, y+dy, s)
    }
    dfs(0,0)

    val st = v.toSet().toIntArray()
    st.sort()
    println("${st.joinToString()}\n")

    for(i in 1..sc.nextInt()) {
        val k = sc.nextInt()
        val x = st.binarySearch(k)
        buf.append("$k\n")
    }

    print(buf)
}
```

8.11 Point in Polygon

```
using Double = __float128;
using Point = pair<Double, Double>;
#define x first
#define y second

int n, m;
vector<Point> poly;
vector<Point> query;
vector<int> ans;

struct Segment {
    Point a, b;
    int id;
};
vector<Segment> segs;

Double Xnow;
inline Double get_y(const Segment &u, Double xnow =
    Xnow) {
    const Point &a = u.a;
    const Point &b = u.b;
    return (a.y * (b.x - xnow) + b.y * (xnow - a.x)) /
        (b.x - a.x);
}
bool operator<(Segment u, Segment v) {
    Double yu = get_y(u);
    Double yv = get_y(v);
    if (yu != yv) return yu < yv;
    return u.id < v.id;
}
rkt<Segment> st;

struct Event {
    int type;  // +1 insert seg, -1 remove seg, 0 query
    Double x, y;
    int id;
};
bool operator<(Event a, Event b) {
    if (a.x != b.x) return a.x < b.x;
    if (a.type != b.type) return a.type < b.type;
    return a.y < b.y;
}
vector<Event> events;

void solve() {
    set<Double> xs;
    set<Point> ps;
    for (int i = 0; i < n; i++) {
        xs.insert(poly[i].x);
        ps.insert(poly[i]);
    }
    for (int i = 0; i < n; i++) {
        Segment s{poly[i], poly[(i + 1) % n], i};
        if (s.a.x > s.b.x || (s.a.x == s.b.x && s.a.y >
             s.b.y)) {
            swap(s.a, s.b);
        }
        segs.push_back(s);

        if (s.a.x != s.b.x) {
            events.push_back({+1, s.a.x + 0.2, s.a.y, i
                });
            events.push_back({-1, s.b.x - 0.2, s.b.y, i
                });
        }
    }
    for (int i = 0; i < m; i++) {
        events.push_back({0, query[i].x, query[i].y, i
            });
    }
    sort(events.begin(), events.end());
    int cnt = 0;
    for (Event e : events) {
        int i = e.id;
        Xnow = e.x;
        if (e.type == 0) {
            Double x = e.x;
            Double y = e.y;
            Segment tmp = {{x - 1, y}, {x + 1, y}, -1};
            auto it = st.lower_bound(tmp);
            if (ps.count(query[i]) > 0) {
                ans[i] = 0;
            } else if (xs.count(x) > 0) {
                ans[i] = -2;
            } else if (it != st.end() && get_y(*it) ==
                get_y(tmp)) {
                ans[i] = 0;
            } else if (it != st.begin() && get_y(*prev(
                it)) == get_y(tmp)) {
                ans[i] = 0;
            } else {
                int rk = st.order_of_key(tmp);
                if (rk % 2 == 1) {
                    ans[i] = 1;
                } else {
                    ans[i] = -1;
                }
            }
        } else if (e.type == 1) {
            st.insert(segs[i]);
            assert((int)st.size() == ++cnt);
        } else if (e.type == -1) {
            st.erase(segs[i]);
            assert((int)st.size() == --cnt);
        }
    }
}

int main() {
    cin.tie(0); cin.sync_with_stdio(0);

    cin >> n >> m;
    poly = vector<Point>(n);
    for (int i = 0; i < n; i++) {
        long long x, y;
        cin >> x >> y;
        poly[i] = {x * (2e9 + 1) + y, y};
    }
    query = vector<Point>(m);
    ans = vector<int>(m);
    for (int i = 0; i < m; i++) {
        long long x, y;
        cin >> x >> y;
        query[i] = {x * (2e9 + 1) + y, y};
    }
    solve();
    for (int i = 0; i < m; i++) {
        int flag = ans[i];
        if (flag == 1) {
            cout << "YES" << '\n';
        } else if (flag == 0) {
            cout << "YES" << '\n';
        } else if (flag == -1) {
            cout << "NO" << '\n';
        } else {
            cout << "UNKNOWN" << '\n';
        }
    }
    return 0;
}
```