Team Notebook

Ho Vu Thanh Binh, Vo Thanh Nguyen, Nguyen Nam Hy

November 16, 2024

Contents

1 DP

1.1 Divide and Conquer

```
int m, n;
vector<long long> dp_before(n), dp_cur(n);

long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};

    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k,
             mid), k});
    }

    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

int solve() {
    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);

    for (int i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }

    return dp_before[n - 1];
}
```

1.2 SOS

```
// sum over subsets O(N * 2^N)
for (int i = 0; i < (1LL << n); i++) {
       sos[i] = cnt[i];
}
for (int i = 0; i < n; i++) {
       for (int mask = 0; mask < (1LL << n); mask++) {
              if (mask & (1 << i)) {
                     sos[mask] += sos[mask ^ (1LL << i)];
              }
       }
}
// list all over subsets O(3^N)
for (int i = 0; i < (1 << n); i++) {
 for (int j = (i - 1) & i; j >= 0; j = (j - 1) & i) { sos[i]
       += a[j]; }

}
```

2 Data Structures

2.1 CentroidDecomp

```
vector<vector<int>> adj;
vector<bool> is_removed;
vector<int> subtree_size;

/** DFS to calculate the size of the subtree rooted at `node
     ` */
int get_subtree_size(int node, int parent = -1) {
 subtree_size[node] = 1;
 for (int child : adj[node]) {
  if (child == parent || is_removed[child]) { continue; }
  subtree_size[node] += get_subtree_size(child, node);
 }
 return subtree_size[node];
}

/**
 * Returns a centroid (a tree may have two centroids) of the
       subtree
 * containing node `node` after node removals
 * @param node current node
 * @param tree_size size of current subtree after node
      removals
 * @param parent parent of u
 * @return first centroid found
 */
int get_centroid(int node, int tree_size, int parent = -1) {
 for (int child : adj[node]) {
  if (child == parent || is_removed[child]) { continue; }
  if (subtree_size[child] * 2 > tree_size) {
   return get_centroid(child, tree_size, node);
  }
 }
 return node;
}

/** Build up the centroid decomposition recursively */
void build_centroid_decomp(int node = 0) {
 int centroid = get_centroid(node, get_subtree_size(node));

 // do something

 is_removed[centroid] = true;


 for (int child : adj[centroid]) {
  if (is_removed[child]) { continue; }
  build_centroid_decomp(child);

 }

}
```

2.2 CoordinateCompression

```
template<typename T>
struct CC {
 vector<T> val;
 bool ckcc = 1;
 void insert(T num) { val.push_back(num); ckcc = 0; }
 void cc() { sort(all(val)), val.erase(unique(all(val)), val
      .end()), ckcc = 1;}
 int idx(T num) { assert(ckcc); return lower_bound(all(val),
       num) - val.begin() + 1; }
};
```

2.3 DSU

```
struct DSU {
       int n;
       vector<int> par, sz;

       DSU(int n): n(n) {
              par = sz = vector<int> (n + 1);
              fill(all(sz), 1LL);
              iota(all(par), 0LL);
       }
       int find(int v) {
              while (v != par[v]) v = par[v] = par[par[v]];
```

```
        return v;
    }

    bool join(int u, int v) {
        u = find(u);
        v = find(v);
        if (u == v) return false;
        if (sz[u] < sz[v]) swap(u, v);
        sz[u] += sz[v];
        par[v] = u;
        return true;
    }

    int same(int u, int v) {
        return find(u) == find(v);
    }
};
```

2.4 FakeFenwick

```
struct FakeFenwick {
    vector<vector<int>> fw, val;
    int n;
    FakeFenwick() {}
    FakeFenwick(int n): n(n), val(n + 1, vector<int>()),
        fw(n + 1) {}
    bool iscc = 0;
    void fakeU(int x, int y) {
        iscc = 0;
        for (; x <= n; x += x & -x) val[x].push_back(y
            );
    }
    void cc() {
        if (iscc) return;
        for (int x = 1; x <= n; x++) {
            sort(all(val[x]));
            val[x].erase(unique(all(val[x])), val[x
                ].end());
            fw[x].resize(val[x].size() + 1);
        }
        iscc = 1;
    }
    void update(int x, int y, int v) {
        assert(iscc);
        for (; x <= n; x += x & -x) {
            int yy = upper_bound(all(val[x]), y) -
                val[x].begin();
            for (; yy <= val[x].size(); yy += yy &
                -yy) {
                fw[x][yy] += v;
            }

        }
    }
    int get(int x, int y) {
        assert(iscc);
        int res = 0;
        for (; x; x -= x & -x) {
            int yy = upper_bound(all(val[x]), y) -
                val[x].begin();
            for (; yy; yy -= yy & -yy) {
                res += fw[x][yy];
            }
        }
        return res;
    }
};
```

2.5 Fenwick

```
template<typename T>
struct fenwick {
 vector<T> f;
 int n;
 fenwick(int n): n(n) {
  f.resize(n + 1);
 }
 void add(int i, T v) {
  dbg(i);
  for (; i <= n; i += i & -i) {
   f[i] += v;
   dbg(i, v);
  }
 }
 T get(int i) {
  T ans = 0;
  for (; i >= 1; i -= i & -i) ans += f[i];
  return ans;
 }
 T find(T k) {
  T sum = 0;
  int pos = 0;
  for (int i = __lg(n); i >= 0; i--)
  {
   if (pos + (1 << i) < n && sum + f[pos + (1 << i)] < k)
   {
    sum += f[pos + (1 << i)];
    pos += (1 << i);
   }
  }
  return pos + 1;
 }
};
```

2.6 HLD

```
struct HLD {
    int n;
    int root;
    vector<int> sz, par, dep, top, in, out, seq;
    vector<vector<int>> adj;
    int curdfs = 0;
    HLD(int n): n(n) {
        par = sz = dep = top = in = out = seq = vector
            <int> (n + 1);
        adj.resize(n + 1);
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void start(int root = 1) {
        this->root = root;
        par[root] = root;
        top[root] = root;
        dfssz(root);
        dfshld(root);
    }
    void dfssz(int u) {
        if (u != root) adj[u].erase(find(all(adj[u]),
            par[u]));
        sz[u] = 1;
        for (int &v : adj[u]) {
            dep[v] = dep[u] + 1;
            par[v] = u;
            dfssz(v);
            sz[u] += sz[v];
            if (sz[v] > sz[adj[u][0]]) swap(adj[u
                ][0], v);
        }
    }
    void dfshld(int u) {
        in[u] = ++curdfs;
        seq[in[u]] = u;
        for (int v : adj[u]) {
            top[v] = (v == adj[u][0] ? top[u] : v);
            dfshld(v);
```

```
            }
            out[u] = curdfs;
        }
        int lca(int u, int v) {
            while (top[u] != top[v]) {
                if (dep[top[u]] > dep[top[v]]) swap(u,
                    v);
  v = par[top[v]];
            }
            return (dep[u] < dep[v] ? u : v);
        }
        bool isancestor(int u, int v) {
            return in[u] <= in[v] && out[v] <= out[u];
        }
};
```

2.7 LinkCutTree

```
typedef long long ll;

typedef struct snode *sn;
struct snode {       //////// VARIABLES
 sn p, c[2];       // parent, children
 bool flip = 0;    // subtree flipped or not
 int sz;           // # nodes in current splay tree
 ll sub, vsub = 0; // vsub stores sum of virtual children
 ll val;           // value in node
 snode(int _val) : val(_val) {
  p = c[0] = c[1] = NULL;
  calc();
 }
 friend int getSz(sn x) { return x ? x->sz : 0; }
 friend ll getSub(sn x) { return x ? x->sub : 0; }
 void prop() { // lazy prop
  if (!flip) return;
  swap(c[0], c[1]);
  flip = 0;
  for (int i = 0; i < 2; i++)
   if (c[i]) c[i]->flip ^= 1;
 }
 void calc() { // recalc vals
  for (int i = 0; i < 2; i++)
   if (c[i]) c[i]->prop();
  sz = 1 + getSz(c[0]) + getSz(c[1]);
  sub = val + getSub(c[0]) + getSub(c[1]) + vsub;
 }
 //////// SPLAY TREE OPERATIONS
 int dir() {
  if (!p) return -2;
  for (int i = 0; i < 2; i++)
   if (p->c[i] == this) return i;
  return -1; // p is path-parent pointer
 } // -> not in current splay tree
 // test if root of current splay tree
 bool isRoot() { return dir() < 0; }
 friend void setLink(sn x, sn y, int d) {
  if (y) y->p = x;
  if (d >= 0) x->c[d] = y;
 }
 void rot() { // assume p and p->p propagated
  assert(!isRoot());
  int x = dir();
  sn pa = p;
  setLink(pa->p, this, pa->dir());
  setLink(pa, c[x ^ 1], x);
  setLink(this, pa, x ^ 1);
  pa->calc();
 }
 void splay() {
  while (!isRoot() && !p->isRoot()) {
   p->p->prop(), p->prop(), prop();
   dir() == p->dir() ? p->rot() : rot();
   rot();
  }
  if (!isRoot()) p->prop(), prop(), rot();
  prop();
  calc();
 }
 sn fbo(int b) { // find by order
  prop();
  int z = getSz(c[0]); // of splay tree
  if (b == z) {
   splay();
   return this;
  }
  return b < z ? c[0]->fbo(b) : c[1]->fbo(b - z - 1);
 }
 //////// BASE OPERATIONS
 void access() { // bring this to top of tree, propagate
  for (sn v = this, pre = NULL; v; v = v->p) {
   v->splay(); // now switch virtual children
   if (pre) v->vsub -= pre->sub;
   if (v->c[1]) v->vsub += v->c[1]->sub;
   v->c[1] = pre;
   v->calc();
   pre = v;
  }
  splay();
  assert(!c[1]); // right subtree is empty
 }
 void makeRoot() {
  access();
  flip ^= 1;
  access();
  assert(!c[0] && !c[1]);
 }
 //////// QUERIES
 friend sn lca(sn x, sn y) {
  if (x == y) return x;
  x->access(), y->access();
  if (!x->p) return NULL;
  x->splay();
  return x->p ?: x; // y was below x in latter case
 } // access at y did not affect x -> not connected
 friend bool connected(sn x, sn y) { return lca(x, y); }
 // # nodes above
 int distRoot() {
  access();
  return getSz(c[0]);
 }
 sn getRoot() { // get root of LCT component
  access();
  sn a = this;
  while (a->c[0]) a = a->c[0], a->prop();
  a->access();
  return a;
 }
 sn getPar(int b) { // get b-th parent on path to root
  access();
  b = getSz(c[0]) - b;
  assert(b >= 0);
  return fbo(b);
 } // can also get min, max on path to root, etc
 //////// MODIFICATIONS
 void set(int v) {
  access();
  val = v;
  calc();
 }
 friend void link(sn x, sn y, bool force = 0) {
  assert(!connected(x, y));
  if (force) y->makeRoot(); // make x par of y
  else {
   y->access();
   assert(!y->c[0]);
  }
  x->access();
  setLink(y, x, 0);
  y->calc();
```

```
 }
 friend void cut(sn y) { // cut y from its parent
  y->access();
  assert(y->c[0]);
  y->c[0]->p = NULL;
  y->c[0] = NULL;
  y->calc();
 }
 friend void cut(sn x, sn y) { // if x, y adj in tree
  x->makeRoot();
  y->access();
  assert(y->c[0] == x && !x->c[0] && !x->c[1]);
  cut(y);
 }
};
```

2.8 OrderedSet

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;
template <class type1>
using ordered_set = tree<type1, null_type, less_equal<type1
    >, rb_tree_tag, tree_order_statistics_node_update>;
.insert(x); // insert x
.order_of_key(val + 1) - .order_of_key(val) > 0; // check
    val exists
.erase(.find_by_order(.order_of_key(c))) // erase c , should
     check exists before
.find_by_order(pos) // s[pos] : 0-indexed
.order_of_key(x) // n.o element less than x
```

2.9 PURQ

```
struct Info {
 int val;
 bool empty;
 Info(): val(0), empty(1) { }
 Info(int val): val(val), empty(0) { }
 friend Info operator + (Info lef, Info rig) {
  if (lef.empty || rig.empty) {
   return lef.empty ? rig : lef;
  }
  Info res;
  res.empty = 0;
  res.val = lef.val + rig.val;
  return res;
 }
};

struct Segtree {
 vector<Info> st;
 int n;
#define lc id << 1
#define rc id << 1 | 1
#define mi ((l + r) >> 1)
 Segtree() {}
 Segtree(int n): n(n), st(n << 2, Info(0)) {
 }
 void update(int i, int x, int id, int l, int r) {
  if (l == r) {
   st[id].val += x;
   return;
  }
  if (i <= mi) update(i, x, lc, l, mi);
  else update(i, x, rc, mi+1, r);
  st[id] = st[lc] + st[rc];
 }
 Info get(int u, int v, int id, int l, int r) {
  if (u > r || l > v) {
   return Info();
  }
  if (u <= l && r <= v) {
   return st[id];
  }
  return get(u, v, lc, l, mi) + get(u, v, rc, mi+1, r);
 }
#undef lc
#undef rc
#undef mi
};
```

2.10 PersistentSegmentTree

```
struct PSTNode {
	int val;
	PSTNode *lef, *rig;
	PSTNode (int val) {
		this->val = val;
		lef = rig = nullptr;
	}
	PSTNode (PSTNode *lef, PSTNode *rig) {
		this->lef = lef;
		this->rig = rig;
		val = 0;
		if (lef) val += lef->val;
		if (rig) val += rig->val;
	}
	PSTNode (PSTNode *p) {
		this->lef = p->lef;
		this->rig = p->rig;
		this->val = p->val;
	}
};

struct PST {
	int n, nv;
	vector<PSTNode*> rt;
	PST() {}
	PST(int n, int nv, vector<int> vec) {
		this->n = n;
		this->nv = nv;
		rt.resize(nv + 1);
		rt[0] = build(1, n, vec);
	}

	PSTNode *build(int l, int r, vector<int> &vec) {
		if (l == r) {
			return new PSTNode(vec[l]);
		}
		int mi = (l + r) >> 1;
		return new PSTNode(build(l, mi, vec), build(mi
		    +1, r, vec));
	}

	int get(int i, PSTNode *p, int l, int r) {
		if (l == r) {
			return p->val;
		}
		int mi = (l + r) >> 1;
		if (i <= mi) return get(i, p->lef, l, mi);
		else return get(i, p->rig, mi+1, r);
	}
	int get(int i, int v) {
		return get(i, rt[v], 1, n);
	}
	PSTNode *update(int i, int val, PSTNode *p, int l,
	    int r) {
		if (l == r) {
			return new PSTNode(val);
		}
		int mi = (l + r) >> 1;
		if (i <= mi) return new PSTNode(update(i, val,
		    p->lef, l, mi), p->rig);
```

```
            else return new PSTNode(p->lef, update(i, val,
                 p->rig, mi+1, r));
        }

        void update(int i, int val, int v) {
            rt[v] = update(i, val, rt[v], 1, n);
        }

        void copy(int t1, int t2) {
            rt[t2] = rt[t1];
        }
};
```

2.11 RMQ

```
struct RMQ {
        int n, lg;
        vector<int> v;
        vector<vector<int>> st;
        RMQ(vector<int> & v): v(v), n(v.size() - 1) {
            lg = __lg(n) + 1;
            st.resize(n + 1, vector<int> (lg + 1));
            for (int i = 1; i <= n; i++) {
                st[i][0] = v[i];
            }
            for (int i = 1; i <= lg; i++) {
                for (int j = 1; j + (1LL << i) - 1 <= n
                     ; j++) {
                    st[j][i] = min(st[j][i - 1], st
                         [j + (1LL << (i - 1))][i -
                         1]);
                }
            }
        }
        int get(int l, int r) {
            assert(l <= r);
            int len = __lg(r - l + 1);
            return min(st[l][len], st[r - (1 << len) + 1][
                 len]);
        }
};
```

2.12 RURQ

```
struct Tag {
 int add;
 Tag(): add(0) {}
 Tag(int add): add(add) {}
};

struct Info {
 int val;
 bool empty = 1;
 Info(): val(0) {
  empty = 1;
 }
 Info(int val): val(val) {
  empty = 0;
 }
 friend Info operator + (Info lef, Info rig) {
  if (lef.empty || rig.empty) {
   return lef.empty ? rig : lef;
  }
  Info res;
  res.empty = 0;
  res.val = lef.val + rig.val;
  return res;
 }
};

struct Segtree {
 vector<Info> st;
 vector<Tag> lz;
 int n;
#define lc id << 1
#define rc id << 1 | 1
#define mi ((l + r) >> 1)
 Segtree() {}
 Segtree(int n): n(n), st(n << 2, Info(0)), lz(n << 2, Tag()
      ) {
 }
 void push(int id, int l, int r) {
  if (lz[id].add == 0) return;
  int x = lz[id].add;
  lz[id].add = 0;
  update(l, mi, x, lc, l, mi);
  update(mi+1, r, x, rc, mi+1, r);
 }
 void update(int u, int v, int x, int id, int l, int r) {
  if (u > r || l > v) {
   return;
  }
  if (u <= l && r <= v) {
   st[id].val += x * (r - l + 1);
   lz[id].add += x;
   return;
  }
  push(id, l, r);
  update(u, v, x, lc, l, mi);
  update(u, v, x, rc, mi+1, r);
  st[id] = st[lc] + st[rc];
 }
 Info get(int u, int v, int id, int l, int r) {
  if (u > r || l > v) {
   return Info();
  }
  if (u <= l && r <= v) {
   return st[id];
  }
  push(id, l, r);
  return get(u, v, lc, l, mi) + get(u, v, rc, mi+1, r);
 }
#undef lc
#undef rc
#undef mi
};
```

2.13 Treap

```
struct Treap {
 Treap *lef, *rig;
 Treap *par;
 int num;
 int prio;
 int sz;
 Treap(int num):
 num(num),
 prio(rnd()),
 sz(1),
 par(nullptr),
 lef(nullptr),
 rig(nullptr) {}
};

int getsz(Treap *p) {
 return (p ? p->sz : 0);
}

int getnum(Treap *p) {
 return (p ? p->num : 0);
}

void pull(Treap *p) {
 if (!p) return;
 p->sz = getsz(p->lef) + 1 + getsz(p->rig);
 p->par = nullptr;
```

```
 if (p->lef) p->lef->par = p;
 if (p->rig) p->rig->par = p;
}

void merge(Treap *&p, Treap *lef, Treap *rig) {
 if (!lef || !rig) {
  p = (lef ? lef : rig);
  return;
 }
 if (lef->prio > rig->prio) {
  merge(lef->rig, lef->rig, rig);
  p = lef;
 }
 else {
  merge(rig->lef, lef, rig->lef);
  p = rig;
 }
 pull(p);
}

void splitbynum(Treap *p, Treap *&lef, Treap *&rig, int k) {
	if (p == nullptr) {
		lef = rig = nullptr;
		return;
	}
	if (getnum(p) <= k) {
		splitbynum(p->rig, p->rig, rig, k);
		lef = p;
	}
	else {
		splitbynum(p->lef, lef, p->lef, k);
		rig = p;
	}
	pull(p);
}

void splitbysz(Treap *p, Treap *&lef, Treap *&rig, int k) {
 if (p == nullptr) {
  lef = rig = nullptr;
  return;
 }
 if (getsz(p->lef) < k) {
  splitbysz(p->rig, p->rig, rig, k - getsz(p->lef) - 1);
  lef = p;
 }
 else {
  splitbysz(p->lef, lef, p->lef, k);
  rig = p;
 }
 pull(p);
}

int index(Treap *p, Treap *pre = nullptr) {
 if (!p) return 0LL;
 if (pre && p->lef == pre) {
  return index(p->par, p);
 }
 else {
  return index(p->par, p) + getsz(p->lef) + 1;
 }
}
```

2.14 Trie

```
struct trie {
 trie* ch[2];
 int sum, cnt, end;
 trie() {
  ch[0] = ch[1] = nullptr;
  sum = cnt = end = 0;
 }
};

int MXVAL = 0; // 2^(MX + 1) - 1

trie *rt = new trie();
void add(int num) {
 auto p = rt;
 for (int i = 30; i >= 0; i--) {
  int b = (num >> i) & 1LL;
  if (p->ch[b] == nullptr) p->ch[b] = new trie();
  p = p->ch[b];
  p->cnt++;
  p->sum += num;
 }
 p->end++;
}
int find(int num) {
 auto p = rt;
 for (int i = 30; i >= 0; i--) {
  int b = (num >> i) & 1LL;
  if (p->ch[b] == nullptr) return false;
  p = p->ch[b];
 }
 return p->end;
}
bool del(trie *p, int num, int i, int d) {
 if (i >= 0) {
  int b = (num >> i) & 1LL;
  if (del(p->ch[b], num, i-1, d)) p->ch[b] = nullptr;
 }
 else {
  p->end -= d;
 }
 if (p != rt) {
  p->cnt -= d;
  p->sum -= num * d;
  if (p->cnt == 0) {
   delete(p);
   return true;
  }
 }
 return false;
}
void del(int num, int t) {
 int d = find(num);
 if (d == 0) return;
 if (t == 0) del(rt, num, 30, 1);
 else del(rt, num, 30, d);
}
int kth(int k) {
 auto p = rt;
 int res = 0;
 for (int i = 30; i >= 0; i--) {
  for (int b = 0; b < 2; b++) {
   if (p->ch[b]) {
    if (p->ch[b]->cnt >= k) {
     p = p->ch[b];
     res += (b << i);
     break;
    }
    else {
     k -= p->ch[b]->cnt;
    }
   }
  }
 }
 return res;
}
int sum(int u, int v, trie *p, int l, int r) {
 if (u > r || l > v || p == nullptr) {
  return 0;
 }
 if (u <= l && r <= v) {
  return p->cnt;
 }
 int mi = (l + r)/2;
 return sum(u, v, p->ch[0], l, mi) + sum(u, v, p->ch[1], mi
     +1, r);
```

```
}
int sum(int l, int r) {
 return sum(l, r, rt, 0, MXVAL);
}
int xr(int num) {
 auto p = rt;
 int res = 0;
 for (int i = 30; i >= 0; i--) {
  int b = (num >> i) & 1LL;
  if (p->ch[1 ^ b] != nullptr) {
   p = p->ch[1 ^ b];
   res |= (1 << i);
  }
  else {
   p = p->ch[b];
  }
 }
 return res;
}
```

2.15 Wavelet

```
struct wavelet {
 wavelet *lc, *rc;
 vector<int> pf;
 int lo, hi;

 wavelet(vector<int> a): wavelet(1 + all(a), *min_element(1
     + all(a)), *max_element(1 + all(a))){}
 wavelet(vector<int>::iterator l, vector<int>::iterator r,
     int lo, int hi): lo(lo), hi(hi) {
  if (lo == hi || l >= r) return;
  int mi = lo + (hi - lo)/2;
  pf.reserve(r - l + 1);
  pf.push_back(0LL);
  for (auto it = l; it != r; it++) {
   pf.push_back(pf.back() + (*it <= mi));
  }
  auto m = stable_partition(l, r, [&](int num) { return num
      <= mi; });
  lc = new wavelet(l, m, lo, mi);
  rc = new wavelet(m, r, mi+1, hi);
 }
 // tim so nho thu k trong doan (l, r);
 int kth(int l, int r, int k) {
  if (l > r) return 0;
  if (lo == hi) {
   return lo;
  }
  int lef = pf[r] - pf[l-1];
  if (lef >= k) return lc->kth(pf[l-1] + 1, pf[r], k);
  else return rc->kth(l - pf[l-1], r - pf[r], k - lef);
 }
 // dem so > k trong doan (l r)
 int count(int l, int r, int num) {
  if (l > r || hi <= num) return 0LL;
  if (lo > num) {
   return (r - l + 1);
  }
  return lc->count(pf[l-1] + 1, pf[r], num) + rc->count(l -
      pf[l-1], r - pf[r], num);
 }
};
```

3 Geometry

3.1 Basic

```
struct Point {
    double x, y;
    Point() { x = y = 0.0; }
    Point(double x, double y) : x(x), y(y) {}

    Point operator + (const Point &a) const { return Point(x
        + a.x, y + a.y); }
    Point operator - (const Point &a) const { return Point(x
        - a.x, y - a.y); }
    Point operator * (double k) const { return Point(x * k, y
         * k); }
    Point operator / (double k) const { return Point(x / k, y
         / k); }
};

struct Line { // Ax + By = C
    double a, b, c;
    Line(double a = 0, double b = 0, double c = 0) : a(a), b(
        b), c(c) {}
    Line(Point A, Point B) {
        a = B.y - A.y;
        b = A.x - B.x;
        c = a * A.x + b * A.y;
    }
};



Line Perpendicular_Bisector(Point A, Point B) {
    Point M = (A + B) / 2;
    Line d = Line(A, B);
    // the equation of a perpendicular line has the form: -Bx
        + Ay = D
    double D = -d.b * M.x + d.a * M.y;
    return Line(-d.b, d.a, D);
}
```

3.2 ConvexHull

```
//  Kiu    im
struct Point {
    int x, y;
};

// Tch  c   hng   ca  AB  v  AC
long long cross(const Point &A, const Point &B, const Point
    &C) {
    return 1LL * (B.x - A.x) * (C.y - A.y) - 1LL * (C.x - A.x
        ) * (B.y - A.y);
}

// A -> B -> C i theo  th   t  theo  chiu  kim   ng    h
    (-1),  thng  hng  (0),   ngc   chiu  kim   ng    h
    (1)
int ccw(const Point &A, const Point &B, const Point &C) {
    long long S = cross(A, B, C);
    if (S < 0) return -1;
    if (S == 0) return 0;
    return 1;
}

//  Tr   v  bao  li   vi   th   t   cc   im      c
     lit   k   ngc    chiu  kim   ng    h
vector<Point> convexHull(vector<Point> p, int n) {
    //  a    im    c  tung       nh   nht  ( v  tri   nht )
          ln    u    tp
    for (int i = 1; i < n; ++i) {
        if (p[0].y > p[i].y || (p[0].y == p[i].y && p[0].x >
            p[i].x)) {
            swap(p[0], p[i]);
        }
    }

    //  Sp   xp   cc    im   I theo gc  to   bi   trc
         honh  theo  chiu   dng   v  OI
    sort(p.begin() + 1, p.end(), [&p](const Point &A, const
        Point &B) {
        int c = ccw(p[0], A, B);
```

```
        if (c > 0) return true;
        if (c < 0) return false;
        return A.x < B.x || (A.x == B.x && A.y < B.y);
    });

    //  Tp  bao  li
    vector<Point> hull;
    hull.push_back(p[0]);

    //  Dng  bao  li
    for (int i = 1; i < n; ++i) {
        while (hull.size() >= 2 && ccw(hull[hull.size() - 2],
              hull.back(), p[i]) < 0) {
            hull.pop_back();
        }
        hull.push_back(p[i]);
    }
    return hull;
}
```

3.3 LineContainer

```
/**
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * Description: Container where you can add lines of the
     form kx+m, and query maximum values at points x.
 *  Useful for dynamic programming (``convex hull trick'').
 * Time: O(\log N)
 * Status: stress-tested
 */
#pragma once

struct Line {
 mutable ll k, m, p;
 bool operator<(const Line& o) const { return k < o.k; }
 bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
 // (for doubles, use inf = 1/.0, div(a,b) = a/b)
 static const ll inf = LLONG_MAX;
 ll div(ll a, ll b) { // floored division
  return a / b - ((a ^ b) < 0 && a % b); }
 bool isect(iterator x, iterator y) {
  if (y == end()) return x->p = inf, 0;
  if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
  else x->p = div(y->m - x->m, x->k - y->k);
  return x->p >= y->p;
 }
 // y = kx + m
 void add(ll k, ll m) {
  auto z = insert({k, m, 0}), y = z++, x = y;
  while (isect(y, z)) z = erase(z);
  if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
  while ((y = x) != begin() && (--x)->p >= y->p)
   isect(x, erase(y));
 }
 ll query(ll x) {
  assert(!empty());
  auto l = *lower_bound(x);
  return l.k * x + l.m;
 }
};
```

4 Graphs

4.1 Articular

```
(u == rt && ch[u].size() > 1) || (low[v] >= num[u] for some
     v in ch[u])
```

4.2 Bellman-Ford

```
struct Edge {
    int a, b, cost;
};

int n, m;
vector<Edge> edges;
const int INF = 1000000000;

void solve()
{
    vector<int> d(n);
    vector<int> p(n, -1);
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (Edge e : edges) {
            if (d[e.a] + e.cost < d[e.b]) {
                d[e.b] = d[e.a] + e.cost;
                p[e.b] = e.a;
                x = e.b;
            }
        }
    }

    if (x == -1) {
        cout << "No negative cycle found.";
    } else {
        for (int i = 0; i < n; ++i)
            x = p[x];

        vector<int> cycle;
        for (int v = x;; v = p[v]) {
            cycle.push_back(v);
            if (v == x && cycle.size() > 1)
                break;
        }
        reverse(cycle.begin(), cycle.end());

        cout << "Negative cycle: ";
        for (int v : cycle)
            cout << v << ' ';
        cout << endl;
    }
```

4.3 Bridge

```
low[u] = num[u]
```

4.4 Matching

```
class matching {
      public:
       vector<vector<int>> g;
       vector<int> pa;
       vector<int> pb;
       vector<int> was;
       int n, m;
       int res;
       int iter;
       matching(int _n, int _m) : n(_n), m(_m) {
              assert(0 <= n && 0 <= m);
              pa = vector<int>(n, -1);
              pb = vector<int>(m, -1);
              was = vector<int>(n, 0);
              g.resize(n);
              res = 0;
```

```
        iter = 0;
    }
    void add(int from, int to) {
        assert(0 <= from && from < n && 0 <= to && to
            < m);
        g[from].push_back(to);
    }
    bool dfs(int v) {
        was[v] = iter;
        for (int u : g[v]) {
            if (pb[u] == -1) {
                pa[v] = u;
                pb[u] = v;
                return true;
            }
        }
        for (int u : g[v]) {
            if (was[pb[u]] != iter && dfs(pb[u])) {
                pa[v] = u;
                pb[u] = v;
                return true;
            }
        }
        return false;
    }
    int solve() {
        while (true) {
            iter++;
            int add = 0;
            for (int i = 0; i < n; i++) {
                if (pa[i] == -1 && dfs(i)) {
                    add++;
                }
            }
            if (add == 0) {
                break;
            }
            res += add;
        }
        return res;
    }
    int run_one(int v) {
        if (pa[v] != -1) {
            return 0;
        }
        iter++;
        return (int)dfs(v);
    }
};
```

4.5 Maximum Flow

```
// https://pastebin.com/exQM152L

template <typename T>
class flow_graph {
 public:
  static constexpr T eps = (T) 1e-9;

  struct edge {
    int to;
    T c;
    T f;
    int rev;
  };

  vector<vector<edge>> g;
  vector<int> ptr;
  vector<int> d;
  vector<int> q;
  vector<int> cnt_on_layer;
  vector<int> prev_edge;
  bool can_reach_sink;

  int n;
  int st, fin;
  T flow;

  flow_graph(int _n, int _st, int _fin) : n(_n), st(_st),
      fin(_fin) {
    assert(0 <= st && st < n && 0 <= fin && fin < n && st !=
        fin);
    g.resize(n);
    ptr.resize(n);
    d.resize(n);
    q.resize(n);
    cnt_on_layer.resize(n + 1);
    prev_edge.resize(n);
    flow = 0;
  }

  void clear_flow() {
    for (int i = 0; i < n; i++) {
      for (edge &e : g[i]) {
        e.f = 0;
      }
    }
    flow = 0;
  }

  void add(int from, int to, T forward_cap, T backward_cap)
      {
    assert(0 <= from && from < n && 0 <= to && to < n);
    int from_size = g[from].size();
    int to_size = g[to].size();
    g[from].push_back({to, forward_cap, 0, to_size});
    g[to].push_back({from, backward_cap, 0, from_size});
  }

  bool expath() {
    fill(d.begin(), d.end(), n);
    q[0] = fin;
    d[fin] = 0;
    fill(cnt_on_layer.begin(), cnt_on_layer.end(), 0);
    cnt_on_layer[n] = n - 1;
    cnt_on_layer[0] = 1;
    int beg = 0, end = 1;
    while (beg < end) {
      int i = q[beg++];
      for (const edge &e : g[i]) {
        const edge &back = g[e.to][e.rev];
        if (back.c - back.f > eps && d[e.to] == n) {
          cnt_on_layer[d[e.to]]--;
          d[e.to] = d[i] + 1;
          cnt_on_layer[d[e.to]]++;
          q[end++] = e.to;
        }
      }
    }
    return (d[st] != n);
  }

  T augment(int &v) {
    T cur = numeric_limits<T>::max();
    int i = fin;
    while (i != st) {
      const edge &e = g[i][prev_edge[i]];
      const edge &back = g[e.to][e.rev];
      cur = min(cur, back.c - back.f);
      i = e.to;
    }
    i = fin;
    while (i != st) {
      edge &e = g[i][prev_edge[i]];
      edge &back = g[e.to][e.rev];
      back.f += cur;
      e.f -= cur;
      i = e.to;
      if (back.c - back.f <= eps) {
        v = i;
```

```
    }
  }
  return cur;
}

int retreat(int v) {
  int new_dist = n - 1;
  for (const edge &e : g[v]) {
    if (e.c - e.f > eps && d[e.to] < new_dist) {
      new_dist = d[e.to];
    }
  }
  cnt_on_layer[d[v]]--;
  if (cnt_on_layer[d[v]] == 0) {
    if (new_dist + 1 > d[v]) {
      can_reach_sink = false;
    }
  }
  d[v] = new_dist + 1;
  cnt_on_layer[d[v]]++;
  if (v != st) {
    v = g[v][prev_edge[v]].to;
  }
  return v;
}

T max_flow() {
  can_reach_sink = true;
  for (int i = 0; i < n; i++) {
    ptr[i] = (int) g[i].size() - 1;
  }
  if (expath()) {
    int v = st;
    while (d[st] < n) {
      while (ptr[v] >= 0) {
        const edge &e = g[v][ptr[v]];
        if (e.c - e.f > eps && d[e.to] == d[v] - 1) {
          prev_edge[e.to] = e.rev;
          v = e.to;
          if (v == fin) {
            flow += augment(v);
          }
          break;
        }
        ptr[v]--;
      }
      if (ptr[v] < 0) {
        ptr[v] = (int) g[v].size() - 1;
        v = retreat(v);
        if (!can_reach_sink) {
          break;
        }
      }
    }
  }
  return flow;
}

vector<bool> min_cut() {
  max_flow();
  assert(!expath());
  vector<bool> ret(n);
  for (int i = 0; i < n; i++) {
    ret[i] = (d[i] != n);
  }
  return ret;
}
};
```

4.6 Minimum Cost Maximum Flow

```
#include <bits/extc++.h>

template <typename T, typename C>
class MCMF {
 public:
  static constexpr T eps = (T) 1e-9;

  struct edge {
    int from;
    int to;
    T c;
    T f;
    C cost;
  };

  int n;
  vector<vector<int>> g;
  vector<edge> edges;
  vector<C> d;
  vector<C> pot;
  __gnu_pbds::priority_queue<pair<C, int>> q;
  vector<typename decltype(q)::point_iterator> its;
  vector<int> pe;
  const C INF_C = numeric_limits<C>::max() / 2;

  explicit MCMF(int n_) : n(n_), g(n), d(n), pot(n, 0), its(
      n), pe(n) {}

  int add(int from, int to, T forward_cap, T backward_cap, C
       edge_cost) {
    assert(0 <= from && from < n && 0 <= to && to < n);
    assert(forward_cap >= 0 && backward_cap >= 0);
    int id = static_cast<int>(edges.size());
    g[from].push_back(id);
    edges.push_back({from, to, forward_cap, 0, edge_cost});
    g[to].push_back(id + 1);
    edges.push_back({to, from, backward_cap, 0, -edge_cost});
    return id;
  }

  void expath(int st) {
    fill(d.begin(), d.end(), INF_C);
    q.clear();
    fill(its.begin(), its.end(), q.end());
    its[st] = q.push({pot[st], st});
    d[st] = 0;
    while (!q.empty()) {
      int i = q.top().second;
      q.pop();
      its[i] = q.end();
      for (int id : g[i]) {
        const edge &e = edges[id];
        int j = e.to;
        if (e.c - e.f > eps && d[i] + e.cost < d[j]) {
          d[j] = d[i] + e.cost;
          pe[j] = id;
          if (its[j] == q.end()) {
            its[j] = q.push({pot[j] - d[j], j});
          } else {
            q.modify(its[j], {pot[j] - d[j], j});
          }
        }
      }
    }
    swap(d, pot);
  }

  pair<T, C> max_flow_min_cost(int st, int fin) {
    T flow = 0;
    C cost = 0;
    bool ok = true;
    for (auto& e : edges) {
      if (e.c - e.f > eps && e.cost + pot[e.from] - pot[e.to]
           < 0) {
        ok = false;
        break;
      }
    }
```

```
if (ok) {
  expath(st);
} else {
  vector<int> deg(n, 0);
  for (int i = 0; i < n; i++) {
    for (int eid : g[i]) {
      auto& e = edges[eid];
      if (e.c - e.f > eps) {
        deg[e.to] += 1;
      }
    }
  }
  vector<int> que;
  for (int i = 0; i < n; i++) {
    if (deg[i] == 0) {
      que.push_back(i);
    }
  }
  for (int b = 0; b < (int) que.size(); b++) {
    for (int eid : g[que[b]]) {
      auto& e = edges[eid];
      if (e.c - e.f > eps) {
        deg[e.to] -= 1;
        if (deg[e.to] == 0) {
          que.push_back(e.to);
        }
      }
    }
  }
  fill(pot.begin(), pot.end(), INF_C);
  pot[st] = 0;
  if (static_cast<int>(que.size()) == n) {
    for (int v : que) {
      if (pot[v] < INF_C) {
        for (int eid : g[v]) {
          auto& e = edges[eid];
          if (e.c - e.f > eps) {
            if (pot[v] + e.cost < pot[e.to]) {
              pot[e.to] = pot[v] + e.cost;
              pe[e.to] = eid;
            }
          }
        }
      }
    }
  } else {
    que.assign(1, st);
    vector<bool> in_queue(n, false);
    in_queue[st] = true;
    for (int b = 0; b < (int) que.size(); b++) {
      int i = que[b];
      in_queue[i] = false;
      for (int id : g[i]) {
        const edge &e = edges[id];
        if (e.c - e.f > eps && pot[i] + e.cost < pot[e.to
            ]) {
          pot[e.to] = pot[i] + e.cost;
          pe[e.to] = id;
          if (!in_queue[e.to]) {
            que.push_back(e.to);
            in_queue[e.to] = true;
          }
        }
      }
    }
  }
  while (pot[fin] < INF_C) {
    T push = numeric_limits<T>::max();
    int v = fin;
    while (v != st) {
      const edge &e = edges[pe[v]];
      push = min(push, e.c - e.f);
      v = e.from;
    }
    v = fin;
    while (v != st) {
      edge &e = edges[pe[v]];
      e.f += push;
      edge &back = edges[pe[v] ^ 1];
      back.f -= push;
      v = e.from;
    }
    flow += push;
    cost += push * pot[fin];
    expath(st);
  }
  return {flow, cost};
 }
};
```

4.7 SCC

```
void tarjan(int u, int p) {
 in[u] = low[u] = ++curdfs;
 st.push(u);
 for (int v : adj[u]) {
  if (!in[v]) {
   tarjan(v, u);
   low[u] = min(low[u], low[v]);
  } else if (!del[v]) {
   low[u] = min(low[u], in[v]);
  }
 }
 if (low[u] == in[u]) {
  int v;
  do {
   v = st.top();
   st.pop();
   scc[u].push_back(v);
   top[v] = u;
   del[v] = true;
  } while (v != u);
 }
};
```

4.8 Topological Sort

```
void kahn(int u) {
 din[u] = -1;
 topo.push_back(u);
 for (int v : aa[u]) {
  din[v]--;
  if (din[v] == 0) kahn(v);
 }
}
```

4.9 TwoSatSolver

```
struct TSS {
       int nvar, nvrt;
       vector<vector<int>> adj;
       vector<bool> res;
       TSS(int nvar): nvar(nvar), nvrt(nvar * 2), adj(nvrt +
            1), res(nvar + 1) {}
       int conj(int u) {
              if (u > nvar) return u - nvar;
              return u + nvar;
       }
       int operator[](int i) { return res[i]; }
       void add(int u, bool nu, int v, bool nv) {
              if (nu) u = conj(u);
              if (nv) v = conj(v);
              adj[u].push_back(v);
       }
       bool solve() {
```

```
            vector<int> in(nvrt + 1), low(nvrt + 1), id(
                nvrt + 1);
            stack<int> st;
            int curdfs = 0, curidx = 2 * nvrt + 1;
            auto tarjan = [&](auto&& f, int u) -> void {
                in[u] = low[u] = ++curdfs;
                st.push(u);
                for (int v : adj[u]) {
                    if (!in[v]) {
                        f(f, v);
                        low[u] = min(low[u], low
                            [v]);
                    } else if (!id[v]) {
                        low[u] = min(low[u], in[
                            v]);
                    }
                }
                if (low[u] == in[u]) {
                    int v;
                    do {
                        v = st.top();
                        st.pop();
                        id[v] = curidx;
                    } while (v != u);
                    --curidx;
                }
            };
            for (int i = 1; i <= nvrt; i++)
                if (!in[i]) tarjan(tarjan, i);
            for (int i = 1; i <= nvar; i++) {
                if (id[i] == id[conj(i)]) return false;
                res[i] = id[i] > id[conj(i)];
            }
            return true;
        }
};
```

5 Maths

5.1 Matrix

```
template<typename type>
struct Matrix {
    vector <vector <type> > data;

    //  S    lng    hng    ca   ma   trn
    int row() const { return data.size(); }
    int col() const { return data[0].size(); }

    auto & operator [] (int i) { return data[i]; }

    const auto & operator[] (int i) const { return data[i]; }

    Matrix() = default;

    Matrix(int r, int c): data(r, vector <type> (c)) { }

    Matrix(const vector <vector <type> > &d): data(d) {

        //  Kim  tra  cc  hng   c   cng  size  khng  v  size  c
            ln   hn  0 hay khng
        // Tuy nhin  khng   thc    s    cn    thit , ta  c
            th    b    cc   dng  /**/  i
        /**/ assert(d.size());
        /**/ int size = d[0].size();
        /**/ assert(size);
        /**/ for (auto x : d) assert(x.size() == size);
    }

    // In ra ma  trn .
    friend ostream & operator << (ostream &out, const Matrix
         &d) {
        for (auto x : d.data) {
            for (auto y : x) out << y << ' ';
            out << '\n';
        }
        return out;
    }

    // Ma  trn    n     v
    static Matrix identity(long long n) {
        Matrix a = Matrix(n, n);
        while (n--) a[n][n] = 1;
        return a;
    }

    //  Nhn  ma  trn
    Matrix operator * (const Matrix &b) {
        Matrix a = *this;

        //  Kim  tra   iu    kin   nhn  ma  trn
        assert(a.col() == b.row());

        Matrix c(a.row(), b.col());
        for (int i = 0; i < a.row(); ++i)
            for (int j = 0; j < b.col(); ++j)
                for (int k = 0; k < a.col(); ++k)
                    c[i][j] += a[i][k] * b[k][j];
        return c;
    }

    // Ly   tha  ma  trn
    Matrix pow(long long exp) {

        //  Kim  tra   iu    kin   ly   tha  ma  trn  ( l  ma
              trn   vung )
        assert(row() == col());

        Matrix base = *this, ans = identity(row());
        for (; exp > 0; exp >>= 1, base = base * base)
            if (exp & 1) ans = ans * base;
        return ans;
    }
};
```

5.2 Mod Int

```
struct M {
    int MOD;
    int v;
    explicit operator int() const { return v; }
    M() {}
    M(int MOD) : v(0), MOD(MOD) {}
    M(int _v, int MOD) : MOD(MOD), v(_v % MOD) { v += (v < 0)
         * MOD; }

    M& operator+=(M o) {
        if ((v += o.v) >= MOD) v -= MOD;
        return *this;
    }
    M& operator-=(M o) {
        if ((v -= o.v) < 0) v += MOD;
        return *this;
    }
    M& operator*=(M o) {
        v = (int)((long long)v * o.v % MOD); // Use long long
             to prevent overflow
        return *this;
    }

    friend M binpow(M a, int p) {
        assert(p >= 0);
        return p == 0 ? M(1, a.MOD) : binpow(a * a, p / 2) *
             (p & 1 ? a : M(1, a.MOD));
    }
```

```
    friend M inv(M a) {
        assert(a.v != 0);
        return binpow(a, a.MOD - 2);
    }

    friend M operator+(M a, M b) { return a += b; }
    friend M operator-(M a, M b) { return a -= b; }
    friend M operator*(M a, M b) { return a *= b; }
    friend M operator/(M a, M b) { return a *= inv(b); }
    friend ostream& operator<<(ostream& out, M a) { return (
        out << a.v); }
};

template<int... MODS>
struct bigM {
    static const int size = sizeof...(MODS);
    M v[size];
    static constexpr array<int, size> mods = {MODS...};
    bigM() {
        for (int i = 0; i < size; ++i) {
            v[i] = M(mods[i]);
        }
    }
    bigM(int _v) {
        for (int i = 0; i < size; ++i) {
            v[i] = M(_v, mods[i]);
        }
    }

    bigM& operator+=(bigM o) {
        for (int i = 0; i < size; i++) {
            v[i] += o.v[i];
        }
        return *this;
    }
    bigM& operator-=(bigM o) {
        for (int i = 0; i < size; i++) {
            v[i] -= o.v[i];
        }
        return *this;
    }
    bigM& operator*=(bigM o) {
        for (int i = 0; i < size; i++) {
            v[i] *= o.v[i];
        }
        return *this;
    }
    bigM& operator/=(bigM o) {
        for (int i = 0; i < size; i++) {
            v[i] /= o.v[i];
        }
        return *this;
    }
    friend bigM binpow(bigM a, int p) {
        assert(p >= 0);
        return p == 0 ? bigM(1) : binpow(a * a, p / 2) * (p &
            1 ? a : bigM(1));
    }
    friend bigM inv(bigM a) {
        bigM res;
        for (int i = 0; i < size; i++) {
            res.v[i] = inv(a.mods[i]);
        }
        return res;
    }
    friend bool operator==(bigM a, bigM b) {
        for (int i = 0; i < size; i++) {
            if (a.v[i].v != b.v[i].v) return false;
        }
        return true;
    }
    friend bool operator!=(bigM a, bigM b) {
        return !(a == b);
    }
    friend bigM operator+(bigM a, bigM b) { return a += b; }
    friend bigM operator-(bigM a, bigM b) { return a -= b; }
    friend bigM operator*(bigM a, bigM b) { return a *= b; }
    friend bigM operator/(bigM a, bigM b) { return a /= b; }
};

using MM = bigM<1035972859, 1704760909, 1234567891>;
```

5.3 Precal Modulo Inverse

```
int n = 10, p = 1000000007;
int inv[n + 1];
inv[1] = 1;
for (int i = 2; i <= n; i ++) inv[i] = 1LL * (p - p / i) *
    inv[p % i] % p;
```

5.4 Rabin-Miller

```
using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(u64 n) { // returns true if n is prime,
    else returns false.
    if (n < 2)
        return false;

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
        37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

6 Miscellaneous

6.1 Clion

```
set(GCC_COVERAGE_COMPILE_FLAGS "-O2 -Dbinhball")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${
    GCC_COVERAGE_COMPILE_FLAGS}" )
```

6.2 Debug

```
#ifdef hvmegy
#define dbg(...) logger(#__VA_ARGS__, __VA_ARGS__)
template<typename ...Args>
void logger(string vars, Args&&... values) {
 cerr << "[" << vars << " : ";
 string delim = "";
 (..., (cerr << delim << values, delim = ", "));
 cerr << "]" << '\n';
}
#else
#define dbg(...)
#endif
```

6.3 mt19937

```
mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch
    ().count());
```

6.4 time

```
cerr << '\n' << clock() * 1000.0 / CLOCKS_PER_SEC << "ms" <<
     '\n';
```

7 Strings

7.1 Aho Corasick

```
struct aho_corasick{
    struct node{
        int suffix_link = -1, exit_link = -1, nxt[128];
        vector<int> leaf;
        node() {fill(nxt, nxt+128, -1);}
    };
    vector<node> g = {node()};
    void insert_string(const string &s, int sidx){
        int p = 0;
        for (char c: s){
            if (g[p].nxt[c] == -1){
                g[p].nxt[c] = g.size();
                g.emplace_back();
            }
            p = g[p].nxt[c];
        }
        g[p].leaf.push_back(sidx);
    }
    void build_automaton(){
        for (deque<int> q = {0}; q.size(); q.pop_front()){
            int v = q.front(), suffix_link = g[v].suffix_link
                ;
            if (v) g[v].exit_link = g[suffix_link].leaf.size
                () ? suffix_link : g[suffix_link].exit_link;
            for (int i=0; i<128; i++){
                int &nxt = g[v].nxt[i], nxt_sf = v ? g[
                    suffix_link].nxt[i] : 0;
                if (nxt == -1) nxt = nxt_sf;
                else{
                    g[nxt].suffix_link = nxt_sf;
                    q.push_back(nxt);
                }
            }
        }
    }
    vector<int> get_sindex(int p){
        vector<int> a;
        for (int v = g[p].leaf.size() ? p : g[p].exit_link; v
             != -1; v = g[v].exit_link)
            for (int j: g[v].leaf)
                a.push_back(j);
        return a;
    }
};
```

7.2 Hash

```
const int mod1 = 1035972859;
const int mod2 = 1704760909;
const int base = ;
struct hashing{
        int mod, n;
        vector<int> h, power;
        int binpow(int a, int n){
                if (n == 0)
                        return 1LL;
                int res = binpow(a, n/2);
                if (n % 2)
                        return res * res % mod * a % mod;
                return res * res % mod;
        }
        hashing() {}
        hashing(string s, int mod): mod(mod){
                this->n = s.length();
                power = h = vector<int> (n);
                power[0] = 1;
                h[0] = s[0];
                for (int i = 1; i < n; i++){
                        power[i] = power[i - 1] * base % mod;
                        h[i] = (h[i - 1] * base + s[i]) % mod;
                }
        }

        int getH(int l, int r){
                return (!l ? h[r] : (h[r] - h[l - 1] * power[r
                     - l + 1] + mod * mod) % mod);
        }
};

struct bighash{
        bighash() {}
        hashing a, b;
        bighash(string s){
                a = hashing(s, mod1);
                b = hashing(s, mod2);
        }
        int get(int l, int r){
                return a.getH(l, r) * b.getH(l, r);
        }
};
```

7.3 KMP

```
int next_x[N];
    int j = next_x[0] = 0;
    n = s.length();
    m = t.length();
    for (int i = 1; i < n; i++){
        while (j > 0 && s[j] != s[i]){
            j = next_x[j - 1];
        }
        if (s[i] == s[j])
            j++;
```

```
    next_x[i] = j;
}
j = 0;
for (int i = 0; i < m; i++){
    while (j > 0 && s[j] != t[i])
        j = next_x[j - 1];
    if (s[j] == t[i])
        j++;
    if (j == n){
        cout << i - n + 2 << " ";
    }
}
```

7.4 Z Function

```
z[0] = n;
    int l = 0, r = 0;
    for (int i = 1; i < n; i++){
        if (i > r){
            l = r = i;
            while (r < n && s[r] == s[r-l]) ++r;
            z[i] = r - l;
            r-=1;
        }
        else{
            int k= i - l;
            if (z[k] < r - i + 1) z[i] = z[k];
            else{
                l = i;
                while (r < n && s[r] == s[r - l]) r++;
                z[i]= r - l;
                r--;
            }
        }
    }
```