Team notebook

NCTU Daisengen

September 7, 2021

Contents

1 Basics

1.1 default code

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

#define endl '\n'
#define pb emplace_back
#define X first
#define Y second
#define SZ(a) ((int)a.size())
#define ALL(x) x.begin(), x.end()
#define CLR(x, y) memset(x, y, sizeof(x))
#define IOS ios::sync_with_stdio(false); cin.tie(nullptr)
#define rep(i, begin, end) for (__typeof(end) i = (begin) -
     ((begin) > (end)); i != (end) - ((begin) > (end)); i +=
     (begin > end ? -1 : 1))
#define debug(args...) { string _s = #args; replace(_s.begin(),
     _s.end(), ',', ' '); stringstream _ss(_s);
     istream_iterator<string> _it(_ss); err(_it, args); }
void err(istream_iterator<string> it) {}
template<typename T, typename... Args>
void err(istream_iterator<string> it, T a, Args... args) {
       cerr << *it << " = " << a << endl;
       err(++it, args...);
}

using ll = long long;
using vi = vector <int>;
using vii = vector <vi>;
using pii = pair <int, int>;
using pll = pair <ll , ll >;

const int MOD = 1000000007;
const ll INF = 0x7f7f7f7f7f7f7f7f; // 922337203685477580;

signed main () {
//     IOS;
}
```

1.2 Shell script

```
g++ -O2 -std=c++17 -Wall -Wextra -Wshadow -o $1 $1.cpp
chmod +x compile.sh
```

1.3 vimrc

```
"This file should be placed at ~/.vimrc"
se nu ai hls et ru ic is sc cul
se re=1 ts=4 sts=4 sw=4 ls=2 mouse=a
syntax on
hi cursorline cterm=none ctermbg=89
set bg=dark
inoremap {<ENTER> {}<LEFT><ENTER><ENTER><UP><TAB>
```

2 Data structures

2.1 Distinct Color

```
struct query {
    int l, r, i;
    void input(int id) {
        i = id;
        cin >> l >> r;
    }
    bool operator<(const query& b) {
        return r < b.r;
    }
} Q[V];

int n, qc, ans[V], lst_vis[V], x[V];
vi xc;

signed main () {
    // input
    for (int i = 0; i < n; ++i)
        cin >> x[i], xc.pb(x[i]);

    for (int i = 0; i < q; ++i)
        Q[i].input(i);

    sort(Q, Q + q); // sort all queries
    sort(ALL(xc)); // discrete all color

    for (int i = 0; i < n; ++i) {
        int j = LB(xc, x[i]);
        if (lst_vis[j])   add(lst_vis[j], -1);
        lst_vis[j] = i + 1; add(lst_vis[j], 1);
        while (qc < q && Q[qc].r == i)
            ans[Q[qc].i] = sum(i + 1) - sum(Q[qc].l), ++qc;
    }
    // output answer
}
```

2.2 hash table

```
/*
 * Micro hash table, can be used as a set. Very efficient vs
      std::set
*/
const int MN = 1001;
struct ht {
  int _s[(MN + 10) >> 5];
  int len;
  void set(int id) {
    len++;
    _s[id >> 5] |= (1LL << (id & 31));
  }
  bool is_set(int id) {
    return _s[id >> 5] & (1LL << (id & 31));
  }
};
```

2.3 heavy light decomposition

```
vi g[V]; // 1-index
int p[V],d[V],sz[V],hs[V];//parent,depth,subtree size,heavy son
int t,tp[V],in[V],rnk[V];//time,top,dfs num(using in DS),rank
struct HeavyLineDCP {
  int n;
  SegmentTree st; // 1-index
  HeavyLineDCP(int n) : n(n) {
    for (int i = 1; i <= n; ++i)
      hs[i] = 0;
      t = 0;
  }
  void DCP(int src=1) {
      d[src] = t = 0;
      d0(src, src);
      d1(src, src, src);
  }
  void d0(int x, int px) {
    p[x] = px;
    sz[x] = 1;

    int h = 0;
    for (int y : g[x])
      if (y != px) {
        d[y] = d[x] + 1;
        d0(y, x);
        sz[x] += sz[y];
        if (sz[y] > h)
          h = sz[y], hs[x] = y;
      }
    }
  void d1(int x, int px, int top) {
    in[x] = ++t;
    rnk[t] = x;
    tp[x] = top;
    if (!hs[x])
      return;
    d1(hs[x], x, top);
    for (int y : g[x])
      if (y != px && y != hs[x])
        d1(y, x, y);
  }
  void build(int w[]) {
    for (int i = 1; i <= n; ++i)
      arr[in[i]] = w[i];
    st = SegmentTree(n);
    st.build(1, 1, n);
```

```
  }
  void upd(int s, int x) {
    st.upd(1, 1, n, in[s], x);
  }
  int query(int a, int b) {
    int ans = 0;
    while (tp[a] != tp[b]) {
      if (d[tp[a]] < d[tp[b]])
        ans=max(ans,st.query(1,1,n,in[tp[b]],in[b])),b=p[tp[b]];
      else
        ans=max(ans,st.query(1,1,n,in[tp[a]],in[a])),a=p[tp[a]];
    }
    if (in[a] > in[b])
      swap(a, b);
    ans = max(ans, st.query(1, 1, n, in[a], in[b]));
    return ans;
  }
};
```

2.4 LiChaoST

```
struct LiChao_min {
  struct line {
    LL m, c;
    line(LL _m = 0, LL _c = 0) {
      m = _m;
      c = _c;
    }
    LL eval(LL x) { return m * x + c; }
  };
  struct node {
    node *l, *r;
    line f;
    node(line v) {
      f = v;
      l = r = NULL;
    }
  };
  typedef node *pnode;
  pnode root;
  int sz;
#define mid ((l + r) >> 1)
  void insert(line &v, int l, int r, pnode &nd) {
    if (!nd) {
      nd = new node(v);
      return;
    }
    LL trl = nd->f.eval(l), trr = nd->f.eval(r);
    LL vl = v.eval(l), vr = v.eval(r);
    if (trl <= vl && trr <= vr) return;
    if (trl > vl && trr > vr) {
      nd->f = v;
      return;
    }
    if (trl > vl) swap(nd->f, v);
    if (nd->f.eval(mid) < v.eval(mid))
      insert(v, mid + 1, r, nd->r);
    else swap(nd->f, v), insert(v, l, mid, nd->l);
  }
  LL query(int x, int l, int r, pnode &nd) {
    if (!nd) return LLONG_MAX;
    if (l == r) return nd->f.eval(x);
    if (mid >= x)
      return min(
        nd->f.eval(x), query(x, l, mid, nd->l));
    return min(
      nd->f.eval(x), query(x, mid + 1, r, nd->r));
  }
  /* -sz <= query_x <= sz */
  void init(int _sz) {
    sz = _sz + 1;
    root = NULL;
  }
  void add_line(LL m, LL c) {
    line v(m, c);
    insert(v, -sz, sz, root);
  }
  LL query(LL x) { return query(x, -sz, sz, root); }
};
```

2.5 link cut tree

```
struct Splay { // xor-sum
  static Splay nil;
  Splay *ch[2], *f;
  int val, sum, rev, size;
  Splay(int _val = 0)
    : val(_val), sum(_val), rev(0), size(1) {
    f = ch[0] = ch[1] = &nil;
  }
  bool isr() {
    return f->ch[0] != this && f->ch[1] != this;
  }
  int dir() { return f->ch[0] == this ? 0 : 1; }
  void setCh(Splay *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;
Splay *nil = &Splay::nil;
void rotate(Splay *x) {
  Splay *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(Splay *x) {
  vector<Splay *> splayVec;
  for (Splay *q = x;; q = q->f) {
    splayVec.pb(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);
  }
}
Splay *access(Splay *x) {
  Splay *q = nil;
  for (; x != nil; x = x->f)
    splay(x), x->setCh(q, 1), q = x;
  return q;
}
void root_path(Splay *x) { access(x), splay(x); }
void chroot(Splay *x) {
  root_path(x), x->rev ^= 1;
  x->push(), x->pull();
}
void split(Splay *x, Splay *y) {
  chroot(x), root_path(y);
}
void link(Splay *x, Splay *y) {
  root_path(x), chroot(y);
  x->setCh(y, 1);
}
void cut(Splay *x, Splay *y) {
  split(x, y);
  if (y->size != 5) return;
  y->push();
  y->ch[0] = y->ch[0]->f = nil;
}
Splay *get_root(Splay *x) {
  for (root_path(x); x->ch[0] != nil; x = x->ch[0])
    x->push();
  splay(x);
  return x;
}
bool conn(Splay *x, Splay *y) {
  return get_root(x) == get_root(y);
}
Splay *lca(Splay *x, Splay *y) {
  access(x), root_path(y);
```

```
  if (y->f == nil) return y;
  return y->f;
}
void change(Splay *x, int val) {
  splay(x), x->val = val, x->pull();
}
int query(Splay *x, Splay *y) {
  split(x, y);
  return y->sum;
}
```

2.6 persistent array

```
struct node {
  node *l, *r;
  int val;

  node (int x) : l(NULL), r(NULL), val(x) {}
  node () : l(NULL), r(NULL), val(-1) {}
};

typedef node* pnode;

pnode update(pnode cur, int l, int r, int at, int what) {
  pnode ans = new node();

  if (cur != NULL) {
    *ans = *cur;
  }
  if (l == r) {
    ans-> val = what;
    return ans;
  }
  int m = (l + r) >> 1;
  if (at <= m) ans-> l = update(ans-> l, l, m, at, what);
  else ans-> r = update(ans-> r, m + 1, r, at, what);
  return ans;
}

int get(pnode cur, int l, int r, int at) {
  if (cur == NULL) return 0;
  if (l == r) return cur-> val;
  int m = (l + r) >> 1;
  if (at <= m) return get(cur-> l, l, m, at);
  else         return get(cur-> r, m + 1, r, at);
}
```

2.7 persistent seg tree

```
/**
 *  Important:
 *  When using lazy propagation remember to create new
 *  versions for each push_down operation!!!
 * */
struct node {
  node *l, *r;
  long long acc;
  int flip;
  node (int x) : l(NULL), r(NULL), acc(x), flip(0) {}
  node () : l(NULL), r(NULL), acc(0), flip(0) {}
};

typedef node* pnode;

pnode create(int l, int r) {
  if (l == r) return new node();
  pnode cur = new node();
  int m = (l + r) >> 1;
  cur-> l = create(l, m);
  cur-> r = create(m + 1, r);
  return cur;
}

pnode copy_node(pnode cur) {
  pnode ans = new node();
  *ans = *cur;
  return ans;
}

void push_down(pnode cur, int l, int r) {
  assert(cur);
  if (cur-> flip) {
    int len = r - l + 1;
    cur-> acc = len - cur-> acc;
    if (cur-> l) {
      cur-> l = copy_node(cur-> l);
      cur-> l -> flip ^= 1;
    }
    if (cur-> r) {
      cur-> r = copy_node(cur-> r);
      cur-> r -> flip ^= 1;
    }
    cur-> flip = 0;
  }
}

int get_val(pnode cur) {
  assert(cur);
  assert((cur-> flip) == 0);
  if (cur) return cur-> acc;
  return 0;
}

pnode update(pnode cur, int l, int r, int at, int what) {
  pnode ans = copy_node(cur);
  if (l == r) {
    assert(l == at);
    ans-> acc = what;
    ans-> flip = 0;
    return ans;
  }
  int m = (l + r) >> 1;
  push_down(ans, l, r);
  if (at <= m) ans-> l = update(ans-> l, l, m, at, what);
  else ans-> r = update(ans-> r, m + 1, r, at, what);

  push_down(ans-> l, l, m);
  push_down(ans-> r, m + 1, r);
  ans-> acc = get_val(ans-> l) + get_val(ans-> r);
  return ans;
}

pnode flip(pnode cur, int l, int r, int a, int b) {
  pnode ans = new node();

  if (cur != NULL) {
    *ans = *cur;
  }
  if (l > b || r < a)
    return ans;

  if (l >= a && r <= b) {
    ans-> flip ^= 1;
    push_down(ans, l, r);
    return ans;
  }

  int m = (l + r) >> 1;
  ans-> l = flip(ans-> l, l, m, a, b);
  ans-> r = flip(ans-> r, m + 1, r, a, b);
  push_down(ans-> l, l, m);
  push_down(ans-> r, m + 1, r);
  ans-> acc = get_val(ans-> l) + get_val(ans-> r);
  return ans;
}

long long get_all(pnode cur, int l, int r) {
  assert(cur);
  push_down(cur, l, r);
  return cur-> acc;
}

void traverse(pnode cur, int l, int r) {
  if (!cur) return;
  cout << l << " - " << r << " : " << (cur-> acc) << " " <<
       (cur-> flip) << endl;
  traverse(cur-> l, l, (l + r) >> 1);
  traverse(cur-> l, 1 + ((l + r) >> 1), r);
}
```

2.8 persistent trie

```
// Persistent binary trie (BST for integers)
const int MD = 31;

struct node_bin {
  node_bin *child[2];
  int val;

  node_bin() : val(0) {
    child[0] = child[1] = NULL;
```

```
  }
};

typedef node_bin* pnode_bin;

pnode_bin copy_node(pnode_bin cur) {
  pnode_bin ans = new node_bin();
  if (cur) *ans = *cur;
  return ans;
}

pnode_bin modify(pnode_bin cur, int key, int inc, int id = MD) {
  pnode_bin ans = copy_node(cur);
  ans->val += inc;
  if (id >= 0) {
    int to = (key >> id) & 1;
    ans->child[to] = modify(ans->child[to], key, inc, id - 1);
  }
  return ans;
}

int sum_smaller(pnode_bin cur, int key, int id = MD) {
  if (cur == NULL) return 0;
  if (id < 0) return 0; // strictly smaller
  // if (id == - 1) return cur->val; // smaller or equal

  int ans = 0;
  int to = (key >> id) & 1;
  if (to) {
    if (cur->child[0]) ans += cur->child[0]->val;
    ans += sum_smaller(cur->child[1], key, id - 1);
  } else {
    ans = sum_smaller(cur->child[0], key, id - 1);
  }
  return ans;
}


// Persistent trie for strings.
const int MAX_CHILD = 26;
struct node {
  node *child[MAX_CHILD];
  int val;
  node() : val(-1) {
    for (int i = 0; i < MAX_CHILD; i++) {
      child[i] = NULL;
    }
  }
};

typedef node* pnode;

pnode copy_node(pnode cur) {
  pnode ans = new node();
  if (cur) *ans = *cur;
  return ans;
}

pnode set_val(pnode cur, string &key, int val, int id = 0) {
  pnode ans = copy_node(cur);
  if (id >= int(key.size())) {
    ans->val = val;
  } else {
    int t = key[id] - 'a';
    ans->child[t] = set_val(ans->child[t], key, val, id + 1);
  }
  return ans;
}

pnode get(pnode cur, string &key, int id = 0) {
  if (id >= int(key.size()) || !cur)
    return cur;
  int t = key[id] - 'a';
  return get(cur->child[t], key, id + 1);
}
```

2.9 segment tree

```
const int MN = 1e5; // limit for array size

struct seg_tree {
  int n; // array size
  int t[2 * MN];

  seg_tree(int _n) : n(_n) {}

  void clear() {
    memset(t, 0, sizeof t);
  }

  void build() { // build the tree
    for (int i = n - 1; i > 0; --i) t[i] = t[i<<1] + t[i<<1|1];
  }

  // Single modification, range query.
  void modify(int p, int value) { // set value at position p
    for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] = t[p] +
        t[p^1];
  }

  int query(int l, int r) { // sum on interval [l, r)
    int res = 0;
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l&1) res += t[l++];
      if (r&1) res += t[--r];
    }
    return res;
  }
};

// Range modification, single query.

void modify(int l, int r, int value) {
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) t[l++] += value;
    if (r&1) t[--r] += value;
  }
}

int query(int p) {
  int res = 0;
  for (p += n; p > 0; p >>= 1) res += t[p];
  return res;
}

/**
 * If at some point after modifications we need to inspect all
      the
 * elements in the array, we can push all the modifications to
      the
 * leaves using the following code. After that we can just
      traverse
 * elements starting with index n. This way we reduce the
      complexity
 * from O(n log(n)) to O(n) similarly to using build instead of
      n modifications.
 * */

void push() {
  for (int i = 1; i < n; ++i) {
    t[i<<1] += t[i];
    t[i<<1|1] += t[i];
    t[i] = 0;
  }
}

// Non commutative combiner functions.

void modify(int p, const S& value) {
  for (t[p += n] = value; p >>= 1; ) t[p] = combine(t[p<<1],
       t[p<<1|1]);
}

S query(int l, int r) {
  S resl, resr;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) resl = combine(resl, t[l++]);
    if (r&1) resr = combine(t[--r], resr);
  }
  return combine(resl, resr);
}


/**
 * segment tree for intervals
 *
 * */

const int MN = 100000 + 100;
struct seg_tree {
  int val[MN * 4 + 4];
  int pending[MN * 4 + 4];

  seg_tree() {
    memset(val, -1, sizeof val);
    memset(pending, -1, sizeof pending);
  }
```

```
  void propagate(int node, int b, int e) {
    if (pending[node] != -1) {
      val[node] = pending[node];
      if (b < e) {
        pending[node << 1] = pending[node];
        pending[node << 1 | 1] = pending[node];
      }
      pending[node] = -1;
    }
  }

  void set(int node, int b, int e, int from, int to, int v) {
    if (b > to || e < from) return;

    if (b >= from && e <= to) {
      pending[node] = v;
      propagate(node, b, e);
      return;
    }

    int mid = (b + e) >> 1;

    set(node << 1, b, mid, from, to, v);
    set(node << 1 | 1, mid + 1, e, from, to, v);
  }



  int query(int node, int b, int e, int pos) {
    propagate(node, b, e);

    if (b == e && b == pos) {
      return val[node];
    }

    int mid = (b + e) >> 1;
    if (pos <= mid)
      return query(node << 1, b, mid, pos);
    return query(node << 1 | 1, mid + 1, e, pos);
  }

  void set(int from, int to, int v) {
    return set(1, 0, MN - 1, from, to, v);
  }

  int query(int pos) {
    return query(1, 0, MN - 1, pos);
  }
};
```

2.10 sparse table

```
const int MN = 100000 + 10; // Max number of elements
const int ML = 18; // ceil(log2(MN));

struct st {
  int data[MN];
  int M[MN][ML];
  int n;

  void build() {
    for (int i = 0; i < n; ++i)
      M[i][0] = data[i];
    for (int j = 1, p = 2, q = 1; p <= n; ++j, p <<= 1, q <<= 1)
      for (int i = 0; i + p - 1 < n; ++i)
        M[i][j] = max(M[i][j - 1], M[i + q][j - 1]);
  }
  int query(int b, int e) {
    int k = log2(e - b + 1);
    return max(M[b][k], M[e + 1 - (1<<k)][k]);
  }
};
```

2.11 splay tree

```
using namespace std;
#include<bits/stdc++.h>
#define D(x) cout<<x<<endl;

typedef int T;

struct node{
  node *left, *right, *parent;
  T key;
  node (T k) : key(k), left(0), right(0), parent(0) {}
};

struct splay_tree{


  node *root;

  void right_rot(node *x) {
    node *p = x->parent;
    if (x->parent = p->parent) {
      if (x->parent->left == p) x->parent->left = x;
      if (x->parent->right == p) x->parent->right = x;
    }
    if (p->left = x->right) p->left->parent = p;
    x->right = p;
    p->parent = x;
  }

  void left_rot(node *x) {
    node *p = x->parent;
    if (x->parent = p->parent) {
      if (x->parent->left == p) x->parent->left = x;
      if (x->parent->right == p) x->parent->right = x;
    }
    if (p->right = x->left) p->right->parent = p;
    x->left = p;
    p->parent = x;
  }

  void splay(node *x, node *fa = 0) {

    while( x->parent != fa and x->parent != 0) {
      node *p = x->parent;
      if (p->parent == fa)
        if (p->right == x)
          left_rot(x);
        else
          right_rot(x);
      else {
        node *gp = p->parent; //grand parent
        if (gp->left == p)
          if (p->left == x)
            right_rot(x),right_rot(x);
          else
            left_rot(x),right_rot(x);
        else
          if (p->left == x)
            right_rot(x), left_rot(x);
          else
            left_rot(x), left_rot(x);
      }
    }
    if (fa == 0) root = x;
  }

  void insert(T key) {
    node *cur = root;
    node *pcur = 0;
    while (cur) {
      pcur = cur;
      if (key > cur->key) cur = cur->right;
      else cur = cur->left;
    }
    cur = new node(key);
    cur->parent = pcur;
    if (!pcur) root = cur;
    else if (key > pcur->key ) pcur->right = cur;
    else pcur->left = cur;
    splay(cur);
  }

  node *find(T key) {
    node *cur = root;
    while (cur) {
      if (key > cur->key) cur = cur->right;
      else if(key < cur->key) cur = cur->left;
      else return cur;
    }
    return 0;
  }

  splay_tree(){ root = 0;};
};
```

2.12 STL order statistics tree II

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree<int,null_type,less<int>,rb_tree_tag,
tree_order_statistics_node_update> order_set;

order_set X;

int get(int y) {
  int l=0,r=1e9+1;
  while(l<r) {
    int m=l+((r-l)>>1);
    if(m-X.order_of_key(m+1)<y)
      l=m+1;
    else
      r=m;
  }
  return l;
}

main(){
  ios::sync_with_stdio(0);
  cin.tie(0);
  int n,m;
  cin>>n>>m;

  for(int i=0;i<m;i++) {
    char a;
    int b;
    cin>>a>>b;
    if(a=='L')
      cout<<get(b)<<endl;
    else
      X.insert(get(b));
  }
}

/***
Input
20 7
L 5
D 5
L 4
L 5
D 5
L 4
L 5

Output
5
4
6
4
7
***/
```

2.13 STL order statistics tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <bits/stdc++.h>

using namespace __gnu_pbds;
using namespace std;

typedef
tree<
  pair<int,int>,
  null_type,
  less<pair<int,int>>,
  rb_tree_tag,
  tree_order_statistics_node_update>
ordered_set;

main()
 {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n;
    int sz=0;
    cin>>n;
    vector<int> ans(n,0);

    ordered_set t;
    int x,y;
    for(int i=0;i<n;i++)
    {
        cin>>x>>y;
        ans[t.order_of_key({x,++sz})]++;
        t.insert({x,sz});
    }

    for(int i=0;i<n;i++)
        cout<<ans[i]<<'\n';
}

/***
Input
5
1 1
5 1
7 1
3 3
5 5

Output
1
2
1
1
0
***/
```

2.14 STL Treap

```
/*
                                :

#include <ext/rope> //header with rope
using namespace __gnu_cxx; //namespace with rope and some
     additional stuff

rope<int> x, y; int s, len;

x.size(),   x.push_back(s),   x.substr(pos, s)  x[i]

x.insert(pos, s or y)
x.erase(pos, len)       [pos, pos + len)
x.replace(pos, s)  x [pos]       s
x.copy(pos, len, y)     [pos, pos + len)   y
*/

#include <iostream>
#include <ext/rope> //header with rope
using namespace std;
using namespace __gnu_cxx; //namespace with rope and some
     additional stuff

int main()
{
    ios_base::sync_with_stdio(false);
    rope <int> v; //use as usual STL container
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; ++i)
        v.push_back(i); //initialization
    int l, r;
    for(int i = 0; i < m; ++i)
    {
        cin >> l >> r;
        --l, --r;
        rope <int> cur = v.substr(l, r - l + 1);
        v.erase(l, r - l + 1);
        v.insert(v.mutable_begin(), cur);
    }
    for(rope <int>::iterator it = v.mutable_begin(); it !=
         v.mutable_end(); ++it)
        cout << *it << " ";
    return 0;
}
```

2.15 wavelet tree

```
/*
                                        :
                    NQ                      [0,i]       kx
          [0,N-1]       lx
this can be tested in the problem:
     http://www.spoj.com/problems/ILKQUERY/
```

```
*/

struct wavelet {
  vector<int> values, ori;
  vector<int> map_left, map_right;
  int l, r, m;
  wavelet *left, *right;
  wavelet() : left(NULL), right(NULL) {}
  wavelet(int a, int b, int c) : l(a), r(b), m(c), left(NULL),
       right(NULL) {}
};

wavelet *init(vector<int> &data, vector<int> &ind, int lo, int
     hi) {
  if (lo > hi || (data.size() == 0)) return NULL;
  int mid = ((long long)(lo) + hi) / 2;
  if (lo + 1 == hi) mid = lo; // handle negative values

  wavelet *node = new wavelet(lo, hi, mid);

  vector<int> data_l, data_r, ind_l, ind_r;
  int ls = 0, rs = 0;
  for (int i = 0; i < int(data.size()); i++) {
    int value = data[i];
    if (value <= mid) {
      data_l.emplace_back(value);
      ind_l.emplace_back(ind[i]);
      ls++;
    } else {
      data_r.emplace_back(value);
      ind_r.emplace_back(ind[i]);
      rs++;
    }
    node->map_left.emplace_back(ls);
    node->map_right.emplace_back(rs);
    node->values.emplace_back(value);
    node->ori.emplace_back(ind[i]);
  }

  if (lo < hi) {
    node->left = init(data_l, ind_l, lo, mid);
    node->right = init(data_r, ind_r, mid + 1, hi);
  }
  return node;
}

int kth(wavelet *node, int to, int k) {
  // returns the kth element in the sorted version of (a[0],
       ..., a[to])
  if (node->l == node->r) return node->m;
  int c = node->map_left[to];
  if (k < c)
    return kth(node->left, c - 1, k);
  return kth(node->right, node->map_right[to] - 1, k - c);
}

int pos_kth_ocurrence(wavelet *node, int val, int k) {
  // returns the position on the original array of the kth
       ocurrence of the value "val"
  if (!node) return -1;
  if (node->l == node->r) {
    if (int(node->ori.size()) <= k)
      return -1;
    return node->ori[k];
  }

  if (val <= node->m)
    return pos_kth_ocurrence(node->left, val, k);
  return pos_kth_ocurrence(node->right, val, k);
}
```

3 DP Optimizations

3.1 convex hull trick

```
/**
 *  Problems:
 *    http://codeforces.com/problemset/problem/319/C
 *    http://codeforces.com/contest/311/problem/B
 *    https://csacademy.com/contest/archive/task/squared-ends
 *    http://codeforces.com/contest/932/problem/F
 * */

struct line {
  long long m, b;
  line (long long a, long long c) : m(a), b(c) {}
  long long eval(long long x) {
    return m * x + b;
  }
};

long double inter(line a, line b) {
  long double den = a.m - b.m;
  long double num = b.b - a.b;
  return num / den;
}

/**
 *  min m_i * x_j + b_i, for all i.
 *     x_j <= x_{j + 1}
 *     m_i >= m_{j + 1}
 * */
struct ordered_cht {
  vector<line> ch;
  int idx; // id of last "best" in query
  ordered_cht() {
    idx = 0;
  }

  void insert_line(long long m, long long b) {
    line cur(m, b);
    // new line's slope is less than all the previous
    while (ch.size() > 1 &&
       (inter(cur, ch[ch.size() - 2]) >= inter(cur,
            ch[ch.size() - 1]))) {
      // f(x) is better in interval [inter(ch.back(), cur),
           inf)
      ch.pop_back();
    }

    ch.push_back(cur);
  }

  long long eval(long long x) { // minimum
    // current x is greater than all the previous x,
    // if that is not the case we can make binary search.
    idx = min<int>(idx, ch.size() - 1);
    while (idx + 1 < (int)ch.size() && ch[idx + 1].eval(x) <=
         ch[idx].eval(x))
      idx++;

    return ch[idx].eval(x);
  }
};


/**
 *  Dynammic convex hull trick
 * */

typedef long long int64;
typedef long double float128;

const int64 is_query = -(1LL<<62), inf = 1e18;

struct Line {
  int64 m, b;
  mutable function<const Line*()> succ;
  bool operator<(const Line& rhs) const {
    if (rhs.b != is_query) return m < rhs.m;
    const Line* s = succ();
    if (!s) return 0;
    int64 x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};

struct HullDynamic : public multiset<Line> { // will maintain
     upper hull for maximum
  bool bad(iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if (z == end()) return y->m == x->m && y->b <= x->b;
    return (float128)(x->b - y->b)*(z->m - y->m) >=
         (float128)(y->b - z->b)*(y->m - x->m);
  }
  void insert_line(int64 m, int64 b) {
    auto y = insert({ m, b });
    y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
    if (bad(y)) { erase(y); return; }
    while (next(y) != end() && bad(next(y))) erase(next(y));
```

```
    while (y != begin() && bad(prev(y))) erase(prev(y));
  }

  int64 eval(int64 x) {
    auto l = *lower_bound((Line) { x, is_query });
    return l.m * x + l.b;
  }
};
```

3.2 divide and conquer

```
/*
 * recurrence:
 *   dp[k][i] = min dp[k-1][j] + c[i][j - 1], for all j > i;
 *
 * "comp" computes dp[k][i] for all i in O(n log n) (k is fixed)
*/

void comp(int l, int r, int le, int re) {
  if (l > r) return;

  int mid = (l + r) >> 1;

  int best = max(mid + 1, le);
  dp[cur][mid] = dp[cur ^ 1][best] + cost(mid, best - 1);
  for (int i = best; i <= re; i++) {
    if (dp[cur][mid] > dp[cur ^ 1][i] + cost(mid, i - 1)) {
      best = i;
      dp[cur][mid] = dp[cur ^ 1][i] + cost(mid, i - 1);
    }
  }

  comp(l, mid - 1, le, best);
  comp(mid + 1, r, best, re);
}
```

4 Else

4.1 Mo's algorithm on trees

```
/**
problems:
    - https://codeforces.com/gym/101161 problem E
*/
void flat(vector<vector<edge>> &g, vector<int> &a,
    vector<int> &le, vector<int> &ri, vector<int> &cost,
    int node, int pi, int &ts, int w) {

  cost[node] = w;
  le[node] = ts;
  a[ts] = node;
  ts++;
  for (auto e : g[node]) {
    if (e.to == pi) continue;
    flat(g, a, le, ri, cost, e.to, node, ts, e.w);
  }
  ri[node] = ts;
  a[ts] = node;
  ts++;
}

/**
 * Case when the cost is in the edges.
 * */
void compute_queries(vector<vector<edge>> &g) {
  // g is undirected
  int n = g.size();

  lca_tree.init(g, 0);

  vector<int> a(2 * n), le(n), ri(n), cost(n);
  // a: nodes in the flatten array
  // le: left id of the given node
  // ri: right id of the given node
  // cost: cost of the edge from the node to the parent

  int ts = 0; // timestamp
  flat(g, a, le, ri, cost, 0, -1, ts, 0);

  int q; cin >> q;
  vector<query> queries(q);
  for (int i = 0; i < q; i++) {
    int u, v;
    cin >> u >> v;
    u--; v--;
    int lca = lca_tree.query(u, v);
    if (le[u] > le[v])
      swap(u, v);
    queries[i].id = i;
    queries[i].lca = lca;
    queries[i].u = u;
    queries[i].v = v;
    if (lca == u) {
      queries[i].a = le[u] + 1;
      queries[i].b = le[v];
    } else {
      queries[i].a = ri[u];
      queries[i].b = le[v];
    }
  }
  solve_mo(queries, a, le, cost); // this is the usal algorithm
}
```

4.2 Mo's algorithm

```
const int MN = 5 * 100000 + 1;
const int SN = 708;

struct Query {
  int a, b, id;
  Query() {}
  Query(int x, int y, int i) : a(x), b(y), id(i) {}

  bool operator<(const Query &o) const {
    if (a / SN != o.a / SN) return a < o.a;
    return a / SN & 1 ? b < o.b : b > o.b;
  }
};

struct DS {
  DS() : {}

  void Insert(int x) {}

  void Erase(int x) {}

  long long Query() {}
};

Query s[MN];
int ans[MN];
DS active;

int main() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (auto &i : a) cin >> i;

  int q;
  cin >> q;
  for (int i = 0; i < q; ++i) {
    int b, e;
    cin >> b >> e;
    b--;
    e--;
    s[i] = Query(b, e, i);
  }

  sort(s, s + q);

  int i = 0;
  int j = -1;
  for (int k = 0; k < (int)q; ++k) {
    int L = s[k].a;
    int R = s[k].b;

    while (j < R) active.Insert(a[++j]);
    while (j > R) active.Erase(a[j--]);
    while (i < L) active.Erase(a[i++]);
    while (i > L) active.Insert(a[--i]);

    ans[s[k].id] = active.Query();
  }

  for (int i = 0; i < q; ++i) {
    cout << ans[i] << endl;
  }

  return 0;
```

```
};
```

5 Flow and Matching

5.1 BoundedFlow

```
struct BoundedFlow { // 0-base
  struct edge {
    int to, cap, flow, rev;
  };
  vector<edge> G[N];
  int n, s, t, dis[N], cur[N], cnt[N];
  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, int lcap, int rcap) {
    cnt[u] -= lcap, cnt[v] += lcap;
    G[u].pb(edge{v, rcap, lcap, SZ(G[v])});
    G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
  }
  void add_edge(int u, int v, int cap) {
    G[u].pb(edge{v, cap, 0, SZ(G[v])});
    G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
  }
  int dfs(int u, int cap) {
    if (u == t || !cap) return cap;
    for (int &i = cur[u]; i < SZ(G[u]); ++i) {
      edge &e = G[u][i];
      if (dis[e.to] == dis[u] + 1 && e.cap != e.flow) {
        int 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.empty()) {
      int u = q.front();
      q.pop();
      for (edge &e : G[u])
        if (!~dis[e.to] && e.flow != e.cap)
          q.push(e.to), dis[e.to] = dis[u] + 1;
    }
    return dis[t] != -1;
  }
  int maxflow(int _s, int _t) {
    s = _s, t = _t;
    int flow = 0, df;
    while (bfs()) {
      fill_n(cur, n + 3, 0);
      while ((df = dfs(s, INF))) flow += df;
    }
    return flow;
  }
  bool solve() {
    int 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;
  }
  int solve(int _s, int _t) {
    add_edge(_t, _s, INF);
    if (!solve()) return -1; // invalid flow
    int x = G[_t].back().flow;
    return G[_t].pop_back(), G[_s].pop_back(), x;
  }
};
```

5.2 Dinic

```
/* solve the number of vertex-disjoint routes */
vector <pii> edges; // edge array
vector <int> g[V]; // adjacency list (edge id)
vi ans;
int n;
bool vis[V];
int dep[V];
bool bfs() {
    CLR(vis, 0);
    queue <int> q;
    q.push(0);
    dep[0] = 0;
    vis[0] = 1;

    while (!q.empty()) {
        int i = q.front(); q.pop();
        for (int id : g[i]) {
            int j = edges[id].X;
            if (!vis[j] && edges[id].Y)
                vis[j] = 1, q.push(j), dep[j] = dep[i] + 1;
        }
    }

    return vis[n - 1];
}
int cur[V];
int dfs(int i, int a) {
    if (i == n - 1 || !a)
        return a;

    int f, flow = 0;
    for (int &id = cur[i]; id < g[i].size(); ++id) {
        int j = edges[g[i][id]].X;
        int &ff = edges[g[i][id]].Y;
        if (dep[i] + 1 == dep[j] && (f = dfs(j, min(a, ff)))) {
            ff -= f;
            edges[g[i][id] ^ 1].Y += f;
            a -= f;
            flow += f;
            if (!a)
                break;
        }
    }
    return flow;
}

int dinic() {
    int cnt = 0;
    while ( bfs() ) {
        CLR(cur, 0);
        cnt += dfs(0, INF);
    }
    return cnt;
}

bool dfs1(int i) {
    ans.pb(i);
    if (i == n - 1)
        return true;
    for (int id : g[i]) {
        if (!(id & 1) && !edges[id].Y) {
            edges[id].Y = 1; // make this edge unusable
            if (dfs1(edges[id].X))
                return true;
        }
    }
    return false;
}

signed main () {
    // input n, m
    for (int i = 0; i < m; ++i) {
        cin >> a >> b;
        g[a].pb(edges.size());
        edges.pb(pii(b, 1));
        g[b].pb(edges.size());
        edges.pb(pii(a, 0));
    }

    int cnt = dinic();
    while (cnt--) {
        ans.clear();
        dfs1(0);
        // print ans
    }
```

```
  return 0;
}
```

5.3 isap

```
/* Better than Dinic without BFS*/
struct Maxflow {
  static const int MAXV = 20010;
  static const int INF = 1000000;
  struct Edge {
    int v, c, r;
    Edge(int _v, int _c, int _r)
      : v(_v), c(_c), r(_r) {}
  };
  int s, t;
  vector<Edge> G[MAXV * 2];
  int iter[MAXV * 2], d[MAXV * 2], gap[MAXV * 2], tot;
  void init(int x) {
    tot = x + 2;
    s = x + 1, t = x + 2;
    for (int i = 0; i <= tot; i++) {
      G[i].clear();
      iter[i] = d[i] = gap[i] = 0;
    }
  }
  void addEdge(int u, int v, int c) {
    G[u].push_back(Edge(v, c, SZ(G[v])));
    G[v].push_back(Edge(u, 0, SZ(G[u]) - 1));
  }
  int dfs(int p, int flow) {
    if (p == t) return flow;
    for (int &i = iter[p]; i < SZ(G[p]); i++) {
      Edge &e = G[p][i];
      if (e.c > 0 && d[p] == d[e.v] + 1) {
        int f = dfs(e.v, min(flow, e.c));
        if (f) {
          e.c -= f;
          G[e.v][e.r].c += f;
          return f;
        }
      }
    }
    if ((--gap[d[p]]) == 0) d[s] = tot;
    else {
      d[p]++;
      iter[p] = 0;
      ++gap[d[p]];
    }
    return 0;
  }
  int solve() {
    int res = 0;
    gap[0] = tot;
    for (res = 0; d[s] < tot; res += dfs(s, INF))
      ;
    return res;
  }
} flow;
```

5.4 Maximum Simple Graph Matching

```
struct GenMatch { // 1-base
  int V, pr[N];
  bool el[N][N], inq[N], inp[N], inb[N];
  int st, ed, nb, bk[N], djs[N], ans;
  void init(int _V) {
    V = _V;
    for (int i = 0; i <= V; ++i) {
      for (int j = 0; j <= V; ++j) el[i][j] = 0;
      pr[i] = bk[i] = djs[i] = 0;
      inq[i] = inp[i] = inb[i] = 0;
    }
  }
  void add_edge(int u, int v) {
    el[u][v] = el[v][u] = 1;
  }
  int lca(int u, int v) {
    fill_n(inp, V + 1, 0);
    while (1)
      if (u = djs[u], inp[u] = true, u == st) break;
      else u = bk[pr[u]];
    while (1)
      if (v = djs[v], inp[v]) return v;
      else v = bk[pr[v]];
    return v;
  }
  void upd(int u) {
    for (int v; djs[u] != nb;) {
      v = pr[u], inb[djs[u]] = inb[djs[v]] = true;
      u = bk[v];
      if (djs[u] != nb) bk[u] = v;
    }
  }
  void blo(int u, int v, queue<int> &qe) {
    nb = lca(u, v), fill_n(inb, V + 1, 0);
    upd(u), upd(v);
    if (djs[u] != nb) bk[u] = v;
    if (djs[v] != nb) bk[v] = u;
    for (int tu = 1; tu <= V; ++tu)
      if (inb[djs[tu]])
        if (djs[tu] = nb, !inq[tu])
          qe.push(tu), inq[tu] = 1;
  }
  void flow() {
    fill_n(inq + 1, V, 0), fill_n(bk + 1, V, 0);
    iota(djs + 1, djs + V + 1, 1);
    queue<int> qe;
    qe.push(st), inq[st] = 1, ed = 0;
    while (!qe.empty()) {
      int u = qe.front();
      qe.pop();
      for (int v = 1; v <= V; ++v)
        if (el[u][v] && djs[u] != djs[v] &&
          pr[u] != v) {
          if ((v == st) ||
            (pr[v] > 0 && bk[pr[v]] > 0))
            blo(u, v, qe);
          else if (!bk[v]) {
            if (bk[v] = u, pr[v] > 0) {
              if (!inq[pr[v]]) qe.push(pr[v]);
            } else return ed = v, void();
          }
        }
    }
  }
  void aug() {
    for (int u = ed, v, w; u > 0;)
      v = bk[u], w = pr[v], pr[v] = u, pr[u] = v,
      u = w;
  }
  int solve() {
    fill_n(pr, V + 1, 0), ans = 0;
    for (int u = 1; u <= V; ++u)
      if (!pr[u])
        if (st = u, flow(), ed > 0) aug(), ++ans;
    return ans;
  }
};
```

5.5 MincostMaxflow

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

5.6 Minimum Weight Matching

```
struct Graph { // 0-base (Perfect Match), n is even
  int n, match[N], onstk[N], stk[N], tp;
  ll edge[N][N], dis[N];
  void init(int _n) {
    n = _n, tp = 0;
    for (int i = 0; i < n; ++i) fill_n(edge[i], n, 0);
  }
  void add_edge(int u, int v, ll w) {
    edge[u][v] = edge[v][u] = w;
  }
  bool SPFA(int u) {
    stk[tp++] = u, onstk[u] = 1;
    for (int v = 0; v < n; ++v)
      if (!onstk[v] && match[u] != v) {
        int m = match[v];
        if (dis[m] >
          dis[u] - edge[v][m] + edge[u][v]) {
          dis[m] = dis[u] - edge[v][m] + edge[u][v];
          onstk[v] = 1, stk[tp++] = v;
          if (onstk[m] || SPFA(m)) return 1;
          --tp, onstk[v] = 0;
        }
      }
    onstk[u] = 0, --tp;
    return 0;
  }
  ll solve() { // find a match
    for (int i = 0; i < n; ++i) match[i] = i ^ 1;
    while (1) {
      int found = 0;
      fill_n(dis, n, 0);
      fill_n(onstk, n, 0);
      for (int i = 0; i < n; ++i)
        if (tp = 0, !onstk[i] && SPFA(i))
          for (found = 1; tp >= 2;) {
            int u = stk[--tp];
            int v = stk[--tp];
            match[u] = v, match[v] = u;
          }
      if (!found) break;
    }
    ll ret = 0;
    for (int i = 0; i < n; ++i)
      ret += edge[i][match[i]];
    return ret >> 1;
  }
};
```

6 Geometry

6.1 convexHull

```
void findConvexHull(int n) {
    sort(v, v + n);
    int u = 0, d = 0;
    for (int i = 0; i < n; ++i) {
        while (d >= 2 && dc[d - 2].cross(dc[d - 1], v[i]) < 0)
            --d, dc.pop_back();
        while (u >= 2 && uc[u - 2].cross(uc[u - 1], v[i]) > 0)
            --u, uc.pop_back();
        dc.pb(v[i]); ++d;
        uc.pb(v[i]); ++u;
    }
}
```

6.2 default code

```
#define forward(a, b) Vector(b.X - a.X, b.Y - a.Y)

using Point = pii;
using Vec = pii;

ll cross(const Vec& u, const Vec& v) {
    return u.X * v.Y - u.Y * v.X;
}

ll cross(const Point& O, const Point& A, const Point& B) {
    return cross(Vec(A.X - O.X, A.Y - O.Y), Vec(B.X - O.X, B.Y
        - O.Y));
}

ll dot(const Vec& u, const Vec& v) {
    return u.X * v.X + u.Y * v.Y;
}

ll dot(const Point& O, const Point& A, const Point& B) {
    return dot(Vec(A.X - O.X, A.Y - O.Y), Vec(B.X - O.X, B.Y -
        O.Y));
}
```

6.3 seg intersection

```
bool intersect(Point p, Point a, Point b) {
    return min(a.X, b.X) <= p.X && p.X <= max(a.X, b.X) &&
          min(a.Y, b.Y) <= p.Y && p.Y <= max(a.Y, b.Y);
}

bool intersect(Point a, Point b, Point c, Point d) {
    int c1 = cross(a, b, c);
    int c2 = cross(a, b, d);
    int c3 = cross(c, d, a);
    int c4 = cross(c, d, b);

    if (c1 && c2 && c3 && c4 && ((c1 < 0) ^ (c2 < 0)) && ((c3 <
         0) ^ (c4 < 0))) return true;

    if (!c1 && intersect(c, a, b)) return true;
    if (!c2 && intersect(d, a, b)) return true;
    if (!c3 && intersect(a, c, d)) return true;
    if (!c4 && intersect(b, c, d)) return true;
    return false;
}
```

6.4 windingNum

```
int n;
Poly poly;

bool checkPointOnSeg(const Point& p, const Point& A, const
     Point& B) {
    return cross(p, A, B) == 0 && dot(p, A, B) <= 0;
}

int boundaryCheck(const Point& p) {
    for (int i = 0; i < n; ++i)
        if (checkPointOnSeg(p, poly[i], poly[i + 1]))
            return cout << "BOUNDARY\n", 1;

    ll windingNumber = 0;
    for (int i = 0; i < n; ++i) {
        if (poly[i].Y <= p.Y) {
            if (poly[i + 1].Y > p.Y)
                if (cross(p, poly[i], poly[i + 1]) > 0)
                    ++windingNumber;
        }
        else {
            if (poly[i + 1].Y <= p.Y)
                if (cross(p, poly[i], poly[i + 1]) < 0)
                    --windingNumber;
        }
    }

    if (windingNumber)
        return cout << "INSIDE\n", 2;
    return cout << "OUTSIDE\n", 0;
}
```

7 Graphs

7.1 BCC Vertex

```
vector<int> G[N]; // 1-base
vector<int> nG[N], bcc[N];
int low[N], dfn[N], Time;
int bcc_id[N], bcc_cnt; // 1-base
bool is_cut[N]; // whether is av
bool cir[N];
int st[N], 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].push_back(t);
        } while (t != v);
        bcc_id[u] = bcc_cnt;
        bcc[bcc_cnt].pb(u);
      }
    } else if (dfn[v] < dfn[u] && v != pa)
      low[u] = min(low[u], dfn[v]);
  if (pa == -1 && 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 && !cir[i]; ++i)
    for (int j : bcc[i])
      if (is_cut[j])
        nG[i].pb(bcc_id[j]), nG[bcc_id[j]].pb(i);
}
```

7.2 bridges

```
struct Graph {
  vector<vector<Edge>> g;
  vector<int> vi, low, d, pi, is_b;
  int bridges_computed;
  int ticks, edges;

  Graph(int n, int m) {
    g.assign(n, vector<Edge>());
    is_b.assign(m, 0);
    vi.resize(n);
    low.resize(n);
    d.resize(n);
    pi.resize(n);
    edges = 0;
    bridges_computed = 0;
  }

  void AddEdge(int u, int v) {
    g[u].push_back(Edge(v, edges));
    g[v].push_back(Edge(u, edges));
    edges++;
  }

  void Dfs(int u) {
    vi[u] = true;
    d[u] = low[u] = ticks++;
    for (int i = 0; i < (int)g[u].size(); ++i) {
      int v = g[u][i].to;
      if (v == pi[u]) continue;
      if (!vi[v]) {
        pi[v] = u;
        Dfs(v);
        if (d[u] < low[v]) is_b[g[u][i].id] = true;

        low[u] = min(low[u], low[v]);
      } else {
        low[u] = min(low[u], d[v]);
      }
    }
  }

  // Multiple edges from a to b are not allowed.
  // (they could be detected as a bridge).
  // If you need to handle this, just count
  // how many edges there are from a to b.
  void CompBridges() {
    fill(pi.begin(), pi.end(), -1);
    fill(vi.begin(), vi.end(), 0);
    fill(low.begin(), low.end(), 0);
    fill(d.begin(), d.end(), 0);
    ticks = 0;
    for (int i = 0; i < (int)g.size(); ++i)
      if (!vi[i]) Dfs(i);
    bridges_computed = true;
  }

  map<int, vector<Edge>> BridgesTree() {
    if (!bridges_computed) CompBridges();
    int n = g.size();
    Dsu dsu(g.size());
    for (int i = 0; i < n; i++)
      for (auto e : g[i])
        if (!is_b[e.id]) dsu.Join(i, e.to);

    map<int, vector<Edge>> tree;
    for (int i = 0; i < n; i++)
      for (auto e : g[i])
        if (is_b[e.id])
          tree[dsu.Find(i)].emplace_back(dsu.Find(e.to), e.id);

    return tree;
  }
};
```

7.3 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;
 }
};
```

7.4 directed mst

```
const int inf = 1000000 + 10;

struct edge {
  int u, v, w;
  edge() {}
  edge(int a,int b,int c) : u(a), v(b), w(c) {}
};

/**
 * Computes the minimum spanning tree for a directed graph
 * - edges : Graph description in the form of list of edges.
 *    each edge is: From node u to node v with cost w
 * - root : Id of the node to start the DMST.
 * - n    : Number of nodes in the graph.
 * */

int dmst(vector<edge> &edges, int root, int n) {
  int ans = 0;
  int cur_nodes = n;
  while (true) {
    vector<int> lo(cur_nodes, inf), pi(cur_nodes, inf);
    for (int i = 0; i < edges.size(); ++i) {
      int u = edges[i].u, v = edges[i].v, w = edges[i].w;
      if (w < lo[v] and u != v) {
        lo[v] = w;
        pi[v] = u;
      }
    }

    lo[root] = 0;
    for (int i = 0; i < lo.size(); ++i) {
      if (i == root) continue;
      if (lo[i] == inf) return -1;
    }
    int cur_id = 0;
    vector<int> id(cur_nodes, -1), mark(cur_nodes, -1);
    for (int i = 0; i < cur_nodes; ++i) {
      ans += lo[i];
      int u = i;
      while (u != root and id[u] < 0 and mark[u] != i) {
        mark[u] = i;
        u = pi[u];
      }
      if (u != root and id[u] < 0) { // Cycle
        for (int v = pi[u]; v != u; v = pi[v])
          id[v] = cur_id;
        id[u] = cur_id++;
      }
    }

    if (cur_id == 0)
      break;

    for (int i = 0; i < cur_nodes; ++i)
      if (id[i] < 0) id[i] = cur_id++;

    for (int i = 0; i < edges.size(); ++i) {
      int u = edges[i].u, v = edges[i].v, w = edges[i].w;
      edges[i].u = id[u];
      edges[i].v = id[v];
      if (id[u] != id[v])
        edges[i].w -= lo[v];
    }
    cur_nodes = cur_id;
    root = id[root];
  }

  return ans;
}
```

7.5 karp min mean cycle

```
/**
 * Finds the min mean cycle, if you need the max mean cycle
 * just add all the edges with negative cost and print
 * ans * -1
 *
 * test: uva, 11090 - Going in Cycle!!
 * */

const int MN = 1000;
struct edge{
  int v;
  long long w;
  edge(){} edge(int v, int w) : v(v), w(w) {}
};

long long d[MN][MN];
// This is a copy of g because increments the size
// pass as reference if this does not matter.
int karp(vector<vector<edge> > g) {
  int n = g.size();

  g.resize(n + 1); // this is important

  for (int i = 0; i < n; ++i)
    if (!g[i].empty())
      g[n].push_back(edge(i,0));
  ++n;

  for(int i = 0;i<n;++i)
    fill(d[i],d[i]+(n+1),INT_MAX);

  d[n - 1][0] = 0;

  for (int k = 1; k <= n; ++k) for (int u = 0; u < n; ++u) {
    if (d[u][k - 1] == INT_MAX) continue;
    for (int i = g[u].size() - 1; i >= 0; --i)
      d[g[u][i].v][k] = min(d[g[u][i].v][k], d[u][k - 1] +
            g[u][i].w);
  }

  bool flag = true;

  for (int i = 0; i < n && flag; ++i)
    if (d[i][n] != INT_MAX)
      flag = false;

  if (flag) {
    return true; // return true if there is no a cycle.
  }

  double ans = 1e15;

  for (int u = 0; u + 1 < n; ++u) {
    if (d[u][n] == INT_MAX) continue;
    double W = -1e15;

    for (int k = 0; k < n; ++k)
      if (d[u][k] != INT_MAX)
        W = max(W, (double)(d[u][n] - d[u][k]) / (n - k));

    ans = min(ans, W);
  }

  // printf("%.2lf\n", ans);
  cout << fixed << setprecision(2) << ans << endl;

  return false;
}
```

7.6 KDTree

```
namespace kdt {
int root, lc[maxn], rc[maxn], xl[maxn], xr[maxn],
  yl[maxn], yr[maxn];
point p[maxn];
int build(int l, int r, int dep = 0) {
  if (l == r) return -1;
  function<bool(const point &, const point &)> f =
    [dep](const point &a, const point &b) {
      if (dep & 1) return a.x < b.x;
      else return a.y < b.y;
    };
  int m = (l + r) >> 1;
  nth_element(p + l, p + m, p + r, f);
  xl[m] = xr[m] = p[m].x;
  yl[m] = yr[m] = p[m].y;
  lc[m] = build(l, m, dep + 1);
  if (~lc[m]) {
    xl[m] = min(xl[m], xl[lc[m]]);
    xr[m] = max(xr[m], xr[lc[m]]);
    yl[m] = min(yl[m], yl[lc[m]]);
    yr[m] = max(yr[m], yr[lc[m]]);
  }
  rc[m] = build(m + 1, r, dep + 1);
  if (~rc[m]) {
    xl[m] = min(xl[m], xl[rc[m]]);
    xr[m] = max(xr[m], xr[rc[m]]);
    yl[m] = min(yl[m], yl[rc[m]]);
    yr[m] = max(yr[m], yr[rc[m]]);
  }
  return m;
}
bool bound(const point &q, int o, long long d) {
  double ds = sqrt(d + 1.0);
  if (q.x < xl[o] - ds || q.x > xr[o] + ds ||
    q.y < yl[o] - ds || q.y > yr[o] + ds)
    return false;
  return true;
}
long long dist(const point &a, const point &b) {
  return (a.x - b.x) * 1ll * (a.x - b.x) +
    (a.y - b.y) * 1ll * (a.y - b.y);
}
void dfs(
  const point &q, long long &d, int o, int dep = 0) {
  if (!bound(q, o, d)) return;
  long long cd = dist(p[o], q);
  if (cd != 0) d = min(d, cd);
  if ((dep & 1) && q.x < p[o].x ||
    !(dep & 1) && q.y < p[o].y) {
    if (~lc[o]) dfs(q, d, lc[o], dep + 1);
    if (~rc[o]) dfs(q, d, rc[o], dep + 1);
  } else {
    if (~rc[o]) dfs(q, d, rc[o], dep + 1);
    if (~lc[o]) dfs(q, d, lc[o], dep + 1);
  }
}
void init(const vector<point> &v) {
  for (int i = 0; i < v.size(); ++i) p[i] = v[i];
  root = build(0, v.size());
}
long long nearest(const point &q) {
  long long res = 1e18;
  dfs(q, res, root);
  return res;
}
} // namespace kdt
```

7.7 konig's theorem

In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover

7.8 minimum path cover in DAG

Given a directed acyclic graph $G = (V, E)$, we are to find the minimum number of vertex-disjoint paths to cover each vertex in V.
We can construct a bipartite graph $G' = (Vout \cup Vin, E')$ from G, where :

$$Vout = \{v \in V : v \ has \ positive \ out - degree\}$$

$$Vin = \{v \in V : v \ has \ positive \ in - degree\}$$

$$E' = \{(u, v) \in Vout \times Vin : (u, v) \in E\}$$

Then it can be shown, via König's theorem, that G' has a matching of size m if and only if there exists $n - m$ vertex-disjoint paths that cover each vertex in G, where n is the number of vertices in G and m is the maximum cardinality bipartite mathching in G'.
Therefore, the problem can be solved by finding the maximum cardinality matching in G' instead.
NOTE: If the paths are note necesarily disjoints, find the transitive closure and solve the problem for disjoint paths.

7.9 SCC kosaraju + 2SAT

```
struct SCC {
  int n, m;
  vi g[V], gt[V], gg[V], v;

  vvi c;
  int comp, id[V], in[V];

  void input() {
    cin >> m >> n;
    int i, j; char a, b;
    while (m--) {
      cin >> a >> i >> b >> j; --i, --j;
      i = (i << 1) | (a == '+'), j = (j << 1) | (b == '+');
      g[i ^ 1].pb(j), gt[j].pb(i ^ 1), g[j ^ 1].pb(i),
          gt[i].pb(j ^ 1);
    }
  }

  SCC() {
    input();
    for (int i = 0; i < n; ++i) v.pb(i);

    for (int i : v) if (!vis[i]) d0(i);
    sort(ALL(v), 0);

    CLR(vis, 0);
    for (int i : v) if (!vis[i]) d1(i), ++comp;

    for (int i = 0; i < n; ++i)
      for (int j : g[i])
        if (id[i] != id[j])
          gg[id[i]].pb(id[j]), ++in[id[j]];

    solve();
  }

  bool vis[V];
  int ans[V];
  int solve() {
    for (int i = 0; i < n; ++i)
      if (id[i << 1] == id[(i << 1) | 1])
        return cout << "IMPOSSIBLE\n", 0;

    queue <int> q; stack <int> s;
    for (int i = 0; i < comp; ++i) if (!in[i]) q.push(i);

    while (!q.empty()) {
      int i = q.front(); q.pop();
      s.push(i);
      for (int j : gg[i]) {
        --in[j]; if (!in[j]) q.push(j);
      }
    }

    CLR(vis, 0);
    while (!s.empty()) {
      int i = s.top(); s.pop();
      bool flag = 0;
      for (int x : c[i])
        if (vis[x >> 1]) { flag = 1; break; }

      if (flag) continue;

      for (int x : c[i]) {
        vis[x >> 1] = 1;
        ans[x >> 1] = (x & 1);
      }
    }
    for (int i = 0; i < n; ++i) cout << "-+"[ans[i]] << ' ';
    return 1;
```

```
  }

  /* old graph */
  int t, out[V];
  void d0(int x) {
    vis[x] = 1;
    for (int y : gt[x]) if (!vis[y]) d0(y);
    out[x] = ++t;
  }

  bool O(int x, int y) {
    return out[x] > out[y];
  }

  /* new graph */
  void d1(int x) {
    vis[x] = 1;
    id[x] = comp;
    c[comp].pb(x);
    for (int y : g[x]) if (!vis[y]) d1(y);
  }
};
```

7.10 Smart Pointer

```
#ifndef REFERENCE_POINTER
#define REFERENCE_POINTER
template <typename T> struct _RefCounter {
  T data;
  int ref;
  _RefCounter(const T &d = 0) : data(d), ref(0) {}
};
template <typename T> struct reference_pointer {
  _RefCounter<T> *p;
  T *operator->() { return &p->data; }
  T &operator*() { return p->data; }
  operator _RefCounter<T> *() { return p; }
  reference_pointer &operator=(
    const reference_pointer &t) {
    if (p && !--p->ref) delete p;
    p = t.p;
    p && ++p->ref;
    return *this;
  }
  reference_pointer(_RefCounter<T> *t = 0) : p(t) {
    p && ++p->ref;
  }
  reference_pointer(const reference_pointer &t)
    : p(t.p) {
    p && ++p->ref;
  }
  ~reference_pointer() {
    if (p && !--p->ref) delete p;
  }
};
template <typename T>
inline reference_pointer<T> new_reference(
  const T &nd) {
  return reference_pointer<T>(new _RefCounter<T>(nd));
}
#endif
// note:
reference_pointer<int> a;
a = new_reference(5);
a = new_reference<int>(5);
a = new_reference((int)5);
reference_pointer<int> b = a;

struct P {
  int a, b;
  P(int _a, int _b) : a(_a), b(_b) {}
} p(2, 3);
reference_pointer<P> a;
c = new_reference(P(1, 2));
c = new_reference<P>(P(1, 2));
c = new_reference(p);
```

7.11 tarjan scc

```
const int MN = 20002;
struct tarjan_scc {
  int scc[MN], low[MN], d[MN], stacked[MN];
  int ticks, current_scc;
  deque<int> s; // used as stack.

  tarjan_scc() {}

  void init () {
    memset(scc, -1, sizeof scc);
    memset(d, -1, sizeof d);
    memset(stacked, 0, sizeof stacked);
    s.clear();
    ticks = current_scc = 0;
  }

  void compute(vector<vector<int> > &g, int u) {
    d[u] = low[u] = ticks++;
    s.push_back(u);
    stacked[u] = true;
    for (int i = 0; i < g[u].size(); ++i) {
      int v = g[u][i];
      if (d[v] == -1)
        compute(g, v);
      if (stacked[v]) {
        low[u] = min(low[u], low[v]);
      }
    }

    if (d[u] == low[u]) { // root
      int v;
      do {
        v = s.back();s.pop_back();
        stacked[v] = false;
        scc[v] = current_scc;
      } while (u != v);
      current_scc++;
    }
  }
};
```

8 Math

8.1 Big number

```
template<typename T>
inline string to_string(const T& x){
  stringstream ss;
  return ss<<x,ss.str();
}
struct bigN:vector<ll>{
  const static int base=1000000000,width=log10(base);
  bool negative;
  bigN(const_iterator a,const_iterator b):vector<ll>(a,b){}
  bigN(string s){
    if(s.empty())return;
    if(s[0]=='-')negative=1,s=s.substr(1);
    else negative=0;
    for(int i=int(s.size())-1;i>=0;i-=width){
      ll t=0;
      for(int j=max(0,i-width+1);j<=i;++j)
        t=t*10+s[j]-'0';
      push_back(t);
    }
    trim();
  }
  template<typename T>
    bigN(const T &x):bigN(to_string(x)){}
  bigN():negative(0){}
  void trim(){
    while(size()&&!back())pop_back();
    if(empty())negative=0;
  }
  void carry(int _base=base){
    for(size_t i=0;i<size();++i){
      if(at(i)>=0&&at(i)<_base)continue;
      if(i+1u==size())push_back(0);
      int r=at(i)%_base;
      if(r<0)r+=_base;
      at(i+1)+=(at(i)-r)/_base,at(i)=r;
    }
  }
  int abscmp(const bigN &b)const{
    if(size()>b.size())return 1;
    if(size()<b.size())return -1;
    for(int i=int(size())-1;i>=0;--i){
      if(at(i)>b[i])return 1;
      if(at(i)<b[i])return -1;
    }
    return 0;
  }
```

```
int cmp(const bigN &b)const{
  if(negative!=b.negative)return negative?-1:1;
  return negative?-abscmp(b):abscmp(b);
}
bool operator<(const bigN&b)const{return cmp(b)<0;}
bool operator>(const bigN&b)const{return cmp(b)>0;}
bool operator<=(const bigN&b)const{return cmp(b)<=0;}
bool operator>=(const bigN&b)const{return cmp(b)>=0;}
bool operator==(const bigN&b)const{return !cmp(b);}
bool operator!=(const bigN&b)const{return cmp(b)!=0;}
bigN abs()const{
  bigN res=*this;
  return res.negative=0, res;
}
bigN operator-()const{
  bigN res=*this;
  return res.negative=!negative,res.trim(),res;
}
bigN operator+(const bigN &b)const{
  if(negative)return -(-(*this)+(-b));
  if(b.negative)return *this-(-b);
  bigN res=*this;
  if(b.size()>size())res.resize(b.size());
  for(size_t i=0;i<b.size();++i)res[i]+=b[i];
  return res.carry(),res.trim(),res;
}
bigN operator-(const bigN &b)const{
  if(negative)return -(-(*this)-(-b));
  if(b.negative)return *this+(-b);
  if(abscmp(b)<0)return -(b-(*this));
  bigN res=*this;
  if(b.size()>size())res.resize(b.size());
  for(size_t i=0;i<b.size();++i)res[i]-=b[i];
  return res.carry(),res.trim(),res;
}
bigN operator*(const bigN &b)const{
  bigN res;
  res.negative=negative!=b.negative;
  res.resize(size()+b.size());
  for(size_t i=0;i<size();++i)
    for(size_t j=0;j<b.size();++j)
      if((res[i+j]+=at(i)*b[j])>=base){
        res[i+j+1]+=res[i+j]/base;
        res[i+j]%=base;
      }
  return res.trim(),res;
}
bigN operator/(const bigN &b)const{
  int norm=base/(b.back()+1);
  bigN x=abs()*norm;
  bigN y=b.abs()*norm;
  bigN q,r;
  q.resize(x.size());
  for(int i=int(x.size())-1;i>=0;--i){
    r=r*base+x[i];
    int s1=r.size()<=y.size()?0:r[y.size()];
    int s2=r.size()<y.size()?0:r[y.size()-1];
    int d=(ll(base)*s1+s2)/y.back();
    r=r-y*d;
    while(r.negative)r=r+y,--d;
    q[i]=d;
  }
  q.negative=negative!=b.negative;
  return q.trim(),q;
}
bigN operator%(const bigN &b)const{
  return *this-(*this/b)*b;
}
friend istream& operator>>(istream &ss,bigN &b){
  string s;
  return ss>>s, b=s, ss;
}
friend ostream& operator<<(ostream &ss,const bigN &b){
  if(b.negative)ss<<'-';
  ss<<(b.empty()?0:b.back());
  for(int i=int(b.size())-2;i>=0;--i)
    ss<<setw(width)<<setfill('0')<<b[i];
  return ss;
}
template<typename T>
  operator T(){
    stringstream ss;
    ss<<*this;
    T res;
    return ss>>res,res;
  }
};
```

8.2 Determinant

```
template<class T>
T det()
{
  T x=1;
  for(int i=0;i<n;++i)
  {
    int mxl=i;
    for(int j=i+1;j<n;++j)
      if(M[j][i]>M[mxl][i]) mxl=j;
    if(mxl!=i)
      M[i].swap(M[mxl]),x*=-1;
    if(M[i][i] == 0) return 0;
    for(int j=i+1;j<n;++j)
    {
      T tmp=-M[j][i]/M[i][i];
      for(int k=i;k<m;++k)
        M[j][k]+=tmp*M[i][k];
    }
  }
  for(int i=0;i<n;++i)
    x=x*M[i][i];
  return x;
}
```

8.3 Fast Fourier Transform

```
template<int MAXN>
struct FFT {
  using c_t = complex<double>;
  const double PI = acos(-1);
  c_t w[MAXN];
  FFT() {
    double arg = 2 * PI / MAXN;
              c_t delta = c_t(cos(arg), sin(arg));
    w[0] = 1;
    for (int i = 1; i < MAXN; ++i) {
      w[i] = w[i-1] * delta;
    }
  }
  void fft(c_t *a, int n, bool inv=false); // see NTT
  // remember to replace ll with c_t
};
```

8.4 Fast Walsh Transform

```
/* x: a[j], y: a[j + (L >> 1)]
or: (y += x * op), and: (x += y * op)
xor: (x, y = (x + y) * op, (x - y) * op)
invop: or, and, xor = -1, -1, 1/2 */
void fwt(int *a, int n, int op) { //or
    for (int L = 2; L <= n; L <<= 1)
        for (int i = 0; i < n; i += L)
            for (int j = i; j < i + (L >> 1); ++j)
                a[j + (L >> 1)] += a[j] * op;
}
const int N = 21;
int f[N][1 << N], g[N][1 << N], h[N][1 << N], ct[1 << N];
void subset_convolution(int *a, int *b, int *c, int L) {
    // c_k = \sum_{i | j = k, i & j = 0} a_i * b_j
    int n = 1 << L;
    for (int i = 1; i < n; ++i)
        ct[i] = ct[i & (i - 1)] + 1;
    for (int i = 0; i < n; ++i)
        f[ct[i]][i] = a[i], g[ct[i]][i] = b[i];
    for (int i = 0; i <= L; ++i)
        fwt(f[i], n, 1), fwt(g[i], n, 1);
    for (int i = 0; i <= L; ++i)
        for (int j = 0; j <= i; ++j)
            for (int x = 0; x < n; ++x)
                h[i][x] += f[j][x] * g[i - j][x];
    for (int i = 0; i <= L; ++i)
        fwt(h[i], n, -1);
    for (int i = 0; i < n; ++i)
        c[i] = h[ct[i]][i];
}
```

8.5 Fraction

```
struct fraction{
  ll n,d;
  fraction(const ll &_n=0,const ll &_d=1):n(_n),d(_d){
    ll t=gcd(n,d);
    n/=t,d/=t;
    if(d<0) n=-n,d=-d;
  }
  fraction operator-()const{
    return fraction(-n,d);
  }
  fraction operator+(const fraction &b)const{
    return fraction(n*b.d+b.n*d,d*b.d);
  }
  fraction operator-(const fraction &b)const{
    return fraction(n*b.d-b.n*d,d*b.d);
  }
  fraction operator*(const fraction &b)const{
    return fraction(n*b.n,d*b.d);
  }
  fraction operator/(const fraction &b)const{
    return fraction(n*b.d,d*b.n);
  }
  bool operator<(const fraction &b)const{
     return n*b.d<b.n*d;
  }
  void print(){
    cout << n;
    if(d!=1) cout << "/" << d;
  }
};
```

8.6 Number Theory Transform

```
//(2^16)+1, 65537, 3
//7*17*(2^23)+1, 998244353, 3
//1255*(2^20)+1, 1315962881, 3
//51*(2^25)+1, 1711276033, 29
template<int MAXN, LL P, LL RT> //MAXN must be 2^k
struct NTT {
  LL w[MAXN];
  LL mpow(LL a, LL n);
  LL minv(LL a) { return mpow(a, P - 2); }
  NTT() {
    LL dw = mpow(RT, (P - 1) / MAXN);
    w[0] = 1;
    for (int i = 1; i < MAXN; ++i) w[i] = w[i - 1] * dw % P;
  }
  void bitrev(LL *a, int n) {
    for (int i = 1, j = 0; i < n; ++i) {
                      for (int l = n >> 1; (j ^= l) < l; l >>=
                           1);
                      if (i > j) swap(a[i], a[j]);
              }
  }
  void operator()(LL *a, int n, bool inv = false) { //0 <= a[i]
       < P
    bitrev(a, n);
    for (int L = 2; L <= n; L <<= 1) { // block size
      int dx = MAXN / L, dl = L >> 1;
      for (int i = 0; i < n; i += L) { // block start
        for (int j = i, x = 0; j < i + dl; ++j, x += dx) {
          LL z = a[j + dl] * w[x] % P;
          if ((a[j + dl] = a[j] - z) < 0) a[j + dl] += P;
          if ((a[j] += z) >= P) a[j] -= P;
        }
      }
    }
    if (inv) {
      reverse(a + 1, a + n);
      LL invn = minv(n);
      for (int i = 0; i < n; ++i) a[i] = a[i] * invn % P;
    }
  }
};
```

8.7 Polynomial Operation

```
#define fi(s, n) for (int i = (int)(s); i < (int)(n); ++i)
template<int MAXN, LL P, LL RT> // MAXN = 2^k
struct Poly : vector<LL> { // coefficients in [0, P)
  using vector<LL>::vector;
  static NTT<MAXN, P, RT> ntt;
  int n() const { return (int)size(); } // n() >= 1
  Poly(const Poly &p, int _n) : vector<LL>(_n) {
    copy_n(p.data(), min(p.n(), _n), data());
  }
  Poly& irev() { return reverse(data(), data() + n()), *this; }
  Poly& isz(int _n) { return resize(_n), *this; }
  Poly& iadd(const Poly &rhs) { // n() == rhs.n()
    fi(0, n()) if (((*this)[i] += rhs[i]) >= P) (*this)[i] -= P;
    return *this;
  }
  Poly& imul(LL k) {
    fi(0, n()) (*this)[i] = (*this)[i] * k % P;
    return *this;
  }
  Poly Mul(const Poly &rhs) const {
    int _n = 1;
    while (_n < n() + rhs.n() - 1) _n <<= 1;
    Poly X(*this, _n), Y(rhs, _n);
    ntt(X.data(), _n), ntt(Y.data(), _n);
    fi(0, _n) X[i] = X[i] * Y[i] % P;
    ntt(X.data(), _n, true);
    return X.isz(n() + rhs.n() - 1);
  }
  Poly Inv() const { // (*this)[0] != 0
    if (n() == 1) return {ntt.minv((*this)[0])};
    int _n = 1;
    while (_n < n() * 2) _n <<= 1;
    Poly Xi = Poly(*this, (n() + 1) / 2).Inv().isz(_n);
    Poly Y(*this, _n);
    ntt(Xi.data(), _n), ntt(Y.data(), _n);
    fi(0, _n) {
      Xi[i] *= (2 - Xi[i] * Y[i]) % P;
      if ((Xi[i] %= P) < 0) Xi[i] += P;
    }
    ntt(Xi.data(), _n, true);
    return Xi.isz(n());
  }
  Poly Sqrt() const { // Jacobi((*this)[0], P) = 1
    if (n() == 1) return {QuadraticResidue((*this)[0], P)};
    Poly X = Poly(*this, (n() + 1) / 2).Sqrt().isz(n());
    return X.iadd(Mul(X.Inv()).isz(n())).imul(P / 2 + 1);
  }
  pair<Poly, Poly> DivMod(const Poly &rhs) const { //
       (rhs.)back() != 0
    if (n() < rhs.n()) return {{0}, *this};
    const int _n = n() - rhs.n() + 1;
    Poly X(rhs); X.irev().isz(_n);
    Poly Y(*this); Y.irev().isz(_n);
    Poly Q = Y.Mul(X.Inv()).isz(_n).irev();
    X = rhs.Mul(Q), Y = *this;
    fi(0, n()) if ((Y[i] -= X[i]) < 0) Y[i] += P;
    return {Q, Y.isz(max(1, rhs.n() - 1))};
  }
  Poly Dx() const {
    Poly ret(n() - 1);
    fi(0, ret.n()) ret[i] = (i + 1) * (*this)[i + 1] % P;
    return ret.isz(max(1, ret.n()));
  }
  Poly Sx() const {
    Poly ret(n() + 1);
    fi(0, n()) ret[i + 1] = ntt.minv(i + 1) * (*this)[i] % P;
    return ret;
  }
  Poly _tmul(int nn, const Poly &rhs) const {
    Poly Y = Mul(rhs).isz(n() + nn - 1);
    return Poly(Y.data() + n() - 1, Y.data() + Y.n());
  }
  vector<LL> _eval(const vector<LL> &x, const vector<Poly> &up)
       const {
    const int _n = (int)x.size();
    if (!_n) return {};
    vector<Poly> down(_n * 2);
    down[1] = DivMod(up[1]).second;
    fi(2, _n * 2) down[i] = down[i / 2].DivMod(up[i]).second;
    /* down[1] =
         Poly(up[1]).irev().isz(n()).Inv().irev()._tmul(_n,
         *this);
    fi(2, _n * 2) down[i] = up[i ^ 1]._tmul(up[i].n() - 1,
         down[i / 2]); */
    vector<LL> y(_n);
    fi(0, _n) y[i] = down[_n + i][0];
    return y;
  }
  static vector<Poly> _tree1(const vector<LL> &x) {
    const int _n = (int)x.size();
    vector<Poly> up(_n * 2);
    fi(0, _n) up[_n + i] = {(x[i] ? P - x[i] : 0), 1};
    for (int i = _n - 1; i > 0; --i) up[i] = up[i * 2].Mul(up[i
         * 2 + 1]);
    return up;
  }
```

```
  vector<LL> Eval(const vector<LL> &x) const {
    auto up = _tree1(x); return _eval(x, up);
  }
  static Poly Interpolate(const vector<LL> &x, const vector<LL>
       &y) {
    const int _n = (int)x.size();
    vector<Poly> up = _tree1(x), down(_n * 2);
    vector<LL> z = up[1].Dx()._eval(x, up);
    fi(0, _n) z[i] = y[i] * ntt.minv(z[i]) % P;
    fi(0, _n) down[_n + i] = {z[i]};
    for (int i = _n - 1; i > 0; --i) down[i] = down[i *
         2].Mul(up[i * 2 + 1]).iadd(down[i * 2 + 1].Mul(up[i *
         2]));
    return down[1];
  }
  Poly Ln() const { // (*this)[0] == 1
    return Dx().Mul(Inv()).Sx().isz(n());
  }
  Poly Exp() const { // (*this)[0] == 0
    if (n() == 1) return {1};
    Poly X = Poly(*this, (n() + 1) / 2).Exp().isz(n());
    Poly Y = X.Ln(); Y[0] = P - 1;
    fi(0, n()) if ((Y[i] = (*this)[i] - Y[i]) < 0) Y[i] += P;
    return X.Mul(Y).isz(n());
  }
  Poly Pow(const string &K) const {
    int nz = 0;
    while (nz < n() && !(*this)[nz]) ++nz;
    LL nk = 0, nk2 = 0;
    for (char c : K) {
      nk = (nk * 10 + c - '0') % P;
      nk2 = nk2 * 10 + c - '0';
      if (nk2 * nz >= n()) return Poly(n());
      nk2 %= P - 1;
    }
    if (!nk && !nk2) return Poly(Poly {1}, n());
    Poly X(data() + nz, data() + nz + n() - nz * nk2);
    LL x0 = X[0];
    return X.imul(ntt.minv(x0)).Ln().imul(nk).Exp()
      .imul(ntt.mpow(x0, nk2)).irev().isz(n()).irev();
  }
  static LL LinearRecursion(const vector<LL> &a, const
       vector<LL> &coef, LL n) { // a_n = \sum c_j a_(n-j)
    const int k = (int)a.size();
    assert((int)coef.size() == k + 1);
    Poly C(k + 1), W(Poly {1}, k), M = {0, 1};
    fi(1, k + 1) C[k - i] = coef[i] ? P - coef[i] : 0;
    C[k] = 1;
    while (n) {
      if (n % 2) W = W.Mul(M).DivMod(C).second;
      n /= 2, M = M.Mul(M).DivMod(C).second;
    }
    LL ret = 0;
    fi(0, k) ret = (ret + W[i] * a[i]) % P;
    return ret;
  }
};
#undef fi
using Poly_t = Poly<131072 * 2, 998244353, 3>;
template<> decltype(Poly_t::ntt) Poly_t::ntt = {};
```

8.8 Simplex Algorithm

```
const int MAXN = 111;
const int MAXM = 111;
const double eps = 1E-10;
double a[MAXN][MAXM], b[MAXN], c[MAXM], d[MAXN][MAXM];
double x[MAXM];
int ix[MAXN + MAXM]; // all 0-index
// 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(double a[MAXN][MAXM], double b[MAXN],
               double c[MAXM], int n, int m){
  ++m;
  int r = n, s = m - 1;
  memset(d, 0, sizeof(d));
  for (int i = 0; i < n + m; ++i) ix[i] = i;
  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;
  }
  for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
  d[n + 1][m - 1] = -1;
  for (double dd;; ) {
    if (r < n) {
      int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
      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 = -1; s = -1;
    for (int j = 0; j < m; ++j)
      if (s < 0 || ix[s] > ix[j]) {
        if (d[n + 1][j] > eps ||
            (d[n + 1][j] > -eps && 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 ||
          (dd = d[r][m] / d[r][s] - d[i][m] / d[i][s]) < -eps ||
          (dd < eps && 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;
  for(int i=0; i<m; i++) x[i] = 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;
}
```

8.9 triangles

Let a, b, c be length of the three sides of a triangle.

$$p = (a + b + c) * 0.5$$

The inradius is defined by:

$$iR = \sqrt{\frac{(p-a)(p-b)(p-c)}{p}}$$

The radius of its circumcircle is given by the formula:

$$cR = \frac{abc}{\sqrt{(a+b+c)(a+b-c)(a+c-b)(b+c-a)}}$$

9 Misc

9.1 black magic

```
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/assoc_container.hpp> //rb_tree
using namespace __gnu_pbds;
typedef __gnu_pbds::priority_queue<int> heap;
int main() {
  heap h1, h2;
  h1.push(1), h1.push(3);
  h2.push(2), h2.push(4);
  h1.join(h2);
  cout << h1.size() << h2.size() << h1.top() << endl; //404
  tree<ll, null_type, less<ll>, rb_tree_tag,
       tree_order_statistics_node_update> st;
  tree<ll, ll, less<ll>, rb_tree_tag,
       tree_order_statistics_node_update> mp;
  for (int x : {0, 2, 3, 4}) st.insert(x);
  cout << *st.find_by_order(2) << st.order_of_key(1) << endl;
       //31
}
//__int128_t,__float128_t
```

9.2 dates

```
int weekday(int y,int m,int d){
  if (m==1||m==2) {m+=12; y--;}
  return (d+2*m+3*(m+1)/5+y+y/4-y/100+y/400+1)%7;
}
```

9.3 readchar

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

9.4 Texas holdem

```
char
     suit[4]={'C','D','H','Y'},ranks[13]={'2','3','4','5','6','7','8','9','T','J','Q','K','A'};
int rk[256];
/*
   for(int i=0;i<13;++i)
   rk[ranks[i]]=i;
   for(int i=0;i<4;++i)
   rk[suit[i]]=i;
   */
struct cards{
  vector<pii> v;
  int suit_count[4],hands;
  void reset(){v.clear(),FILL(suit_count,0),hands=-1;}
  void insert(char a,char b){//suit,rank
    ++suit_count[rk[a]];
    int flag=0;
    for(auto &i:v)
      if(i.Y==rk[b])
      {
        ++i.X,flag=1;
        break;
      }
    if(!flag) v.pb(pii(1,rk[b]));
  }
  void insert(string s){insert(s[0],s[1]);}
  void ready(){
    int
         Straight=0,Flush=(*max_element(suit_count,suit_count+4)==5);
    sort(ALL(v),[](ii a,ii b){return a>b;});
    if(SZ(v)==5&&v[0].Y==v[1].Y+1&&v[1].Y==v[2].Y+1&&v[2].Y==v[3].Y+1&&v[3].Y==v[4].Y+1)
      Straight=1;
    else
         if(SZ(v)==5&&v[0].Y==12&&v[1].Y==3&&v[2].Y==2&&v[3].Y==1&&v[4].Y==0)
      v[0].Y=3,v[1].Y=2,v[2].Y=1,v[1].Y=0,v[0].Y=-1,Straight=1;
    if(Straight&&Flush) hands=1;
    else if(v[0].X==4) hands=2;
    else if(v[0].X==3&&v[1].X==2) hands=3;
    else if(Flush) hands=4;
    else if(Straight) hands=5;
    else if(v[0].X==3) hands=6;
    else if(v[0].X==2&&v[1].X==2) hands=7;
    else if(v[0].X==2) hands=8;
    else hands=9;
  }
  bool operator>(const cards &a)const{
    if(hands==a.hands) return v>a.v;
    return hands<a.hands;
  }
};
```

10 Number theory

10.1 chineseRemainder

```
LL solve(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;
  int p, q;
  extgcd(m1, m2, p, q);
  LL lcm = m1 * m2 * g;
  LL res = p * (x2 - x1) * m1 + x1;
  return (res % lcm + lcm) % lcm;
}
```

10.2 convolution

```
//check x is 2^a
inline bool is_pow2(LL x);

inline int ceil_log2(LL x) {
  int ans = 0;
  --x;
  while (x != 0) {
    x >>= 1;
    ans++;
  }
  return ans;
}

/* Returns the convolution of the two given vectors in time
     proportional to n*log(n).
 * The number of roots of unity to use nroots_unity must be set
     so that the product of the first
 * nroots_unity primes of the vector nth_roots_unity is greater
     than the maximum value of the
 * convolution. Never use sizes of vectors bigger than 2^24, if
     you need to change the values of
 * the nth roots of unity to appropriate primes for those sizes.
 */
vector<LL> convolve(const vector<LL> &a, const vector<LL> &b,
     int nroots_unity = 2) {
  int N = 1 << ceil_log2(a.size() + b.size());
  vector<LL> ans(N,0), fA(N), fB(N), fC(N);
  LL modulo = 1;
  for (int times = 0; times < nroots_unity; times++) {
    fill(fA.begin(), fA.end(), 0);
    fill(fB.begin(), fB.end(), 0);
    for (int i = 0; i < a.size(); i++) fA[i] = a[i];
    for (int i = 0; i < b.size(); i++) fB[i] = b[i];
    LL prime = nth_roots_unity[times].first;
    LL inv_modulo = mod_inv(modulo % prime, prime);
    LL normalize = mod_inv(N, prime);
    ntfft(fA, 1, nth_roots_unity[times]);
    ntfft(fB, 1, nth_roots_unity[times]);
    for (int i = 0; i < N; i++) fC[i] = (fA[i] * fB[i]) % prime;
    ntfft(fC, -1, nth_roots_unity[times]);
    for (int i = 0; i < N; i++) {
      LL curr = (fC[i] * normalize) % prime;
      LL k = (curr - (ans[i] % prime) + prime) % prime;
      k = (k * inv_modulo) % prime;
      ans[i] += modulo * k;
    }
    modulo *= prime;
  }
  return ans;
}
```

10.3 diophantine equations

```
ll gcd(ll a, ll b, ll &x, ll &y) {
  if (a == 0)
    return x = 0, y = 1, b;

  ll x1, y1;
  ll d = gcd(b % a, a, x1, y1);
  x = y1 - (b / a) * x1;
  y = x1;
  return d;
}

bool find_any_solution(ll a, ll b, ll c, ll &x0,
    ll &y0, ll &g) {
  g = gcd(abs(a), abs(b), x0, y0);
  if (c % g)
    return false;

  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
```

```
void shift_solution(ll &x, ll &y, ll a, ll b,
    ll cnt) {
  x += cnt * b;
  y -= cnt * a;
}

ll find_all_solutions(ll a, ll b, ll c,
    ll minx, ll maxx, ll miny,
    ll maxy) {
  ll x, y, g;
  if (!find_any_solution(a, b, c, x, y, g)) return 0;
  a /= g;
  b /= g;

  ll sign_a = a > 0 ? +1 : -1;
  ll sign_b = b > 0 ? +1 : -1;

  shift_solution(x, y, a, b, (minx - x) / b);
  if (x < minx) shift_solution(x, y, a, b, sign_b);
  if (x > maxx) return 0;
  ll lx1 = x;

  shift_solution(x, y, a, b, (maxx - x) / b);
  if (x > maxx) shift_solution(x, y, a, b, -sign_b);
  ll rx1 = x;

  shift_solution(x, y, a, b, -(miny - y) / a);
  if (y < miny) shift_solution(x, y, a, b, -sign_a);
  if (y > maxy) return 0;
  ll lx2 = x;

  shift_solution(x, y, a, b, -(maxy - y) / a);
  if (y > maxy) shift_solution(x, y, a, b, sign_a);
  ll rx2 = x;

  if (lx2 > rx2) swap(lx2, rx2);
  ll lx = max(lx1, lx2);
  ll rx = min(rx1, rx2);

  if (lx > rx) return 0;
  return (rx - lx) / abs(b) + 1;
}
```

10.4 discrete logarithm

```
// Computes x which a ^ x = b mod n.
ll d_log(ll a, ll b, ll n) {
  ll m = ceil(sqrt(n));
  ll aj = 1;
  map<ll, ll> M;
  for (int i = 0; i < m; ++i) {
    if (!M.count(aj))
      M[aj] = i;
    aj = (aj * a) % n;
  }
  ll coef = qp(a, n - 2, n);
  coef = qp(coef, m, n);
  // coef = a ^ (-m)
  ll gamma = b;
  for (int i = 0; i < m; ++i) {
    if (M.count(gamma)) {
      return i * m + M[gamma];
    } else {
      gamma = (gamma * coef) % n;
    }
  }
  return -1;
}
```

10.5 fibonacci properties

Let A, B and n be integer numbers.

$$k = A - B \tag{1}$$

$$F_A F_B = F_{k+1} F_A^2 + F_k F_A F_{A-1} \tag{2}$$

$$\sum_{i=0}^{n} F_i^2 = F_{n+1} F_n \tag{3}$$

$ev(n) =$ returns 1 if n is even.

$$\sum_{i=0}^{n} F_i F_{i+1} = F_{n+1}^2 - ev(n) \tag{4}$$

$$\sum_{i=0}^{n} F_i F_{i-1} = \sum_{i=0}^{n-1} F_i F_{i+1} \tag{5}$$

10.6 floor ceil

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

10.7 floor sum

```
ll q(ll a, ll b) {
      a += b * MOD;
      return a / b - MOD;
}
ll f(ll n, ll m, ll a, ll b) {
      if (!n) return 0;
      if (!a) return b / m * n;
      ll x = (a * (n - 1) + b) / m;
      ll y = a - b - 1;
      ll r = (y % a + a) % a;
      return q(a - b - 1, a) + n * x - q(m * x - b + a - 1,
           a) - f(x, a, m % a, r) - x * (x - 1) / 2 * (m / a)
           - (y - r) / a * x;
}
// sum^{n-1}_0 floor((a * i + b) / m) in log(n + m + a + b)
```

10.8 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 Miller_Rabin(ll a, ll n) {
  if((a = a % n) == 0) return 1;
  if((n & 1) ^ 1) return n == 2;
  ll tmp = (n - 1) / ((n - 1) & (1 - n));
  ll t = __lg(((n - 1) & (1 - n))), x = 1;
  for(; tmp; tmp >>= 1, a = mul(a, a, n))
    if(tmp & 1) x = mul(x, a, n);
  if(x == 1 || x == n - 1) return 1;
  while(--t)
    if((x = mul(x, x, n)) == n - 1) return 1;
  return 0;
}
```

10.9 Mobius Inversion

```
struct INFO {
    int freq;
    int parity;
    int prod;
} info[V];
/* find all coprime pairs */
// [n == 1] = \sum_{d | n} \mu(d)
// substitute n to gcd(i, j)
signed main () {
    // input n
    for (i = 2; i < V; ++i) // init
        info[i].freq = info[i].parity = 0, info[i].prod = 1;
```

```
    for (i = 0; i < n; ++i)
        cin >> x, ++info[x].freq, M = max(M, x);

    for (i = 2; i * i <= M; ++i) // preprocess
        if (info[i].prod == 1)
            for (j = i, k = i; j <= M; j += i) {
                if (--k) {
                    info[j].parity ^= 1;
                    info[j].prod *= i;
                }
                else {
                    info[j].prod = 0;
                    k = i;
                }
            }
    for (; i <= M; ++i)
        if (info[i].prod && info[i].prod != i)
            info[i].parity ^= 1;

    ll ans = (n * (n - 1)) >> 1, c;
    for (i = 2; i <= M; ++i)
        if (info[i].prod) {
            c = 0;
            for (j = i; j <= M; j += i)
                c += info[j].freq;
            ans -= ((info[i].parity << 1) - 1) * ((c * (c - 1))
                 >> 1);
        }
}
```

10.10 Pollard Rho

```
ll f(ll x,ll mod){ return add(mul(x,x,mod),1,mod); }
ll pollard_rho(ll n){
  if(!(n&1)) return 2;
  while(1){
    ll y=2,x=rand()%(n-1)+1,res=1;
    for(int sz=2;res==1;y=x,sz<<=1)
      for(int i=0;i<sz&&res<=1;++i)
        x=f(x,n),res=gcd(abs(x-y),n);
    if(res!=0&&res!=n) return res;
  }
}
vector<ll> factorize(ll n) {
  vector<ll> ans;
  if (n == 1)
    return ans;
  if (miller_rabin(n)) {
    ans.push_back(n);
  }
  else {
    ll d = 1;
    while (d == 1)
      d = pollard_rho(n);
    vector<ll> dd = factorize(d);
    ans = factorize(n / d);
    for (int i = 0; i < dd.size(); ++i)
      ans.push_back(dd[i]);
  }
  return ans;
}
```

10.11 Prime Count

```
int64_t PrimeCount(int64_t n) {
  if (n <= 1) return 0;
  const int v = sqrt(n);
  vector<int> smalls(v + 1);
  for (int i = 2; i <= v; ++i) smalls[i] = (i + 1) / 2;
  int s = (v + 1) / 2;
  vector<int> roughs(s);
  for (int i = 0; i < s; ++i) roughs[i] = 2 * i + 1;
  vector<int64_t> larges(s);
  for (int i = 0; i < s; ++i) larges[i] = (n / (2 * i + 1) + 1)
       / 2;
  vector<bool> skip(v + 1);
  int pc = 0;
  for (int p = 3; p <= v; ++p) {
    if (smalls[p] > smalls[p - 1]) {
      int q = p * p;
      pc++;
      if (1LL * q * q > n) break;
      skip[p] = true;
      for (int i = q; i <= v; i += 2 * p) skip[i] = true;
      int ns = 0;
      for (int k = 0; k < s; ++k) {
        int i = roughs[k];
        if (skip[i]) continue;
        int64_t d = 1LL * i * p;
        larges[ns] = larges[k] - (d <= v ? larges[smalls[d] -
             pc] : smalls[n / d]) + pc;
        roughs[ns++] = i;
      }
      s = ns;
      for (int j = v / p; j >= p; --j) {
        int c = smalls[j] - pc;
        for (int i = j * p, e = min(i + p, v + 1); i < e; ++i)
             smalls[i] -= c;
      }
    }
  }
  for (int k = 1; k < s; ++k) {
    const int64_t m = n / roughs[k];
    int64_t s = larges[k] - (pc + k - 1);
    for (int l = 1; l < k; ++l) {
      int p = roughs[l];
      if (1LL * p * p > m) break;
      s -= smalls[m / p] - (pc + l - 1);
    }
    larges[0] -= s;
  }
  return larges[0];
}
```

10.12 Primes

```
/*
   12721 13331 14341 75577 123457 222557 556679 999983
        1097774749 1076767633 100102021 999997771 1001010013
        1000512343 987654361 999991231 999888733 98789101
        987777733 999991921 1010101333 1010102101 1000000000039
        1000000000000037 2305843009213693951
        4611686018427387847 9223372036854775783
        18446744073709551557
*/
```

10.13 QuadraticResidue

```
int Jacobi(int a, int m) {
  int s = 1;
  for (; m > 1; ) {
    a %= m;
    if (a == 0) return 0;
    const int r = __builtin_ctz(a);
    if ((r & 1) && ((m + 2) & 4)) s = -s;
    a >>= r;
    if (a & m & 2) s = -s;
    swap(a, m);
  }
  return s;
}

int QuadraticResidue(int a, int p) {
  if (p == 2) return a & 1;
  const int jc = Jacobi(a, p);
  if (jc == 0) return 0;
  if (jc == -1) return -1;
  int b, d;
  for (; ; ) {
    b = rand() % p;
    d = (1LL * b * b + p - a) % p;
    if (Jacobi(d, p) == -1) break;
  }
  int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
  for (int e = (1LL + p) >> 1; e; e >>= 1) {
    if (e & 1) {
      tmp = (1LL * g0 * f0 + 1LL * d * (1LL * g1 * f1 % p)) % p;
      g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
      g0 = tmp;
    }
    tmp = (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1 % p)) % p;
    f1 = (2LL * f0 * f1) % p;
    f0 = tmp;
  }
  return g0;
}
```

11 Strings

11.1 Incremental Aho Corasick

```
class IncrementalAhoCorasic {
  static const int Alphabets = 26;
  static const int AlphabetBase = 'a';
  struct Node {
    Node *fail;
    Node *next[Alphabets];
    int sum;
    Node() : fail(NULL), next{}, sum(0) { }
  };

  struct String {
    string str;
    int sign;
  };

public:
  //totalLen = sum of (len + 1)
  void init(int totalLen) {
    nodes.resize(totalLen);
    nNodes = 0;
    strings.clear();
    roots.clear();
    sizes.clear();
    que.resize(totalLen);
  }

  void insert(const string &str, int sign) {
    strings.push_back(String{ str, sign });
    roots.push_back(nodes.data() + nNodes);
    sizes.push_back(1);
    nNodes += (int)str.size() + 1;
    auto check = [&]() { return sizes.size() > 1 &&
         sizes.end()[-1] == sizes.end()[-2]; };
    if(!check())
      makePMA(strings.end() - 1, strings.end(), roots.back(),
          que);
    while(check()) {
      int m = sizes.back();
      roots.pop_back();
      sizes.pop_back();
      sizes.back() += m;
      if(!check())
        makePMA(strings.end() - m * 2, strings.end(),
            roots.back(), que);
    }
  }

  int match(const string &str) const {
    int res = 0;
    for(const Node *t : roots)
      res += matchPMA(t, str);
    return res;
  }

private:
```

```
static void makePMA(vector<String>::const_iterator begin,
    vector<String>::const_iterator end, Node *nodes,
    vector<Node*> &que) {
  int nNodes = 0;
  Node *root = new(&nodes[nNodes ++]) Node();
  for(auto it = begin; it != end; ++ it) {
    Node *t = root;
    for(char c : it->str) {
      Node *&n = t->next[c - AlphabetBase];
      if(n == nullptr)
        n = new(&nodes[nNodes ++]) Node();
      t = n;
    }
    t->sum += it->sign;
  }
  int qt = 0;
  for(Node *&n : root->next) {
    if(n != nullptr) {
      n->fail = root;
      que[qt ++] = n;
    } else {
      n = root;
    }
  }
  for(int qh = 0; qh != qt; ++ qh) {
    Node *t = que[qh];
    int a = 0;
    for(Node *n : t->next) {
      if(n != nullptr) {
        que[qt ++] = n;
        Node *r = t->fail;
        while(r->next[a] == nullptr)
          r = r->fail;
        n->fail = r->next[a];
        n->sum += r->next[a]->sum;
      }
      ++ a;
    }
  }
}

static int matchPMA(const Node *t, const string &str) {
  int res = 0;
  for(char c : str) {
    int a = c - AlphabetBase;
    while(t->next[a] == nullptr)
      t = t->fail;
    t = t->next[a];
    res += t->sum;
  }
  return res;
}


vector<Node> nodes;
int nNodes;
vector<String> strings;
vector<Node*> roots;
vector<int> sizes;
vector<Node*> que;
```

```
};

int main() {
  int m;
  while(~scanf("%d", &m)) {
    IncrementalAhoCorasic iac;
    iac.init(600000);
    rep(i, m) {
      int ty;
      char s[300001];
      scanf("%d%s", &ty, s);
      if(ty == 1) {
        iac.insert(s, +1);
      } else if(ty == 2) {
        iac.insert(s, -1);
      } else if(ty == 3) {
        int ans = iac.match(s);
        printf("%d\n", ans);
        fflush(stdout);
      } else {
        abort();
      }
    }
  }
  return 0;
}
```

11.2 KMP

```
/* partial matching list */
string s;
int len=s.length(),list[len]={0};
for(int i=1;i<len;i++) {
    int j=list[i-1];
    while(j>0&&s[i]!=s[j])
        j=list[j-1];
    if(s[i]==s[j])list[i]=j+1;
}

/* finding substring by list */
string p;
int j=0;
bool ans=false;
for(int i=0;i<p.length();i++) {
    if(p[i]==s[j]) {
        j++;
        if(j==len) {
            ans=true;
            break;
        }
    }
    else {
        while(j>0&&p[i]!=s[j])
            j=list[j-1];
        if(p[i]==s[j]) ++j;
        else j = ;
        if(len-j>s.length()-i-1)break; //
```

```
  }
}
```

11.3 minimal string rotation

```
// Lexicographically minimal string rotation
int lmsr() {
  string s;
  cin >> s;
  int n = s.size();
  s += s;
  vector<int> f(s.size(), -1);
  int k = 0;
  for (int j = 1; j < 2 * n; ++j) {
    int i = f[j - k - 1];
    while (i != -1 && s[j] != s[k + i + 1]) {
      if (s[j] < s[k + i + 1])
        k = j - i - 1;
      i = f[i];
    }
    if (i == -1 && s[j] != s[k + i + 1]) {
      if (s[j] < s[k + i + 1]) {
        k = j;
      }
      f[j - k] = -1;
    } else {
      f[j - k] = i + 1;
    }
  }
  return k;
}
```

11.4 suffix array

```
/**
 * O (n log^2 (n))
 * See http://web.stanford.edu/class/cs97si/suffix-array.pdf
      for reference
 * */

struct entry{
  int a, b, p;
  entry(){}
  entry(int x, int y, int z): a(x), b(y), p(z){}
  bool operator < (const entry &o) const {
    return (a == o.a) ? (b == o.b) ? ( p < o.p) : (b < o.b) :
         (a < o.a);
  }
};

struct SuffixArray{
  const int N;
  string s;
  vector<vector<int> > P;
  vector<entry> M;

  SuffixArray(const string &s) : N(s.length()) , s(s), P(1,
       vector<int> (N, 0)), M(N) {
    for (int i = 0; i < N; ++i)
      P[0][i] = (int) s[i];

    for (int skip = 1, level = 1; skip < N; skip *= 2, level++)
         {
      P.push_back(vector<int>(N, 0));
      for (int i = 0 ; i < N; ++i) {
        int next = ((i + skip) < N) ? P[level - 1][i + skip] :
             -10000;
        M[i] = entry(P[level - 1][i], next, i);
      }
      sort(M.begin(), M.end());
      for (int i = 0; i < N; ++i)
        P[level][M[i].p] = (i > 0 and M[i].a == M[i - 1].a and
             M[i].b == M[i - 1].b) ? P[level][M[i - 1].p] : i;
    }
  }

  vector<int> getSuffixArray(){
    vector<int> &rank = P.back();
    vector<pair<int, int> > inv(rank.size());
    for (int i = 0; i < rank.size(); ++i)
      inv[i] = make_pair(rank[i], i);
    sort(inv.begin(), inv.end());
    vector<int> sa(rank.size());
    for (int i = 0; i < rank.size(); ++i)
      sa[i] = inv[i].second;
    return sa;
  }

  // returns the length of the longest common prefix of
       s[i...L-1] and s[j...L-1]
  int lcp(int i, int j) {
    int len = 0;
    if (i == j) return N - i;
    for (int k = P.size() - 1; k >= 0 && i < N && j < N; --k) {
      if (P[k][i] == P[k][j]) {
        i += 1 << k;
        j += 1 << k;
        len += 1 << k;
      }
    }
    return len;
  }
};
```

11.5 suffix automaton

```
/*
 * Suffix automaton:
 * This implementation was extended to maintain (online) the
 * number of different substrings. This is equivalent to compute
 * the number of paths from the initial state to all the other
 * states.
 *
 * The overall complexity is O(n)
 * can be tested here:
       https://www.urionlinejudge.com.br/judge/en/problems/view/1530
 * */

struct state {
  int len, link;
  long long num_paths;
  map<int, int> next;
};

const int MN = 200011;
state sa[MN << 1];
int sz, last;
long long tot_paths;

void sa_init() {
  sz = 1;
  last = 0;
  sa[0].len = 0;
  sa[0].link = -1;
  sa[0].next.clear();
  sa[0].num_paths = 1;
  tot_paths = 0;
}

void sa_extend(int c) {
  int cur = sz++;
  sa[cur].len = sa[last].len + 1;
  sa[cur].next.clear();
  sa[cur].num_paths = 0;
  int p;
  for (p = last; p != -1 && !sa[p].next.count(c); p =
       sa[p].link) {
    sa[p].next[c] = cur;
    sa[cur].num_paths += sa[p].num_paths;
    tot_paths += sa[p].num_paths;
  }

  if (p == -1) {
    sa[cur].link = 0;
  } else {
    int q = sa[p].next[c];
    if (sa[p].len + 1 == sa[q].len) {
      sa[cur].link = q;
    } else {
      int clone = sz++;
      sa[clone].len = sa[p].len + 1;
      sa[clone].next = sa[q].next;
      sa[clone].num_paths = 0;
      sa[clone].link = sa[q].link;
      for (; p!= -1 && sa[p].next[c] == q; p = sa[p].link) {
        sa[p].next[c] = clone;
        sa[q].num_paths -= sa[p].num_paths;
        sa[clone].num_paths += sa[p].num_paths;
      }
      sa[q].link = sa[cur].link = clone;
    }
```

```
  }
  last = cur;
}
```

11.6 z algorithm

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

vector<int> compute_z(const string &s){
  int n = s.size();
  vector<int> z(n,0);
  int l,r;
  r = l = 0;
  for(int i = 1; i < n; ++i){
    if(i > r) {
      l = r = i;
      while(r < n and s[r - l] == s[r])r++;
      z[i] = r - l;r--;
    }else{
      int k = i-l;
      if(z[k] < r - i +1) z[i] = z[k];
      else {
        l = i;
        while(r < n and s[r - l] == s[r])r++;
        z[i] = r - l;r--;
      }
    }
  }
  return z;
}

int main(){
  //string line;cin>>line;
  string line = "alfalfa";
  vector<int> z = compute_z(line);

  for(int i = 0; i < z.size(); ++i ){
    if(i)cout<<" ";
    cout<<z[i];
  }
  cout<<endl;

  // must print "0 0 0 4 0 0 1"

  return 0;
}
```