Contents

Sublime Build

```
{
    "cmd" : ["g++ -std=c++14 -DSONIC $file_name -o $file_base_name && timeout 4s ./$file_base_name<inputf.in>outputf.in"],
    "selector" : "source.cpp",
    "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
    "shell": true,
    "working_dir" : "$file_path"
}
```

vimrc

```
set mouse=a
  set termguicolors
  filetype plugin indent on
  syntax on

" Some useful settings
  set smartindent expandtab ignorecase smartcase incsearch relativenumber nowrap autoread splitright splitbelow
  set tabstop=4       "the width of a tab
  set shiftwidth=4    "the width for indent
  colorscheme torte

"auto pair curlybraces
  inoremap {<ENTER> {}<LEFT><CR><ESC><S-o>

" mapping jj to esc
  inoremap jj <ESC>

  "compile and run using file input put
  autocmd filetype cpp map <F5> :wa<CR>:!clear && g++ % -D LOCAL -std=c++17 -Wall -Wextra -Wconversion -Wshadow -Wfloat-equal -o ~/Codes/prog && (timeout 5 ~/Codes/prog < ~/Codes/in) > ~/Codes/out<CR>
  "copy to input file
  map <F4> :!xclip -o -sel clip > ~/Codes/in<CR><CR>
  map <F6> :vsplit ~/Codes/in<CR>:split ~/Codes/out<CR><C-w>=20<C-w><<C-w><C-h>

 " Leader key
  let mapleader=',,'

 " Copy template
  noremap <Leader>t :!cp ~/Codes/temp.cpp %<CR><CR>
  :autocmd BufNewFile *.cpp 0r ~/Codes/temp.cpp

  "note if vim-features +clipboard is not found, it will not work
  "for fast check :echo has('clipboard') = 0 if clipboard features not present,
  "need vim-gtk / vim-gtk3 package for this
  set clipboard=unnamedplus
```

1 All Macros

```cpp
//#pragma GCC optimize("Ofast")
//#pragma GCC optimization ("O3")
//#pragma comment(linker, "/stack:200000000")
//#pragma GCC optimize("unroll-loops")
//#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
    //find_by_order(k) --> returns iterator to the kth largest element counting from 0
    //order_of_key(val) --> returns the number of items in a set that are strictly smaller than our item
template <typename DT>
using ordered_set = tree <DT, null_type, less<DT>, rb_tree_tag, tree_order_statistics_node_update>;

/*--- DEBUG TEMPLATE STARTS HERE ---*/
#ifdef LEL
void show(int x) {cerr << x;}
void show(long long x) {cerr << x;}
void show(double x) {cerr << x;}
void show(char x) {cerr << '\'' << x << '\'';}
void show(const string &x) {cerr << '\"' << x << '\"';}
void show(bool x) {cerr << (x ? "true" : "false");}

template<typename T, typename V>
void show(pair<T, V> x) { cerr << '{'; show(x.first); cerr << ", "; show(x.second); cerr << '}'; }
template<typename T>
void show(T x) {int f = 0; cerr << "{"; for (auto &i: x) cerr << (f++ ? ", " : ""), show(i); cerr << "}";}

void debug_out(string s) {
    cerr << '\n';
}
template <typename T, typename... V>
void debug_out(string s, T t, V... v) {
    s.erase(remove(s.begin(), s.end(), ' '), s.end());
    cerr << "        "; // 8 spaces
    cerr << s.substr(0, s.find(','));
    s = s.substr(s.find(',') + 1);
    cerr << " = ";
    show(t);
    cerr << endl;
    if(sizeof...(v)) debug_out(s, v...);
}

#define debug(x...) cerr << "LINE: " << __LINE__ << endl; debug_out(#x, x); cerr << endl;
#else
#define debug(x...)
#endif
```

2 DP

2.1 1D-1D

```cpp
/// Author: anachor

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

/// Solves dp[i] = min(dp[j] + cost(j+1, i)) given that cost() is QF
long long solve1D(int n, long long cost(int, int)) {
  vector<long long> dp(n + 1), opt(n + 1);
  deque<pair<int, int>> dq;
  dq.push_back({0, 1});
  dp[0] = 0;

  for (int i = 1; i <= n; i++) {
    opt[i] = dq.front().first;
    dp[i] = dp[opt[i]] + cost(opt[i] + 1, i);
    if (i == n) break;

    dq[0].second++;
    if (dq.size() > 1 && dq[0].second == dq[1].second) dq.pop_front();

    int en = n;
    while (dq.size()) {
      int o = dq.back().first, st = dq.back().second;
      if (dp[o] + cost(o + 1, st) >= dp[i] + cost(i + 1, st))
        dq.pop_back();
      else {
        int lo = st, hi = en;
        while (lo < hi) {
          int mid = (lo + hi + 1) / 2;
          if (dp[o] + cost(o + 1, mid) < dp[i] + cost(i + 1, mid))
            lo = mid;
          else
            hi = mid - 1;
        }
        if (lo < n) dq.push_back({i, lo + 1});
        break;
      }
      en = st - 1;
    }
    if (dq.empty()) dq.push_back({i, i + 1});
  }
  return dp[n];
}

/// Solves https://open.kattis.com/problems/coveredwalkway
const int N = 1e6 + 7;
long long x[N];
int c;
long long cost(int l, int r) { return (x[r] - x[l]) * (x[r] - x[l]) + c; }

int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);

  int n;
  cin >> n >> c;
  for (int i = 1; i <= n; i++) cin >> x[i];
  cout << solve1D(n, cost) << endl;
}
```

2.2 Convex Hull Trick

```cpp
struct line {
  ll m, c;
  line() {}
  line(ll m, ll c) : m(m), c(c) {}
};
struct convex_hull_trick {
  vector<line> lines;
  int ptr = 0;
  convex_hull_trick() {}
  bool bad(line a, line b, line c) {
    return 1.0 * (c.c - a.c) * (a.m - b.m) < 1.0 * (b.c - a.c) * (a.m - c.m);
  }
  void add(line L) {
    int sz = lines.size();
    while (sz >= 2 && bad(lines[sz - 2], lines[sz - 1], L)) {
      lines.pop_back();
      sz--;
    }
    lines.pb(L);
  }
  ll get(int idx, int x) { return (1ll * lines[idx].m * x + lines[idx].c); }
  ll query(int x) {
    if (lines.empty()) return 0;
    if (ptr >= lines.size()) ptr = lines.size() - 1;
    while (ptr < lines.size() - 1 && get(ptr, x) > get(ptr + 1, x)) ptr++;
    return get(ptr, x);
  }
};
ll sum[MAX];
ll dp[MAX];
int arr[MAX];
int main() {
  fastio;
  int t;
  cin >> t;
  while (t--) {
    int n, a, b, c;
    cin >> n >> a >> b >> c;
    for (int i = 1; i <= n; i++) cin >> sum[i];
    for (int i = 1; i <= n; i++) dp[i] = 0, sum[i] += sum[i - 1];
```

```
    convex_hull_trick cht;
    cht.add(line(0, 0));
    for (int pos = 1; pos <= n; pos++) {
      dp[pos] = cht.query(sum[pos]) - 1ll *
          a * sqr(sum[pos]) - c;
      cht.add(line(2ll * a * sum[pos], dp[
          pos] - a * sqr(sum[pos])));
    }
    ll ans = (-1ll * dp[n]);
    ans += (1ll * sum[n] * b);
    cout << ans << "\n";
  }
}
```

2.3 Divide and Conquer dp

```
const int K = 805, N = 4005;
LL dp[2][N], _cost[N][N];
// 1-indexed for convenience
LL cost(int l, int r) {
  return _cost[r][r] - _cost[l - 1][r] -
      _cost[r][l - 1] + _cost[l - 1][l - 1]
      >> 1;
}
void compute(int cnt, int l, int r, int optl
    , int optr) {
  if (l > r) return;
  int mid = l + r >> 1;
  LL best = INT_MAX;
  int opt = -1;
  for (int i = optl; i <= min(mid, optr); i
      ++) {
    LL cur = dp[cnt ^ 1][i - 1] + cost(i,
        mid);
    if (cur < best) best = cur, opt = i;
  }
  dp[cnt][mid] = best;
  compute(cnt, l, mid - 1, optl, opt);
  compute(cnt, mid + 1, r, opt, optr);
}
LL dnc_dp(int k, int n) {
  fill(dp[0] + 1, dp[0] + n + 1, INT_MAX);
  for (int cnt = 1; cnt <= k; cnt++) {
    compute(cnt & 1, 1, n, 1, n);
  }
  return dp[k & 1][n];
}
```

2.4 Knuth optimization

```
const int N = 1005;
LL dp[N][N], a[N];
int opt[N][N];
LL cost(int i, int j) { return a[j + 1] - a[
    i]; }
LL knuth_optimization(int n) {
  for (int i = 0; i < n; i++) {
    dp[i][i] = 0;
    opt[i][i] = i;
  }
  for (int i = n - 2; i >= 0; i--) {
    for (int j = i + 1; j < n; j++) {
      LL mn = LLONG_MAX;
      LL c = cost(i, j);
      for (int k = opt[i][j - 1]; k <= min(j
          - 1, opt[i + 1][j]); k++) {
        if (mn > dp[i][k] + dp[k + 1][j] + c
            ) {
          mn = dp[i][k] + dp[k + 1][j] + c;
          opt[i][j] = k;
        }
      }
      dp[i][j] = mn;
    }
  }
  return dp[0][n - 1];
}
```

2.5 Li Chao Tree

```
struct line {
  LL m, c;
  line(LL m = 0, LL c = 0) : m(m), c(c) {}
};
LL calc(line L, LL x) { return 1LL * L.m * x
    + L.c; }
struct node {
  LL m, c;
  line L;
  node *lft, *rt;
  node(LL m = 0, LL c = 0, node *lft = NULL,
      node *rt = NULL)
      : L(line(m, c)), lft(lft), rt(rt) {}
};
struct LiChao {
  node *root;
  LiChao() { root = new node(); }
  void update(node *now, int L, int R, line
      newline) {
    int mid = L + (R - L) / 2;
    line lo = now->L, hi = newline;
    if (calc(lo, L) > calc(hi, L)) swap(lo,
        hi);
    if (calc(lo, R) <= calc(hi, R)) {
      now->L = hi;
      return;
    }
    if (calc(lo, mid) < calc(hi, mid)) {
      now->L = hi;
      if (now->rt == NULL) now->rt = new
          node();
      update(now->rt, mid + 1, R, lo);
    } else {
      now->L = lo;
      if (now->lft == NULL) now->lft = new
          node();
      update(now->lft, L, mid, hi);
    }
  }
  LL query(node *now, int L, int R, LL x) {
    if (now == NULL) return -inf;
    int mid = L + (R - L) / 2;
    if (x <= mid)
      return max(calc(now->L, x), query(now
          ->lft, L, mid, x));
    else
      return max(calc(now->L, x), query(now
          ->rt, mid + 1, R, x));
  }
};
```

3 Data Structure

3.1 BIT-2D

```
const int N = 1008;
int bit[N][N], n, m;
int a[N][N], q;
void update(int x, int y, int val) {
  for (; x < N; x += -x & x)
    for (int j = y; j < N; j += -j & j) bit[
        x][j] += val;
}
int get(int x, int y) {
  int ans = 0;
  for (; x; x -= x & -x)
    for (int j = y; j; j -= j & -j) ans +=
        bit[x][j];
  return ans;
}
int get(int x1, int y1, int x2, int y2) {
  return get(x2, y2) - get(x1 - 1, y2) - get
      (x2, y1 - 1) + get(x1 - 1, y1 - 1);
}
```

3.2 Divide And Conquer Query Offline

```
namespace up {
int l[N], r[N], u[N], v[N], tm;
void push(int _l, int _r, int _u, int _v) {
  l[tm] = _l, r[tm] = _r, u[tm] = _u, v[tm]
      = _v;
  tm++;
}
}  // namespace up
namespace que {
int node[N], tm;
LL ans[N];
void push(int _node) { node[++tm] = _node; }
}  // namespace que
namespace edge_set {
void push(int i) { dsu ::merge(up ::u[i], up
    ::v[i]); }
void pop(int t) { dsu ::rollback(t); }
int time() { return dsu ::op.size(); }
LL query(int u) { return a[dsu ::root(u)]; }
}  // namespace edge_set
namespace dncq {
vector<int> tree[4 * N];
void update(int idx, int l = 0, int r = que
    ::tm, int node = 1) {
  int ul = up ::l[idx], ur = up ::r[idx];
  if (r < ul or ur < l) return;
  if (ul <= l and r <= ur) {
    tree[node].push_back(idx);
    return;
  }
  int m = l + r >> 1;
  update(idx, l, m, node << 1);
  update(idx, m + 1, r, node << 1 | 1);
}
void dfs(int l = 0, int r = que ::tm, int
    node = 1) {
  int cur = edge_set ::time();
  for (int e : tree[node]) edge_set ::push(e
      );
  if (l == r) {
    que ::ans[l] = edge_set ::query(que ::
        node[l]);
  } else {
    int m = l + r >> 1;
    dfs(l, m, node << 1);
    dfs(m + 1, r, node << 1 | 1);
  }
  edge_set ::pop(cur);
}
}  // namespace dncq
void push_updates() {
  for (int i = 0; i < up ::tm; i++) dncq ::
      update(i);
}
```

3.3 MO with Update

```
const int N = 1e5 + 5, sz = 2700, bs = 25;
int arr[N], freq[2 * N], cnt[2 * N], id[N],
    ans[N];
struct query {
  int l, r, t, L, R;
  query(int l = 1, int r = 0, int t = 1, int
      id = -1)
      : l(l), r(r), t(t), L(l / sz), R(r /
          sz) {}
  bool operator<(const query &rhs) const {
    return (L < rhs.L) or (L == rhs.L and R
        < rhs.R) or
          (L == rhs.L and R == rhs.R and t <
              rhs.t);
  }
} Q[N];
struct update {
  int idx, val, last;
} Up[N];
int qi = 0, ui = 0;
int l = 1, r = 0, t = 0;

void add(int idx) {
  --cnt[freq[arr[idx]]];
  freq[arr[idx]]++;
  cnt[freq[arr[idx]]]++;
}
void remove(int idx) {
  --cnt[freq[arr[idx]]];
  freq[arr[idx]]--;
  cnt[freq[arr[idx]]]++;
}
void apply(int t) {
  const bool f = l <= Up[t].idx and Up[t].
      idx <= r;
  if (f) remove(Up[t].idx);
  arr[Up[t].idx] = Up[t].val;
  if (f) add(Up[t].idx);
}
void undo(int t) {
```

```
  const bool f = l <= Up[t].idx and Up[t].
      idx <= r;
  if (f) remove(Up[t].idx);
  arr[Up[t].idx] = Up[t].last;
  if (f) add(Up[t].idx);
}
int mex() {
  for (int i = 1; i <= N; i++)
    if (!cnt[i]) return i;
  assert(0);
}
int main() {
  int n, q;
  cin >> n >> q;
  int counter = 0;
  map<int, int> M;
  for (int i = 1; i <= n; i++) {
    cin >> arr[i];
    if (!M[arr[i]]) M[arr[i]] = ++counter;
    arr[i] = M[arr[i]];
  }
  iota(id, id + N, 0);
  while (q--) {
    int tp, x, y;
    cin >> tp >> x >> y;
    if (tp == 1)
      Q[++qi] = query(x, y, ui);
    else {
      if (!M[y]) M[y] = ++counter;
      y = M[y];
      Up[++ui] = {x, y, arr[x]};
      arr[x] = y;
    }
  }
  t = ui;
  cnt[0] = 3 * n;
  sort(id + 1, id + qi + 1, [&](int x, int y
      ) { return Q[x] < Q[y]; });
  for (int i = 1; i <= qi; i++) {
    int x = id[i];
    while (Q[x].t > t) apply(++t);
    while (Q[x].t < t) undo(t--);
    while (Q[x].l < l) add(--l);
    while (Q[x].r > r) add(++r);
    while (Q[x].l > l) remove(l++);
    while (Q[x].r < r) remove(r--);
    ans[x] = mex();
  }
  for (int i = 1; i <= qi; i++) cout << ans[
      i] << '\n';
}
```

3.4 Merge Sort Tree

```
vector<LL> Tree[4 * MAXN];
LL arr[MAXN];

vector<LL> merge(vector<LL> v1, vector<LL>
    v2) {
  LL i = 0, j = 0;
  vector<LL> ret;

  while (i < v1.size() || j < v2.size()) {
    if (i == v1.size()) {
      ret.push_back(v2[j]);
      j++;
    } else if (j == v2.size()) {
      ret.push_back(v1[i]);
      i++;
    } else {
      if (v1[i] < v2[j]) {
        ret.push_back(v1[i]);
        i++;
      } else {
        ret.push_back(v2[j]);
        j++;
      }
    }
  }

  return ret;
}

void Build(LL node, LL bg, LL ed) {
  if (bg == ed) {
    Tree[node].push_back(arr[bg]);
    return;
  }

  LL leftNode = 2 * node, rightNode = 2 *
      node + 1;
  LL mid = (bg + ed) / 2;

  Build(leftNode, bg, mid);
  Build(rightNode, mid + 1, ed);

  Tree[node] = merge(Tree[leftNode], Tree[
      rightNode]);
}

LL query(LL node, LL bg, LL ed, LL l, LL r,
    LL k) {
  if (ed < l || bg > r) return 0;

  if (l <= bg && ed <= r)
    return upper_bound(Tree[node].begin(),
        Tree[node].end(), k) -
          Tree[node].begin();

  LL leftNode = 2 * node, rightNode = 2 *
      node + 1;
  LL mid = (bg + ed) / 2;

  return query(leftNode, bg, mid, l, r, k) +
         query(rightNode, mid + 1, ed, l, r,
            k);
}
```

3.5 SparseTable (Rectangle Query)

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

const int MAXN = 505;
const int LOGN = 9;

// O(n^2 (logn)^2
// Supports Rectangular Query
int A[MAXN][MAXN];
int M[MAXN][MAXN][LOGN][LOGN];

void Build2DSparse(int N) {
  for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
      M[i][j][0][0] = A[i][j];
    }
    for (int q = 1; (1 << q) <= N; q++) {
      int add = 1 << (q - 1);
      for (int j = 1; j + add <= N; j++) {
        M[i][j][0][q] = max(M[i][j][0][q -
            1], M[i][j + add][0][q - 1]);
      }
    }
  }

  for (int p = 1; (1 << p) <= N; p++) {
    int add = 1 << (p - 1);
    for (int i = 1; i + add <= N; i++) {
      for (int q = 0; (1 << q) <= N; q++) {
        for (int j = 1; j <= N; j++) {
          M[i][j][p][q] = max(M[i][j][p - 1][
              q], M[i + add][j][p - 1][q]);
        }
      }
    }
  }
}

// returns max of all A[i][j], where x1<=i<=
    x2 and y1<=j<=y2
int Query(int x1, int y1, int x2, int y2) {
  int kX = log2(x2 - x1 + 1);
  int kY = log2(y2 - y1 + 1);
  int addX = 1 << kX;
  int addY = 1 << kY;

  int ret1 = max(M[x1][y1][kX][kY], M[x1][y2
      - addY + 1][kX][kY]);
  int ret2 = max(M[x2 - addX + 1][y1][kX][kY
      ],
                 M[x2 - addX + 1][y2 - addY +
                     1][kX][kY]);
  return max(ret1, ret2);
}
```

3.6 Segment Tree

```
const int N = 1000006;

using DT = LL;
using LT = LL;
constexpr DT I = 0;
constexpr LT None = 0;
DT val[4 * N];
LT lazy[4 * N];
int L, R;

void pull(int s, int e, int node) {
  val[node] = val[node << 1] + val[node << 1
       | 1];
}
void apply(const LT &U, int s, int e, int
    node) {
  val[node] += (e - s + 1) * U;
  lazy[node] += U;
}
void reset(int node) { lazy[node] = None; }
DT merge(const DT &a, const DT &b) { return
    a + b; }
DT get(int s, int e, int node) { return val[
    node]; }
void push(int s, int e, int node) {
  if (s == e) return;
  apply(lazy[node], s, s + e >> 1, node <<
      1);
  apply(lazy[node], s + e + 2 >> 1, e, node
      << 1 | 1);
  reset(node);
}
void build(int s, int e, vector<DT> &v, int
    node = 1) {
  int m = s + e >> 1;
  if (s == e) {
    val[node] = v[s];
    return;
  }
  build(s, m, v, node * 2);
  build(m + 1, e, v, node * 2 + 1);
  pull(s, e, node);
}
void update(int S, int E, LT uval, int s = L
    , int e = R, int node = 1) {
  if (S > E) return;
  if (S == s and E == e) {
    apply(uval, s, e, node);
    return;
  }
  push(s, e, node);
  int m = s + e >> 1;
  update(S, min(m, E), uval, s, m, node * 2)
      ;
  update(max(S, m + 1), E, uval, m + 1, e,
      node * 2 + 1);
  pull(s, e, node);
}
DT query(int S, int E, int s = L, int e = R,
     int node = 1) {
  if (S > E) return I;
  if (s == S and e == E) return get(s, e,
      node);
  push(s, e, node);
  int m = s + e >> 1;
  DT L = query(S, min(m, E), s, m, node * 2)
      ;
  DT R = query(max(S, m + 1), E, m + 1, e,
      node * 2 + 1);
  return merge(L, R);
}
void init(int _L, int _R, vector<DT> &v) {
  L = _L, R = _R;
  build(L, R, v);
}
```

3.7 Persistent Segment Tree

```cpp
struct Node {
  Node *l, *r;
  int sum;

  Node(int val) : l(nullptr), r(nullptr), sum(val) {}
  Node(Node* l, Node* r) : l(l), r(r), sum(0) {
    if (l) sum += l->sum;
    if (r) sum += r->sum;
  }
};

int a[MAXN];
Node* root[MAXN];

Node* Build(int bg, int ed) {
  if (bg == ed) return new Node(a[bg]);
  int mid = (bg + ed) / 2;
  return new Node(Build(bg, mid), Build(mid + 1, ed));
}

int Query(Node* v, int bg, int ed, int l, int r) {
  if (l > ed || r < bg) return 0;
  if (l <= bg && ed <= r) return v->sum;
  int mid = (bg + ed) / 2;
  return Query(v->l, bg, mid, l, r) + Query(v->r, mid + 1, ed, l, r);
}

Node* Update(Node* v, int bg, int ed, int pos, int new_val) {
  if (bg == ed) return new Node(v->sum + new_val);
  int mid = (bg + ed) / 2;
  if (pos <= mid)
    return new Node(Update(v->l, bg, mid, pos, new_val), v->r);
  else
    return new Node(v->l, Update(v->r, mid + 1, ed, pos, new_val));
}
```

3.8 Implicit Segment Tree

```cpp
struct node {
  int val;
  node *lft, *rt;
  node() {}
  node(int val = 0) : val(val), lft(NULL), rt(NULL) {}
};

struct implicit_segtree {
  node *root;
  implicit_segtree() {}
  implicit_segtree(int n) {
    root = new node(n);
  }
  void update(node *now, int L, int R, int idx, int val) {
    if (L == R) {
      now -> val += val;
      return;
    }
    int mid = L + (R - L) / 2;
    if (now->lft == NULL) now->lft = new node(mid - L + 1);
    if (now->rt == NULL) now->rt = new node(R - mid);
    if (idx <= mid) update(now->lft, L, mid, idx, val);
    else update(now->rt, mid + 1, R, idx, val);
    now->val = (now->lft)->val + (now->rt)->val;
  }

  int query(node *now, int L, int R, int k) {
    if (L == R) return L;
    int mid = L + (R - L) / 2;
    if (now->lft == NULL) now->lft = new node(mid - L + 1);
    if (now->rt == NULL) now->rt = new node(R - mid);
    if (k <= (now->lft)->val) return query(now->lft, L, mid, k);
    else return query(now->rt, mid + 1, R, k - (now->lft)->val);
  }
};
```

3.9 HashTable

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
using namespace std;
using namespace __gnu_pbds;

const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch().count();
unsigned hash_f(unsigned x) {
  x = ((x >> 16) ^ x) * 0x45d9f3b;
  x = ((x >> 16) ^ x) * 0x45d9f3b;
  x = (x >> 16) ^ x;
  return x;
}

unsigned hash_combine(unsigned a, unsigned b) { return a * 31 + b; }
struct chash {
  int operator()(int x) const { return hash_f(x); }
};
typedef gp_hash_table<int, int, chash> gp;
gp table;
```

3.10 Treap

```cpp
mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
typedef struct node* pnode;
struct node {
  int prior, val, sz;
  ll sum;
  node *lft, *rt;
  node(int val = 0, node *lft = NULL, node *rt = NULL) :
    lft(lft), rt(rt), prior(rnd()), val(val), sz(1), sum(0) {}
};
struct treap {
  pnode root;
  treap() {
    root = NULL;
  }
  int get_sz(pnode now) {
    return now ? now->sz : 0;
  }
  void update_sz(pnode now) {
    if (!now) return;
    now->sz = 1 + get_sz(now->lft) + get_sz(now->rt);
  }
  ll get(pnode now) {
    return now ? now->sum : 0;
  }
  void push(pnode now) {}
  void combine(pnode now) {
    if (!now) return;
    now->sum = now->val + get(now->lft) + get(now->rt);
  }
  pnode unite(pnode lft, pnode rt) {
    if (!lft || !rt) return lft ? lft : rt;
    // push(lft), push(rt); this not tested
    if (lft->prior < rt->prior) swap(lft, rt);
    pnode l, r;
    split(rt, l, r, lft->val);
    lft->lft = unite(lft->lft, l), update_sz(lft);
    lft->rt = unite(lft->rt, r), update_sz(lft);
    // combine(lft); this not tested
    return lft;
  }
  ///value < val goes to left, value >= val goes to right
  void split(pnode now, pnode &lft, pnode &rt, int val, int add = 0) {
    push(now);
    if (!now) return void(lft = rt = NULL);
    if (now->val < val) split(now->rt, now->rt, rt, val), lft = now;
    else split(now->lft, lft, now->lft, val), rt = now;
    update_sz(now), combine(now);
  }
  void merge(pnode &now, pnode lft, pnode rt) {
    push(lft), push(rt);
    if (!lft || !rt) now = lft ? lft : rt;
    else if (lft->prior > rt->prior) merge(lft->rt, lft->rt, rt), now = lft;
    else merge(rt->lft, lft, rt->lft), now = rt;
    update_sz(now), combine(now);
  }
  void insert(pnode &now, pnode notun) {
    if (!now) return void(now = notun);
    push(now);
    if (notun->prior > now->prior) split(now, notun->lft, notun->rt, notun->val), now = notun;
    else insert(notun->val < now->val ? now->lft : now->rt, notun);
    update_sz(now), combine(now);
  }
  void erase(pnode &now, int val) {
    push(now);
    if (now->val == val) {
      pnode temp = now;
      merge(now, now->lft, now->rt);
      delete(temp);
    } else erase(val < now->val ? now->lft : now->rt, val);
    update_sz(now), combine(now);
  }
  int get_idx(pnode &now, int val) {
    if (!now) return INT_MIN;
    else if (now->val == val) return 1 + get_sz(now->lft);
    else if (val < now->val) return get_idx(now->lft, val);
    else return (1 + get_sz(now->lft) + get_idx(now->rt, val));
  }
  int find_kth(pnode &now, int k) {
    if (k < 1 || k > get_sz(now)) return -1;
    if (get_sz(now->lft) + 1 == k) return now->val;
    if (k <= get_sz(now->lft)) return find_kth(now->lft, k);
    return find_kth(now->rt, k - get_sz(now->lft) - 1);
  }
  ll prefix_sum(pnode &now, int k) {
    if (k < 1 || k > get_sz(now)) return -inf;
    if (get_sz(now->lft) + 1 == k) return get(now->lft) + now->val;
    if (k <= get_sz(now->lft)) return prefix_sum(now->lft, k);
    return get(now->lft) + now->val + prefix_sum(now->rt, k - get_sz(now->lft) - 1);
  }
  pnode get_rng(int l, int r) { ///gets all l <= values <= r
    pnode lft, rt, mid;
    split(root, lft, mid, l);
    split(mid, mid, rt, r + 1);
    merge(root, lft, rt);
    return mid;
  }
  void output(pnode now, vector<int>&v) {
    if (!now) return;
    output(now->lft, v);
```

```
    v.pb(now->val);
    output(now->rt, v);
  }
  vector<int>get_arr() {
    vector<int>ret;
    output(root, ret);
    return ret;
  }
};
```

3.11 DSU With Rollbacks

```
struct Rollback_DSU {
  int n;
  vector<int> par, sz;
  vector<pair<int, int>> op;
  Rollback_DSU(int n) : par(n), sz(n, 1) {
    iota(par.begin(), par.end(), 0);
    op.reserve(n);
  }
  int Anc(int node) {
    for (; node != par[node]; node = par[
        node])
      ; // no path compression
    return node;
  }
  void Unite(int x, int y) {
    if (sz[x = Anc(x)] < sz[y = Anc(y)])
        swap(x, y);
    op.emplace_back(x, y);
    par[y] = x;
    sz[x] += sz[y];
  }
  void Undo(int t) {
    for (; op.size() > t; op.pop_back()) {
      par[op.back().second] = op.back().
          second;
      sz[op.back().first] -= sz[op.back().
          second];
    }
  }
};
```

3.12 Binary Trie

```
const int N = 1e7 + 5, b = 30;
int tc = 1;
struct node {
  int vis = 0;
  int to[2] = {0, 0};
  int val[2] = {0, 0};
  void update() {
    to[0] = to[1] = 0;
    val[0] = val[1] = 0;
    vis = tc;
  }
} T[N + 2];
node *root = T;
int ptr = 0;
node *nxt(node *cur, int x) {
  if (cur->to[x] == 0) cur->to[x] = ++ptr;
  assert(ptr < N);
  int idx = cur->to[x];
  if (T[idx].vis < tc) T[idx].update();
  return T + idx;
}
int query(int j, int aj) {
  int ans = 0, jaj = j ^ aj;
  node *cur = root;
  for (int k = b - 1; ~k; k--) {
    maximize(ans, nxt(cur, (jaj >> k & 1) ^
        1)->val[1 ^ (aj >> k & 1)]);
    cur = nxt(cur, (jaj >> k & 1));
  }
  return ans;
}
void insert(int j, int aj, int val) {
  int jaj = j ^ aj;
  node *cur = root;
  for (int k = b - 1; ~k; k--) {
    cur = nxt(cur, (jaj >> k & 1));
    maximize(cur->val[j >> k & 1], val);
  }
}
void clear() {
  tc++;
  ptr = 0;
  root->update();
}
```

4 Geometry

4.1 Point

```
typedef double Tf;
typedef double Ti; /// use long long for
    exactness
const Tf PI = acos(-1), EPS = 1e-9;
int dcmp(Tf x) { return abs(x) < EPS ? 0 : (
    x < 0 ? -1 : 1); }

struct Point {
  Ti x, y;
  Point(Ti x = 0, Ti y = 0) : x(x), y(y) {}

  Point operator+(const Point& u) const {
      return Point(x + u.x, y + u.y); }
  Point operator-(const Point& u) const {
      return Point(x - u.x, y - u.y); }
  Point operator*(const LL u) const { return
       Point(x * u, y * u); }
  Point operator*(const Tf u) const { return
       Point(x * u, y * u); }
  Point operator/(const Tf u) const { return
       Point(x / u, y / u); }

  bool operator==(const Point& u) const {
    return dcmp(x - u.x) == 0 && dcmp(y - u.
        y) == 0;
  }
  bool operator!=(const Point& u) const {
      return !(*this == u); }
  bool operator<(const Point& u) const {
    return dcmp(x - u.x) < 0 || (dcmp(x - u.
        x) == 0 && dcmp(y - u.y) < 0);
  }
};
Ti dot(Point a, Point b) { return a.x * b.x
    + a.y * b.y; }
Ti cross(Point a, Point b) { return a.x * b.
    y - a.y * b.x; }
Tf length(Point a) { return sqrt(dot(a, a));
     }
Ti sqLength(Point a) { return dot(a, a); }
Tf distance(Point a, Point b) { return
    length(a - b); }
Tf angle(Point u) { return atan2(u.y, u.x);
    }

// returns angle between oa, ob in (-PI, PI]
Tf angleBetween(Point a, Point b) {
  Tf ans = angle(b) - angle(a);
  return ans <= -PI ? ans + 2 * PI : (ans >
      PI ? ans - 2 * PI : ans);
}
// Rotate a ccw by rad radians, Tf Ti same
Point rotate(Point a, Tf rad) {
  return Point(a.x * cos(rad) - a.y * sin(
      rad),
               a.x * sin(rad) + a.y * cos(rad
                   ));
}
// rotate a ccw by angle th with cos(th) =
    co && sin(th) = si, tf ti same
Point rotatePrecise(Point a, Tf co, Tf si) {
  return Point(a.x * co - a.y * si, a.y * co
      + a.x * si);
}
Point rotate90(Point a) { return Point(-a.y,
    a.x); }
// scales vector a by s such that length of
    a becomes s, Tf Ti same
Point scale(Point a, Tf s) { return a /
    length(a) * s; }
// returns an unit vector perpendicular to
    vector a, Tf Ti same
Point normal(Point a) {
  Tf l = length(a);
  return Point(-a.y / l, a.x / l);
}
// returns 1 if c is left of ab, 0 if on ab
    && -1 if right of ab
int orient(Point a, Point b, Point c) {
  return dcmp(cross(b - a, c - a)); }
/// Use as sort(v.begin(), v.end(),
    polarComp(O, dir))
/// Polar comparator around O starting at
    direction dir
struct polarComp {
  Point O, dir;
  polarComp(Point O = Point(0, 0), Point dir
       = Point(1, 0)) : O(O), dir(dir) {}
  bool half(Point p) {
    return dcmp(cross(dir, p)) < 0 ||
        (dcmp(cross(dir, p)) == 0 && dcmp(
            dot(dir, p)) > 0);
  }
  bool operator()(Point p, Point q) {
    return make_tuple(half(p), 0) <
        make_tuple(half(q), cross(p, q));
  }
};
struct Segment {
  Point a, b;
  Segment(Point aa, Point bb) : a(aa), b(bb)
       {}
};
typedef Segment Line;
struct Circle {
  Point o;
  Tf r;
  Circle(Point o = Point(0, 0), Tf r = 0) :
      o(o), r(r) {}
  // returns true if point p is in || on the
       circle
  bool contains(Point p) { return dcmp(
      sqLength(p - o) - r * r) <= 0; }
  // returns a point on the circle rad
      radians away from +X CCW
  Point point(Tf rad) {
    static_assert(is_same<Tf, Ti>::value);
    return Point(o.x + cos(rad) * r, o.y +
        sin(rad) * r);
  }
  // area of a circular sector with central
      angle rad
  Tf area(Tf rad = PI + PI) { return rad * r
       * r / 2; }
  // area of the circular sector cut by a
      chord with central angle alpha
  Tf sector(Tf alpha) { return r * r * 0.5 *
       (alpha - sin(alpha)); }
};
```

4.2 Linear

```
// **** LINE LINE INTERSECTION START ****
// returns true if point p is on segment s
bool onSegment(Point p, Segment s) {
  return dcmp(cross(s.a - p, s.b - p)) == 0
      && dcmp(dot(s.a - p, s.b - p)) <= 0;
}
// returns true if segment p && q touch or
    intersect
bool segmentsIntersect(Segment p, Segment q)
     {
  if (onSegment(p.a, q) || onSegment(p.b, q)
      ) return true;
  if (onSegment(q.a, p) || onSegment(q.b, p)
      ) return true;

  Ti c1 = cross(p.b - p.a, q.a - p.a);
  Ti c2 = cross(p.b - p.a, q.b - p.a);
  Ti c3 = cross(q.b - q.a, p.a - q.a);
  Ti c4 = cross(q.b - q.a, p.b - q.a);
  return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3)
       * dcmp(c4) < 0;
}
bool linesParallel(Line p, Line q) {
  return dcmp(cross(p.b - p.a, q.b - q.a))
      == 0;
}
// lines are represented as a ray from a
    point: (point, vector)
```

```
// returns false if two lines (p, v) && (q, w) are parallel or collinear
// true otherwise, intersection point is stored at o via reference, Tf Ti Same
bool lineLineIntersection(Point p, Point v, Point q, Point w, Point& o) {
  if (dcmp(cross(v, w)) == 0) return false;
  Point u = p - q;
  o = p + v * (cross(w, u) / cross(v, w));
  return true;
}
// returns false if two lines p && q are parallel or collinear
// true otherwise, intersection point is stored at o via reference
bool lineLineIntersection(Line p, Line q, Point& o) {
  return lineLineIntersection(p.a, p.b - p.a, q.a, q.b - q.a, o);
}
// returns the distance from point a to line l
// **** LINE LINE INTERSECTION FINISH ****
Tf distancePointLine(Point p, Line l) {
  return abs(cross(l.b - l.a, p - l.a) / length(l.b - l.a));
}
// returns the shortest distance from point a to segment s
Tf distancePointSegment(Point p, Segment s) {
  if (s.a == s.b) return length(p - s.a);
  Point v1 = s.b - s.a, v2 = p - s.a, v3 = p - s.b;
  if (dcmp(dot(v1, v2)) < 0)
    return length(v2);
  else if (dcmp(dot(v1, v3)) > 0)
    return length(v3);
  else
    return abs(cross(v1, v2) / length(v1));
}
// returns the shortest distance from segment p to segment q
Tf distanceSegmentSegment(Segment p, Segment q) {
  if (segmentsIntersect(p, q)) return 0;
  Tf ans = distancePointSegment(p.a, q);
  ans = min(ans, distancePointSegment(p.b, q));
  ans = min(ans, distancePointSegment(q.a, p));
  ans = min(ans, distancePointSegment(q.b, p));
  return ans;
}
// returns the projection of point p on line l, Tf Ti Same
Point projectPointLine(Point p, Line l) {
  Point v = l.b - l.a;
  return l.a + v * ((Tf)dot(v, p - l.a) / dot(v, v));
}
```

4.3 Circular

```
// Extremely inaccurate for finding near touches
// compute intersection of line l with circle c
// The intersections are given in order of the ray (l.a, l.b), Tf Ti same
vector<Point> circleLineIntersection(Circle c, Line l) {
  vector<Point> ret;
  Point b = l.b - l.a, a = l.a - c.o;
  Tf A = dot(b, b), B = dot(a, b);
  Tf C = dot(a, a) - c.r * c.r, D = B * B - A * C;
  if (D < -EPS) return ret;
  ret.push_back(l.a + b * (-B - sqrt(D + EPS)) / A);
  if (D > EPS) ret.push_back(l.a + b * (-B + sqrt(D)) / A);
  return ret;
}
// signed area of intersection of circle(c.o, c.r) &&
// triangle(c.o, s.a, s.b) [cross(a-o, b-o)/2]
Tf circleTriangleIntersectionArea(Circle c, Segment s) {
  using Linear::distancePointSegment;
  Tf OA = length(c.o - s.a);
  Tf OB = length(c.o - s.b);
  // sector
  if (dcmp(distancePointSegment(c.o, s) - c.r) >= 0)
    return angleBetween(s.a - c.o, s.b - c.o) * (c.r * c.r) / 2.0;
  // triangle
  if (dcmp(OA - c.r) <= 0 && dcmp(OB - c.r) <= 0)
    return cross(c.o - s.b, s.a - s.b) / 2.0;
  // three part: (A, a) (a, b) (b, B)
  vector<Point> Sect = circleLineIntersection(c, s);
  return circleTriangleIntersectionArea(c, Segment(s.a, Sect[0])) +
      circleTriangleIntersectionArea(c, Segment(Sect[0], Sect[1])) +
      circleTriangleIntersectionArea(c, Segment(Sect[1], s.b));
}
// area of intersecion of circle(c.o, c.r) && simple polyson(p[])
Tf circlePolyIntersectionArea(Circle c, Polygon p) {
  Tf res = 0;
  int n = p.size();
  for (int i = 0; i < n; ++i)
    res += circleTriangleIntersectionArea(c, Segment(p[i], p[(i + 1) % n]));
  return abs(res);
}
// locates circle c2 relative to c1
// interior             (d < R - r)         ----> -2
// interior tangents (d = R - r)         ----> -1
// concentric        (d = 0)
// secants              (R - r < d < R + r) ----> 0
// exterior tangents (d = R + r)         ----> 1
// exterior             (d > R + r)         ----> 2
int circleCirclePosition(Circle c1, Circle c2) {
  Tf d = length(c1.o - c2.o);
  int in = dcmp(d - abs(c1.r - c2.r)), ex = dcmp(d - (c1.r + c2.r));
  return in < 0 ? -2 : in == 0 ? -1 : ex == 0 ? 1 : ex > 0 ? 2 : 0;
}
// compute the intersection points between two circles c1 && c2, Tf Ti same
vector<Point> circleCircleIntersection(Circle c1, Circle c2) {
  vector<Point> ret;
  Tf d = length(c1.o - c2.o);
  if (dcmp(d) == 0) return ret;
  if (dcmp(c1.r + c2.r - d) < 0) return ret;
  if (dcmp(abs(c1.r - c2.r) - d) > 0) return ret;

  Point v = c2.o - c1.o;
  Tf co = (c1.r * c1.r + sqLength(v) - c2.r * c2.r) / (2 * c1.r * length(v));
  Tf si = sqrt(abs(1.0 - co * co));
  Point p1 = scale(rotatePrecise(v, co, -si), c1.r) + c1.o;
  Point p2 = scale(rotatePrecise(v, co, si), c1.r) + c1.o;

  ret.push_back(p1);
  if (p1 != p2) ret.push_back(p2);
  return ret;
}
// intersection area between two circles c1, c2
Tf circleCircleIntersectionArea(Circle c1, Circle c2) {
  Point AB = c2.o - c1.o;
  Tf d = length(AB);
  if (d >= c1.r + c2.r) return 0;
  if (d + c1.r <= c2.r) return PI * c1.r * c1.r;
  if (d + c2.r <= c1.r) return PI * c2.r * c2.r;

  Tf alpha1 = acos((c1.r * c1.r + d * d - c2.r * c2.r) / (2.0 * c1.r * d));
  Tf alpha2 = acos((c2.r * c2.r + d * d - c1.r * c1.r) / (2.0 * c2.r * d));
  return c1.sector(2 * alpha1) + c2.sector(2 * alpha2);
}
// returns tangents from a point p to circle c, Tf Ti same
vector<Point> pointCircleTangents(Point p, Circle c) {
  vector<Point> ret;
  Point u = c.o - p;
  Tf d = length(u);
  if (d < c.r)
    ;
  else if (dcmp(d - c.r) == 0) {
    ret = {rotate(u, PI / 2)};
  } else {
    Tf ang = asin(c.r / d);
    ret = {rotate(u, -ang), rotate(u, ang)};
  }
  return ret;
}
// returns the points on tangents that touches the circle, Tf Ti Same
vector<Point> pointCircleTangencyPoints(Point p, Circle c) {
  Point u = p - c.o;
  Tf d = length(u);
  if (d < c.r)
    return {};
  else if (dcmp(d - c.r) == 0)
    return {c.o + u};
  else {
    Tf ang = acos(c.r / d);
    u = u / length(u) * c.r;
    return {c.o + rotate(u, -ang), c.o + rotate(u, ang)};
  }
}
// for two circles c1 && c2, returns two list of points a && b
// such that a[i] is on c1 && b[i] is c2 && for every i
// Line(a[i], b[i]) is a tangent to both circles
// CAUTION: a[i] = b[i] in case they touch | -1 for c1 = c2
int circleCircleTangencyPoints(Circle c1, Circle c2, vector<Point> &a,
                               vector<Point> &b) {
  a.clear(), b.clear();
  int cnt = 0;
  if (dcmp(c1.r - c2.r) < 0) {
    swap(c1, c2);
    swap(a, b);
  }
  Tf d2 = sqLength(c1.o - c2.o);
  Tf rdif = c1.r - c2.r, rsum = c1.r + c2.r;
  if (dcmp(d2 - rdif * rdif) < 0) return 0;
  if (dcmp(d2) == 0 && dcmp(c1.r - c2.r) == 0) return -1;

  Tf base = angle(c2.o - c1.o);
  if (dcmp(d2 - rdif * rdif) == 0) {
    a.push_back(c1.point(base));
    b.push_back(c2.point(base));
    cnt++;
    return cnt;
  }
```

```
  Tf ang = acos((c1.r - c2.r) / sqrt(d2));
  a.push_back(c1.point(base + ang));
  b.push_back(c2.point(base + ang));
  cnt++;
  a.push_back(c1.point(base - ang));
  b.push_back(c2.point(base - ang));
  cnt++;

  if (dcmp(d2 - rsum * rsum) == 0) {
    a.push_back(c1.point(base));
    b.push_back(c2.point(PI + base));
    cnt++;
  } else if (dcmp(d2 - rsum * rsum) > 0) {
    Tf ang = acos((c1.r + c2.r) / sqrt(d2));
    a.push_back(c1.point(base + ang));
    b.push_back(c2.point(PI + base + ang));
    cnt++;
    a.push_back(c1.point(base - ang));
    b.push_back(c2.point(PI + base - ang));
    cnt++;
  }
  return cnt;
}
```

4.4 Convex

```
/// minkowski sum of two polygons in O(n)
Polygon minkowskiSum(Polygon A, Polygon B) {
  int n = A.size(), m = B.size();
  rotate(A.begin(), min_element(A.begin(), A
      .end()), A.end());
  rotate(B.begin(), min_element(B.begin(), B
      .end()), B.end());

  A.push_back(A[0]);
  B.push_back(B[0]);
  for (int i = 0; i < n; i++) A[i] = A[i +
      1] - A[i];
  for (int i = 0; i < m; i++) B[i] = B[i +
      1] - B[i];

  Polygon C(n + m + 1);
  C[0] = A.back() + B.back();
  merge(A.begin(), A.end() - 1, B.begin(), B
      .end() - 1, C.begin() + 1,
       polarComp(Point(0, 0), Point(0, -1))
         );
  for (int i = 1; i < C.size(); i++) C[i] =
      C[i] + C[i - 1];
  C.pop_back();
  return C;
}
// finds the rectangle with minimum area
     enclosing a convex polygon and
// the rectangle with minimum perimeter
     enclosing a convex polygon
// Tf Ti Same
pair<Tf, Tf> rotatingCalipersBoundingBox(
     const Polygon &p) {
  using Linear::distancePointLine;
  int n = p.size();
  int l = 1, r = 1, j = 1;
  Tf area = 1e100;
  Tf perimeter = 1e100;
  for (int i = 0; i < n; i++) {
    Point v = (p[(i + 1) % n] - p[i]) /
         length(p[(i + 1) % n] - p[i]);
    while (dcmp(dot(v, p[r % n] - p[i]) -
         dot(v, p[(r + 1) % n] - p[i])) < 0)
      r++;
    while (j < r || dcmp(cross(v, p[j % n] -
          p[i]) -
                    cross(v, p[(j + 1) %
                           n] - p[i])) < 0)
      j++;
    while (l < j ||
           dcmp(dot(v, p[l % n] - p[i]) - dot
                (v, p[(l + 1) % n] - p[i])) >
                 0)
      l++;
    Tf w = dot(v, p[r % n] - p[i]) - dot(v,
         p[l % n] - p[i]);
    Tf h = distancePointLine(p[j % n], Line(
         p[i], p[(i + 1) % n]));
    area = min(area, w * h);
    perimeter = min(perimeter, 2 * w + 2 * h
         );
  }
  return make_pair(area, perimeter);
}
// returns the left side of polygon u after
     cutting it by ray a->b
Polygon cutPolygon(Polygon u, Point a, Point
      b) {
  using Linear::lineLineIntersection;
  using Linear::onSegment;

  Polygon ret;
  int n = u.size();
  for (int i = 0; i < n; i++) {
    Point c = u[i], d = u[(i + 1) % n];
    if (dcmp(cross(b - a, c - a)) >= 0) ret.
         push_back(c);
    if (dcmp(cross(b - a, d - c)) != 0) {
      Point t;
      lineLineIntersection(a, b - a, c, d -
           c, t);
      if (onSegment(t, Segment(c, d))) ret.
           push_back(t);
    }
  }
  return ret;
}
// returns true if point p is in or on
     triangle abc
bool pointInTriangle(Point a, Point b, Point
      c, Point p) {
  return dcmp(cross(b - a, p - a)) >= 0 &&
         dcmp(cross(c - b, p - b)) >= 0 &&
          dcmp(cross(a - c, p - c)) >= 0;
}
// pt must be in ccw order with no three
     collinear points
// returns inside = -1, on = 0, outside = 1
int pointInConvexPolygon(const Polygon &pt,
     Point p) {
  int n = pt.size();
  assert(n >= 3);

  int lo = 1, hi = n - 1;
  while (hi - lo > 1) {
    int mid = (lo + hi) / 2;
    if (dcmp(cross(pt[mid] - pt[0], p - pt
         [0])) > 0)
      lo = mid;
    else
      hi = mid;
  }

  bool in = pointInTriangle(pt[0], pt[lo],
       pt[hi], p);
  if (!in) return 1;

  if (dcmp(cross(pt[lo] - pt[lo - 1], p - pt
       [lo - 1])) == 0) return 0;
  if (dcmp(cross(pt[hi] - pt[lo], p - pt[lo
       ])) == 0) return 0;
  if (dcmp(cross(pt[hi] - pt[(hi + 1) % n],
       p - pt[(hi + 1) % n])) == 0)
    return 0;
  return -1;
}
// Extreme Point for a direction is the
     farthest point in that direction
// u is the direction for extremeness
int extremePoint(const Polygon &poly, Point
     u) {
  int n = (int)poly.size();
  int a = 0, b = n;
  while (b - a > 1) {
    int c = (a + b) / 2;
    if (dcmp(dot(poly[c] - poly[(c + 1) % n
         ], u)) >= 0 &&
        dcmp(dot(poly[c] - poly[(c - 1 + n)
             % n], u)) >= 0) {
      return c;
    }

    bool a_up = dcmp(dot(poly[(a + 1) % n] -
         poly[a], u)) >= 0;
    bool c_up = dcmp(dot(poly[(c + 1) % n] -
         poly[c], u)) >= 0;
    bool a_above_c = dcmp(dot(poly[a] - poly
         [c], u)) > 0;

    if (a_up && !c_up)
      b = c;
    else if (!a_up && c_up)
      a = c;
    else if (a_up && c_up) {
      if (a_above_c)
        b = c;
      else
        a = c;
    } else {
      if (!a_above_c)
        b = c;
      else
        a = c;
    }
  }

  if (dcmp(dot(poly[a] - poly[(a + 1) % n],
       u)) > 0 &&
      dcmp(dot(poly[a] - poly[(a - 1 + n) %
           n], u)) > 0)
    return a;
  return b % n;
}
// For a convex polygon p and a line l,
     returns a list of segments
// of p that touch or intersect line l.
// the i'th segment is considered (p[i], p[(
     i + 1) modulo |p|])
// #1 If a segment is collinear with the
     line, only that is returned
// #2 Else if l goes through i'th point, the
      i'th segment is added
// Complexity: O(lg |p|)
vector<int> lineConvexPolyIntersection(const
      Polygon &p, Line l) {
  assert((int)p.size() >= 3);
  assert(l.a != l.b);

  int n = p.size();
  vector<int> ret;

  Point v = l.b - l.a;
  int lf = extremePoint(p, rotate90(v));
  int rt = extremePoint(p, rotate90(v) * Ti
       (-1));
  int olf = orient(l.a, l.b, p[lf]);
  int ort = orient(l.a, l.b, p[rt]);

  if (!olf || !ort) {
    int idx = (!olf ? lf : rt);
    if (orient(l.a, l.b, p[(idx - 1 + n) % n
         ]) == 0)
      ret.push_back((idx - 1 + n) % n);
    else
      ret.push_back(idx);
    return ret;
  }
  if (olf == ort) return ret;

  for (int i = 0; i < 2; ++i) {
    int lo = i ? rt : lf;
    int hi = i ? lf : rt;
    int olo = i ? ort : olf;

    while (true) {
      int gap = (hi - lo + n) % n;
      if (gap < 2) break;

      int mid = (lo + gap / 2) % n;
      int omid = orient(l.a, l.b, p[mid]);
      if (!omid) {
        lo = mid;
        break;
      }
      if (omid == olo)
        lo = mid;
```

```
      else
        hi = mid;
    }
    ret.push_back(lo);
  }
  return ret;
}
// Calculate [ACW, CW] tangent pair from an external point
constexpr int CW = -1, ACW = 1;
bool isGood(Point u, Point v, Point Q, int dir) {
  return orient(Q, u, v) != -dir;
}
Point better(Point u, Point v, Point Q, int dir) {
  return orient(Q, u, v) == dir ? u : v;
}
Point pointPolyTangent(const Polygon &pt, Point Q, int dir, int lo, int hi) {
  while (hi - lo > 1) {
    int mid = (lo + hi) / 2;
    bool pvs = isGood(pt[mid], pt[mid - 1], Q, dir);
    bool nxt = isGood(pt[mid], pt[mid + 1], Q, dir);

    if (pvs && nxt) return pt[mid];
    if (!(pvs || nxt)) {
      Point p1 = pointPolyTangent(pt, Q, dir, mid + 1, hi);
      Point p2 = pointPolyTangent(pt, Q, dir, lo, mid - 1);
      return better(p1, p2, Q, dir);
    }

    if (!pvs) {
      if (orient(Q, pt[mid], pt[lo]) == dir)
        hi = mid - 1;
      else if (better(pt[lo], pt[hi], Q, dir) == pt[lo])
        hi = mid - 1;
      else
        lo = mid + 1;
    }
    if (!nxt) {
      if (orient(Q, pt[mid], pt[lo]) == dir)
        lo = mid + 1;
      else if (better(pt[lo], pt[hi], Q, dir) == pt[lo])
        hi = mid - 1;
      else
        lo = mid + 1;
    }
  }

  Point ret = pt[lo];
  for (int i = lo + 1; i <= hi; i++) ret = better(ret, pt[i], Q, dir);
  return ret;
}
// [ACW, CW] Tangent
pair<Point, Point> pointPolyTangents(const Polygon &pt, Point Q) {
  int n = pt.size();
  Point acw_tan = pointPolyTangent(pt, Q, ACW, 0, n - 1);
  Point cw_tan = pointPolyTangent(pt, Q, CW, 0, n - 1);
  return make_pair(acw_tan, cw_tan);
}
```

4.5 Polygon

```
typedef vector<Point> Polygon;
// removes redundant colinear points
// polygon can't be all colinear points
Polygon RemoveCollinear(const Polygon &poly) {
  Polygon ret;
  int n = poly.size();
  for (int i = 0; i < n; i++) {
    Point a = poly[i];
    Point b = poly[(i + 1) % n];
    Point c = poly[(i + 2) % n];
    if (dcmp(cross(b - a, c - b)) != 0 && (ret.empty() || b != ret.back()))
      ret.push_back(b);
  }
  return ret;
}
// returns the signed area of polygon p of n vertices
Tf signedPolygonArea(const Polygon &p) {
  Tf ret = 0;
  for (int i = 0; i < (int)p.size() - 1; i++)
    ret += cross(p[i] - p[0], p[i + 1] - p[0]);
  return ret / 2;
}
// given a polygon p of n vertices, generates the convex hull in in CCW
// Tested on https://acm.timus.ru/problem.aspx?space=1&num=1185
// Caution: when all points are colinear AND removeRedundant == false
// output will be contain duplicate points (from upper hull) at back
Polygon convexHull(Polygon p, bool removeRedundant) {
  int check = removeRedundant ? 0 : -1;
  sort(p.begin(), p.end());
  p.erase(unique(p.begin(), p.end()), p.end());

  int n = p.size();
  Polygon ch(n + n);
  int m = 0; // preparing lower hull
  for (int i = 0; i < n; i++) {
    while (m > 1 &&
           dcmp(cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 1])) <= check)
      m--;
    ch[m++] = p[i];
  }
  int k = m; // preparing upper hull
  for (int i = n - 2; i >= 0; i--) {
    while (m > k &&
           dcmp(cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2])) <= check)
      m--;
    ch[m++] = p[i];
  }
  if (n > 1) m--;
  ch.resize(m);
  return ch;
}
// returns inside = -1, on = 0, outside = 1
int pointInPolygon(const Polygon &p, Point o) {
  using Linear::onSegment;
  int wn = 0, n = p.size();
  for (int i = 0; i < n; i++) {
    int j = (i + 1) % n;
    if (onSegment(o, Segment(p[i], p[j])) || o == p[i]) return 0;
    int k = dcmp(cross(p[j] - p[i], o - p[i]));
    int d1 = dcmp(p[i].y - o.y);
    int d2 = dcmp(p[j].y - o.y);
    if (k > 0 && d1 <= 0 && d2 > 0) wn++;
    if (k < 0 && d2 <= 0 && d1 > 0) wn--;
  }
  return wn ? -1 : 1;
}
// Given a simple polygon p, and a line l, returns (x, y)
// x = longest segment of l in p, y = total length of l in p.
pair<Tf, Tf> linePolygonIntersection(Line l, const Polygon &p) {
  using Linear::lineLineIntersection;
  int n = p.size();
  vector<pair<Tf, int>> ev;
  for (int i = 0; i < n; ++i) {
    Point a = p[i], b = p[(i + 1) % n], z = p[(i - 1 + n) % n];
    int ora = orient(l.a, l.b, a), orb = orient(l.a, l.b, b),
        orz = orient(l.a, l.b, z);
    if (!ora) {
      Tf d = dot(a - l.a, l.b - l.a);
      if (orz && orb) {
        if (orz != orb) ev.emplace_back(d, 0);
        // else // Point Touch
      } else if (orz)
        ev.emplace_back(d, orz);
      else if (orb)
        ev.emplace_back(d, orb);
    } else if (ora == -orb) {
      Point ins;
      lineLineIntersection(l, Line(a, b), ins);
      ev.emplace_back(dot(ins - l.a, l.b - l.a), 0);
    }
  }
  sort(ev.begin(), ev.end());

  Tf ans = 0, len = 0, last = 0, tot = 0;
  bool active = false;
  int sign = 0;
  for (auto &qq : ev) {
    int tp = qq.second;
    Tf d = qq.first; /// current Segment is (last, d)
    if (sign) {     /// On Border
      len += d - last;
      tot += d - last;
      ans = max(ans, len);
      if (tp != sign) active = !active;
      sign = 0;
    } else {
      if (active) { /// Strictly Inside
        len += d - last;
        tot += d - last;
        ans = max(ans, len);
      }
      if (tp == 0)
        active = !active;
      else
        sign = tp;
    }
    last = d;
    if (!active) len = 0;
  }
  ans /= length(l.b - l.a);
  tot /= length(l.b - l.a);
  return {ans, tot};
}
```

4.6 Intersecting Segments

```
// Given a list of segments v, finds a pair (i, j)
// st v[i], v[j] intersects. If none, returns {-1, -1}
// Tested Timus 1469, CF 1359F
struct Event {
  Tf x;
  int tp, id;
  bool operator<(const Event &p) const {
    if (dcmp(x - p.x)) return x < p.x;
    return tp > p.tp;
  }
};
pair<int, int> anyIntersection(const vector&lt;Segment&gt; &v) {
  using Linear::segmentsIntersect;
  static_assert(is_same<Tf, Ti>::value);

  vector<Event> ev;
  for (int i = 0; i < v.size(); i++) {
    ev.push_back({min(v[i].a.x, v[i].b.x), +1, i});
    ev.push_back({max(v[i].a.x, v[i].b.x), -1, i});
  }
  sort(ev.begin(), ev.end());

  auto comp = [&v](int i, int j) {
```

```cpp
    Segment p = v[i], q = v[j];
    Tf x = max(min(p.a.x, p.b.x), min(q.a.x,
        q.b.x));
    auto yvalSegment = [&x](const Line &s) {
      if (dcmp(s.a.x - s.b.x) == 0) return s
          .a.y;
      return s.a.y + (s.b.y - s.a.y) * (x -
          s.a.x) / (s.b.x - s.a.x);
    };
    return dcmp(yvalSegment(p) - yvalSegment
        (q)) < 0;
  };

  multiset<int, decltype(comp)> st(comp);
  typedef decltype(st)::iterator iter;
  auto prev = [&st](iter it) { return it ==
      st.begin() ? st.end() : --it; };
  auto next = [&st](iter it) { return it ==
      st.end() ? st.end() : ++it; };

  vector<iter> pos(v.size());
  for (auto &cur : ev) {
    int id = cur.id;
    if (cur.tp == 1) {
      iter nxt = st.lower_bound(id);
      iter pre = prev(nxt);
      if (pre != st.end() &&
          segmentsIntersect(v[*pre], v[id])
          )
        return {*pre, id};
      if (nxt != st.end() &&
          segmentsIntersect(v[*nxt], v[id])
          )
        return {*nxt, id};
      pos[id] = st.insert(nxt, id);
    } else {
      iter nxt = next(pos[id]);
      iter pre = prev(pos[id]);
      if (pre != st.end() && nxt != st.end()
           &&
          segmentsIntersect(v[*pre], v[*nxt])
            )
        return {*pre, *nxt};
      st.erase(pos[id]);
    }
  }
  return {-1, -1};
}
```

5 Graph

5.1 Link Cut Tree

```cpp
struct SplayTree {
  struct node {
    int ch[2] = {0, 0}, p = 0;
    ll self = 0, path = 0;
    ll sub = 0, extra = 0;
    bool rev = false;
  };
  vector<node> T;
  SplayTree(int n) : T(n + 1) {}
  void push(int x) {
    if (!x) return;
    int l = T[x].ch[0], r = T[x].ch[1];
    if (T[x].rev) {
      T[l].rev ^= true, T[r].rev ^= true;
      swap(T[x].ch[0], T[x].ch[1]);
      T[x].rev = false;
    }
  }
  void pull(int x) {
    int l = T[x].ch[0], r = T[x].ch[1];
    push(l), push(r);
    T[x].path = T[x].self + T[l].path + T[r
        ].path;
    T[x].sub = T[x].self + T[x].extra + T[l
        ].sub + T[r].sub;
  }
  void set(int parent, int child, int d) {
    T[parent].ch[d] = child;
    T[child].p = parent;
    pull(parent);
  }
  int dir(int x) {
    int parent = T[x].p;
    if (!parent) return -1;
    return (T[parent].ch[0] == x) ? 0 : (T[
        parent].ch[1] == x) ? 1 : -1;
  }
  void rotate(int x) {
    int parent = T[x].p, gparent = T[parent
        ].p;
    int dx = dir(x), dp = dir(parent);
    set(parent, T[x].ch[!dx], dx);
    set(x, parent, !dx);
    if (~dp) set(gparent, x, dp);
    T[x].p = gparent;
  }
  void splay(int x) {
    push(x);
    while (~dir(x)) {
      int parent = T[x].p;
      int gparent = T[parent].p;
      push(gparent), push(parent), push(x);
      int dx = dir(x), dp = dir(parent);
      if (~dp) rotate(dx != dp ? x : parent)
          ;
      rotate(x);
    }
  }
};
struct LinkCut : SplayTree {
  LinkCut(int n) : SplayTree(n) {}
  void cut_right(int x) {
    splay(x);
    int r = T[x].ch[1];
    T[x].extra += T[r].sub;
    T[x].ch[1] = 0, pull(x);
  }
  int access(int x) {
    int u = x, v = 0;
    for (; u; v = u, u = T[u].p) {
      cut_right(u);
      T[u].extra -= T[v].sub;
      T[u].ch[1] = v, pull(u);
    }
    return splay(x), v;
  }
  void make_root(int x) {
    access(x);
    T[x].rev ^= true, push(x);
  }
  void link(int u, int v) {
    make_root(v), access(u);
    T[u].extra += T[v].sub;
    T[v].p = u, pull(u);
  }
  void cut(int u) {
    access(u);
    T[u].ch[0] = T[T[u].ch[0]].p = 0;
    pull(u);
  }
  void cut(int u, int v) {
    make_root(u), access(v);
    T[v].ch[0] = T[u].p = 0, pull(v);
  }
  int find_root(int u) {
    access(u), push(u);
    while (T[u].ch[0]) {
      u = T[u].ch[0], push(u);
    }
    return splay(u), u;
  }
  int lca(int u, int v) {
    if (u == v) return u;
    access(u);
    int ret = access(v);
    return T[u].p ? ret : 0;
  }
  // subtree query of u if v is the root
  ll subtree(int u, int v) {
    make_root(v), access(u);
    return T[u].self + T[u].extra;
  }
  ll path(int u, int v) {
    make_root(u), access(v);
    return T[v].path;
  }
  // point update
  void update(int u, ll val) {
    access(u);
    T[u].self = val, pull(u);
  }
};
```

5.2 Graph Template

```cpp
struct edge {
  int u, v;
  edge(int u = 0, int v = 0) : u(u), v(v) {}
  int to(int node) { return u ^ v ^ node; }
};
struct graph {
  int n;
  vector<vector<int>> adj;
  vector<edge> edges;
  graph(int n = 0) : n(n), adj(n) {}
  void addEdge(int u, int v, bool dir = 1) {
    adj[u].push_back(edges.size());
    if (dir) adj[v].push_back(edges.size());
    edges.emplace_back(u, v);
  }
  int addNode() {
    adj.emplace_back();
    return n++;
  }
  edge &operator()(int idx) { return edges[
      idx]; }
  vector<int> &operator[](int u) { return
      adj[u]; }
};
```

5.3 LCA, ETT, VT

```cpp
struct lca_table {
  tree &T;
  int n, LOG = 20;
  vector<vector<int>> anc;
  vector<int> level;

  void setupLifting(int node, int par) {
    for (int v : T[node])
      if (v != par) {
        anc[v][0] = node, level[v] = level[
            node] + 1;
        for (int k = 1; k < LOG; k++) anc[v
            ][k] = anc[anc[v][k - 1]][k -
            1];
        setupLifting(v, node);
      }
  }
  lca_table(tree &T, int root = 0) : T(T), n
      (T.n) {
    LOG = 33 - __builtin_clz(n);
    anc.assign(n, vector<int>(LOG, root));
    level.resize(n);
    setupLifting(root, root);
  }
  int lca(int u, int v) {
    if (level[u] > level[v]) swap(u, v);
    for (int k = LOG - 1; ~k; k--)
      if (level[u] + (1 << k) <= level[v]) v
          = anc[v][k];
    if (u == v) return u;
    for (int k = LOG - 1; ~k; k--)
      if (anc[u][k] != anc[v][k]) u = anc[u
          ][k], v = anc[v][k];
    return anc[u][0];
  }
  int getAncestor(int node, int ht) {
    for (int k = 0; k < LOG; k++)
      if (ht & (1 << k)) node = anc[node][k
          ];
    return node;
  }
  int distance(int u, int v) {
    int g = lca(u, v);
    return level[u] + level[v] - 2 * level[g
        ];
  }
};
struct euler_tour {
  int time = 0;
```

```
  tree &T;
  int n;
  vector<int> start, finish, level, par;
  euler_tour(tree &T, int root = 0)
      : T(T), n(T.n), start(n), finish(n),
        level(n), par(n) {
    time = 0;
    call(root);
  }
  void call(int node, int p = -1) {
    if (p != -1) level[node] = level[p] + 1;
    start[node] = time++;
    for (int e : T[node])
      if (e != p) call(e, node);
    par[node] = p;
    finish[node] = time++;
  }
  bool isAncestor(int node, int par) {
    return start[par] <= start[node] and
        finish[par] >= finish[node];
  }
  int subtreeSize(int node) { return finish[
      node] - start[node] + 1 >> 1; }
};
tree virtual_tree(vector<int> &nodes,
    lca_table &table, euler_tour &tour) {
  sort(nodes.begin(), nodes.end(),
      [&](int x, int y) { return tour.start
          [x] < tour.start[y]; });
  int n = nodes.size();
  for (int i = 0; i + 1 < n; i++)
    nodes.push_back(table.lca(nodes[i],
        nodes[i + 1]));
  sort(nodes.begin(), nodes.end());
  nodes.erase(unique(nodes.begin(), nodes.
      end()), nodes.end());
  sort(nodes.begin(), nodes.end(),
      [&](int x, int y) { return tour.start
          [x] < tour.start[y]; });
  n = nodes.size();
  stack<int> st;
  st.push(0);
  tree ans(n);
  for (int i = 1; i < n; i++) {
    while (!tour.isAncestor(nodes[i], nodes[
        st.top()])) st.pop();
    ans.addEdge(st.top(), i);
    st.push(i);
  }
  return ans;
}
set<int> getCenters(tree &T) {
  int n = T.n;
  vector<int> deg(n), q;
  set<int> s;
  for (int i = 0; i < n; i++) {
    deg[i] = T[i].size();
    if (deg[i] == 1) q.push_back(i);
    s.insert(i);
  }
  for (vector<int> t; s.size() > 2; q = t) {
    for (auto x : q) {
      for (auto e : T[x])
        if (--deg[e] == 1) t.push_back(e);
      s.erase(x);
    }
  }
  return s;
}
```

5.4 SCC

```
typedef long long LL;
const LL N = 1e6 + 7;

bool vis[N];
vector<int> adj[N], adjr[N];
vector<int> order, component;
// tp = 0 ,finding topo order, tp = 1 ,
    reverse edge traversal

void dfs(int u, int tp = 0) {
  vis[u] = true;
  if (tp) component.push_back(u);
  auto& ad = (tp ? adjr : adj);
  for (int v : ad[u])
    if (!vis[v]) dfs(v, tp);
  if (!tp) order.push_back(u);
}
int main() {
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) dfs(i);
  }
  memset(vis, 0, sizeof vis);
  reverse(order.begin(), order.end());
  for (int i : order) {
    if (!vis[i]) {
      // one component is found
      dfs(i, 1), component.clear();
    }
  }
}
```

5.5 Euler Tour on Edge

```
// for simplicity, G[idx] contains the
    adjacency list of a node
// while G(e) is a reference to the e-th
    edge.
const int N = 2e5 + 5;
int in[N], out[N], fwd[N], bck[N];
int t = 0;
void dfs(graph &G, int node, int par) {
  out[node] = t;
  for (int e : G[node]) {
    int v = G(e).to(node);
    if (v == par) continue;
    fwd[e] = t++;
    dfs(G, v, node);
    bck[e] = t++;
  }
  in[node] = t - 1;
}
void init(graph &G, int node) {
  t = 0;
  dfs(G, node, node);
}
```

5.6 LCA In O(1)

```
/* LCA in O(1)
 * depth calculates weighted distance
 * level calculates distance by number of
    edges
 * Preprocessing in NlongN */
LL depth[N];
int level[N];

int st[N], en[N], LOG[N], par[N];
int a[N], id[N], table[L][N];

vector<PII> adj[N];
int n, root, Time, cur;

void init(int nodes, int root_) {
  n = nodes, root = root_, LOG[0] = LOG[1] =
      0;
  for (int i = 2; i <= n; i++) LOG[i] = LOG[
      i >> 1] + 1;
  for (int i = 0; i <= n; i++) adj[i].clear
      ();
}

void addEdge(int u, int v, int w) {
  adj[u].push_back(PII(v, w));
  adj[v].push_back(PII(u, w));
}

int lca(int u, int v) {
  if (en[u] > en[v]) swap(u, v);
  if (st[v] <= st[u] && en[u] <= en[v])
      return v;

  int l = LOG[id[v] - id[u] + 1];
  int p1 = id[u], p2 = id[v] - (1 << l) + 1;
  int d1 = level[table[l][p1]], d2 = level[
      table[l][p2]];

  if (d1 < d2)
    return par[table[l][p1]];
  else
    return par[table[l][p2]];
}

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

/* Euler tour */
void dfs(int u, int p) {
  st[u] = ++Time, par[u] = p;

  for (auto [v, w] : adj[u]) {
    if (v == p) continue;
    depth[v] = depth[u] + w;
    level[v] = level[u] + 1;
    dfs(v, u);
  }

  en[u] = ++Time;
  a[++cur] = u, id[u] = cur;
}

/* RMQ */
void pre() {
  cur = Time = 0, dfs(root, root);
  for (int i = 1; i <= n; i++) table[0][i] =
      a[i];

  for (int l = 0; l < L - 1; l++) {
    for (int i = 1; i <= n; i++) {
      table[l + 1][i] = table[l][i];

      bool C1 = (1 << l) + i <= n;
      bool C2 = level[table[l][i + (1 << l)
          ]] < level[table[l][i]];

      if (C1 && C2) table[l + 1][i] = table[
          l][i + (1 << l)];
    }
  }
}
```

5.7 HLD

```
const int N = 1e6 + 7;
template <typename DT>
struct Segtree {
  // write lazy segtree here
};
Segtree<int> tree(N);
vector<int> adj[N];
int depth[N], par[N], pos[N];
int head[N], heavy[N], cnt;

int dfs(int u, int p) {
  int SZ = 1, mxsz = 0, heavyc;
  depth[u] = depth[p] + 1;

  for (auto v : adj[u]) {
    if (v == p) continue;
    par[v] = u;
    int subsz = dfs(v, u);
    if (subsz > mxsz) heavy[u] = v, mxsz =
        subsz;
    SZ += subsz;
  }
  return SZ;
}
void decompose(int u, int h) {
  head[u] = h, pos[u] = ++cnt;
  if (heavy[u] != -1) decompose(heavy[u], h)
      ;

  for (int v : adj[u]) {
    if (v == par[u]) continue;
    if (v != heavy[u]) decompose(v, v);
  }
}
int query(int a, int b) {
  int ret = 0;
  for (; head[a] != head[b]; b = par[head[b
      ]]) {
```

```
    if (depth[head[a]] > depth[head[b]])
      swap(a, b);
    ret += tree.query(1, 0, cnt, pos[head[b
      ]], pos[b]);
  }

  if (depth[a] > depth[b]) swap(a, b);
  ret += tree.query(1, 0, cnt, pos[a], pos[b
    ]);
  return ret;
}
```

5.8 Centroid Decomposition

```
class Centroid_Decomposition {
  vector<bool> blocked;
  vector<int> CompSize;
  int CompDFS(tree &T, int node, int par) {
    CompSize[node] = 1;
    for (int &e : T[node])
      if (e != par and !blocked[e]) CompSize
        [node] += CompDFS(T, e, node);
    return CompSize[node];
  }
  int FindCentroid(tree &T, int node, int
    par, int sz) {
    for (int &e : T[node])
      if (e != par and !blocked[e])
        if (CompSize[e] > sz / 2) return
          FindCentroid(T, e, node, sz);
    return node;
  }
  pair<int, int> GetCentroid(tree &T, int
    entry) {
    int sz = CompDFS(T, entry, entry);
    return {FindCentroid(T, entry, entry, sz
      ), sz};
  }
  c_vector<LL> left[2], right[2];
  int GMin, GMax;
  void dfs(tree &T, int node, int par, int
    Min, int Max, int sum) {
    if (blocked[node]) return;
    right[Max < sum or Min > sum][sum]++;
    Max = max(Max, sum), Min = min(Min, sum)
      ;
    GMin = min(GMin, sum), GMax = max(GMax,
      sum);
    for (int i = 0; i < T[node].size(); i++)
      if (T[node][i] != par) {
        dfs(T, T[node][i], node, Min, Max,
          sum + T.col[node][i]);
      }
  }
  LL solve(tree &T, int c, int sz) {
    LL ans = 0;
    left[0].clear(-sz, sz), left[1].clear(-
      sz, sz);
    for (int i = 0; i < T[c].size(); i++) {
      GMin = 1, GMax = -1;
      dfs(T, T[c][i], c, GMin, GMax, T.col[c
        ][i]);
      ans += right[0][0] + left[1][0] *
        right[1][0];
      for (int j : {0, 1})
        for (int k = GMin; k <= GMax; k++) {
          ans += right[j][k] * (left[0][-k] +
            (j == 0) * left[1][-k]);
        }
      for (int j : {0, 1})
        for (int k = GMin; k <= GMax; k++) {
          left[j][k] += right[j][k];
          right[j][k] = 0;
        }
    }
    return ans;
  }

 public:
  LL operator()(tree &T, int entry) {
    blocked.resize(T.n);
    CompSize.resize(T.n);
    for (int i : {0, 1})
      left[i].resize(2 * T.n + 5), right[i].
        resize(2 * T.n + 5);
    auto [c, sz] = GetCentroid(T, entry);
    LL ans = solve(T, c, sz);
    blocked[c] = true;
    for (int e : T[c])
      if (!blocked[e]) ans += (*this)(T, e);
    return ans;
  }
};
```

5.9 Dinic Max Flow

```
/// flow with demand(lower bound) only for
  DAG
// create new src and sink
// add_edge(new src, u, sum(in_demand[u]))
// add_edge(u, new sink, sum(out_demand[u]))
// add_edge(old sink, old src, inf)
//  if (sum of lower bound == flow) then
  demand satisfied
// flow in every edge i = demand[i] + e.flow

using Ti = long long;
const Ti INF = 1LL << 60;
struct edge {
  int v, u;
  Ti cap, flow = 0;
  edge(int v, int u, Ti cap) : v(v), u(u),
    cap(cap) {}
};
const int N = 1e5 + 50;
vector<edge> edges;
vector<int> adj[N];
int m = 0, n;
int level[N], ptr[N];
queue<int> q;
bool bfs(int s, int t) {
  for (q.push(s), level[s] = 0; !q.empty();
    q.pop()) {
    for (int id : adj[q.front()]) {
      auto &ed = edges[id];
      if (ed.cap - ed.flow > 0 and level[ed.
        u] == -1)
        level[ed.u] = level[ed.v] + 1, q.
          push(ed.u);
    }
  }
  return level[t] != -1;
}
Ti dfs(int v, Ti pushed, int t) {
  if (pushed == 0) return 0;
  if (v == t) return pushed;
  for (int &cid = ptr[v]; cid < adj[v].size
    (); cid++) {
    int id = adj[v][cid];
    auto &ed = edges[id];
    if (level[v] + 1 != level[ed.u] || ed.
      cap - ed.flow < 1) continue;
    Ti tr = dfs(ed.u, min(pushed, ed.cap -
      ed.flow), t);
    if (tr == 0) continue;
    ed.flow += tr;
    edges[id ^ 1].flow -= tr;
    return tr;
  }
  return 0;
}
void init(int nodes) {
  m = 0, n = nodes;
  for (int i = 0; i < n; i++) level[i] = -1,
    ptr[i] = 0, adj[i].clear();
}
void addEdge(int v, int u, Ti cap) {
  edges.emplace_back(v, u, cap), adj[v].
    push_back(m++);
  edges.emplace_back(u, v, 0), adj[u].
    push_back(m++);
}
Ti maxFlow(int s, int t) {
  Ti f = 0;
  for (auto &ed : edges) ed.flow = 0;
  for (; bfs(s, t); memset(level, -1, n * 4)
    ) {
    for (memset(ptr, 0, n * 4); Ti pushed =
      dfs(s, INF, t); f += pushed)
      ;
  }
  return f;
}
```

5.10 Min Cost Max Flow

```
mt19937 rnd(chrono::steady_clock::now().
    time_since_epoch().count());
const LL inf = 1e9;
struct edge {
  int v, rev;
  LL cap, cost, flow;
  edge() {}
  edge(int v, int rev, LL cap, LL cost)
    : v(v), rev(rev), cap(cap), cost(cost)
      , flow(0) {}
};
struct mcmf {
  int src, sink, n;
  vector<int> par, idx, Q;
  vector<bool> inq;
  vector<LL> dis;
  vector<vector<edge>> g;
  mcmf() {}
  mcmf(int src, int sink, int n)
    : src(src),
      sink(sink),
      n(n),
      par(n),
      idx(n),
      inq(n),
      dis(n),
      g(n),
      Q(10000005) {} // use Q(n) if not
        using random
  void add_edge(int u, int v, LL cap, LL
    cost, bool directed = true) {
    edge _u = edge(v, g[v].size(), cap, cost
      );
    edge _v = edge(u, g[u].size(), 0, -cost)
      ;
    g[u].pb(_u);
    g[v].pb(_v);
    if (!directed) add_edge(v, u, cap, cost,
      true);
  }
  bool spfa() {
    for (int i = 0; i < n; i++) {
      dis[i] = inf, inq[i] = false;
    }
    int f = 0, l = 0;
    dis[src] = 0, par[src] = -1, Q[l++] =
      src, inq[src] = true;
    while (f < l) {
      int u = Q[f++];
      for (int i = 0; i < g[u].size(); i++)
        {
        edge &e = g[u][i];
        if (e.cap <= e.flow) continue;
        if (dis[e.v] > dis[u] + e.cost) {
          dis[e.v] = dis[u] + e.cost;
          par[e.v] = u, idx[e.v] = i;
          if (!inq[e.v]) inq[e.v] = true, Q[l
            ++] = e.v;
          // if (!inq[e.v]) {
          //   inq[e.v] = true;
          //   if (f && rnd() & 7) Q[--f] = e
            .v;
          //   else Q[l++] = e.v;
          // }
        }
      }
      inq[u] = false;
    }
    return (dis[sink] != inf);
  }
  pair<LL, LL> solve() {
    LL mincost = 0, maxflow = 0;
    while (spfa()) {
      LL bottleneck = inf;
      for (int u = par[sink], v = idx[sink];
        u != -1; v = idx[u], u = par[u])
        {
        edge &e = g[u][v];
```

```
      bottleneck = min(bottleneck, e.cap -
          e.flow);
    }
    for (int u = par[sink], v = idx[sink];
        u != -1; v = idx[u], u = par[u])
        {
      edge &e = g[u][v];
      e.flow += bottleneck;
      g[e.v][e.rev].flow -= bottleneck;
    }
    mincost += bottleneck * dis[sink],
        maxflow += bottleneck;
  }
  return make_pair(mincost, maxflow);
 }
};
// want to minimize cost and don't care
    about flow
// add edge from sink to dummy sink (cap =
    inf, cost = 0)
// add edge from source to sink (cap = inf,
    cost = 0)
// run mcmf, cost returned is the minimum
    cost
```

5.11 StoerWanger

```
/* for finding the min cut of a graph
    without specifing the source and the
   sink. all the edges are directed and no
        need to make any edge bidirectional.
*/
const int N = 1407;
// O(n^3) but faster, 1 indexed

mt19937 rnd(chrono::steady_clock::now().
    time_since_epoch().count());
struct StoerWagner {
  int n, idx[N];
  LL G[N][N], dis[N];
  bool vis[N];
  const LL inf = 1e18;

  StoerWagner() {}
  StoerWagner(int _n) {
    n = _n;
    memset(G, 0, sizeof G);
  }
  void add_edge(
      int u, int v,
      LL w) { // undirected edge, multiple
          edges are merged into one edge
    if (u != v) G[u][v] += w, G[v][u] += w;
  }

  LL solve() {
    LL ans = inf;
    for (int i = 0; i < n; ++i) idx[i] = i +
        1;
    shuffle(idx, idx + n, rnd);

    while (n > 1) {
      int t = 1, s = 0;
      for (int i = 1; i < n; ++i) {
        dis[idx[i]] = G[idx[0]][idx[i]];
        if (dis[idx[i]] > dis[idx[t]]) t = i
            ;
      }

      memset(vis, 0, sizeof vis);
      vis[idx[0]] = true;

      for (int i = 1; i < n; ++i) {
        if (i == n - 1) {
          if (ans > dis[idx[t]])
            ans =
                dis[idx[t]]; // idx[s] - idx[
                    t] is in two halves of
                    the mincut
          if (ans == 0) return 0;
          for (int j = 0; j < n; ++j) {
            G[idx[s]][idx[j]] += G[idx[j]][
                idx[t]];
            G[idx[j]][idx[s]] += G[idx[j]][
                idx[t]];
          }
          idx[t] = idx[--n];
        }

        vis[idx[t]] = true;
        s = t, t = -1;

        for (int j = 1; j < n; ++j) {
          if (!vis[idx[j]]) {
            dis[idx[j]] += G[idx[s]][idx[j]];
            if (t == -1 || dis[idx[t]] < dis[
                idx[j]]) t = j;
          }
        }
      }
    }
    return ans;
  }
};
```

5.12 Hungarian

```
/*  For max cost, negate cost matrix and
     negate output.
   Complexity: O(n^2 m). n must not be
        greater than m.
   Input: (n+1) x (m+1) cost matrix. (0th
        row and column are useless)
   Output: (ans, ml), where ml[i] = match
        for node i on the left. */
#include <bits/stdc++.h>
using namespace std;

template <typename T>
pair<T, vector<int>> Hungarian(const vector<
    vector<T>> &cost) {
  const T INF = numeric_limits<T>::max();
  int n = cost.size() - 1, m = cost[0].size
      () - 1;
  vector<T> U(n + 1), V(n + 1);
  vector<int> mr(m + 1), way(m + 1), ml(n +
      1);

  for (int i = 1; i <= n; i++) {
    mr[0] = i;
    int lastJ = 0;
    vector<T> minV(m + 1, INF);
    vector<bool> used(m + 1);
    do {
      used[lastJ] = true;
      int lastI = mr[lastJ], nextJ;
      T delta = INF;
      for (int j = 1; j <= m; j++) {
        if (used[j]) continue;
        T diffCost = cost[lastI][j] - U[
            lastI] - V[j];
        if (diffCost < minV[j]) minV[j] =
            diffCost, way[j] = lastJ;
        if (minV[j] < delta) delta = minV[j
            ], nextJ = j;
      }
      for (int j = 0; j <= m; j++) {
        if (used[j])
          U[mr[j]] += delta, V[j] -= delta;
        else
          minV[j] -= delta;
      }
      lastJ = nextJ;
    } while (mr[lastJ] != 0);
    do {
      int prevJ = way[lastJ];
      mr[lastJ] = mr[prevJ];
      lastJ = prevJ;
    } while (lastJ != 0);
  }
  for (int i = 1; i <= m; i++) ml[mr[i]] = i
      ;
  return {-V[0], ml};
}

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

  int n;
  cin >> n;

  vector<vector<long long>> cost(n + 1,
      vector<long long>(n + 1));
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++) cin >> cost
        [i][j];

  auto [ans, match] = Hungarian(cost);
  cout << ans << endl;
  for (int i = 1; i <= n; i++) cout << match
      [i] - 1 << " ";
}
```

5.13 Block Cut Tree

```
vector<vector<int> > components;
vector<int> cutpoints, start, low;
vector<bool> is_cutpoint;
stack<int> st;
void find_cutpoints(int node, graph &G, int
    par = -1, int d = 0) {
  low[node] = start[node] = d++;
  st.push(node);
  int cnt = 0;
  for (int e : G[node])
    if (int to = G(e).to(node); to != par) {
      if (start[to] == -1) {
        find_cutpoints(to, G, node, d + 1);
        cnt++;
        if (low[to] >= start[node]) {
          is_cutpoint[node] = par != -1 or
              cnt > 1;
          components.push_back({node}); //
              starting a new block with the
              point
          while (st.top() != node)
            components.back().push_back(st.
                top()), st.pop();
        }
      }
      low[node] = min(low[node], low[to]);
    }
}
graph tree;
vector<int> id;
void init(graph &G) {
  int n = G.n;
  start.assign(n, -1), low.resize(n),
      is_cutpoint.resize(n), id.assign(n,
      -1);
  find_cutpoints(0, G);
  for (int u = 0; u < n; ++u)
    if (is_cutpoint[u]) id[u] = tree.addNode
        ();
  for (auto &comp : components) {
    int node = tree.addNode();
    for (int u : comp)
      if (!is_cutpoint[u])
        id[u] = node;
      else
        tree.addEdge(node, id[u]);
  }
  if (id[0] == -1) // corner - 1
    id[0] = tree.addNode();
}
```

5.14 Bridge Tree

```
vector<vector<int>> components;
vector<int> depth, low;
stack<int> st;
vector<int> id;
vector<edge> bridges;
graph tree;
void find_bridges(int node, graph &G, int
    par = -1, int d = 0) {
  low[node] = depth[node] = d;
  st.push(node);
  for (int id : G[node]) {
    int to = G(id).to(node);
    if (par != to) {
      if (depth[to] == -1) {
        find_bridges(to, G, node, d + 1);
        if (low[to] > depth[node]) {
```

```
        bridges.emplace_back(node, to);
        components.push_back({});
        for (int x = -1; x != to; x = st.
            top(), st.pop())
          components.back().push_back(st.
              top());
      }
    }
    low[node] = min(low[node], low[to]);
  }
}
if (par == -1) {
  components.push_back({});
  while (!st.empty()) components.back().
      push_back(st.top()), st.pop();
}
}
graph &create_tree() {
  for (auto &comp : components) {
    int idx = tree.addNode();
    for (auto &e : comp) id[e] = idx;
  }
  for (auto &[l, r] : bridges) tree.addEdge(
      id[l], id[r]);
  return tree;
}
void init(graph &G) {
  int n = G.n;
  depth.assign(n, -1), id.assign(n, -1), low
      .resize(n);
  for (int i = 0; i < n; i++)
    if (depth[i] == -1) find_bridges(i, G);
}
```

5.15 Tree Isomorphism

```
mp["01"] = 1;
ind = 1;
int dfs(int u, int p) {
  int cnt = 0;
  vector<int> vs;
  for (auto v : g1[u]) {
    if (v != p) {
      int got = dfs(v, u);
      vs.pb(got);
      cnt++;
    }
  }
  if (!cnt) return 1;

  sort(vs.begin(), vs.end());
  string s = "0";
  for (auto i : vs) s += to_string(i);
  vs.clear();
  s.pb('1');
  if (mp.find(s) == mp.end()) mp[s] = ++ind;
  int ret = mp[s];
  return ret;
}
```

6 Math

6.1 Adaptive Simpsons

```
/*
  For finding the length of an arc in a
      range
  L = integrate(ds) from start to end of
      range
  where ds = sqrt(1+(d/dy(x))^2)dy
*/
const double SIMPSON_TERMINAL_EPS = 1e-12;
/// Function whose integration is to be
    calculated
double F(double x);
double simpson(double minx, double maxx) {
  return (maxx - minx) / 6 * (F(minx) + 4 *
      F((minx + maxx) / 2.) + F(maxx));
}
double adaptive_simpson(double minx, double
    maxx, double c, double EPS) {
  //  if(maxx - minx <
      SIMPSON_TERMINAL_EPS) return 0;

  double midx = (minx + maxx) / 2;
  double a = simpson(minx, midx);
  double b = simpson(midx, maxx);

  if (fabs(a + b - c) < 15 * EPS) return a +
      b + (a + b - c) / 15.0;

  return adaptive_simpson(minx, midx, a, EPS
      / 2.) +
       adaptive_simpson(midx, maxx, b, EPS
          / 2.);
}
double adaptive_simpson(double minx, double
    maxx, double EPS) {
  return adaptive_simpson(minx, maxx,
      simpson(minx, maxx, i), EPS);
}
```

6.2 Berlekamp Massey

```
struct berlekamp_massey { // for linear
    recursion
  typedef long long LL;
  static const int SZ = 2e5 + 5;
  static const int MOD = 1e9 + 7; /// mod
      must be a prime
  LL m , a[SZ] , h[SZ] , t_[SZ] , s[SZ] , t[
      SZ];
  // bigmod goes here
  inline vector <LL> BM( vector <LL> &x ) {
    LL lf , ld;
    vector <LL> ls , cur;
    for ( int i = 0; i < int(x.size()); ++i
        ) {
      LL t = 0;
      for ( int j = 0; j < int(cur.size());
          ++j ) t = (t + x[i - j - 1] * cur
          [j]) % MOD;
      if ( (t - x[i]) % MOD == 0 ) continue;
      if ( !cur.size() ) {
        cur.resize( i + 1 );
        lf = i; ld = (t - x[i]) % MOD;
        continue;
      }
      LL k = -(x[i] - t) * bigmod( ld , MOD
          - 2 , MOD ) % MOD;
      vector <LL> c(i - lf - 1);
      c.push_back( k );
      for ( int j = 0; j < int(ls.size());
          ++j ) c.push_back(-ls[j] * k %
          MOD);
      if ( c.size() < cur.size() ) c.resize(
           cur.size() );
      for ( int j = 0; j < int(cur.size());
          ++j ) c[j] = (c[j] + cur[j]) %
          MOD;
      if (i - lf + (int)ls.size() >= (int)
          cur.size() ) ls = cur, lf = i, ld
           = (t - x[i]) % MOD;
      cur = c;
    }
    for ( int i = 0; i < int(cur.size()); ++
        i ) cur[i] = (cur[i] % MOD + MOD) %
        MOD;
    return cur;
  }
  inline void mull( LL *p , LL *q ) {
    for ( int i = 0; i < m + m; ++i ) t_[i]
        = 0;
    for ( int i = 0; i < m; ++i ) if ( p[i]
        )
      for ( int j = 0; j < m; ++j ) t_[i +
          j] = (t_[i + j] + p[i] * q[j])
          % MOD;
    for ( int i = m + m - 1; i >= m; --i )
        if ( t_[i] )
      for ( int j = m - 1; ~j; --j ) t_[i
          - j - 1] = (t_[i - j - 1] + t_[i
          ] * h[j]) % MOD;
    for ( int i = 0; i < m; ++i ) p[i] = t_[
        i];
  }
  inline LL calc( LL K ) {
    for ( int i = m; ~i; --i ) s[i] = t[i] =
         0;
    s[0] = 1; if ( m != 1 ) t[1] = 1; else t
        [0] = h[0];
    while ( K ) {
      if ( K & 1 ) mull( s , t );
      mull( t , t ); K >>= 1;
    }
    LL su = 0;
    for ( int i = 0; i < m; ++i ) su = (su +
         s[i] * a[i]) % MOD;
    return (su % MOD + MOD) % MOD;
  }
  /// already calculated upto k , now
      calculate upto n.
  inline vector <LL> process( vector <LL> &x
      , int n , int k ) {
    auto re = BM( x );
    x.resize( n + 1 );
    for ( int i = k + 1; i <= n; i++ ) {
      for ( int j = 0; j < re.size(); j++ )
        {
        x[i] += 1LL * x[i - j - 1] % MOD *
            re[j] % MOD; x[i] %= MOD;
      }
    }
    return x;
  }
  inline LL work( vector <LL> &x , LL n ) {
    if ( n < int(x.size()) ) return x[n] %
        MOD;
    vector <LL> v = BM( x ); m = v.size();
        if ( !m ) return 0;
    for ( int i = 0; i < m; ++i ) h[i] = v[i
        ], a[i] = x[i];
    return calc( n ) % MOD;
  }
} rec;
vector <LL> v;
void solve() {
  int n;
  cin >> n;
  cout << rec.work(v, n - 1) << endl;
}
```

6.3 Fractional Binary Search

```
/**
Given a function f and n, finds the smallest
    fraction p / q in [0, 1] or [0,n]
such that f(p / q) is true, and p, q <= n.
Time: O(log(n))
**/
struct frac { long long p, q; };
bool f(frac x) {
  return 6 + 8 * x.p >= 17 * x.q + 12;
}
frac fracBS(long long n) {
  bool dir = 1, A = 1, B = 1;
  frac lo{0, 1}, hi{1, 0}; // Set hi to 1/0
      to search within [0, n] and {1, 1} to
       search within [0, 1]
  if (f(lo)) return lo;
  assert(f(hi)); //checking if any solution
      exists or not
  while (A || B) {
    long long adv = 0, step = 1; // move hi
        if dir, else lo
    for (int si = 0; step; (step *= 2) >>=
        si) {
      adv += step;
      frac mid{lo.p * adv + hi.p, lo.q * adv
          + hi.q};
      if (abs(mid.p) > n || mid.q > n || dir
          == !f(mid)) {
        adv -= step; si = 2;
      }
    }
    hi.p += lo.p * adv;
    hi.q += lo.q * adv;
    dir = !dir;
    swap(lo, hi);
    A = B; B = !!adv;
  }
  return dir ? hi : lo;
}
```

6.4 Gaussian Elimination

```cpp
double gaussian_elimination(int row, int col) {
  int basis[30];
  for (int j = 0; j < row; j++) {
    MAT[j][j + col] = 1;
  }
  memset(basis, -1, sizeof basis);
  double det = 1;
  for (int i = 0; i < col; i++) {
    for (int p = 0; p < row; p++) {
      for (int q = 0; q < col; q++) cout << MAT[p][q] << ' ';
      cout << '\n';
    }
    int x = -1;
    for (int k = 0; k < row; k++) {
      if (abs(MAT[k][i]) > eps and basis[k] == -1) {
        x = k, det *= MAT[k][i], basis[x] = i;
        break;
      }
    }
    if (x < 0) continue;
    for (int j = 0; j < col; j++)
      if (j != i)
        for (int k = 0; k < row; k++)
          if (k != x) MAT[k][j] -= (MAT[k][i] * MAT[x][j]) / MAT[x][i];
    for (int k = 0; k < col; k++)
      if (k != i) MAT[x][k] /= MAT[x][i];
    for (int j = 0; j < row; j++) MAT[j][i] = (j == i);
  }
  for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) cout << MAT[i][j] << ' ';
    cout << '\n';
  }
  for (int k = 0; k < row; k++)
    if (basis[k] == -1) return 0;
  return det;
}
```

6.5 Green Hackenbush

```cpp
// Green Hackenbush
// Description:
//   Consider a two player game on a graph with a specified vertex (root).
//   In each turn, a player eliminates one edge.
//   Then, if a subgraph that is disconnected from the root, it is removed.
//   If a player cannot select an edge (i.e., the graph is singleton),
//   he will lose.
//
//   Compute the Grundy number of the given graph.
//
// Algorithm:
//   We use two principles:
//     1. Colon Principle: Grundy number of a tree is the xor of
//        Grundy number of child subtrees.
//        (Proof: easy).
//
//     2. Fusion Principle: Consider a pair of adjacent vertices u, v
//        that has another path (i.e., they are in a cycle). Then,
//        we can contract u and v without changing Grundy number.
//        (Proof: difficult)
//
//   We first decompose graph into two-edge connected components.
//   Then, by contracting each components by using Fusion Principle,
//   we obtain a tree (and many self loops) that has the same Grundy
//   number to the original graph. By using Colon Principle, we can
//   compute the Grundy number.
//
// Complexity:
//   O(m + n).
//
// Verified:
//   SPOJ 1477: Play with a Tree
//   IPSC 2003 G: Got Root?

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())
struct hackenbush {
  int n;
  vector<vector<int>> adj;
  hackenbush(int n) : n(n), adj(n) {}
  void add_edge(int u, int v) {
    adj[u].push_back(v);
    if (u != v) adj[v].push_back(u);
  }
  // r is the only root connecting to the ground
  int grundy(int r) {
    vector<int> num(n), low(n);
    int t = 0;
    function<int(int, int)> dfs = [&](int p, int u) {
      num[u] = low[u] = ++t;
      int ans = 0;
      for (int v : adj[u]) {
        if (v == p) {
          p += 2 * n;
          continue;
        }
        if (num[v] == 0) {
          int res = dfs(u, v);
          low[u] = min(low[u], low[v]);
          if (low[v] > num[u])
            ans ^= (1 + res) ^ 1; // bridge
          else
            ans ^= res; // non bridge
        } else
          low[u] = min(low[u], num[v]);
      }
      if (p > n) p -= 2 * n;
      for (int v : adj[u])
        if (v != p && num[u] <= num[v]) ans ^= 1;
      return ans;
    };
    return dfs(-1, r);
  }
};
```

6.6 Lagrange

```cpp
// p is a polynomial with n points.
// p(0), p(1), p(2), ... p(n-1) are given.
// Find p(x).
LL Lagrange(vector<LL> &p, LL x) {
  LL n = p.size(), L, i, ret;
  if (x < n) return p[x];
  L = 1;
  for (i = 1; i < n; i++) {
    L = (L * (x - i)) % MOD;
    L = (L * bigmod(MOD - i, MOD - 2)) % MOD;
  }
  ret = (L * p[0]) % MOD;
  for (i = 1; i < n; i++) {
    L = (L * (x - i + 1)) % MOD;
    L = (L * bigmod(x - i, MOD - 2)) % MOD;
    L = (L * bigmod(i, MOD - 2)) % MOD;
    L = (L * (MOD + i - n)) % MOD;
    ret = (ret + L * p[i]) % MOD;
  }
  return ret;
}
```

6.7 Matrix Exponentiation

```cpp
typedef vector<vector<LL>> Mat;
Mat Mul(Mat A, Mat B) {
  Mat ret(A.size(), vector<LL>(B[0].size()));
  LL i, j, k;
  for (i = 0; i < ret.size(); i++) {
    for (j = 0; j < ret[0].size(); j++) {
      for (k = 0; k < A[0].size(); k++)
        ret[i][j] = (ret[i][j] + (A[i][k] * B[k][j]) % MOD) % MOD;
    }
  }
  return ret;
}
Mat Pow(Mat A, LL p) {
  Mat ret(A.size(), vector<LL>(A[0].size()));
  for (LL i = 0; i < ret.size(); i++) ret[i][i] = 1;
  while (p) {
    if (p & 1) ret = Mul(ret, A);
    A = Mul(A, A);
    p >>= 1;
  }
  return ret;
}
```

6.8 Pollard Rho

```cpp
LL mul(LL a, LL b, LL mod) {
  return (__int128)a * b % mod;
  // LL ans = a * b - mod * (LL) (1.L / mod * a * b);
  // return ans + mod * (ans < 0) - mod * (ans >= (LL) mod);
}
LL bigmod(LL num, LL pow, LL mod) {
  LL ans = 1;
  for (; pow > 0; pow >>= 1, num = mul(num, num, mod))
    if (pow & 1) ans = mul(ans, num, mod);
  return ans;
}
bool is_prime(LL n) {
  if (n < 2 or n % 6 % 4 != 1) return (n | 1) == 3;
  LL a[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
  LL s = __builtin_ctzll(n - 1), d = n >> s;
  for (LL x : a) {
    LL p = bigmod(x % n, d, n), i = s;
    for (; p != 1 and p != n - 1 and x % n and i--; p = mul(p, p, n))
      ;
    if (p != n - 1 and i != s) return false;
  }
  return true;
}
LL get_factor(LL n) {
  auto f = [&](LL x) { return mul(x, x, n) + 1; };
  LL x = 0, y = 0, t = 0, prod = 2, i = 2, q;
  for (; t++ % 40 or gcd(prod, n) == 1; x = f(x), y = f(f(y))) {
    (x == y) ? x = i++, y = f(x) : 0;
    prod = (q = mul(prod, max(x, y) - min(x, y), n)) ? q : prod;
  }
  return gcd(prod, n);
}
map<LL, int> factorize(LL n) {
  map<LL, int> res;
  if (n < 2) return res;
  LL small_primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41,
                       43, 47, 53, 59, 61, 67,
                       71, 73, 79, 83,
                       89, 97};
  for (LL p : small_primes)
    for (; n % p == 0; n /= p, res[p]++)
      ;

  auto _factor = [&](LL n, auto &_factor) {
    if (n == 1) return;
    if (is_prime(n))
      res[n]++;
    else {
      LL x = get_factor(n);
```

```
      _factor(x, _factor);
      _factor(n / x, _factor);
    }
  };
  _factor(n, _factor);
  return res;
}
```

6.9 Shanks' Baby Step, Giant Step

```
// Finds a^x = b (mod p)

LL bigmod(LL b, LL p, LL m) {}

LL babyStepGiantStep(LL a, LL b, LL p) {
  LL i, j, c, sq = sqrt(p);
  map<LL, LL> babyTable;

  for (j = 0, c = 1; j <= sq; j++, c = (c *
      a) % p) babyTable[c] = j;

  LL giant = bigmod(a, sq * (p - 2), p);

  for (i = 0, c = 1; i <= sq; i++, c = (c *
      giant) % p) {
    if (babyTable.find((c * b) % p) !=
        babyTable.end())
      return i * sq + babyTable[(c * b) % p
          ];
  }

  return -1;
}
```

6.10 Stirling Numbers

```
//stirling number 2nd kind variation(number
    of ways to place n marbles in k boxes
    so that each box has at least x marbles
    )
ll solve(int marble, int box) {
  if (marble < 1ll * box * x) return 0;
  if (box == 1 && marble >= x) return 1;
  if (vis[marble][box] == cs) return dp[
      marble][box];
  vis[marble][box] = cs;
  ll a = ( 1ll * box * solve(marble - 1, box
      ) ) % MOD;
  ll b = ( 1ll * box * ncr(marble - 1, x -
      1) ) % MOD;
  b = (b * solve(marble - x, box - 1)) % MOD
      ;
  ll ret = (a + b) % MOD;
  return dp[marble][box] = ret;
}
//number of ways to place n marbles in k
    boxes so that no box is empty
ll stir(ll n, ll k) {
  ll ret = 0;
  for (int i = 0; i <= k; i++) {
    ll v = ncr(k, i) * bigmod(i, n) % MOD;
    if ( (k - i) % 2 == 0 ) ret = (ret + v)
        % MOD;
    else ret = (ret - v + MOD) % MOD;
  }
  return ret;
}
```

6.11 Xor Basis

```
struct XorBasis {
  static const int sz = 64;
  array<ULL, sz> base = {0}, back;
  array<int, sz> pos;
  void insert(ULL x, int p) {
    ULL cur = 0;
    for (int i = sz - 1; ~i; i--)
      if (x >> i & 1) {
        if (!base[i]) {
          base[i] = x, back[i] = cur, pos[i]
              = p;
          break;
        } else x ^= base[i], cur |= 1ULL <<
            i;
      }
  }
  pair<ULL, vector<int>> construct(ULL mask)
      {
    ULL ok = 0, x = mask;
    for (int i = sz - 1; ~i; i--)
      if (mask >> i & 1 and base[i]) mask ^=
          base[i], ok |= 1ULL << i;
    vector<int> ans;
    for (int i = 0; i < sz; i++)
      if (ok >> i & 1) {
        ans.push_back(pos[i]);
        ok ^= back[i];
      }
    return {x ^ mask, ans};
  }
};
```

6.12 Combi

```
array<int, N + 1> fact, inv, inv_fact;
void init() {
  fact[0] = inv_fact[0] = 1;
  for (int i = 1; i <= N; i++) {
    inv[i] = i == 1 ? 1 : (LL)inv[i - mod %
        i] * (mod / i + 1) % mod;
    fact[i] = (LL)fact[i - 1] * i % mod;
    inv_fact[i] = (LL)inv_fact[i - 1] * inv[
        i] % mod;
  }
}
LL C(int n, int r) {
  return (r < 0 or r > n) ? 0 : (LL)fact[n]
      * inv_fact[r] % mod * inv_fact[n - r]
      % mod;
}
```

6.13 Linear Sieve

```
const int N = 1e7;
vector<int> primes;
int spf[N + 5], phi[N + 5], NOD[N + 5], cnt[
    N + 5], POW[N + 5];
bool prime[N + 5];
int SOD[N + 5];
void init() {
  fill(prime + 2, prime + N + 1, 1);
  SOD[1] = NOD[1] = phi[1] = spf[1] = 1;
  for (LL i = 2; i <= N; i++) {
    if (prime[i]) {
      primes.push_back(i), spf[i] = i;
      phi[i] = i - 1;
      NOD[i] = 2, cnt[i] = 1;
      SOD[i] = i + 1, POW[i] = i;
    }
    for (auto p : primes) {
      if (p * i > N or p > spf[i]) break;
      prime[p * i] = false, spf[p * i] = p;
      if (i % p == 0) {
        phi[p * i] = p * phi[i];
        NOD[p * i] = NOD[i] / (cnt[i] + 1) *
            (cnt[i] + 2),
              cnt[p * i] = cnt[i] + 1;
        SOD[p * i] = SOD[i] / SOD[POW[i]] *
            (SOD[POW[i]] + p * POW[i]),
              POW[p * i] = p * POW[i];
        break;
      } else {
        phi[p * i] = phi[p] * phi[i];
        NOD[p * i] = NOD[p] * NOD[i], cnt[p
            * i] = 1;
        SOD[p * i] = SOD[p] * SOD[i], POW[p
            * i] = p;
      }
    }
  }
}
```

6.14 Chinese Remainder Theorem

```
// given a, b will find solutions for
// ax + by = 1
tuple<LL, LL, LL> EGCD(LL a, LL b) {
  if (b == 0)
    return {1, 0, a};
  else {
    auto [x, y, g] = EGCD(b, a % b);
    return {y, x - a / b * y, g};
  }
}
// given modulo equations, will apply CRT
PLL CRT(vector<PLL> &v) {
  LL V = 0, M = 1;
  for (auto &[v, m] : v) { // value % mod
    auto [x, y, g] = EGCD(M, m);
    if ((v - V) % g != 0) return {-1, 0};
    V += x * (v - V) / g % (m / g) * M, M *=
        m / g;
    V = (V % M + M) % M;
  }
  return make_pair(V, M);
}
```

6.15 Mobius Function

```
const int N = 1e6 + 5;
int mob[N];
void mobius() {
  memset(mob, -1, sizeof mob);
  mob[1] = 1;
  for (int i = 2; i < N; i++)
    if (mob[i]) {
      for (int j = i + i; j < N; j += i) mob
          [j] -= mob[i];
    }
}
```

6.16 FFT

```
using CD = complex<double>;
typedef long long LL;
const double PI = acos(-1.0L);

int N;
vector<int> perm;
vector<CD> wp[2];
void precalculate(int n) {
  assert((n & (n - 1)) == 0), N = n;
  perm = vector<int>(N, 0);
  for (int k = 1; k < N; k <<= 1) {
    for (int i = 0; i < k; i++) {
      perm[i] <<= 1;
      perm[i + k] = 1 + perm[i];
    }
  }
  wp[0] = wp[1] = vector<CD>(N);
  for (int i = 0; i < N; i++) {
    wp[0][i] = CD(cos(2 * PI * i / N), sin(2
        * PI * i / N));
    wp[1][i] = CD(cos(2 * PI * i / N), -sin
        (2 * PI * i / N));
  }
}
void fft(vector<CD> &v, bool invert = false)
    {
  if (v.size() != perm.size()) precalculate(
      v.size());
  for (int i = 0; i < N; i++)
    if (i < perm[i]) swap(v[i], v[perm[i]]);
  for (int len = 2; len <= N; len *= 2) {
    for (int i = 0, d = N / len; i < N; i +=
        len) {
      for (int j = 0, idx = 0; j < len / 2;
          j++, idx += d) {
        CD x = v[i + j];
        CD y = wp[invert][idx] * v[i + j +
            len / 2];
        v[i + j] = x + y;
        v[i + j + len / 2] = x - y;
      }
    }
  }
  if (invert) {
    for (int i = 0; i < N; i++) v[i] /= N;
  }
}
void pairfft(vector<CD> &a, vector<CD> &b,
    bool invert = false) {
  int N = a.size();
  vector<CD> p(N);
  for (int i = 0; i < N; i++) p[i] = a[i] +
      b[i] * CD(0, 1);
  fft(p, invert);
```

```cpp
  p.push_back(p[0]);
  for (int i = 0; i < N; i++) {
    if (invert) {
      a[i] = CD(p[i].real(), 0);
      b[i] = CD(p[i].imag(), 0);
    } else {
      a[i] = (p[i] + conj(p[N - i])) * CD(0.5, 0);
      b[i] = (p[i] - conj(p[N - i])) * CD(0, -0.5);
    }
  }
}
vector<LL> multiply(const vector<LL> &a, const vector<LL> &b) {
  int n = 1;
  while (n < a.size() + b.size()) n <<= 1;
  vector<CD> fa(a.begin(), a.end()), fb(b.begin(), b.end());
  fa.resize(n);
  fb.resize(n);
  //       fft(fa); fft(fb);
  pairfft(fa, fb);
  for (int i = 0; i < n; i++) fa[i] = fa[i] * fb[i];
  fft(fa, true);
  vector<LL> ans(n);
  for (int i = 0; i < n; i++) ans[i] = round(fa[i].real());
  return ans;
}
const int M = 1e9 + 7, B = sqrt(M) + 1;
vector<LL> anyMod(const vector<LL> &a, const vector<LL> &b) {
  int n = 1;
  while (n < a.size() + b.size()) n <<= 1;
  vector<CD> al(n), ar(n), bl(n), br(n);
  for (int i = 0; i < a.size(); i++) al[i] = a[i] % M / B, ar[i] = a[i] % M % B;
  for (int i = 0; i < b.size(); i++) bl[i] = b[i] % M / B, br[i] = b[i] % M % B;
  pairfft(al, ar);
  pairfft(bl, br);
  //       fft(al); fft(ar); fft(bl); fft(br);
  for (int i = 0; i < n; i++) {
    CD ll = (al[i] * bl[i]), lr = (al[i] * br[i]);
    CD rl = (ar[i] * bl[i]), rr = (ar[i] * br[i]);
    al[i] = ll;
    ar[i] = lr;
    bl[i] = rl;
    br[i] = rr;
  }
  pairfft(al, ar, true);
  pairfft(bl, br, true);
  //       fft(al, true); fft(ar, true); fft(bl, true); fft(br, true);
  vector<LL> ans(n);
  for (int i = 0; i < n; i++) {
    LL right = round(br[i].real()), left = round(al[i].real());
    ;
    LL mid = round(round(bl[i].real()) + round(ar[i].real()));
    ans[i] = ((left % M) * B * B + (mid % M) * B + right) % M;
  }
  return ans;
}
```

6.17 NTT

```cpp
const LL N = 1 << 18;
const LL MOD = 786433;

vector<LL> P[N];
LL rev[N], w[N | 1], a[N], b[N], inv_n, g;
LL Pow(LL b, LL p) {
  LL ret = 1;
  while (p) {
    if (p & 1) ret = (ret * b) % MOD;
    b = (b * b) % MOD;
    p >>= 1;
  }
  return ret;
}
LL primitive_root(LL p) {
  vector<LL> factor;
  LL phi = p - 1, n = phi;
  for (LL i = 2; i * i <= n; i++) {
    if (n % i) continue;
    factor.emplace_back(i);
    while (n % i == 0) n /= i;
  }
  if (n > 1) factor.emplace_back(n);
  for (LL res = 2; res <= p; res++) {
    bool ok = true;
    for (LL i = 0; i < factor.size() && ok; i++)
      ok &= Pow(res, phi / factor[i]) != 1;
    if (ok) return res;
  }
  return -1;
}
void prepare(LL n) {
  LL sz = abs(31 - __builtin_clz(n));
  LL r = Pow(g, (MOD - 1) / n);
  inv_n = Pow(n, MOD - 2);
  w[0] = w[n] = 1;
  for (LL i = 1; i < n; i++) w[i] = (w[i - 1] * r) % MOD;
  for (LL i = 1; i < n; i++)
    rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (sz - 1));
}
void NTT(LL *a, LL n, LL dir = 0) {
  for (LL i = 1; i < n - 1; i++)
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (LL m = 2; m <= n; m <<= 1) {
    for (LL i = 0; i < n; i += m) {
      for (LL j = 0; j < (m >> 1); j++) {
        LL &u = a[i + j], &v = a[i + j + (m >> 1)];
        LL t = v * w[dir ? n - n / m * j : n / m * j] % MOD;
        v = u - t < 0 ? u - t + MOD : u - t;
        u = u + t >= MOD ? u + t - MOD : u + t;
      }
    }
  }
  if (dir)
    for (LL i = 0; i < n; i++) a[i] = (inv_n * a[i]) % MOD;
}
vector<LL> mul(vector<LL> p, vector<LL> q) {
  LL n = p.size(), m = q.size();
  LL t = n + m - 1, sz = 1;
  while (sz < t) sz <<= 1;
  prepare(sz);

  for (LL i = 0; i < n; i++) a[i] = p[i];
  for (LL i = 0; i < m; i++) b[i] = q[i];
  for (LL i = n; i < sz; i++) a[i] = 0;
  for (LL i = m; i < sz; i++) b[i] = 0;

  NTT(a, sz);
  NTT(b, sz);
  for (LL i = 0; i < sz; i++) a[i] = (a[i] * b[i]) % MOD;
  NTT(a, sz, 1);

  vector<LL> c(a, a + sz);
  while (c.size() && c.back() == 0) c.pop_back();
  return c;
}
```

6.18 WalshHadamard

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
#define bitwiseXOR 1
// #define bitwiseAND 2
// #define bitwiseOR 3
const LL MOD = 30011;
LL BigMod(LL b, LL p) {
  LL ret = 1;
  while (p > 0) {
    if (p % 2 == 1) {
      ret = (ret * b) % MOD;
    }
    p = p / 2;
    b = (b * b) % MOD;
  }
  return ret % MOD;
}

void FWHT(vector<LL>& p, bool inverse) {
  LL n = p.size();
  assert((n & (n - 1)) == 0);

  for (LL len = 1; 2 * len <= n; len <<= 1) {
    for (LL i = 0; i < n; i += len + len) {
      for (LL j = 0; j < len; j++) {
        LL u = p[i + j];
        LL v = p[i + len + j];

#ifdef bitwiseXOR
        p[i + j] = (u + v) % MOD;
        p[i + len + j] = (u - v + MOD) % MOD;
#endif // bitwiseXOR

#ifdef bitwiseAND
        if (!inverse) {
          p[i + j] = v % MOD;
          p[i + len + j] = (u + v) % MOD;
        } else {
          p[i + j] = (-u + v) % MOD;
          p[i + len + j] = u % MOD;
        }
#endif // bitwiseAND

#ifdef bitwiseOR
        if (!inverse) {
          p[i + j] = u + v;
          p[i + len + j] = u;
        } else {
          p[i + j] = v;
          p[i + len + j] = u - v;
        }
#endif // bitwiseOR
      }
    }
  }

#ifdef bitwiseXOR
  if (inverse) {
    LL val = BigMod(n, MOD - 2); // Option 2: Exclude
    for (LL i = 0; i < n; i++) {
      // assert(p[i]%n==0); //Option 2: Include
      p[i] = (p[i] * val) % MOD; // Option 2: p[i]/=n;
    }
  }
#endif // bitwiseXOR
}
```

7 String

7.1 Aho Corasick

```cpp
const int sg = 26, N = 1e3 + 9;
struct aho_corasick {
  struct node {
    node *link, *out, *par;
    bool leaf;
    LL val;
    int cnt, last, len;
    char p_ch;
    array<node*, sg> to;
    node(node* par = NULL, char p_ch = '$', int len = 0)
      : par(par), p_ch(p_ch), len(len) {
      val = leaf = cnt = last = 0;
      link = out = NULL;
```

```
    }
  };
  vector<node> trie;
  node* root;
  aho_corasick() {
    trie.reserve(N), trie.emplace_back();
    root = &trie[0];
    root->link = root->out = root;
  }
  inline int f(char c) { return c - 'a'; }
  inline node* add_node(node* par = NULL,
      char p_ch = '$', int len = 0) {
    trie.emplace_back(par, p_ch, len);
    return &trie.back();
  }
  void add_str(const string& s, LL val = 1)
      {
    node* now = root;
    for (char c : s) {
      int i = f(c);
      if (!now->to[i]) now->to[i] = add_node
          (now, c, now->len + 1);
      now = now->to[i];
    }
    now->leaf = true, now->val++;
  }
  void push_links() {
    queue<node*> q;
    for (q.push(root); q.empty(); q.pop()) {
      node *cur = q.front(), *link = cur->
          link;
      cur->out = link->leaf ? link : link->
          out;
      int idx = 0;
      for (auto& next : cur->to) {
        if (next != NULL) {
          next->link = cur != root ? link->to
              [idx++] : root;
          q.push(next);
        } else
          next = link->to[idx++];
      }
    }
    cur->val += link->val;
  }
};
```

7.2 Double hash

```
ostream& operator<<(ostream& os, PLL hash) {
  return os << "(" << hash.ff << ", " <<
      hash.ss << ")";
}

PLL operator+(PLL a, LL x) { return PLL(a.ff
    + x, a.ss + x); }
PLL operator-(PLL a, LL x) { return PLL(a.ff
    - x, a.ss - x); }
PLL operator*(PLL a, LL x) { return PLL(a.ff
    * x, a.ss * x); }
PLL operator+(PLL a, PLL x) { return PLL(a.
    ff + x.ff, a.ss + x.ss); }
PLL operator-(PLL a, PLL x) { return PLL(a.
    ff - x.ff, a.ss - x.ss); }
PLL operator*(PLL a, PLL x) { return PLL(a.
    ff * x.ff, a.ss * x.ss); }
PLL operator%(PLL a, PLL m) { return PLL(a.
    ff % m.ff, a.ss % m.ss); }

PLL base(1949313259, 1997293877);
PLL mod(2091573227, 2117566807);

PLL power(PLL a, LL p) {
  if (!p) return PLL(1, 1);
  PLL ans = power(a, p / 2);
  ans = (ans * ans) % mod;
  if (p % 2) ans = (ans * a) % mod;
  return ans;
}

PLL inverse(PLL a) { return power(a, (mod.ff
    - 1) * (mod.ss - 1) - 1); }
PLL inv_base = inverse(base);

PLL val;
vector<PLL> P;

void hash_init(int n) {
  P.resize(n + 1);
  P[0] = PLL(1, 1);
  for (int i = 1; i <= n; i++) P[i] = (P[i -
      1] * base) % mod;
}

/// appends c to string
PLL append(PLL cur, char c) { return (cur *
    base + c) % mod; }

/// prepends c to string with size k
PLL prepend(PLL cur, int k, char c) { return
    (P[k] * c + cur) % mod; }

/// replaces the i-th (0-indexed) character
    from right from a to b;
PLL replace(PLL cur, int i, char a, char b)
    {
  cur = (cur + P[i] * (b - a)) % mod;
  return (cur + mod) % mod;
}

/// Erases c from the back of the string
PLL pop_back(PLL hash, char c) {
  return (((hash - c) * inv_base) % mod +
      mod) % mod;
}

/// Erases c from front of the string with
    size len
PLL pop_front(PLL hash, int len, char c) {
  return ((hash - P[len - 1] * c) % mod +
      mod) % mod;
}
/// concatenates two strings where length of
    the right is k
PLL concat(PLL left, PLL right, int k) {
    return (left * P[k] + right) % mod; }
/// Calculates hash of string with size len
    repeated cnt times
/// This is O(log n). For O(1), pre-
    calculate inverses
PLL repeat(PLL hash, int len, LL cnt) {
  PLL mul = (P[len * cnt] - 1) * inverse(P[
      len] - 1);
  mul = (mul % mod + mod) % mod;
  PLL ret = (hash * mul) % mod;

  if (P[len].ff == 1) ret.ff = hash.ff * cnt
      ;
  if (P[len].ss == 1) ret.ss = hash.ss * cnt
      ;
  return ret;
}
LL get(PLL hash) { return ((hash.ff << 32) ^
    hash.ss); }
struct hashlist {
  int len;
  vector<PLL> H, R;

  hashlist() {}
  hashlist(string& s) {
    len = (int)s.size();
    hash_init(len);
    H.resize(len + 1, PLL(0, 0)), R.resize(
        len + 2, PLL(0, 0));
    for (int i = 1; i <= len; i++) H[i] =
        append(H[i - 1], s[i - 1]);
    for (int i = len; i >= 1; i--) R[i] =
        append(R[i + 1], s[i - 1]);
  }

  /// 1-indexed
  inline PLL range_hash(int l, int r) {
    int len = r - l + 1;
    return ((H[r] - H[l - 1] * P[len]) % mod
        + mod) % mod;
  }

  inline PLL reverse_hash(int l, int r) {
    int len = r - l + 1;
    return ((R[l] - R[r + 1] * P[len]) % mod
        + mod) % mod;
  }

  inline PLL concat_range_hash(int l1, int
      r1, int l2, int r2) {
    int len_2 = r2 - l2 + 1;
    return concat(range_hash(l1, r1),
        range_hash(l2, r2), len_2);
  }

  inline PLL concat_reverse_hash(int l1, int
      r1, int l2, int r2) {
    int len_1 = r1 - l1 + 1;
    return concat(reverse_hash(l2, r2),
        reverse_hash(l1, r1), len_1);
  }
};
```

7.3 Manacher's

```
vector<int> d1(n);
// d[i] = number of palindromes taking s[i]
    as center
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 1 : min(d1[l + r - i], r
      - i + 1);
  while (0 <= i - k && i + k < n && s[i - k]
      == s[i + k]) k++;
  d1[i] = k--;
  if (i + k > r) l = i - k, r = i + k;
}

vector<int> d2(n);
// d[i] = number of palindromes taking s[i
    -1] and s[i] as center
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 0 : min(d2[l + r - i +
      1], r - i + 1);
  while (0 <= i - k - 1 && i + k < n && s[i
      - k - 1] == s[i + k]) k++;
  d2[i] = k--;
  if (i + k > r) l = i - k - 1, r = i + k;
}
```

7.4 Palindromic Tree

```
struct state {
  int len, link;
  map<char, int> next;
};
state st[MAX];
int id, last;
string s;
ll ans[MAX];
void init() {
  for (int i = 0; i <= id; i++) {
    st[i].len = 0; st[i].link = 0;
    st[i].next.clear(); ans[i] = 0;
  }
  st[1].len = -1; st[1].link = 1;
  st[2].len = 0; st[2].link = 1;
  id = 2; last = 2;
}
void extend(int pos) {
  while (s[pos - st[last].len - 1] != s[pos
      ]) last = st[last].link;
  int newlink = st[last].link;
  char c = s[pos];
  while (s[pos - st[newlink].len - 1] != s[
      pos]) newlink = st[newlink].link;
  if (!st[last].next.count(c)) {
    st[last].next[c] = ++id;
    st[id].len = st[last].len + 2;
    st[id].link = (st[id].len == 1 ? 2 : st[
        newlink].next[c]);
    ans[id] += ans[st[id].link];
    if (st[id].len > 2) {
      int l = st[id].len / 2 + (st[id].len %
          2 ? 1 : 0);
      if (h.range_hash(pos - st[id].len + 1,
          pos - st[id].len + l) == h.
          reverse_hash(pos - st[id].len +
          1, pos - st[id].len + l)) ans[id
          ]++;
```

```
    }
  }
  last = st[last].next[c];
}
```

7.5 String Match FFT

```
//find occurrences of t in s where '?'s are
    automatically matched with any
    character
//res[i + m - 1] = sum_j=0 to m - 1_{s[i + j
    ] * t[j] * (s[i + j] - t[j])
vector<int> string_matching(string &s,
    string &t) {
  int n = s.size(), m = t.size();
  vector<int> s1(n), s2(n), s3(n);
  for(int i = 0; i < n; i++) s1[i] = s[i] ==
      '?' ? 0 : s[i] - 'a' + 1; //assign
      any non zero number for non '?'s
  for(int i = 0; i < n; i++) s2[i] = s1[i] *
      s1[i];
  for(int i = 0; i < n; i++) s3[i] = s1[i] *
      s2[i];
  vector<int> t1(m), t2(m), t3(m);
  for(int i = 0; i < m; i++) t1[i] = t[i] ==
      '?' ? 0 : t[i] - 'a' + 1;
  for(int i = 0; i < m; i++) t2[i] = t1[i] *
      t1[i];
  for(int i = 0; i < m; i++) t3[i] = t1[i] *
      t2[i];
  reverse(t1.begin(), t1.end());
  reverse(t2.begin(), t2.end());
  reverse(t3.begin(), t3.end());
  vector<int> s1t3 = multiply(s1, t3);
  vector<int> s2t2 = multiply(s2, t2);
  vector<int> s3t1 = multiply(s3, t1);
  vector<int> res(n);
  for(int i = 0; i < n; i++) res[i] = s1t3[i
      ] - s2t2[i] * 2 + s3t1[i];
  vector<int> oc;
  for(int i = m - 1; i < n; i++) if(res[i]
      == 0) oc.push_back(i - m + 1);
  return oc;
}
```

7.6 Suffix Array

```
/** Suffix Array Construction: O(NlogN)
    LCP Array Construction: O(NlogN)
    Suffix LCP: O(logN) **/
typedef pair<int, int> PII;
typedef vector<int> VI;

/// Equivalence Class INFO
vector<VI> c;
VI sort_cyclic_shifts(const string &s) {
  int n = s.size();
  const int alphabet = 256;
  VI p(n), cnt(alphabet, 0);

  c.clear();
  c.emplace_back();
  c[0].resize(n);

  for (int i = 0; i < n; i++) cnt[s[i]]++;
  for (int i = 1; i < alphabet; i++) cnt[i]
      += cnt[i - 1];
  for (int i = 0; i < n; i++) p[--cnt[s[i]]]
      = i;

  c[0][p[0]] = 0;
  int classes = 1;

  for (int i = 1; i < n; i++) {
    if (s[p[i]] != s[p[i - 1]]) classes++;
    c[0][p[i]] = classes - 1;
  }

  VI pn(n), cn(n);
  cnt.resize(n);

  for (int h = 0; (1 << h) < n; h++) {
    for (int i = 0; i < n; i++) {
      pn[i] = p[i] - (1 << h);
      if (pn[i] < 0) pn[i] += n;
    }
    fill(cnt.begin(), cnt.end(), 0);

    /// radix sort
    for (int i = 0; i < n; i++) cnt[c[h][pn[
        i]]]++;
    for (int i = 1; i < classes; i++) cnt[i]
        += cnt[i - 1];
    for (int i = n - 1; i >= 0; i--) p[--cnt
        [c[h][pn[i]]]] = pn[i];

    cn[p[0]] = 0;
    classes = 1;

    for (int i = 1; i < n; i++) {
      PII cur = {c[h][p[i]], c[h][(p[i] + (1
          << h)) % n]};
      PII prev = {c[h][p[i - 1]], c[h][(p[i
          - 1] + (1 << h)) % n]};
      if (cur != prev) ++classes;
      cn[p[i]] = classes - 1;
    }
    c.push_back(cn);
  }
  return p;
}

VI suffix_array_construction(string s) {
  s += "!";
  VI sorted_shifts = sort_cyclic_shifts(s);
  sorted_shifts.erase(sorted_shifts.begin())
      ;
  return sorted_shifts;
}

/// LCP between the ith and jth (i != j)
    suffix of the STRING
int suffixLCP(int i, int j) {
  assert(i != j);
  int log_n = c.size() - 1;

  int ans = 0;
  for (int k = log_n; k >= 0; k--) {
    if (c[k][i] == c[k][j]) {
      ans += 1 << k;
      i += 1 << k;
      j += 1 << k;
    }
  }
  return ans;
}

VI lcp_construction(const string &s, const
    VI &sa) {
  int n = s.size();
  VI rank(n, 0);
  VI lcp(n - 1, 0);

  for (int i = 0; i < n; i++) rank[sa[i]] =
      i;

  for (int i = 0, k = 0; i < n; i++) {
    if (rank[i] == n - 1) {
      k = 0;
      continue;
    }

    int j = sa[rank[i] + 1];
    while (i + k < n && j + k < n && s[i + k
        ] == s[j + k]) k++;
    lcp[rank[i]] = k;
    if (k) k--;
  }
  return lcp;
}

const int MX = 1e6 + 7, K = 20;
int lg[MX];

void pre() {
  lg[1] = 0;
  for (int i = 2; i < MX; i++) lg[i] = lg[i
      / 2] + 1;
}
struct RMQ {
  int N;
  VI v[K];
  RMQ(const VI &a) {
    N = a.size();
    v[0] = a;

    for (int k = 0; (1 << (k + 1)) <= N; k
        ++) {
      v[k + 1].resize(N);
      for (int i = 0; i - 1 + (1 << (k + 1))
          < N; i++) {
        v[k + 1][i] = min(v[k][i], v[k][i +
            (1 << k)]);
      }
    }
  }

  int findMin(int i, int j) {
    int k = lg[j - i + 1];
    return min(v[k][i], v[k][j + 1 - (1 << k
        )]);
  }
};
```

7.7 Suffix Automata

```
/**
    Linear Time Suffix Automata contruction.
    Build Complexity: O(n * alphabet)
    To achieve better build complexity and
        linear space,
    use map for transitions.
**/

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

const int MAXN = 1e5+7, ALPHA = 26;
int len[2*MAXN], link[2*MAXN], nxt[2*MAXN][
    ALPHA];
int sz;
int last;

void sa_init() {
    memset(nxt, -1, sizeof nxt);

    len[0] = 0;
    link[0] = -1;
    sz = 1;
    last = 0;
}

void add(char ch) {
    int c = ch-'a';

    int cur = sz++;
        //create new node
    len[cur] = len[last]+1;

    int u = last;
    while (u != -1 && nxt[u][c] == -1) {
        nxt[u][c] = cur;
        u = link[u];
    }

    if (u == -1) {
        link[cur] = 0;
    }
    else {
        int v = nxt[u][c];
        if (len[v] == len[u]+1) {
            link[cur] = v;
        }
        else {
            int clone = sz++;
                //create node by cloning
            len[clone] = 1 + len[u];
            link[clone] = link[v];

            for (int i=0; i<ALPHA; i++)
                nxt[clone][i] = nxt[v][i];
```

```
            while (u != -1 && nxt[u][c] == v)
                  {
                nxt[u][c] = clone;
                u = link[u];
            }

            link[v] = link[cur] = clone;
        }
    }
    last = cur;
}

vector<int> edge[2*MAXN];
///Optional, Call after adding all
     characters
void makeEdge() {
    for (int i=0; i<sz; i++) {
        edge[i].clear();
        for (int j=0; j<ALPHA; j++)
            if (nxt[i][j]!=-1)
                edge[i].push_back(j);
    }
}

// The following code solves SPOJ SUBLEX
// Given a string S, you have to answer some
      queries:
// If all distinct substrings of string S
     were sorted
// lexicographically, which one will be the
     K-th smallest?

long long dp[2*MAXN];
bool vis[2*MAXN];

void dfs(int u) {
    if (vis[u]) return;
    vis[u] = 1;
    dp[u] = 1;
    for (int i: edge[u]) {
        if (nxt[u][i] == -1)  continue;
        dfs(nxt[u][i]);
        dp[u] += dp[nxt[u][i]];
    }
}

void go(int u, long long rem, string &s) {
    if (rem == 1) return;
    long long sum = 1;
    for (int i: edge[u]) {
        if (nxt[u][i] == -1)  continue;
        if (sum + dp[nxt[u][i]] < rem) {
            sum += dp[nxt[u][i]];
        }
        else {
            s += ('a' + i);
            go(nxt[u][i], rem-sum, s);
            return;
        }
    }
}

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

    string s;
    cin>>s;

    sa_init();
    for (char c: s) add(c);
    makeEdge();


    dfs(0);
    int q;
    cin>>q;

    while (q--) {
        long long x;
        cin>>x;
        x++;
        string s;
        go(0, x, s);
        cout<<s<<"\n";
    }
}
```

7.8 Z Algo

```
vector<int> calcz(string s) {
    int n = s.size();
    vector<int> z(n);
    int l, r; l = 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--;
        } 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--;
            }
        }
    }
    return z;
}
```

8 Equations and Formulas

8.1 Catalan Numbers

$C_n = \frac{1}{n+1}\binom{2n}{n}$ $C_0 = 1, C_1 = 1$ and $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$

The number of ways to completely parenthesize n+1 factors.
The number of triangulations of a convex polygon with n+2 sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
The number of ways to connect the $2n$ points on a circle to form n disjoint i.e. non-intersecting chords.
The number of rooted full binary trees with n+1 leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
Number of permutations of $1, \ldots, n$ that avoid the pattern 123 (or any of the other patterns of length 3); that is, the number of permutations with no three-term increasing sub-sequence. For $n = 3$, these permutations are 132, 213, 231, 312 and 321.

8.2 Stirling Numbers First Kind

The Stirling numbers of the first kind count permutations according to their number of cycles (counting fixed points as cycles of length one).
$S(n,k)$ counts the number of permutations of n elements with k disjoint cycles.
$S(n,k) = (n-1)\cdot S(n-1,k) + S(n-1,k-1)$, $where$, $S(0,0) = 1, S(n,0) = S(0,n) = 0$ $\sum_{k=0}^{n} S(n,k) = n!$
The unsigned Stirling numbers may also be defined algebraically, as the coefficient of the rising factorial:

$$x^{\bar{n}} = x(x+1)...(x+n-1) = \sum_{k=0}^{n} S(n,k)x^k$$

Lets $[n,k]$ be the stirling number of the first kind, then

$$\left[{}_{n}\,{}^{n}_{-}\,{}_{k}\right] = \sum_{0\le i_1<i_2<i_k<n} i_1 i_2 \ldots . i_k.$$

8.3 Stirling Numbers Second Kind

Stirling number of the second kind is the number of ways to partition a set of n objects into k non-empty subsets.
$S(n,k) = k\cdot S(n-1,k) + S(n-1,k-1)$, $where$ $S(0,0) = 1, S(n,0) = S(0,n) = 0$ $S(n,2) = 2^{n-1} - 1$ $S(n,k)\cdot k! =$ number of ways to color n nodes using colors from 1 to k such that each color is used at least once.
An r-associated Stirling number of the second kind is the number of ways to partition a set of n objects into k subsets, with each subset containing at least r elements. It is denoted by $S_r(n,k)$ and obeys the recurrence relation. $S_r(n+1,k) = kS_r(n,k) + \binom{n}{r-1} S_r(n-r+1,k-1)$
Denote the n objects to partition by the integers $1, 2, \ldots, n$. Define the reduced Stirling numbers of the second kind, denoted $S^d(n,k)$, to be the number of ways to partition the integers $1, 2, \ldots, n$ into k nonempty subsets such that all elements in each subset have pairwise distance at least d. That is, for any integers i and j in a given subset, it is required that $|i-j| \ge d$. It has been shown that these numbers satisfy, $S^d(n,k) = S(n-d+1, k-d+1), n \ge k \ge d$

8.4 Other Combinatorial Identities

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

$\sum_{i=0}^{k}\binom{n+i}{i} = \sum_{i=0}^{k}\binom{n+i}{n} = \binom{n+k+1}{k}$

$n, r \in N, n > r, \sum_{i=r}^{n}\binom{i}{r} = \binom{n+1}{r+1}$

If $P(n) = \sum_{k=0}^{n}\binom{n}{k}\cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n}(-1)^{n-k}\binom{n}{k}\cdot P(k)$$

If $P(n) = \sum_{k=0}^{n}(-1)^k\binom{n}{k}\cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n}(-1)^k\binom{n}{k}\cdot P(k)$$

8.5 Different Math Formulas

Picks Theorem : $A = i + b/2 - 1$
Deragements : $d(i) = (i-1)\times(d(i-1) + d(i-2))$

$$\frac{n}{ab} \quad - \quad \left\{\frac{b\prime n}{a}\right\} \quad - \quad \left\{\frac{a\prime n}{b}\right\} \quad + \quad 1$$

8.6 GCD and LCM

if m is any integer, then $\gcd(a + m{\cdot}b, b) = \gcd(a, b)$
The gcd is a multiplicative function in the following sense: if a_1 and a_2 are relatively prime, then $\gcd(a_1\cdot a_2, b) = \gcd(a_1, b)\cdot\gcd(a_2, b)$.
$\gcd(a, \mathrm{lcm}(b,c)) = \mathrm{lcm}(\gcd(a,b), \gcd(a,c))$.
$\mathrm{lcm}(a, \gcd(b,c)) = \gcd(\mathrm{lcm}(a,b), \mathrm{lcm}(a,c))$.
For non-negative integers a and b, where a and b are not both zero, $\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1$
$\gcd(a,b) = \sum_{k|a \text{ and } k|b} \phi(k)$

$\sum_{i=1}^{n}[\gcd(i,n) = k] = \phi\left(\frac{n}{k}\right)$

$\sum_{k=1}^{n}\gcd(k,n) = \sum_{d|n} d\cdot\phi\left(\frac{n}{d}\right)$

$\sum_{k=1}^{n} x^{\gcd(k,n)} = \sum_{d|n} x^d\cdot\phi\left(\frac{n}{d}\right)$

$\sum_{k=1}^{n}\frac{1}{\gcd(k,n)} = \sum_{d|n}\frac{1}{d}\cdot\phi\left(\frac{n}{d}\right) = \frac{1}{n}\sum_{d|n} d\cdot\phi(d)$

$\sum_{k=1}^{n}\frac{k}{\gcd(k,n)} = \frac{n}{2}\cdot\sum_{d|n}\frac{1}{d}\cdot\phi\left(\frac{n}{d}\right) = \frac{n}{2}\cdot\frac{1}{n}\cdot\sum_{d|n} d\cdot\phi(d)$

$\sum_{k=1}^{n}\frac{n}{\gcd(k,n)} = 2*\sum_{k=1}^{n}\frac{k}{\gcd(k,n)} - 1$, for $n > 1$

$\sum_{i=1}^{n}\sum_{j=1}^{n}[\gcd(i,j) = 1] = \sum_{d=1}^{n}\mu(d)\lfloor\frac{n}{d}\rfloor^2$

$\sum_{i=1}^{n}\sum_{j=1}^{n}\gcd(i,j) = \sum_{d=1}^{n}\phi(d)\lfloor\frac{n}{d}\rfloor^2$

$\sum_{i=1}^{n}\sum_{j=1}^{n} i\cdot j[\gcd(i,j) = 1] = \sum_{i=1}^{n}\phi(i)i^2$

$F(n) = \sum_{i=1}^{n}\sum_{j=1}^{n}\mathrm{lcm}(i,j) = \sum_{l=1}^{n}\left(\frac{\left(1 + \lfloor\frac{n}{l}\rfloor\right)\left(\lfloor\frac{n}{l}\rfloor\right)}{2}\right)^2\sum_{d|l}\mu(d)ld$