Team Note of tony9402

Compiled on February 23, 2025

Contents

1 Data Structure

1.1 2D Segment Tree

```cpp
// Time Complexity: Space O(N^2) Query O(log^2N) Update O(log^2N)
// [l, r]
template<typename T> struct Segment2D{
  vector<vector<T>> tree;
  int sizY, sizX;

  Segment2D() { }
  Segment2D(int y, int x){ setSize(y, x); }

  void setSize(int y, int x){
    sizY = sizX = 1;
    while(sizY <= y)sizY <<= 1;
    while(sizX <= x)sizX <<= 1;
    tree.resize(2 * sizY);
    for(int i=0; i < sizY * 2; i++) tree[i].resize(2 * sizX);
  }
  void putItem(int y, int x, T data){ tree[sizY + y][sizX + x]  = data; }
  void addItem(int y, int x, T data){ tree[sizY + y][sizX + x] += data; }
  void build(){
    for(int i=sizY;i<sizY*2;i++)
      for(int j=sizX-1;j;j--)
        tree[i][j] = merge(tree[i][j<<1], tree[i][j<<1|1]);

    for(int i=sizY-1;i;i--)
      for(int j=0;j<2*sizX;j++)
        tree[i][j]=merge(tree[i<<1][j], tree[i<<1|1][j]);
  }
```

```
  void update(int y, int x, T data, bool add=false){
    if(add) addItem(y, x, data);
    else  putItem(y, x, data);
    x += sizX; y += sizY;
    for(int i = x >> 1; i; i >>= 1) tree[y][i] = merge(tree[y][i << 1], tree[y][i << 1 |
    1]);

    for(int i = y >> 1; i; i >>= 1)
      for(int j = x; j; j >>= 1)
        tree[i][j] = merge(tree[i<<1][j], tree[i<<1|1][j]);
  }
  T query1D(int y, int l, int r){
    T ret = 0;
    for(l += sizX, r += sizX + 1; l < r; l >>= 1, r >>= 1){
      if(l & 1) ret += tree[y][l++];
      if(r & 1) ret += tree[y][--r];
    }
    return ret;
  }
  T query(int y1, int x1, int y2, int x2){
    T ret = 0;
    for(y1 += sizY, y2 += sizY + 1; y1 < y2; y1 >>= 1, y2 >>= 1){
      if(y1&1) ret += query1D(y1++, x1, x2);
      if(y2&1) ret += query1D(--y2, x1, x2);
    }
    return ret;
  }
  T merge(T, T);
};
template<typename T> T Segment2D<T>::merge(T a, T b) { return a + b; }
```

1.2 Cht

```
// Not Tested, 17526 source code
#include <bits/stdc++.h>

#define p pair<ll, ll>
#define mp make_pair
using namespace std;
typedef long long ll;

ll arr[100005], l[100005], r[100005];
ll dp[100005];

struct CHT{
  p vc[100005];
  int siz=0;
  double cross(p x, p y){
    return 1.0 * (x.second - y.second) / (y.first - x.first);
  }

  void add(p x){
    while(siz >= 2 && cross(vc[siz-2], vc[siz-1]) >= cross(vc[siz-1],x))siz--;
    vc[siz++]=x;
  }

  ll query(int x){
    int s, e;
    s = 0, e = siz - 1;
    while(s != e){
      int m = (s + e) >> 1;
      if(vc[m].first * x + vc[m].second < vc[m+1].first * x + vc[m+1].second)e = m;
      else s = m + 1;
    }
    return vc[s].first * x + vc[s].second;
  }
};

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

  CHT DCHT;
  int n;
  cin >> n;
  for(int i=n-1;i;i--)cin >> arr[i];
  for(int i=n-1;i;i--)cin >> l[i] >> r[i];
  for(int i=1;i<n;i++)arr[i] = arr[i] + arr[i-1];
  for(int i=1;i<n;i++){
    DCHT.add(mp(-arr[i-1], dp[i-1]));
    dp[i] = DCHT.query(r[i]) + arr[i] * r[i] + l[i];
  }
  cout << dp[n-1];
}
```

1.3 Dynamic Segment Tree

```
// Query O(logN) Update O(logN)
const int MAXL = 1000000000;
template<typename T>
struct DynamicSegment{
  struct Node{
    int l, r; // range
    T data;
    Node *left, *right;
    Node():l(1),r(MAXL),data(0),left(nullptr),right(nullptr) { }
    void extend(){
      if(l == r)return;
      if(left == nullptr){ //if leaf node
        left  = new Node();
        right =  new Node();
        int mid = (l + r) / 2;
        left->l  = l;
        left->r  = mid;
        right->l = mid + 1;
        right->r = r;
      }
      return;
    }
  };
  Node *tree;

  DynamicSegment() { tree = new Node(); }
  void update(Node *cur, int x, T data){
    if(x < cur->l || cur->r < x)return;
```

```
    if(cur->l == cur->r)return cur->data = data, (void)0;
    cur->extend();
    update(cur->left, x, data);
    update(cur->right, x, data);
    cur->data = mergeNode(cur->left->data, cur->right->data);
  }
  void update(int x, T data){ update(tree, x, data); }

  T query(Node *cur, int l, int r){
    if(cur->l > cur->r || cur->r < l || cur->l > r)return T(0);
    if(l <= cur->l && cur->r <= r)return cur->data;
    cur->extend();
    return mergeNode(query(cur->left, l, r), query(cur->right, l, r));
  }
  T query(int l, int r){ return query(tree, l, r); }
  T mergeNode(T a, T b){ return a + b; }
};
DynamicSegment<long long> tree;
```

1.4 Dynamic Segment Tree With Lazy

```
// Query O(logN) Update O(logN)
const int MAXL = 1000000000;
template<typename T>
struct DynamicSegmentLazy{
  struct Node{
    int l, r; // range
    T data, lazy;
    Node *left, *right;
    Node():l(1),r(MAXL),data(0),lazy(0),left(0),right(0) { }
    void extend(T lzy=0){
      if(l == r)return;
      if(left == 0){ //if leaf node
        left  = new Node();
        right =  new Node();
        int m = (l + r) / 2;
        left->l  = l;
        left->r  = m;
        right->l = m + 1;
        right->r = r;
      }
      left->lazy += lzy;
      right->lazy += lzy;
      return;
    }
  };
  Node *tree;

  DynamicSegmentLazy() { tree = new Node(); }
  void pushdown(Node *cur){
    if(cur->lazy){
      cur->data += (cur->r - cur->l + 1) * cur->lazy;
      cur->extend(cur->lazy);
      cur->lazy = 0;
    }
  }

  void update(Node *cur, int l, int r, T data){
    pushdown(cur);
    if(cur->l > cur->r || cur->l > r || l > cur->r)return;
    if(l <= cur->l && cur->r <= r){
      cur->data += (cur->r - cur->l + 1) * data;
      if(cur->l != cur->r)cur->extend(data);
      return;
    }
    cur->extend();
    update(cur->left, l, r, data);
    update(cur->right, l, r, data);
    cur->data = mergeNode(cur->left->data, cur->right->data);
  }
  void update(int l, int r, T data){ update(tree, l, r, data); }

  T query(Node *cur, int l, int r){
    if(cur->l > cur->r || cur->l > r || l > cur->r)return T(0);
    pushdown(cur);
    if(l <= cur->l && cur->r <= r)return cur->data;
    cur->extend();
    return mergeNode(query(cur->left, l, r), query(cur->right, l, r));
  }
  T query(int l, int r){ return query(tree, l, r); }
  T merge(T a, T b) {
    return a + b;
  }
};
```

1.5 Fenwick

```
// Query O(logN) Update O(logN)
template <typename T>
struct Fenwick {
  int N;
  vector<T> tree;
  Fenwick(int _N):N(_N) { tree.resize(N + 1); }
  void update(int idx, T data) {
    for( ; idx <= N; idx += idx & -idx) tree[idx] += data;
  }
  T query(int idx) {
    T ret = 0;
    for( ; idx; idx -= idx & -idx) ret += tree[idx];
    return ret;
  }
  T query(int l, int r) {
    return query(r) - query(l - 1);
  }
};
```

1.6 Hld

```
// Query O(logN) or O(log^2N) Update O(logN) or O(log^2N)
struct HLD {
  Graph<int> G;
  vector<int> par, top, dep, siz, in, out;
  Segment<int> seg; // Option
  int id;
  HLD(Graph<int> G):G(G) {
    int N = (int)G.size();
```

```
    siz = par = top = dep = in = out = vector<int>(N);
    seg = Segment<int>(N); // Option
    id = 0;
  }
  void dfs(int cur=1, int prev=0) {
    siz[cur] = 1;
    par[cur] = prev;
    dep[cur] = dep[prev] + 1;
    for(int &nxt : G[cur]) {
      if(nxt == prev) continue;
      dfs(nxt, cur);
      siz[cur] += siz[nxt];
      if(siz[nxt] > siz[G[cur][0]]) swap(nxt, G[cur][0]);
    }
  }
  void dfs2(int cur=1, int prev=0) {
    in[cur] = ++id;
    if(cur == 1) top[cur] = 1;
    for(int nxt: G[cur]) {
      if(nxt == prev)continue;
      top[nxt] = (nxt == G[cur][0] ? top[cur] : nxt);
      dfs2(nxt, cur);
    }
    out[cur] = id;
  }
  int lca(int a, int b) {
    while(top[a] != top[b]) {
      if(dep[top[a]] < dep[top[b]]) swap(a, b);
      a = par[top[a]];
    }
    if(in[a] > in[b]) swap(a, b);
    return a;
  }
  void update(int, int);
  int query(int, int);
};
```

1.7 Kdtree

```
// Time Complexity: Build O(Nlog^2N) Query O(logN)
template <typename T>
inline T INF() {
  return numeric_limits<T>::max() / 2;
}
template<typename T> inline T square(T x) { return x * x; }
template<typename T> struct KDTree {
  // axis == 1 ? y : x
  struct Node {
    T x, y;
    int axis;
    T mnx, mxx, mny, mxy;

    Node() {
      mnx = mny = INF<T>();
      mxx = mxy = -INF<T>();
      axis = 0;
    }
    void update(T y, T x) {
      mnx = min(mnx, x); mny = min(mny, y);
      mxx = max(mxx, x); mxy = max(mxy, y);
    }
    T dis(pair<T, T> point) {
      T a = point.first - y, b = point.second - x;
      return square(a) + square(b);
    }
    bool operator==(pair<T, T> point) { return make_pair(y, x) == point; }
    bool operator!=(pair<T, T> point) { return make_pair(y, x) != point; }
    bool operator<(pair<T, T> point) { return make_pair(y, x) < point; }
    bool operator>(pair<T, T> point) { return make_pair(y, x) > point; }
};

vector<pair<T, T>> points;
vector<Node> tree;
vector<bool> exist;
T query_answer;
int siz;
KDTree(int N = 1 << 17) {
  for(siz = 1; siz < N; siz <<= 1);
  tree.resize(siz << 1);
  exist.resize(siz << 1);
}
KDTree(const vector<pair<T, T>> &V) : KDTree(V.size()) { points = V; }
void build(int l, int r, int pos) {
  Node cur;
  for(int i = l; i <= r; ++i) {
    auto [y, x] = points[i];
    cur.update(y, x);
  }
  tree[pos] = cur;
  exist[pos] = true;
  if(pos == 1) tree[pos].axis = 0;
  else tree[pos].axis = 1 - tree[pos >> 1].axis;
  if(tree[pos].axis) sort(points.begin() + l, points.begin() + r + 1);
  else sort(points.begin() + l, points.begin() + r + 1, [&](const pair<T, T> &a, const
  pair<T, T> &b) { return a.second != b.second ? a.second < b.second : a.first < b.first;
  });
  int mid = (l + r) / 2;
  tree[pos].y = points[mid].first;
  tree[pos].x = points[mid].second;
  if(l <= mid - 1) build(l, mid - 1, pos << 1);
  if(mid + 1 <= r) build(mid + 1, r, pos << 1 | 1);
}
void build() { build(0, (int)points.size() - 1, 1); }
void query(int pos, pair<T, T> point) {
  if(tree[pos] != point) query_answer = min(query_answer, tree[pos].dis(point));
  if(tree[pos].axis) { // y
    if(point.first < tree[pos].y) {
      if(exist[pos << 1]) query(pos << 1, point);
      if(exist[pos << 1 | 1] && square(tree[pos << 1 | 1].mny - point.first) <
      query_answer) query(pos << 1 | 1, point);
    }
    else {
      if(exist[pos << 1 | 1]) query(pos << 1 | 1, point);
```

```
      if(exist[pos << 1] && square(tree[pos << 1].mxy - point.first) < query_answer)
      query(pos << 1, point);
    }
  }
  else {
    if(point.second < tree[pos].x) {
      if(exist[pos << 1]) query(pos << 1, point);
      if(exist[pos << 1 | 1] && square(tree[pos << 1 | 1].mnx - point.second) <
      query_answer) query(pos << 1 | 1, point);
    }
    else {
      if(exist[pos << 1 | 1]) query(pos << 1 | 1, point);
      if(exist[pos << 1] && square(tree[pos << 1].mxx - point.second) < query_answer)
      query(pos << 1, point);
    }
  }
 }
 T query(pair<T, T> point) {
  query_answer = INF<T>();
  query(1, point);
  return query_answer;
 }
};
```

1.8 Lichao

```
// Not Tested, 12795 source code
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;

struct Line {
 ll a, b;
 Line() : Line(0) { }
 Line(ll _a) : Line(_a, -1e18) { }
 Line(ll _a, ll _b) : a(_a), b(_b) { }
 ll operator[](const ll x) { return a * x + b; }
};

struct LiChao {
 struct Node {
  int l, r;
  ll xl, xr;
  Line line;
 };
 vector<Node> nodes;

 void init(ll xmn, ll xmx) {
  nodes.push_back({-1, -1, xmn, xmx, {0, static_cast<ll>(-1e18)}});
 }

 void insert(int pos, Line data) {
  ll xl = nodes[pos].xl, xr = nodes[pos].xr;
  ll xm = xl + xr >> 1;

  Line low = nodes[pos].line;
  Line high = data;
  if(low[xl] >= high[xl]) swap(low, high);
  if(low[xr] <= high[xr]) {
   nodes[pos].line = high;
   return;
  }
  else if(low[xm] <= high[xm]) {
   nodes[pos].line = high;
   if(!~nodes[pos].r) {
    nodes[pos].r = nodes.size();
    nodes.push_back({-1, -1, xm + 1, xr, {0, static_cast<ll>(-1e18)}});
   }
   insert(nodes[pos].r, low);
  }
  else {
   nodes[pos].line = low;
   if(!~nodes[pos].l) {
    nodes[pos].l = nodes.size();
    nodes.push_back({-1,-1,xl,xm,{0,static_cast<ll>(-1e18)}});
   }
   insert(nodes[pos].l, high);
  }
 }
 void insert(Line newLine) { insert(0, newLine); }

 ll get(int pos, ll x) {
  if(!~pos) return static_cast<ll>(-1e18);
  ll xl = nodes[pos].xl, xr = nodes[pos].xr;
  ll xm = xl + xr >> 1;
  if(x <= xm) return max(nodes[pos].line[x], get(nodes[pos].l, x));
  else return max(nodes[pos].line[x], get(nodes[pos].r, x));
 }
 ll get(ll x) { return get(0, x); }
};

LiChao tree;

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

 int Q; cin >> Q;
 tree.init(-2e12, 2e12);
 while(Q --> 0) {
  int cmd; cin >> cmd;
  if(cmd == 1) {
   ll a, b; cin >> a >> b;
   tree.insert({a, b});
  }
  else {
   ll x; cin >> x;
   cout << tree.get(x) << '\n';
  }
 }

 return 0;
}
```

1.9 Merge Sort Tree

```
template <typename T>
struct MergesortTree {
  vector<vector<T>> tree;
  int siz;
  MergesortTree(int N) {
    for(siz = 1; siz < N; siz <<= 1);
    tree.resize(siz << 1);
  }
  void build(const vector<T> &V) {
    int N = (int)V.size();
    for(int i = 0; i < N; ++i) {
      tree[siz + i].push_back(V[i]);
    }
    for(int i = siz - 1; i; --i) {
      auto &L = tree[i << 1];
      auto &R = tree[i << 1 | 1];
      merge(L.begin(), L.end(), R.begin(), R.end(), back_inserter(tree[i]));
    }
  }
  int query(int l, int r, int s, int e, int pos, int k) {
    if(s <= l && r <= e) return tree[pos].end() - upper_bound(tree[pos].begin(),
    tree[pos].end(), k);
    if(e < l || r < s) return 0;
    int mid = (l + r) / 2;
    return query(l, mid, s, e, pos << 1, k) + query(mid + 1, r, s, e, pos << 1 | 1, k);
  }
  int query(int s, int e, int k) {
    return query(0, siz - 1, s, e, 1, k);
  }
};
```

1.10 Palindrome Tree

```
// Not Tested
template <typename T = int>
struct PalindromeTree {
  struct Node {
    int len, suffix_link;
    T cnt;
    map<char, int> nxt;

    Node() : Node(0, 0) { }
    Node(int _len, int _link) {
      len = _len; suffix_link = _link;
      cnt = T();
    }
  };

  vector<Node> tree;
  int cnt, last_suffix;

  PalindromeTree(int N) {
    tree.resize(N);
    cnt = last_suffix = 2;
    tree[1] = Node(-1, 1);
    tree[2] = Node(0, 1);
  }
  void init(const string &S) {
    auto chk = [&](int idx, int cur) {
      return idx - tree[cur].len - 1 >= 0 && S[idx - tree[cur].len - 1] == S[idx];
    };
    for(int i = 0; i < (int)S.size(); ++i) {
      int cur = last_suffix;
      while(!chk(i, cur)) cur = tree[cur].suffix_link;
      if(tree[cur].nxt.count(S[i])) {
        last_suffix = tree[cur].nxt[S[i]];
        ++ tree[last_suffix].cnt;
        continue;
      }
      last_suffix = tree[cur].nxt[S[i]] = ++cnt;
      int nxt = cnt;
      tree[nxt].len = tree[cur].len + 2;
      ++ tree[nxt].cnt;
      if(tree[nxt].len == 1) {
        tree[nxt].suffix_link = 2;
        continue;
      }
      while(cur > 1) {
        cur = tree[cur].suffix_link;
        if(chk(i, cur)) {
          tree[nxt].suffix_link = tree[cur].nxt[S[i]];
          break;
        }
      }
    }
  }
  Node& operator[](const int &idx) { return tree[idx]; }
};
```

1.11 Pbds

```
// 시간복잡도 set이랑 동일하다고 보면 됨.
#include <ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define ordered_set tree<int, null_type, less_equal<int>,
rb_tree_tag,tree_order_statistics_node_update>
// multiset처럼 less_equal<int>
// set처럼 less<int>
ordered_set pbds;
pbds.insert(x);
pbds.erase(x); // multiset처럼 쓸 때 주의
*pbds.find_by_order(x);
*pbds.find_by_key(x);
```

1.12 Pst

```
// Query O(logN)
template <typename T>
struct PST {
  struct Node {
    Node *left, *right;
    T data;
    Node(Node *l = nullptr, Node *r = nullptr, T v=0):left(l), right(r), data(v) { }
```

```
    Node *push(int l, int r, int x, T _data) {
      if(r < x || x < l) return this;
      if(l == r) return new Node(0, 0, this->data + _data);
      int mid = l + (r - l) / 2;
      Node *L = left->push(l, mid, x, _data);
      Node *R = right->push(mid + 1, r, x, _data);
      return new Node(L, R, L->data + R->data);
    }
  };
  Node *roots[100002];
  int siz;

  PST() { setting(); }
  PST(int N) { setting(N); }
  void setting(int N = 2e9 + 10){
    siz = N;
    roots[0] = new Node();
    roots[0]->left = roots[0]->right = roots[0];
  }

  void expand(int p){ roots[p] = roots[p - 1]; }
  void update(int p, int idx, T data, bool _expand=false){
    if(_expand) expand(p);
    roots[p] = roots[p]->push(1, siz, idx, data);
  }

  T query(Node *cur, int l, int r, int s, int e){
    if(s <= l && r <= e)return cur->data;
    if(e <  l || r <  s)return 0;
    int mid = l + (r - l) / 2;
    return query(cur->left, l, mid, s, e) + query(cur->right, mid + 1, r, s, e);
  }
  T query(int s, int e, int p){ return query(roots[p], 1, siz, s, e); }

  T kth(Node *s, Node *e, int l, int r, int k){
    if(l == r)return l;
    int mid = l + (r - l) / 2;
    T data = e->left->data - s->left->data;
    if(data >= k)return kth(s->left, e->left, l, mid, k);
    return kth(s->right, e->right, mid + 1, r, k - data);
  }
  T kth(int s, int e, int k){ return kth(roots[s], roots[e], 1, siz, k); }
};
```

1.13 Removal Priority Queue

```
// Not Tested
template <typename T, typename Compare = less<T>>
struct RPQ {
  struct Node {
    T data;
    int id;
    Node():Node(T(), -1) { }
    Node(T _data, int _id):data(_data),id(_id) {}
  };
  vector<Node> heap;
  vector<int> inverted_index;
  Compare comp;
  RPQ():comp() { heap.resize(1); }
  RPQ(const vector<T> &V):comp() {
    int N = (int)V.size();
    heap.reserve(N + 1); heap.resize(N + 1);
    inverted_index.reserve(N + 1); inverted_index.resize(N + 1);
    for(int i = 1; i <= N; ++i) {
      heap[i] = Node(V[i - 1], i);
      inverted_index[i] = i;
    }
    for(int i = N; i >= 1; --i) heapify(i);
  }
  void heapify(int start_index = 1) {
    int cur = start_index, N = size();
    while(cur * 2 + 1 <= N) {
      int nxt = cur;
      if(comp(heap[nxt].data, heap[cur * 2].data)) nxt = cur * 2;
      if(comp(heap[nxt].data, heap[cur * 2 + 1].data)) nxt = cur * 2 + 1;
      if(cur != nxt) {
        swap(heap[cur], heap[nxt]);
        inverted_index[heap[cur].id] = cur;
        inverted_index[heap[nxt].id] = nxt;
        cur = nxt;
      }
      else break;
    }
  }
  T top() {
    assert((int)heap.size() > 0);
    return heap[1].data;
  }
  void pop() {
    heap[1] = heap.back(); heap.pop_back();
    heapify();
  }
  void remove(int idx) {
    assert(0 <= idx && idx < (int)inverted_index.size());
    int x = inverted_index[idx];
    assert(x > 0);
    heap[x] = heap.back(); heap.pop_back();
    heapify(x);
    inverted_index[idx] = -1;
  }
  void push_update() {
    int idx = size();
    while(idx != 1) {
      if(comp(heap[idx >> 1].data, heap[idx].data)) {
        swap(heap[idx], heap[idx >> 1]);
        inverted_index[heap[idx].id] = idx;
        idx >>= 1;
        inverted_index[heap[idx].id] = idx;
      }
      else break;
    }
  }
  int push(T data) {
    int id = inverted_index.size();
```

```
    heap.emplace_back(data, id);
    inverted_index.push_back(size());
    push_update();
    return id;
  }
  void update(int idx, T data) {
    assert(0 <= idx && idx < (int)inverted_index.size());
    int x = inverted_index[idx], N = size();
    assert(x > 0);
    heap[x].data = data;
    swap(heap[x], heap[N]);
    inverted_index[heap[x].id] = x;
    inverted_index[heap[N].id] = N;
    heapify(x);
    push_update();
  }
  int size() { return (int)heap.size() - 1; }
  bool empty() { return size() == 0; }
};
```

1.14 Rmq

```
// Need Graph Template 1-indexed
// build O(NlogN), query O(1) RMQ
template<typename T> struct RMQ {
  vector<int> L;
  vector<vector<T>> table;
  RMQ() { }
  RMQ(const vector<T> &V) {
    int N = (int)V.size() - 1;
    L = vector<int>(N + 1);
    for(int i = 2; i <= N; ++i) L[i] = L[i / 2] + 1;
    table = vector<vector<T>>(L[N] + 1, vector<T>(N + 1));
    for(int i = 1; i <= N; ++i) table[0][i] = V[i];
    for(int i = 1; i <= L[N]; ++i) {
      int k = 1 << (i - 1);
      for(int j = 1; j + k <= N; ++j) {
        table[i][j] = merge(table[i - 1][j], table[i - 1][j + k]);
      }
    }
  }
  T query(int l, int r) {
    int d = L[r - l + 1];
    return merge(table[d][l], table[d][r - (1 << d) + 1]);
  }
  T merge(T a, T b) { return min(a, b); }
};
// build O(NlogN), query O(1) LCA
// 1-indexed
struct LCA {
  Graph<int> G;
  RMQ<pair<int, int>> rmq;
  vector<int> in, dep;
  LCA(Graph<int> _G):G(_G) {
    int id = 0;
    int N = G.size();
    in = dep = vector<int>(N + 1);
    vector<pair<int, int>> V(1);
    function<void(int, int)> dfs = [&](int cur, int prev) -> void {
      in[cur] = ++id;
      dep[cur] = dep[prev] + 1;
      V.emplace_back(dep[cur], cur);
      for(int nxt: G[cur]) {
        if(nxt == prev) continue;
        dfs(nxt, cur);
        V.emplace_back(dep[cur], cur);
        ++ id;
      }
    };
    dfs(1, 0);
    rmq = RMQ<pair<int, int>>(V);
  }
  int lca(int u, int v) {
    if(in[u] > in[v]) swap(u, v);
    return rmq.query(in[u], in[v]).second;
  }
};
```

1.15 Rope

```
// O(logN)
#include <ext/rope>
using namespace __gnu_cxx;
string S;
crope rp = S.c_str();
rp.push_back('a');
rp.insert(0, "asdf");
rp.erase(0, 1);
rp.replace(0, 1, "asdf");
rp.substr(0, 2); // idx, cnt
rp.pop_back();
rp += rp2;
```

1.16 Segment Tree

```
template <typename T>
struct Segment {
  vector<T> tree;
  int siz;

  Segment(int N = 1 << 17) {
    for(siz = 1; siz < N; siz <<= 1);
    tree = vector<T>(siz << 1);
  }
  void build() {
    for(int i = siz - 1; i > 0; --i) {
      tree[i] = tree[i << 1] + tree[i << 1 | 1];
    }
  }
  void update(int idx, T data) {
    tree[idx += siz] = data;
    while(idx >>= 1) tree[idx] = tree[idx << 1] + tree[idx << 1 | 1];
  }
  T query(int l, int r) {
    T ret_L = T(), ret_R = T();
    for(l += siz, r += siz; l <= r; l >>= 1, r >>= 1) {
```

```
      if(l & 1) ret_L = ret_L + tree[l ++];
      if(~r & 1) ret_R = tree[r --] + ret_R;
    }
    return ret_L + ret_R;
  }
  T& operator[](const int &idx) { return tree[idx + siz]; }
};
```

1.17 Segment Tree With Lazy

```
template <typename T>
struct SegmentLazy {
  vector<T> tree, lazy;
  int siz;

  SegmentLazy(int N = 1 << 17) {
    for(siz = 1; siz < N; siz <<= 1);
    lazy = tree = vector<T>(siz << 1);
  }
  void putItem(int idx, T data) { tree[idx + siz] = data; }
  void build() {
    for(int i = siz - 1; i; --i) tree[i] = merge(tree[i << 1], tree[i << 1 | 1]);
  }
  void propagate(int l, int r, int pos) {
    if(!lazy[pos]) return;
    if(l != r) {
      lazy[pos << 1] = merge(lazy[pos << 1], lazy[pos]);
      lazy[pos << 1 | 1] = merge(lazy[pos << 1 | 1], lazy[pos]);
    }
    tree[pos] += lazy[pos] * (r - l + 1);
    lazy[pos] = 0;
  }
  void update(int l, int r, int s, int e, int pos, T data) {
    if(s <= l && r <= e) {
      lazy[pos] += data;
      propagate(l, r, pos);
      return;
    }
    propagate(l, r, pos);
    if(e <  l || r <  s) return;
    int mid = (l + r) / 2;
    update(l, mid, s, e, pos << 1, data);
    update(mid + 1, r, s, e, pos << 1 | 1, data);
    tree[pos] = merge(tree[pos << 1], tree[pos << 1 | 1]);
  }
  void update(int s, int e, T data) { update(0, siz - 1, s, e, 1, data); }
  T query(int l, int r, int s, int e, int pos) {
    propagate(l, r, pos);
    if(s <= l && r <= e) return tree[pos];
    if(e <  l || r <  s) return 0;
    int mid = (l + r) / 2;
    return merge(query(l, mid, s, e, pos << 1), query(mid + 1, r, s, e, pos << 1 | 1));
  }
  T query(int s, int e) { return query(0, siz - 1, s, e, 1); }
  T merge(T a, T b) {
    return a + b;
  }
};
```

1.18 Splay

```
template <typename T, const int node_siz = 500000>
struct SplayTree {
  struct Data {
    int sz;
    T mn, mx, sum, value;
    bool flip, dummy;

    Data(T _value = 0) : value(_value) {
      init();
      flip = dummy = false;
    }

    void init() {
      sz = 1;
      mn = mx = sum = value;
    }
  };

  struct Node {
    Node *l, *r, *p;
    Data data;

    Node():Node(0){}
    Node(T _value) : Node(_value, nullptr) {}

    Node(T _value, Node *_p) {
      p = _p;
      l = r = nullptr;
      data = Data(_value);
    }

    bool is_left() { return this == p->l; }

    bool is_right() { return this == p->r; }

    bool is_root() { return p == nullptr || (!is_left() && !is_right()); }

    void merge(Data o) {
      data.sz += o.sz;
      data.mn = min(data.mn, o.mn);
      data.mx = max(data.mx, o.mx);
      data.sum += o.sum;
    }

    void update() {
      data.init();
      if (l) merge(l->data);
      if (r) merge(r->data);
    }

    void push() {
      if (data.flip == false) return;
      swap(l, r);
      data.flip = false;
```

```
    if (l) l->data.flip ^= 1;
    if (r) r->data.flip ^= 1;
  }
};

Node *root;
Node *node[node_siz];

SplayTree() {}

SplayTree(int N, const vector<T> &V) { init(N, V); }

void init(int N, const vector<T> &V) {
  const T INF = numeric_limits<T>::max() / 2;
  Node *cur = root = new Node(-INF);
  for (int i = 1; i <= N; ++i) {
    node[V[i]] = cur->r = new Node(V[i], cur);
    cur = cur->r;
  }
  node[N + 1] = cur->r = new Node(INF, cur);
  root->data.dummy = cur->r->data.dummy = true;
  for (int i = N + 1; i >= 1; --i) node[i]->update();
}

void rotate(Node *cur) {
  if (cur->p == nullptr) return;
  Node *p = cur->p;
  p->push();
  cur->push();

  if (cur->is_left()) {
    if (cur->r) cur->r->p = p;
    p->l = cur->r;
    cur->r = p;
  }
  else {
    if (cur->l) cur->l->p = p;
    p->r = cur->l;
    cur->l = p;
  }

  if (!p->is_root()) {
    if (p->is_left()) p->p->l = cur;
    else p->p->r = cur;
  }
  else root = cur;

  cur->p = p->p;
  p->p = cur;

  p->update();
  cur->update();
}

Node *splay(Node *cur, Node *g = nullptr) {
  while (!cur->is_root() && cur->p != g) {
    if (cur->p->is_root() || cur->p->p == g) {
      rotate(cur);
      continue;
    }
    if (cur->p->p != g) rotate((cur->is_left() ^ cur->p->is_left()) ? cur : cur->p);
    rotate(cur);
  }
  if (g == nullptr) root = cur;
  return root;
}

Node *kth(int k) {
  Node *cur = root;
  cur->push();
  while (true) {
    while (cur->l && cur->l->data.sz > k) {
      cur = cur->l;
      cur->push();
    }
    if (cur->l) k -= cur->l->data.sz;
    if (!k--) break;
    cur = cur->r;
    cur->push();
  }
  return splay(cur);
}

Node *gather(int l, int r) {
  Node *Left = kth(r + 1);
  Node *Right = kth(l - 1);
  return splay(Left, Right)->r->l;
}

Node *flip(int l, int r) {
  Node *cur = gather(l, r);
  cur->data.flip ^= 1;
  return cur;
}

Node *shift(int l, int r, int x) {
  Node *cur = gather(l, r);
  if (x >= 0) {
    x %= (r - l + 1);
    if (x == 0) return cur;
    flip(l, r);
    flip(l, l + x - 1);
    flip(l + x, r);
  }
  else {
    x *= -1;
    x %= (r - l + 1);
    flip(l, r);
    flip(l, r - x);
    flip(r - x + 1, r);
  }
  return gather(l, r);
```

```
  }

  int operator[](const int k) {
    return splay(node[k])->l->data.sz;
  }

  // Debuging
  void inorder(Node *cur) {
    if(cur == nullptr) return;
    cur->push();
    inorder(cur->l);
    if (cur->data.dummy == false) cout << cur->data << ' ';
    inorder(cur->r);
  }

  void inorder() { inorder(root); }
};
```

1.19 Union Find Roll Back

```
struct UnionFind {
  vector<int> par, rank;
  stack<tuple<int, int, int>> st;
  UnionFind(int N) {
    par = rank = vector<int>(N + 1);
    iota(par.begin(), par.end(), 0);
  }

  int find(int x) { return par[x] == x ? x : find(par[x]); }
  bool merge(int u, int v) {
    u = find(u); v = find(v);
    if(u == v) return false;
    if(rank[u] < rank[v]) swap(u, v);
    par[v] = u;
    st.emplace(u, v, rank[u] == rank[v]);
    if(rank[u] == rank[v]) ++rank[u];
    return true;
  }
  void revert(int cnt) {
    while(cnt --> 0) {
      auto [u, v, c] = st.top(); st.pop();
      par[v] = v;
      if(c) -- rank[u];
    }
  }
  int conn(int u, int v) { return find(u) == find(v); }
};
```

2 Graph

2.1 Dinic

```
// O(V^2E)
struct Dinic {
  struct Node {
    int node_idx, cost, flow, rev;
    Node(int _nxt = -1, int _cost = 0, int _rev =
    -1):node_idx(_nxt),cost(_cost),flow(0),rev(_rev) { }
    int spare() { return cost - flow; }
    void setRev(int _rev) { rev = _rev; }
  };
  vector<Node> nodes;
  vector<vector<int>> G;
  vector<int> level;
  vector<int> work;

  int src, snk, asrc, asnk, N;
  Dinic(int _N) {
    src = _N + 1;
    snk = src + 1;
    asrc = snk + 1;
    asnk = asrc + 1;
    N = asnk;
    G.resize(N + 1);
  }
  bool bfs(int s, int e) {
    level = vector<int>(N + 1, -1);
    level[s] = 0;
    queue<int> Q; Q.push(s);
    while(!Q.empty()) {
      int cur = Q.front(); Q.pop();
      for(const int &x: G[cur]) {
        Node &nxt = nodes[x];
        if(nxt.spare() > 0 && level[nxt.node_idx] == -1) {
          level[nxt.node_idx] = level[cur] + 1;
          Q.push(nxt.node_idx);
        }
      }
    }
    return ~level[e];
  }
  int dfs(int s, int e, int f) {
    if(s == e) return f;
    for(int &i = work[s]; i < (int)G[s].size(); ++i) {
      Node &nxt = nodes[G[s][i]];
      if(nxt.spare() > 0 && level[nxt.node_idx] == level[s] + 1) {
        int ret = dfs(nxt.node_idx, e, min(f, nxt.spare()));
        if(ret > 0) {
          nxt.flow += ret;
          nodes[nxt.rev].flow -= ret;
          return ret;
        }
      }
    }
    return 0;
  }
  int flow(int s, int e) {
    int ret = 0;
    while(bfs(s, e)) {
      work = vector<int>(N + 1, 0);
      while(true) {
        int x = dfs(s, e, numeric_limits<int>::max());
        if(x == 0) break;
        ret += x;
      }
```

```
    }
    return ret;
  }
  void addEdge(int u, int v, int cost, bool is_directed = true, bool is_unique = false) {
    if(is_unique) {
      for(const int &x: G[u]) {
        if(nodes[x].node_idx == v) {
          nodes[x].cost += cost;
          if(!is_directed) return;
          break;
        }
      }
      if(!is_directed) {
        for(const int &x: G[v]) {
          if(nodes[x].node_idx == u) {
            nodes[x].cost += cost;
            return;
          }
        }
      }
    }
    int a = (int)nodes.size(), b = a + 1;
    Node uv = Node(v, cost, b);
    Node vu = Node(u, is_directed ? 0 : cost, a);
    nodes.push_back(uv); nodes.push_back(vu);
    G[u].push_back(a); G[v].push_back(b);
  }
  void addLREdge(int u, int v, int lower, int upper) {
    if(lower) {
      addEdge(asrc, v, lower);
      addEdge(u, asnk, lower);
    }
    addEdge(u, v, upper - lower);
  }
  int flow() { return flow(src, snk); }
  int lrflow() { return flow(asrc, asnk); }
};
```

2.2 Hungarian

```
// Kactl Template, O(N^3)
pair<int, vector<int>> hungarian(const vector<vector<int>> &a) {
  if (a.empty()) return {0, {}};
  int n = (int)a.size() + 1, m = (int)(a[0].size()) + 1;
  vector<int> u(n), v(m), p(m), ans(n - 1);
  for(int i = 1; i < n; ++i) {
    p[0] = i;
    int j0 = 0; // add "dummy" worker 0
    vector<int> dist(m, INT_MAX), pre(m, -1);
    vector<bool> done(m + 1);
    do { // dijkstra
      done[j0] = true;
      int i0 = p[j0], j1, delta = INT_MAX;
      for(int j = 1; j < m; ++j) if (!done[j]) {
        auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
        if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
        if (dist[j] < delta) delta = dist[j], j1 = j;
      }
      for(int j = 0; j < m; ++j) {
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
      }
      j0 = j1;
    } while (p[j0]);
    while (j0) { // update alternating path
      int j1 = pre[j0];
      p[j0] = p[j1], j0 = j1;
    }
  }
  for(int j = 1; j < m; ++j) if (p[j]) ans[p[j] - 1] = j - 1;
  return {-v[0], ans}; // min cost
}
```

2.3 Mcmf

```
// < O(VEf)
template <typename T>
struct MinCostMaxFlow {
  struct Edge {
    int edge_id, node_idx, cost, flow, rev;
    T dist;
    Edge(int _edge_id, int _node_idx, int _cost, T _dist, int
    _rev):edge_id(_edge_id),node_idx(_node_idx),cost(_cost),flow(0),dist(_dist),rev(_rev) {
    }
    int spare() { return cost - flow; }
  };

  vector<Edge> edges;
  vector<vector<int>> G;
  vector<pair<int, int>> par;
  vector<T> dist;

  int src, snk, N;
  T INF;

  MinCostMaxFlow(int _N) {
    src = _N + 1;
    snk = src + 1;
    N = snk;

    INF = numeric_limits<T>::max();

    G.resize(N + 1);
    par.resize(N + 1, make_pair(-1, -1));
  }
  bool spfa(int s, int e) {
    vector<int> InQ(N + 1);
    dist = vector<T>(N + 1, INF);

    dist[s] = 0;
    deque<int> dq; dq.push_back(s);
    InQ[s] = 1;

    while(!dq.empty()) {
      int cur = dq.front(); dq.pop_front();
```

```
      InQ[cur] = 0;
      for(const int &x: G[cur]) {
        Edge &e = edges[x];
        if(e.spare() > 0 && dist[e.node_idx] > dist[cur] + e.dist) {
          dist[e.node_idx] = dist[cur] + e.dist;
          par[e.node_idx] = make_pair(cur, e.edge_id);
          if(InQ[e.node_idx] == 0) {
            dq.push_back(e.node_idx);
            InQ[e.node_idx] = 1;
          }
        }
      }
    }
    return dist[e] != INF;
  }

  // min_cost, max_flow
  pair<T, int> flow_after_spfa(int s, int e) {
    int mn = numeric_limits<int>::max();
    for(int cur = e; cur != s; cur = par[cur].first) {
      mn = min(mn, edges[par[cur].second].spare());
    }
    if(mn == 0) return make_pair<T, int>(-1, -1);
    T min_cost = 0;
    int max_flow = mn;
    for(int cur = e; cur != s; cur = par[cur].first) {
      min_cost += (T)mn * edges[par[cur].second].dist;
      edges[par[cur].second].flow += mn;
      edges[edges[par[cur].second].rev].flow -= mn;
    }
    return make_pair(min_cost, max_flow);
  }
  pair<T, int> flow(int s, int e) {
    pair<T, int> ret;
    while (spfa(s, e)) {
      pair<T, int> cur = flow_after_spfa(s, e);
      if (cur.first == -1) break;
      ret.first += cur.first;
      ret.second += cur.second;
    }
    return ret;
  }

  // addEdge
  void addEdge(int u, int v, int cost, T dist) {
    int a = edges.size();
    int b = a + 1;

    Edge uv = Edge(a, v, cost, dist, b);
    Edge vu = Edge(b, u, 0, -dist, a);

    edges.push_back(uv);
    edges.push_back(vu);

    G[u].push_back(a);
    G[v].push_back(b);
  }

  pair<T, int> flow() { return flow(src, snk); }
};
```

2.4 2Sat

```
// 1-indexed, a xor b = (a or b) and (¬a or ¬b)
int getIdx(int x) { return abs(x) << 1 | (x < 0); }
void addEdge(Graph<int> &G, int u, int v) {
  u = getIdx(u), v = getIdx(v);
  G.addEdge(u ^ 1, v); G.addEdge(v ^ 1, u);
}
bool avaiable(Graph<int> &G) {
  SCC scc(G);
  int N = G.size() - 2 >> 1;
  for(int i = 1; i <= N; ++i) {
    if(scc.scc_id[i << 1] == scc.scc_id[i << 1 | 1]) return false;
  }
  return true;
}
```

2.5 Bcc

```
// Need Graph template, 1-indexed
// O(V+E)
struct BCC {
  int N, dfs_id;
  Graph<int> G;
  vector<int> dfsn;
  vector<vector<pair<int, int>>> bcc;
  stack<pair<int, int>> S;
  BCC(Graph<int> _G):G(_G) {
    N = G.size();
    dfsn.resize(N + 1);
    dfs_id = 0;
    for(int i = 1; i <= N; ++i) {
      if(dfsn[i] == 0) dfs(i, 0);
    }
  }

  int dfs(int cur, int prev) {
    int res = dfsn[cur] = ++ dfs_id;
    for(int nxt: G[cur]) {
      if(nxt == prev) continue;
      if(dfsn[cur] > dfsn[nxt]) S.emplace(cur, nxt);
      if(dfsn[nxt] > 0) res = min(res, dfsn[nxt]);
      else {
        int tmp = dfs(nxt, cur);
        res = min(res, tmp);
        if(tmp >= dfsn[cur]) {
          vector<pair<int, int>> curBCC;
          while(!S.empty() && S.top() != make_pair(cur, nxt)) {
            curBCC.push_back(S.top());
            S.pop();
          }
          curBCC.push_back(S.top());
          S.pop();
          bcc.push_back(curBCC);
```

```
      }
    }
  }
  return res;
}
vector<int> cut_vertex() {
  vector<int> cnt(N + 1), last(N + 1, -1);
  for(int i = 0; i < (int)bcc.size(); ++i) {
    for(auto [u, v]: bcc[i]) {
      if(last[u] < i) ++ cnt[u], last[u] = i;
      if(last[v] < i) ++ cnt[v], last[v] = i;
    }
  }
  vector<int> vertex;
  for(int i = 1; i <= N; ++i) {
    if(cnt[i] > 1) vertex.push_back(i);
  }
  return vertex;
}
vector<pair<int, int>> cut_edge() {
  vector<pair<int, int>> edges;
  for(int i = 0; i < (int)bcc.size(); ++i) {
    if(bcc[i].size() > 1) continue;
    auto [u, v] = bcc[i][0];
    edges.emplace_back(min(u, v), max(u, v));
  }
  return edges;
}
};
```

2.6 Scc

```
// 1-indexed, Need Graph template
// O(V+E)
struct SCC {
  int N, id;
  Graph<int> G;
  vector<int> D, scc_id;
  vector<vector<int>> scc;
  stack<int> st;

  SCC(const Graph<int> &_G):G(_G) {
    id = 0;
    N = G.size();
    D.resize(N + 1);
    scc_id.resize(N + 1, -1);
    for(int i = 1; i <= N; ++i) if(!D[i]) dfs(i);
  }
  int dfs(int cur) {
    D[cur] = ++id;
    st.push(cur);
    int par = D[cur];
    for(const auto &nxt: G[cur]) {
      if(!D[nxt]) par = min(par, dfs(nxt));
      else if(scc_id[nxt] == -1) par = min(par, D[nxt]);
    }
    if(par == D[cur]) {
      scc.emplace_back();
      while(!st.empty()) {
        int x = st.top(); st.pop();
        scc_id[x] = (int)scc.size() - 1;
        scc.back().push_back(x);
        if(x == cur) break;
      }
    }
    return par;
  }
  int size() { return scc.size(); }
  vector<int> &operator[] (const int idx) { return scc[idx]; }
  Graph<int> graph() {
    int K = size();
    Graph<int> sccG(K);
    for(int i = 1; i <= N; ++i) {
      for(const int &nxt: G[i]) {
        if(scc_id[i] == scc_id[nxt]) continue;
        sccG.addEdge(scc_id[i], scc_id[nxt]);
      }
    }
    for(int i = 0; i < K; ++i) {
      sort(sccG[i].begin(), sccG[i].end());
      sccG[i].erase(unique(sccG[i].begin(), sccG[i].end()), sccG[i].end());
    }
    return sccG;
  }
};
```

2.7 Dominator Tree

```
// O((V+E)logV)
vector<int> DominatorTree(const vector<vector<int>> &G, int start_node) {
  int N = (int)G.size();
  vector<vector<int>> rG(N);
  for (int cur = 0; cur < N; ++cur) {
    for (int nxt : G[cur]) rG[nxt].push_back(cur);
  }

  vector<int> uf(N), sdom_id(N), idom(N, -1), sdom(N, -1);
  for (int i = 0; i < N; ++i) uf[i] = sdom_id[i] = i;
  function<int(int)> find = [&](int x) -> int {
    if (uf[x] == x) return x;
    int tmp = find(uf[x]);
    if (sdom[sdom_id[x]] > sdom[sdom_id[uf[x]]]) sdom_id[x] = sdom_id[uf[x]];
    return uf[x] = tmp;
  };
  vector<int> numbering, par(N);
  function<void(int)> dfs = [&](int cur) -> void {
    sdom[cur] = numbering.size();
    numbering.push_back(cur);
    for (int nxt : G[cur]) {
      if (sdom[nxt] != -1) continue;
      par[nxt] = cur;
      dfs(nxt);
    }
  };
  dfs(start_node);
```

```
  int K = (int)numbering.size();
  vector<vector<int>> buf(N);
  vector<int> final_uf(N);
  for (int i = K - 1; i >= 0; --i) {
    int u = numbering[i];
    if (sdom[u] == -1) continue;
    for (int v : rG[u]) {
      if (sdom[v] == -1) continue;
      find(v);
      if (sdom[u] > sdom[sdom_id[v]]) sdom[u] = sdom[sdom_id[v]];
    }
    buf[numbering[sdom[u]]].push_back(u);
    for (int nxt : buf[par[u]]) {
      find(nxt);
      final_uf[nxt] = sdom_id[nxt];
    }
    buf[par[u]].clear();
    uf[u] = par[u];
  }
  idom[start_node] = start_node;
  for (const int &x : numbering) {
    if (sdom[x] == sdom[final_uf[x]]) idom[x] = sdom[x];
    else idom[x] = idom[final_uf[x]];
  }
  for (const int &x : numbering) {
    if (x != start_node) idom[x] = numbering[idom[x]];
  }
  return idom;
}
```

2.8 Gomory Hu

```
vector<int> par(N);
int ans = 0;
for (int i = 1; i < N; ++i) {
  Dinic dinic(N);
  for (auto [u, v]: edges) dinic.addEdge(u, v, 1, false);
  int src = i, snk = par[i];
  int flow = dinic.flow(src, snk);
  ans = max(ans, flow);
  for (int j = i + 1; j < N; ++j) {
    if (dinic.level[j] != -1 && par[j] == par[i]) par[j] = i;
  }
}
```

2.9 Lca

```
// 1-index, dist (ll), Need Graph Template
struct LCA {
  int N, sz;
  Graph<pair<int, int>> G;
  vector<int> dep;
  vector<ll> dist;
  vector<vector<int>> par;
  LCA(const Graph<pair<int, int>> &_G):G(_G) {
    for(sz = 1; (1 << sz) < N; ++ sz);
    N = G.size();
    dep = vector<int>(N + 1);
    dist = vector<ll>(N + 1);
    par = vector<vector<int>>(sz, vector<int>(N + 1));
    dfs(1, 0);
    for(int j = 1; j < sz; ++j) for(int i = 1; i <= N; ++i) par[j][i] = par[j - 1][par[j -
    1][i]];
  }
  void dfs(int cur, int prev) {
    dep[cur] = dep[prev] + 1;
    for(const auto &[nxt, w]: G[cur]) {
      if(nxt == prev) continue;
      par[0][nxt] = cur;
      dist[nxt] = dist[cur] + w;
      dfs(nxt, cur);
    }
  }
  int lca(int u, int v) {
    if(dep[u] > dep[v]) swap(u, v);
    for(int i = sz - 1; ~i; --i) if(dep[u] <= dep[par[i][v]]) v = par[i][v];
    if(u == v) return u;
    for(int i = sz - 1; ~i; --i) if(par[i][u] != par[i][v]) u = par[i][u], v = par[i][v];
    return par[0][u];
  }
  ll distance(int u, int v) { return dist[u] + dist[v] - 2 * dist[lca(u, v)]; }
  int kth(int u, int v, int k) {
    int l = lca(u, v), dif = dep[u] - dep[l] + 1;
    if(dif < k) k = dep[v] - dep[l] + dif - k, u = v, v = l;
    else --k;
    for(int i = sz - 1; ~i; --i) if(k & (1 << i)) u = par[i][u];
    return u;
  }
};
```

2.10 Tree Isomorphism

```
// Need Graph Template
struct TreeIsomorphism {
  string tree_str;
  TreeIsomorphism(Graph<int> &G) {
    int N = G.size();
    function<vector<int>()> get_center = [&]() -> vector<int> {
      vector<int> ind(N), cand;
      for (int i = 0; i < N; ++i) {
        ind[i] = G[i].size();
        if (ind[i] < 2) cand.push_back(i);
      }
      int cnt = N;
      while (cnt > 2) {
        vector<int> tmp;
        for (int x : cand) {
          --cnt;
          for (int y : G[x]) if (--ind[y] == 1) tmp.push_back(y);
        }
        cand = tmp;
      }
      return cand;
    };

    function<string(int, int)> make_string = [&](int cur, int prev) -> string {
```

```
      vector<string> child;
      for (int nxt : G[cur]) {
        if (nxt == prev) continue;
        child.push_back(make_string(nxt, cur));
      }
      sort(child.begin(), child.end());
      string ret = "";
      for (const string &s : child) ret += s;
      return "(" + ret + ")";
    };

    if (N == 0) { }
    else {
      vector<int> center = get_center();
      if (center.size() == 1) tree_str = make_string(center[0], -1);
      else tree_str = min(make_string(center[0], -1), make_string(center[1], -1));
    }
  }
  string get() { return tree_str; }
};
```

3 Others

3.1 Fastinput

```
// eof 추가해야함.
#define BUFFERMAX 1 << 19
struct IO {
  char buf[BUFFERMAX];
  char _read() {
    static int idx = BUFFERMAX;
    if(idx == BUFFERMAX) fread(buf, 1, BUFFERMAX, stdin), idx = 0;
    return buf[idx++];
  }
  char readChar() {
    char ret = _read();
    while(ret == 10 || ret == 32) ret = _read();
    return ret;
  }
  string readString() {
    string ret = "";
    char now = _read();
    while(now == 10 || now == 32) now = _read();
    while(true) {
      ret += now;
      now = _read();
      if(now == 10 || now == 32) break;
    }
    return ret;
  }
  template<typename T> T readInt() {
    T ret = 0;
    bool minus = false;
    char now = _read();
    while(now == 10 || now == 32) now = _read();
    if(now == '-') minus = true, now = _read();
    while(48 <= now && now <= 57) {
      ret = ret * 10 + now - 48;
      now = _read();
    }
    if(minus) ret *= -1;
    return ret;
  }
  void read(int &x) { x = readInt<int>(); }
  void read(long long &x) { x = readInt<long long>(); }
  void read(char &x) { x = readChar(); }
  void read(string &x) { x = readString(); }
  template<typename Type, typename... Types> void read(Type &arg, Types &...args) {
  read(arg); read(args...); }
} io;

template<typename T>
IO& operator>> (IO& in, T &x) { in.read(x); return in; }

#define cin io
#define istream IO
```

3.2 Main

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

using namespace std;

#define all(x) (x).begin(), (x).end()
#define rall(x) (x).rbegin(), (x).rend()
#define sz(x) ((int)(x).size())
#define sortall(x) sort(all(x))
#define Unique(x) (x).erase(unique(all(x)), (x).end())
#define compress(x) sortall(x); Unique(x)

typedef bool i1;
typedef char i8;
typedef short i16;
typedef int i32;
typedef long long i64;

typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned int u32;
typedef unsigned long long u64;

typedef float f16;
typedef double f32;
typedef long double f64;

template<typename T> using Vec = vector<T>;
template<typename T> using Que = queue<T>;
template<typename T> using Dec = deque<T>;

template<int fp=0> struct fastio { fastio() { ios::sync_with_stdio(false); cin.tie(0);
if(fp)cout<<fixed<<' '<<setprecision(fp); } };

template<typename First, typename Second> inline istream& operator>>(istream &in,
pair<First, Second> &_data) { in>>_data.first>>_data.second; return in; }
template<typename First, typename Second> inline ostream& operator<<(ostream &out,
pair<First, Second> &_data) { out<<_data.first<<' '<<_data.second; return out; }
```

```cpp
template<typename First, typename Second, typename Third> inline istream&
operator>>(istream &in, tuple<First, Second, Third> &_data) {
in>>get<0>(_data)>>get<1>(_data)>>get<2>(_data); return in; }
template<typename First, typename Second, typename Third> inline ostream&
operator<<(ostream &out, tuple<First, Second, Third> &_data) { out<<get<0>(_data)<<'
'<<get<1>(_data)<<' '<<get<2>(_data); return out; }

template<typename T> auto Vector(const int N, const T& value) { return vector(N, value); }
template<typename...Ts> auto Vector(const int N, Ts... args) { return vector(N,
Vector(args...)); }
template<typename InputType> void in(InputType& x) { cin>>x; }
template<typename InputType, typename... InputTypes> void in(InputType& x, InputTypes&
...y) { cin>>x; in(y...); }
template<typename IterableInputType> void vin(IterableInputType &V, int skip=0) { for(auto
&x: V) if(--skip < 0) cin >> x; }

template<const int p=0, typename OutputType> void out(OutputType x) { cout<<x<<' '; }
template<const int p=0, typename OutputType, typename... OutputTypes> void out(OutputType
x, OutputTypes ...y) { cout<<fixed<<setprecision(p)<<x<<' '; out<p>(y...); }
template<const int p=0, typename IterableOutputType> void vout(const IterableOutputType &V,
int skip=0) { for(auto &x: V) if(--skip<0) out<p>(x); }

template<i64 modulo=numeric_limits<i64>::max(), typename... T> i64 Sum(T... x) { return
(... + x) % modulo; }
template<i64 modulo=numeric_limits<i64>::max(), typename... T> i64 Mul(T... x) { return
(... * x) % modulo; }

constexpr int dy[] = {-1,1,0,0,-1,-1,1,1,-2,-1,1,2,2,1,-1,-2};
constexpr int dx[] = {0,0,-1,1,-1,1,-1,1,1,2,2,1,-1,-2,-2,-1};

int main() {
  fastio<>();



  return 0;
}
```

3.3 Random

```cpp
// (unsigned)chrono::steady_clock::now().time_since_epoch().count())
struct Random {
  mt19937 rd;
  Random() : rd(0x9402) {}
  Random(int seed) : rd(seed) {}
  template <typename T = int> T randint(T l = 0, T r = 32767) { return
  uniform_int_distribution<T>(l, r)(rd); }
  double randouble(double l = 0, double r = 1) { return
  uniform_real_distribution<double>(l, r)(rd); }
};
```

4 Math

4.1 Convolution

```cpp
template <typename T>
void SupersetZetaTransform(vector<T> &V) {
  int N = (int)V.size();
  assert((N & (N - 1)) == 0);
  for(int j = 1; j < N; j <<= 1) {
    for(int i = 0; i < N; ++i) {
      if(i & j) V[i ^ j] += V[i];
    }
  }
}
template<typename T> void SupersetMobiusTransform(vector<T> &V) {
  int N = (int)V.size();
  assert((N & (N - 1)) == 0);
  for(int j = 1; j < N; j <<= 1) {
    for(int i = 0; i < N; ++i) {
      if(i & j) V[i ^ j] -= V[i];
    }
  }
}
template<typename T> void SubsetZetaTransform(vector<T> &V) {
  int N = (int)V.size();
  assert((N & (N - 1)) == 0);
  for(int j = 1; j < N; j <<= 1) {
    for(int i = 0; i < N; ++i) {
      if(i & j) V[i] += V[i ^ j];
    }
  }
}
template<typename T> void SubsetMobiusTransform(vector<T> &V) {
  int N = (int)V.size();
  assert((N & (N - 1)) == 0);
  for(int j = 1; j < N; j <<= 1) {
    for(int i = 0; i < N; ++i) {
      if(i & j) V[i] -= V[i ^ j];
    }
  }
}
template<typename T> vector<T> AndConvolution(vector<T> A, vector<T> B) {
  SupersetZetaTransform(A);
  SupersetZetaTransform(B);
  for(int i = 0; i < A.size(); ++i) A[i] *= B[i];
  SupersetMobiusTransform(A);
  return A;
}

template<typename T> vector<T> OrConvolution(vector<T> A, vector<T> B) {
  SubsetZetaTransform(A);
  SubsetZetaTransform(B);
  for(int i = 0; i < A.size(); ++i) A[i] *= B[i];
  SubsetMobiusTransform(A);
  return A;
}
template<typename T> T AND(const vector<T> &A, const int K) {
  T ret = AndConvolution(A, A)[K];
  return ret - A[K] >> 1;
}
template<typename T> T OR(const vector<T> &A, const int K) {
  T ret = OrConvolution(A, A)[K];
  return ret - A[K] >> 1;
}
```

```cpp
template<typename T> T XOR(const vector<T> &A, const int K) {
  T ret = 0;
  for(int i = 0; i < A.size(); ++i) ret += A[i] * A[i ^ K];
  if(K == 0) for(int x: A) ret -= x;
  return ret >> 1;
}
```

4.2 Crt

```cpp
template <typename T>
pair<T, T> ext_gcd(T a, T b) {
  if (b == 0) return make_pair(1, 0);
  auto [u, v] = ext_gcd(b, a % b);
  return make_pair(v, u - a / b * v);
}
template<typename T> pair<T, T> crt(const vector<T> &A, const vector<T> &M) {
  int N = (int)A.size();
  T x = A[0], m = M[0];
  for(int i = 1; i < N; ++i) {
    T x2 = A[i], m2 = M[i];
    T g = gcd(m, m2);
    if(x % g != x2 % g) return make_pair(-1, -1);
    auto [p, q] = ext_gcd(m / g, m2 / g);
    T mod = m / g * m2;
    x = (x * (m2 / g) * q % mod + x2 * (m / g) * p % mod) % mod;
    if(x < 0) x += mod;
    m = mod;
  }
  return make_pair(x, m);
}
```

4.3 Euler Phi

```cpp
template <typename T>
struct EulerPhi {
  int N;
  bool isBig;
  vector<T> phi, primes;
  EulerPhi(int _N):N(_N) {
    if(N <= 5000000) {
      isBig = false;
      phi.resize(N + 1); iota(phi.begin(), phi.end(), 0);
      phi[0] = 0;
      for(int i = 2; i <= N; ++i) {
        if(phi[i] != i) continue;
        for(int j = i; j <= N; j += i) phi[j] = phi[j] / i * (i - 1);
      }
    }
    else {
      isBig = true;
      T sq = (T)sqrtl(N);
      vector<int> chk(sq + 1);
      for(T i = 2; i * i <= N; ++i) {
        if(chk[i]) continue;
        primes.push_back(i);
        for(T j = i + i; j * j <= N; j += i) chk[j] = 1;
      }
    }
  }
  T getPhi(T N) {
    if(N == 1) return 1;
    if(!isBig) return phi[N];
    T res = 1;
    for(T p: primes) {
      T x = 1;
      while(N % p == 0) x *= p, N /= p;
      res *= x - x / p;
    }
    if(N != 1) res *= N - 1;
    return res;
  }
};
```

4.4 Fft

```cpp
using ll = long long;
using cpx = complex<double>;
void FFT(vector<cpx> &a, bool inv = false) {
  int N = (int)a.size();
  vector<cpx> root(N / 2);
  for(int i = 1, j = 0; i < N; ++i) {
    int bit = N >> 1;
    while(j >= bit) j -= bit, bit >>= 1;
    j += bit;
    if(i < j) swap(a[i], a[j]);
  }
  double ang = 2 * acos(-1) / N * (inv ? -1 : 1);
  for(int i = 0; i < N / 2; ++i) root[i] = cpx(cos(ang * i), sin(ang * i));
  /*
  XOR convolution: set roots[:] = 1.
  OR convolution: set roots[:] = 1 and do following
  if(!inv) a[j + k] = u + v, a[j + k + i / 2] = u;
  else a[j + k] = v, a[j + k + i / 2] = u - v;
  */
  for(int i = 2; i <= N; i <<= 1) {
    int step = N / i;
    for(int j = 0; j < N; j += i) {
      for(int k = 0; k < i / 2; ++k) {
        cpx u = a[j | k], v = a[j | k | i >> 1] * root[step * k];
        a[j | k] = u + v; a[j | k | i >> 1] = u - v;
      }
    }
  }
  if(inv) for(int i = 0; i < N; ++i) a[i] /= N;
}
vector<ll> multiply(const vector<ll> &va, const vector<ll> &vb) {
  vector<cpx> a(va.begin(), va.end()), b(vb.begin(), vb.end());
  int N = 2;
  while(N < a.size() + b.size()) N <<= 1;
  a.resize(N); b.resize(N);
  FFT(a); FFT(b);
  for(int i = 0; i < N; ++i) a[i] *= b[i];
  FFT(a, true);
  vector<ll> res(N);
  for(int i = 0; i < N; ++i) res[i] = llround(a[i].real());
  ret
```

4.5 Miller Rabin

```cpp
// Don't need power template
using ll = long long;
ll Mul(ll x, ll y, ll MOD) {
  ll ret = x * y - MOD * (unsigned long long)(1.L / MOD * x * y);
  return ret + MOD * (ret < 0) - MOD * (ret >= (ll)MOD);
}
template<typename T> T power(T a, T b, T mod) {
  T ret = 1;
  while(b) {
    if(b & 1) ret = Mul(ret, a, mod);
    a = Mul(a, a, mod);
    b >>= 1;
  }
  return ret;
}
bool MillerRabin(ll a, ll b) {
  if(a % b == 0) return false;
  for(ll d = a - 1; ; d >>= 1) {
    ll cur = power<ll>(b, d, a);
    if(d & 1) return cur != 1 && cur != a - 1;
    if(cur == a - 1) return false;
  }
  return true;
}
bool isPrime(ll x) {
  if(x == 2 || x == 3 || x == 5 || x == 7) return true;
  if(x % 2 == 0 || x % 3 == 0 || x % 5 == 0 || x % 7 == 0) return false;
  if(x < 121) return x > 1;
  if(x < (1ULL << 32)) {
    for(const ll &y: {2, 7, 61}) {
      if(x == y) return true;
      if(x > y && MillerRabin(x, y)) return false;
    }
  }
  else {
    for (const ll &y : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
      if(x == y) return true;
      if(x > y && MillerRabin(x, y)) return false;
    }
  }
  return true;
}
```

4.6 Mobius Inversion

```cpp
vector<int> MobiusInversion(int N) {
  vector<int> mu(N + 1);
  mu[1] = 1;
  for(int i = 1; i <= N; ++i) {
    for(int j = i + i; j <= N; j += i) mu[j] -= mu[i];
  }
  return mu;
}
```

4.7 Ntt

```cpp
// Need power template
using ll = long long;
// (MOD) 104,857,601   =  25 * 2^22 + 1, w = 3
// (MOD) 998,244,353   = 119 * 2^23 + 1, w = 3
// (MOD) 2,281,701,377 =  17 * 2^27 + 1, w = 3
// (MOD) 2,483,027,969 =  37 * 2^26 + 1, w = 3
// (MOD) 2,113,929,217 =  63 * 2^25 + 1, w = 5
// (MOD) 1,092,616,193 = 521 * 2^21 + 1, w = 3
template<ll W, ll MOD> void NTT(vector<ll> &V, bool inv=false) {
  int N = (int)V.size();
  vector<ll> root(N >> 1);
  for(int i = 1, j = 0; i < N; ++i) {
    int bit = N >> 1;
    while(j >= bit) j -= bit, bit >>= 1;
    j += bit;
    if(i < j) swap(V[i], V[j]);
  }
  ll ang = power<ll>(W, (MOD - 1) / N, MOD);
  if(inv) ang = power<ll>(ang, MOD - 2, MOD);
  root[0] = 1;
  for(int i = 1; i * 2 < N; ++i) root[i] = root[i - 1] * ang % MOD;
  for(int i = 2; i <= N; i <<= 1) {
    int step = N / i;
    for(int j = 0; j < N; j += i) {
      for(int k = 0; k * 2 < i; ++k) {
        ll u = V[j | k], v = V[j | k | i >> 1] * root[step * k] % MOD;
        V[j | k] = (u + v) % MOD;
        V[j | k | i >> 1] = ((u - v) % MOD + MOD) % MOD;
      }
    }
  }
  if(inv) {
    ll t = power<ll>(N, MOD - 2, MOD);
    for(int i = 0; i < N; ++i) V[i] = V[i] * t % MOD;
  }
}
template<ll W, ll MOD> vector<ll> multiply(const vector<ll> &va, const vector<ll> &vb) {
  vector<ll> a(va.begin(), va.end()), b(vb.begin(), vb.end());
  int N = 2;
  while(N < a.size() + b.size()) N <<= 1;
  a.resize(N); b.resize(N);
  NTT<W, MOD>(a); NTT<W, MOD>(b);
  for(int i = 0; i < N; ++i) a[i] *= b[i];
  NTT<W, MOD>(a, true);
  return a;
}
```

4.8 Pollard Rho

```cpp
// 5615
// Need Random Template, Miller_rabin Template
Random Rand;
void _factorize(ll N, vector<ll> &V) {
  if(N == 1) return;
  if(~N & 1) {
    V.push_back(2);
    _factorize(N >> 1, V);
    return;
```

```
  }
  if(isPrime(N)) {
    V.push_back(N);
    return;
  }
  ll a, b, c, g = N;
  auto F = [&](ll x) -> ll { return (c + Mul(x, x, N)) % N; };
  do {
    if(g == N) {
      a = b = Rand.randint(0LL, N - 3) + 2;
      c = Rand.randint(0LL, 19LL) + 1;
    }
    a = F(a);
    b = F(F(b));
    g = gcd(abs(a - b), N);
  } while(g == 1);
  _factorize(g, V); _factorize(N / g, V);
}
vector<ll> factorize(ll N) {
  vector<ll> res;
  _factorize(N, res);
  sort(res.begin(), res.end());
  return res;
}
```

4.9 Power

```
template <typename T>
T power(T a, T b, T mod) {
  T ret = 1;
  while(b) {
    if(b & 1) ret = ret * a % mod;
    a = a * a % mod;
    b >>= 1;
  }
  return ret;
}
```

5 String

5.1 Aho Corasick

```
struct AhoCorasick {
  struct Trie {
    Trie *nxt[26];
    Trie *fail;
    bool output;

    Trie() {
      for(int i=0;i<26;++i) nxt[i]=nullptr;
      fail=nullptr;
      output=false;
    }
    ~Trie() {
      for(int i=0;i<26;++i) if(nxt[i]) delete nxt[i];
    }
  } *root;
  AhoCorasick() { root = new Trie(); }
  void insert(const string &S) {
    Trie *cur = root;
    int N = (int)S.size();
    for(int i = 0; i < N; ++i) {
      int nxt = S[i] - 'a';
      if(cur->nxt[nxt] == nullptr) cur->nxt[nxt] = new Trie();
      cur = cur->nxt[nxt];
    }
    cur->output=true;
  }
  void build() {
    queue<Trie*> Q;
    root->fail = root;
    Q.push(root);
    while(!Q.empty()) {
      Trie* cur = Q.front(); Q.pop();

      for(int i = 0; i < 26; ++i) {
        Trie *next = cur->nxt[i];
        if(next == nullptr) continue;
        if(cur == root) next->fail = root;
        else {
          Trie *dst = cur->fail;
          while(dst != root && dst->nxt[i] == nullptr) dst = dst->fail;
          if(dst->nxt[i]) dst = dst->nxt[i];
          next->fail = dst;
        }
        if(next->fail->output) next->output = true;
        Q.push(next);
      }
    }
  }
  bool find(const string &S) {
    Trie *cur = root;
    int N = (int)S.size();
    for(int i = 0; i < N; ++i) {
      int nxt = S[i] - 'a';
      while(cur != root && cur->nxt[nxt] == nullptr) cur = cur->fail;
      if(cur->nxt[nxt]) cur = cur->nxt[nxt];
      if(cur->output) return true;
    }
    return false;
  }
};
```

5.2 Hash

```
// 31, 998244353
template <long long C, long long HASH_MOD>
struct Hashing {
  vector<long long> H, B;
  template<typename T> void build(const T& S) {
    H.resize(S.size() + 1);
    B.resize(S.size() + 1); B[0] = 1;
    for(int i = 1; i <= (int)S.size(); ++i) H[i] = (H[i - 1] * C + S[i - 1]) % HASH_MOD;
    for(int i = 1; i <= (int)S.size(); ++i) B[i] = B[i - 1] * C % HASH_MOD;
  }
  long long get(int s, int e) {
```

```
    long long ret = (H[e] - H[s - 1] * B[e - s + 1]) % HASH_MOD;
    if(ret < 0) ret += HASH_MOD;
    return ret;
  }
  void chk_setting() { assert(gcd(C, HASH_MOD) == 1); }
};
```

5.3 Kmp

```
template <typename T>
struct KMP {
  vector<int> fail;
  vector<int> failure(const T& Q) {
    fail.resize((int)Q.size() + 1);
    for(int i = 1, j = 0; i < (int)Q.size(); ++i) {
      while(j > 0 && Q[i] != Q[j]) j = fail[j - 1];
      if(Q[i] == Q[j]) fail[i] = ++j;
    }
    return fail;
  }
  vector<int> kmp(const T& P, const T& Q) {
    if(fail.size() == 0) failure(Q);
    vector<int> res;
    for(int i = 0, j = 0; i < (int)P.size(); ++i) {
      while(j > 0 && P[i] != Q[j]) j = fail[j - 1];
      if(P[i] == Q[j]) {
        if(j + 1 == (int)Q.size()) res.push_back(i - (int)Q.size() + 1), j = fail[j];
        else ++j;
      }
    }
    return res;
  }
};
```

5.4 Manacher

```
struct Manacher {
  vector<int> P;
  Manacher(string S) {
    string T = "$";
    for(char ch: S) T += ch, T += '$';
    int N = (int)T.size();
    P.resize(N);
    for(int i = 0, r = 0, c = 0; i < N; ++i) {
      if(2 * c >= i) P[i] = max(0, min(P[2 * c - i], r - i));
      while(0 <= i - P[i] - 1 && i + P[i] + 1 < N && T[i - P[i] - 1] == T[i + P[i] + 1]) ++
      P[i];
      if(r < i + P[i]) r = i + P[i], c = i;
    }
  }
  int& operator[](int idx) { return P[idx]; }
};
```

5.5 Suffix Array And Lcp

```
// O(NlogN)
struct SuffixArray {
  vector<int> SA, LCP;
  int N;
  SuffixArray() { }
  SuffixArray(int N) {
    SA  = vector<int>(N);
    LCP = vector<int>(N);
  }
  SuffixArray(string S, int M = 26) {
    N   = S.size();
    S   = " " + S;
    SA  = vector<int>(N + 5);
    LCP = vector<int>(N + 5);

    // SuffixArray
    vector<int> cnt(max(M, N) + 1, 0), x(N + 1, 0), y(N + 1, 0);
    int i, j, k = 0;
    for (i = 1; i <= N; i++) cnt[x[i] = S[i] - 'a' + 1]++;
    for (i = 1; i <= M; i++) cnt[i] += cnt[i - 1];
    for (i = N; i > 0; i--) SA[cnt[x[i]]--] = i;
    for (int l = 1, p = 1; p < N; l <<= 1, M = p) {
      for (p = 0, i = N - l; ++i <= N;) y[++p] = i;
      for (i = 1; i <= N; i++) if (SA[i] > l) y[++p] = SA[i] - l;
      for (i = 0; i <= M; i++) cnt[i] = 0;
      for (i = 1; i <= N; i++) cnt[x[y[i]]]++;
      for (i = 1; i <= M; i++) cnt[i] += cnt[i - 1];
      for (i = N; i > 0; i--) SA[cnt[x[y[i]]]--] = y[i];
      swap(x, y); p = 1; x[SA[1]] = 1;
      for (i = 1; i < N; i++)
        x[SA[i + 1]] = SA[i] + l <= N && SA[i + 1] + l <= N && y[SA[i]] == y[SA[i + 1]] &&
        y[SA[i] + l] == y[SA[i + 1] + l] ? p : ++p;
    }

    // LCP
    vector <int> rank(N + 1, 0);
    for (i = 1; i <= N; i++) rank[SA[i]] = i;
    for (i = 1; i <= N; LCP[rank[i++]] = k) for (k ? k-- : 0, j = SA[rank[i] - 1]; S[i + k]
    == S[j + k]; k++);
  }
};
```

5.6 Z

```
template <typename T>
vector<int> Z(const T &V) {
  int N = (int)V.size();
  vector<int> ret(N); ret[0] = N;
  for(int i = 1, l = 0, r = 0; i < N; ++i) {
    if(i < r) ret[i] = min(r - i - 1, ret[i - l]);
    while(i + ret[i] < N && V[i + ret[i]] == V[ret[i]]) ++ ret[i];
    if(i + ret[i] > r) r = i + ret[i], l = i;
  }
  return ret;
}
```

6 Geometry

6.1 Bulldozer Trick

```
// Not Tested, 9484 source code
#include <bits/stdc++.h>
```

```cpp
using namespace std;
typedef long long ll;

struct Point {
  ll x, y;
  bool operator < (const Point &o) const { return tie(x, y) <  tie(o.x, o.y); }
  bool operator ==(const Point &o) const { return tie(x, y) == tie(o.x, o.y); }
};

struct Line {
  ll i, j, dx, dy;
  Line(int _i, int _j, const Point &pi, const Point &pj) {
    i = _i; j = _j;
    dx = pj.x - pi.x;
    dy = pj.y - pi.y;
  }
  bool operator< (const Line &l) const {
    ll left  = dy * l.dx;
    ll right = l.dy * dx;
    return tie(left, i, j) < tie(right, l.i, l.j);
  }
  bool operator==(const Line &l) const {
    return dy * l.dx == l.dy * dx;
  }
};

ll area(const Point &a, const Point &b, const Point &c) {
  return abs((b.x - a.x) * (c.y - b.y) - (c.x - b.x) * (b.y - a.y));
}

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

  while(true) {
    int N; cin >> N;
    if(N == 0) break;
    vector<Point> points(N + 1);
    for(int i = 1; i <= N; ++i) cin >> points[i].x >> points[i].y;
    sort(points.begin() + 1, points.end());
    vector<int> pos(N + 1);
    for(int i = 1; i <= N; ++i) pos[i] = i;

    vector<Line> V;
    for(int i = 1; i <= N; ++i) {
      for(int j = i + 1; j <= N; ++j) {
        V.emplace_back(i, j, points[i], points[j]);
      }
    }
    sort(V.begin(), V.end());

    ll mn = LLONG_MAX / 3, mx = LLONG_MIN / 3;
    for(int i = 0, j = 0; i < (int)V.size(); i = j) {
      while(j < (int)V.size() && V[i] == V[j]) ++j;
      for(int k = i; k < j; ++k) {
        int u = V[k].i, v = V[k].j;
        swap(pos[u], pos[v]); swap(points[pos[u]], points[pos[v]]);
        if(pos[u] > pos[v]) swap(u, v);
        if(pos[u] > 1) {
          mn = min(mn, area(points[pos[u]], points[pos[v]], points[pos[u] - 1]));
          mx = max(mx, area(points[pos[u]], points[pos[v]], points[1]));
        }
        if(pos[v] < N) {
          mn = min(mn, area(points[pos[u]], points[pos[v]], points[pos[v] + 1]));
          mx = max(mx, area(points[pos[u]], points[pos[v]], points[N]));
        }
      }
    }
    cout << mn / 2 << '.' << mn % 2 * 5 << ' ';
    cout << mx / 2 << '.' << mx % 2 * 5 << '\n';
  }

  return 0;
}
```

6.2 Ccw

```cpp
template <typename T>
struct Point {
  T x, y;
  Point() : Point(0, 0) {}
  Point(T _x, T _y):x(_x),y(_y) { }
  Point operator+(Point p) { return Point(x+p.x,y+p.y); }
  Point operator-(Point p) { return Point(x-p.x,y-p.y); }
  T operator*(Point p) { return x*p.y-y*p.x; }
  bool operator==(Point p) const { return x == p.x && y == p.y; }
  bool operator<(Point p) const { return x == p.x ? y < p.y : x < p.x; }
  template<typename OT> void operator=(Point<OT> p) { *this=Point(p.x,p.y); }
  void t() { swap(x, y); }
};
template<typename T> inline istream& operator>>(istream &in, Point<T> &o) { in >> o.x >>
o.y; return in; }
template<typename T> inline ostream& operator<<(ostream &out, Point<T> &o) { out << o.x <<
' ' << o.y; return out; }
// -1: 반시계, 0: 평행, 1: 시계
template<typename T> int ccw(Point<T> a, Point<T> b, Point<T> c) {
  T x = a * b + b * c + c * a;
  return (x > 0) - (x < 0);
}
template<typename T> T dist(Point<T> a, Point<T> b) {
  T x = (a.x - b.x), y = (a.y - b.y);
  return x * x + y * y;
}
template<typename T> struct Line {
  Point<T> p1, p2;
  Line():Line(0, 0) {}
  Line(T a, T b):Line(Point<T>(0, 0), Point<T>(a, b)) { }
  Line(Point<T> a, Point<T> b):p1(a),p2(b) {
    if(p1.x > p2.x) swap(p1, p2);
    else if(p1.x == p2.x && p1.y > p2.y) swap(p1, p2);
  }
  T dx() { return p1.x - p2.x; }
  T dy() { return p1.y - p2.y; }
  T ccw() { return p1 * p2; }
```

```
  void t() { p1.t(); p2.t(); }
};
// 0: 교점 0개, 1: 교점 1개 (끝점 O), 2: 교점 1개 (끝점 X), 3: 교점 ∞개
// 4: 평행 교점 1개, 5: 평행 교점 ∞개
template<typename T> int intersect(Line<T> l1, Line<T> l2) {
  int ca = ccw(l1.p1, l1.p2, l2.p1), cb = ccw(l1.p1, l1.p2, l2.p2);
  int cc = ccw(l2.p1, l2.p2, l1.p1), cd = ccw(l2.p1, l2.p2, l1.p2);
  if(ca == 0 && cb == 0 && cc == 0 && cd == 0) {
    if(l1.p1.x == l1.p2.x && l2.p1.x == l2.p2.x && l1.p2.x == l2.p1.x) l1.t(), l2.t();
    int A = l1.p1.x, B = l1.p2.x, C = l2.p1.x, D = l2.p2.x;
    if(A > D || B < C) return 0;
    if(A == D || B == C) return 4;
    return 5;
  }
  if(ca * cb <= 0 && cc * cd <= 0) return (!ca || !cb || !cc || !cd) ? 1 : 2;
  return 0;
}
template<typename T, typename AT> pair<int, Point<AT>> intersection_point(Line<T> l1,
Line<T> l2) {
  int chk = intersect(l1, l2);
  if(chk == 0 || chk == 3) return make_pair(chk, Point<AT>());
  if(chk == 1 || chk == 4) {
    Point<AT> ans;
    if(l1.p1 == l2.p1 || l1.p1 == l2.p2) ans = l1.p1;
    else if(l1.p2 == l2.p1 || l1.p2 == l2.p2) ans = l1.p2;
    else if(ccw(l1.p1, l1.p2, l2.p1) == 0) ans = l2.p1;
    else if(ccw(l1.p1, l1.p2, l2.p2) == 0) ans = l2.p2;
    else if(ccw(l2.p1, l2.p2, l1.p1) == 0) ans = l1.p1;
    else if(ccw(l2.p1, l2.p2, l1.p2) == 0) ans = l1.p2;
    return make_pair(1, ans);
  }
  T a = l1.ccw() * l2.dx() - l1.dx() * l2.ccw();
  T b = l1.ccw() * l2.dy() - l1.dy() * l2.ccw();
  T d = l1.dx() * l2.dy() - l1.dy() * l2.dx();
  return make_pair(chk, Point<AT>(1. * a / d, 1. * b / d));
}
template<typename T> bool is_inner(const vector<Point<T>> &V, Point<T> p, bool
on_line=false) {
  int cnt = 0;
  Point<T> inf(INT_MAX, p.y + 1);
  int N = (int)V.size();
  for(int i = 0, j = 1; i < N; ++i, j = (j + 1) % N) {
    if(V[i] == p) return true;
    if(on_line && intersect(Line<T>(V[i], V[j]), Line<T>(p, p)) != 0) return true;
    cnt += intersect(Line<T>(V[i], V[j]), Line<T>(p, inf)) != 0;
  }
  return cnt & 1;
}
// Box와 직선 또는 선분이 교차하는지 (Box 안에 선분이 존재하는걸 포함)
```

6.3 Convex Hull

```
// 더 추가해야함.
template <typename T>
vector<Point<T>> ConvexHull(vector<Point<T>> V) {
  swap(V[0], *min_element(V.begin(), V.end()));
  sort(V.begin() + 1, V.end(), [&](Point<T> a, Point<T> b) {
    int w = ccw(V[0], a, b);
    return w ? w > 0 : dist(V[0], a) < dist(V[0], b);
  });
  int idx = (int)V.size() - 1;
  while(idx > 1 && ccw(V[0], V[idx], V[idx - 1]) == 0) --idx;
  reverse(V.begin() + idx, V.end());
  vector<int> st;
  for(int i = 0; i < (int)V.size(); ++i) {
    // line ok < or <=
    while(st.size() > 1 && ccw(V[st[st.size() - 2]], V[st.back()], V[i]) < 0)
    st.pop_back();
    st.push_back(i);
  }
  vector<Point<T>> res;
  for(int x: st) res.push_back(V[x]);
  return res;
}
template<typename T> pair<Point<T>, Point<T>> get_far_two_point(vector<Point<T>> V) {
  int N = (int)V.size();
  T d = 0;
  pair<Point<T>, Point<T>> res;
  auto upd = [&](Point<T> a, Point<T> b) {
    T cur = dist(a, b);
    if(d < cur) d = cur, res = make_pair(a, b);
  };
  for(int i = 0, r = 0; i < N; ++i) {
    while(r + 1 < N && ccw(Point<T>(), V[(i + 1) % N] - V[i], V[(r + 1) % N] - V[r]) >= 0)
    upd(V[i], V[r++]);
    upd(V[i], V[r]);
  }
  return res;
}
template<typename T> bool IsPointInConvex(const vector<Point<T>> &V, Point<T> p) {
  if(V[0].x >= p.x) return false;
  int N = (int)V.size();

  int l = 0, r = N - 1;
  while(l <= r) {
    int mid = (l + r) / 2;
    if(ccw(V[0], V[mid], p) >= 0) l = mid + 1;
    else r = mid - 1;
  }
  l = (l + N) % N; r = (r + N) % N;
  if(ccw(V[0], V[r], p) == 0) return p < V[r];
  int nxt = (r + 1) % N;
  return ccw(V[r], V[nxt], p) > 0;
}
```

6.4 Enclosing Circle

```
// D: dimension, must be |P| % D == 0
template <typename T>
struct EnclosingCircle {
  int D;
  vector<T> P;
  EnclosingCircle(const vector<T> &_P, int _D):D(_D),P(_P) {}
  vector<T> train(T precision=-1, int epoch=100000, T learning_rate=0.1, T
  weight_decay=0.999) {
```

```
    vector<T> res(D);
    int N = (int)P.size() / D;
    for(int j = 0; j < D; ++j) {
      for(int i = 0; i < N; ++i) res[j] += P[i * D + j];
      res[j] /= N;
    }
    while(~--epoch) {
      T mx = 0; int mx_idx = 0;
      for(int i = 0; i < N; ++i) {
        T cur = cal(res, i);
        if(cur > mx) {
          mx = cur;
          mx_idx = i;
        }
      }
      T mx_dif = 0;
      for(int i = 0; i < D; ++i) {
        T dif = (P[mx_idx * D + i] - res[i]) * learning_rate;
        mx_dif = max(mx_dif, fabs(dif));
        res[i] += dif;
      }
      if(precision > 0 && mx_dif < precision) break;
      learning_rate *= weight_decay;
    }
    if(precision > 0) for(int i = 0; i < D; ++i) if(res[i] < 0 && res[i] > -precision)
    res[i] = -res[i];
    return res;
  }
  T cal(const vector<T> &x, int idx) {
    T ret = 0;
    for(int i = 0; i < D; ++i) {
      ret += (x[i] - P[idx * D + i]) * (x[i] - P[idx * D + i]);
    }
    return ret;
  }
};
```

6.5 Gradient Descent

```
template <typename T>
struct GradientDescent {
  const long double eps = 0.0000001;
  int D;
  vector<T> P;
  GradientDescent(const vector<T> &_P, int _D):D(_D),P(_P) {}
  vector<T> train(T precision=-1, int epoch=100000, T learning_rate=1000000, T
  weight_decay=0.999) {
    vector<T> res(D);
    int N = (int)P.size() / D;
    for(int j = 0; j < D; ++j) {
      for(int i = 0; i < N; ++i) res[j] += P[i * D + j];
      res[j] /= N;
    }
    while(~--epoch) {
      vector<T> g = gradient(res);
      if(g.empty()) break;
      T mx_dif = 0;
      for(int i = 0; i < D; ++i) {
        res[i] -= learning_rate * g[i];
        mx_dif = max(mx_dif, fabs(g[i]));
      }
      if(precision > 0 && mx_dif < precision) break;
      learning_rate *= weight_decay;
    }
    if(precision > 0) for(int i = 0; i < D; ++i) if(res[i] < 0 && res[i] > -precision)
    res[i] = -res[i];
    return res;
  }
  // Customizing
  vector<T> f(const vector<T> &V) {
    int N = (int)P.size() / D;
    vector<T> ret(N);
    for(int i = 0; i < N; ++i) {
      for(int j = 0; j < D; ++j) {
        ret[i] += (P[i * D + j] - V[j] + eps) * (P[i * D + j] - V[j] + eps);
      }
      ret[i] = sqrtl(ret[i]);
    }
    return ret;
  }
  vector<T> gradient(const vector<T> &V) {
    vector<T> W = f(V);
    for(int i = 0; i < D; ++i) if(W[i] < 0.0000001) return {};
    int N = (int)P.size() / D;
    vector<T> res(D);
    for(int j = 0; j < D; ++j) {
      for(int i = 0; i < N; ++i) {
        res[j] += (V[j] - P[i * D + j]) / W[i];
      }
    }
    return res;
  }
};
```

6.6 Nearest Two Point

```
// Need Point(CCW) template
template <typename T>
T nearest_two_points(vector<Point<T>> P) {
  int N = (int)P.size();
  const T MIN = numeric_limits<T>::min();
  const T MAX = numeric_limits<T>::max();
  sort(P.begin(), P.end(), [&](Point<T> a, Point<T> b) {
    a.t(); b.t();
    return a < b;
  });
  set<Point<T>> st({P[0], P[1]});
  T ret = dist(P[0], P[1]);
  for(int i = 2, j = 0; i < N; ++i) {
    while(j < i && (P[i].y - P[j].y) * (P[i].y - P[j].y) >= ret) st.erase(P[j ++]);
    T d = sqrtl(ret) + 2;
    auto it1 = st.lower_bound(Point<T>(P[i].x - d, MIN));
    auto it2 = st.upper_bound(Point<T>(P[i].x + d, MAX));
    while(it1 != it2) ret = min(ret, dist(P[i], *it1 ++));
    st.insert(P[i]);
```

```
  }
  return ret;
}
```

< 10^k	number	divisors	2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59	61	67
1	6	4	1	1																	
2	60	12	2	1	1																
3	840	32	3	1	1	1															
4	7560	64	3	3	1	1															
5	83160	128	3	3	1	1	1														
6	720720	240	4	2	1	1	1	1													
7	8648640	448	6	3	1	1	1	1													
8	73513440	768	5	3	1	1	1	1	1												
9	735134400	1344	6	3	2	1	1	1	1												
10	6983776800	2304	5	3	2	1	1	1	1	1											
11	97772875200	4032	6	3	2	2	1	1	1	1											
12	963761198400	6720	6	4	2	1	1	1	1	1	1										
13	9316358251200	10752	6	3	2	1	1	1	1	1	1	1									
14	97821761637600	17280	5	4	2	2	1	1	1	1	1	1									
15	866421317361600	26880	6	4	2	1	1	1	1	1	1	1	1								
16	8086598962041600	41472	8	3	2	2	1	1	1	1	1	1	1								
17	74801040398884800	64512	6	3	2	2	1	1	1	1	1	1	1	1							
18	897612484786617600	103680	8	4	2	2	1	1	1	1	1	1	1	1							

< 10^k	prime	# of prime	< 10^k	prime
1	7	4	10	9999999967
2	97	25	11	99999999977
3	997	168	12	999999999989
4	9973	1229	13	9999999999971
5	99991	9592	14	99999999999973
6	999983	78498	15	999999999999989
7	9999991	664579	16	9999999999999937
8	99999989	5761455	17	99999999999999997
9	999999937	50847534	18	999999999999999989

- Burnside's Lemma
  - 수식
  G=(X,A): 집합X와 액션A로 정의되는 군G에 대해, $|A||X/A| = sum(|\text{Fixed points of a}|, \text{for all a in A})$
  X/A 는 Action으로 서로 변형가능한 X의 원소들을 동치로 묶었을때 동치류(파티션) 집합
  - 풀어쓰기
  orbit: 그룹에 대해 두 원소 a,b와 액션f에 대해 f(a)=b인거에 간선연결한 컴포넌트(연결집합)
  orbit개수 = sum(각 액션 g에 대해 f(x)=x인 x(고정점)개수)/액션개수
  - 자유도 치트시트
  회전 n개: 회전i의 고정점 자유도=gcd(n,i)
  임의뒤집기 n=홀수: n개 원소중심축(자유도 (n+1)/2)
  임의뒤집기 n=짝수: n/2개 원소중심축(자유도 n/2+1) + n/2개 원소안지나는축(자유도 n/2)
- 알고리즘 게임
  - Nim Game의 해법(마지막에 가져가는 사람이 승) : XOR = 0이면 후공 승, 0 아니면 선공 승
  - Subtraction Game : 한 번에 k 개까지의 돌만 가져갈 수 있는 경우, 각 더미의 돌의 개수를 k + 1로 나눈 나머지를 XOR 합하여 판단한다.
  - Index-k Nim : 한 번에 최대 k개의 더미를 골라 각각의 더미에서 아무렇게나 돌을 제거할 수 있을 때, 각 binary digit에 대하여 합을 k + 1로 나눈 나머지를 계산한다. 만약 이 나머지가 모든 digit에 대하여 0이라면 두번째, 하나라도 0이 아니라면 첫번째 플레이어가 승리.
  - Misere Nim : 모든 돌 무더기가 1이면 N이 홀수일 때 후공 승, 그렇지 않은 경우 XOR 합 0이면 후공 승
- Pick's Theorem
  격자점으로 구성된 simple polygon이 주어짐. I 는 polygon 내부의 격자점 수, B 는 polygon 선분 위 격자점 수, A는 polygon의 넓이라고 할 때, 다음과 같은 식이 성립한다. $A = I + B/2 - 1$
- 홀의 결혼 정리 : 이분그래프(L-R)에서, 모든 L을 매칭하는 필요충분 조건 = L에서 임의의 부분집합 S를 골랐을 때, 반드시 (S의 크기) <= (S와 연결되어있는 모든 R의 크기)이다.
- Simpson 공식 (적분) : Simpson 공식, $S_n(f) = \frac{h}{3}[f(x_0) + f(x_n) + 4\sum f(x_{2i+1}) + 2\sum f(x_{2i})]$
  - $M = \max|f^4(x)|$이라고 하면 오차 범위는 최대 $E_n \le \frac{M(b-a)}{180}h^4$
- 브라마굽타 : 원에 내접하는 사각형의 각 선분의 길이가 a, b, c, d일 때
  사각형의 넓이 $S = \sqrt{(s-a)(s-b)(s-c)(s-d)}$, $s = (a+b+c+d)/2$
- 브레치나이더 : 임의의 사각형의 각 변의 길이를 a, b, c, d라고 하고, 마주보는 두 각의 합을 2로 나눈 값을 θ라 하면, $S = \sqrt{(s-a)(s-b)(s-c)(s-d) - abcd \times cos^2\theta}$
- 페르마 포인트 : 삼각형의 세 꼭짓점으로부터 거리의 합이 최소가 되는 점
  $2\pi/3$ 보다 큰 각이 있으면 그 점이 페르마 포인트, 그렇지 않으면 각 변마다 정삼각형 그린 다음, 정삼각형의 끝점에서 반대쪽 삼각형의 꼭짓점으로 연결한 선분의 교점
  $2\pi/3$ 보다 큰 각이 없으면 거리의 합은 $\sqrt{(a^2+b^2+c^2+4\sqrt{3}S)/2}$, S는 넓이
- 오일러 정리: 서로소인 두 정수 a, n에 대해 $a^{\phi(n)} \equiv 1 \pmod n$
  모든 정수에 대해 $a^n \equiv a^{n-\phi(n)} \pmod n$
  $m \ge log_2 n$이면 $a^m \equiv a^{m\%\phi(n)+\phi(n)} \pmod n$
- $g^0 + g^1 + g^2 + \cdots g^{p-2} \equiv -1 \pmod p$ iff $g = 1$, otherwise 0.
- if $n \equiv 0 \pmod 2$, then $1^n + 2^n + \cdots + (n-1)^n \equiv 0 \pmod n$
- Eulerian numbers
  Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \ge j$, k j:s s.t. $\pi(j) > j$.
  $E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$
  $E(n,0) = E(n,n-1) = 1$
  $E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$
- Pythagorean triple: $a^2 + b^2 = c^2$이고 서로소인 (a,b,c) 생성
  $(a,b,c) = (st, \frac{s^2-t^2}{2}, \frac{s^2+t^2}{2})$, $gcd(s,t) = 1, s > t$