Nea1's Code Library

Nea1

ORZ

He is Nea1

Contents

Intro

Main template

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

#define FOR(x,n) for(int x=0;x<n;x++)
#define forn(i, n) for (int i = 0; i < int(n); i++)
#define all(v) v.begin(),v.end()
using ll = long long;
using ld = long double;
using pii = pair<int, int>;
const char nl = '\n';

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cout << fixed << setprecision(20);
  // mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
}
```

Fast IO

```cpp
namespace io {
constexpr int SIZE = 1 << 16;
char buf[SIZE], *head, *tail;
char get_char() {
  if (head == tail) tail = (head = buf) + fread(buf, 1, SIZE, stdin);
  return *head++;
}
ll read() {
  ll x = 0, f = 1;
  char c = get_char();
  for (; !isdigit(c); c = get_char()) (c == '-') && (f = -1);
  for (; isdigit(c); c = get_char()) x = x * 10 + c - '0';
  return x * f;
}
string read_s() {
  string str;
  char c = get_char();
  while (c == ' ' || c == '\n' || c == '\r') c = get_char();
  while (c != ' ' && c != '\n' && c != '\r') str += c, c = get_char();
  return str;
}
void print(int x) {
  if (x > 9) print(x / 10);
  putchar(x % 10 | '0');
}
void println(int x) { print(x), putchar('\n'); }
struct Read {
  Read& operator>>(ll& x) { return x = read(), *this; }
  Read& operator>>(long double& x) { return x = stold(read_s()), *this; }
} in;
}  // namespace io
```

Pragmas (lol)

```cpp
#pragma GCC optimize(2)
#pragma GCC optimize(3)
#pragma GCC optimize("Ofast")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-ffast-math")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("unroll-loops")
```

```cpp
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-funroll-loops")
#pragma GCC optimize("-fwhole-program")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-fstrict-overflow")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-skip-blocks")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("-funsafe-loop-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4.1,sse4.2,avx,avx2,popcnt,tune=native")
```

Data Structures

Segment Tree

Recursive

- Implicit segment tree, range query + point update

```cpp
struct Node {
  int lc, rc, p;
};

struct SegTree {
  vector<Node> t = {{}};
  SegTree(int n) { t.reserve(n * 40); }
  int modify(int p, int l, int r, int x, int v) {
    int u = p;
    if (p == 0) {
      t.push_back(t[p]);
      u = (int)t.size() - 1;
    }
    if (r - l == 1) {
      t[u].p = t[p].p + v;
    } else {
      int m = (l + r) / 2;
      if (x < m) {
        t[u].lc = modify(t[p].lc, l, m, x, v); // ub before c++17
      } else {
        t[u].rc = modify(t[p].rc, m, r, x, v);
      }
      t[u].p = t[t[u].lc].p + t[t[u].rc].p;
    }
    return u;
  }
```

```
  int query(int p, int l, int r, int x, int y) {
    if (x <= l && r <= y) return t[p].p;
    int m = (l + r) / 2, res = 0;
    if (x < m) res += query(t[p].lc, l, m, x, y);
    if (y > m) res += query(t[p].rc, m, r, x, y);
    return res;
  }
};
```

- Persistent implicit, range query + point update

```
struct Node {
  int lc = 0, rc = 0, p = 0;
};

struct SegTree {
  vector<Node> t = {{}};  // init all
  SegTree() = default;
  SegTree(int n) { t.reserve(n * 20); }
  int modify(int p, int l, int r, int x, int v) {
    // p: original node, update a[x] -> v
    t.push_back(t[p]);
    int u = (int)t.size() - 1;
    if (r - l == 1) {
      t[u].p = v;
    } else {
      int m = (l + r) / 2;
      if (x < m) {
        t[u].lc = modify(t[p].lc, l, m, x, v);
        t[u].rc = t[p].rc;
      } else {
        t[u].lc = t[p].lc;
        t[u].rc = modify(t[p].rc, m, r, x, v);
      }
      t[u].p = t[t[u].lc].p + t[t[u].rc].p;
    }
    return u;
  }
  int query(int p, int l, int r, int x, int y) {
    // query sum a[x]...a[y-1] rooted at p
    // t[p] holds the info of [l, r)
    if (x <= l && r <= y) return t[p].p;
    int m = (l + r) / 2, res = 0;
    if (x < m) res += query(t[p].lc, l, m, x, y);
    if (y > m) res += query(t[p].rc, m, r, x, y);
    return res;
  }
};
```

Iterating

- Iterating, range query + point update

```
struct Node {
  ll v = 0, init = 0;
};

Node pull(const Node &a, const Node &b) {
  if (!a.init) return b;
  if (!b.init) return a;
  Node c;
  return c;
}

struct SegTree {
  ll n;
  vector<Node> t;
  SegTree(ll _n) : n(_n), t(2 * n){};
  void modify(ll p, const Node &v) {
    t[p += n] = v;
    for (p /= 2; p; p /= 2) t[p] = pull(t[p * 2], t[p * 2 + 1]);
```

```
  }
  Node query(ll l, ll r) {
    Node left, right;
    for (l += n, r += n; l < r; l /= 2, r /= 2) {
      if (l & 1) left = pull(left, t[l++]);
      if (r & 1) right = pull(t[--r], right);
    }
    return pull(left, right);
  }
};
```

- Iterating, range query + range update

```
struct SegTree {
  ll n, h = 0;
  vector<Node> t;
  SegTree(ll _n) : n(_n), h((ll)log2(n)), t(n * 2) {}
  void apply(ll x, ll v) {
    if (v == 0) {
      t[x].one = 0;
    } else {
      t[x].one = t[x].total;
    }
    t[x].lazy = v;
  }
  void build(ll l) {
    for (l = (l + n) / 2; l > 0; l /= 2) {
      if (t[l].lazy == -1) {
        t[l] = pull(t[l * 2], t[l * 2 + 1]);
      }
    }
  }
  void push(ll l) {
    l += n;
    for (ll s = h; s > 0; s--) {
      ll i = l >> s;
      if (t[i].lazy != -1) {
        apply(2 * i, t[i].lazy);
        apply(2 * i + 1, t[i].lazy);
      }
      t[i].lazy = -1;
    }
  }
  void modify(ll l, ll r, int v) {
    push(l), push(r - 1);
    ll l0 = l, r0 = r;
    for (l += n, r += n; l < r; l /= 2, r /= 2) {
      if (l & 1) apply(l++, v);
      if (r & 1) apply(--r, v);
    }
    build(l0), build(r0 - 1);
  }
  Node query(ll l, ll r) {
    push(l), push(r - 1);
    Node left, right;
    for (l += n, r += n; l < r; l /= 2, r /= 2) {
      if (l & 1) left = pull(left, t[l++]);
      if (r & 1) right = pull(t[--r], right);
    }
    return pull(left, right);
  }
};
```

- AtCoder Segment Tree (recursive structure but iterative)

```
template <class T> struct PointSegmentTree {
  int size = 1;
  vector<T> tree;
  PointSegmentTree(int n) : PointSegmentTree(vector<T>(n)) {}
  PointSegmentTree(vector<T>& arr) {
    while(size < (int)arr.size())
      size <<= 1;
```

```
    tree = vector<T>(size << 1);
    for(int i = size + arr.size() - 1; i >= 1; i--)
      if(i >= size) tree[i] = arr[i - size];
      else consume(i);
  }
  void set(int i, T val) {
    tree[i += size] = val;
    for(i >>= 1; i >= 1; i >>= 1)
      consume(i);
  }
  T get(int i) { return tree[i + size]; }
  T query(int l, int r) {
    T resl, resr;
    for(l += size, r += size + 1; l < r; l >>= 1, r >>= 1) {
      if(l & 1) resl = resl * tree[l++];
      if(r & 1) resr = tree[--r] * resr;
    }
    return resl * resr;
  }
  T query_all() { return tree[1]; }
  void consume(int i) { tree[i] = tree[i << 1] * tree[i << 1 | 1]; }
};


struct SegInfo {
  ll v;
  SegInfo() : SegInfo(0) {}
  SegInfo(ll val) : v(val) {}
  SegInfo operator*(SegInfo b) {
    return SegInfo(v + b.v);
  }
};
```

Union Find

```
vector<int> p(n);
iota(p.begin(), p.end(), 0);
function<int(int)> find = [&](int x) { return p[x] == x ? x : (p[x] = find(p[x])); };
auto merge = [&](int x, int y) { p[find(x)] = find(y); };
```

- Persistent version

```
struct Node {
  int lc, rc, p;
};

struct SegTree {
  vector<Node> t = {{0, 0, -1}};  // init all
  SegTree() = default;
  SegTree(int n) { t.reserve(n * 20); }
  int modify(int p, int l, int r, int x, int v) {
    // p: original node, update a[x] -> v
    t.push_back(t[p]);
    int u = (int)t.size() - 1;
    if (r - l == 1) {
      t[u].p = v;
    } else {
      int m = (l + r) / 2;
      if (x < m) {
        t[u].lc = modify(t[p].lc, l, m, x, v);
        t[u].rc = t[p].rc;
      } else {
        t[u].lc = t[p].lc;
        t[u].rc = modify(t[p].rc, m, r, x, v);
      }
      t[u].p = t[t[u].lc].p + t[t[u].rc].p;
    }
    return u;
  }
  int query(int p, int l, int r, int x, int y) {
    // query sum a[x]...a[y-1] rooted at p
```

```
    // t[p] holds the info of [l, r)
    if (x <= l && r <= y) return t[p].p;
    int m = (l + r) / 2, res = 0;
    if (x < m) res += query(t[p].lc, l, m, x, y);
    if (y > m) res += query(t[p].rc, m, r, x, y);
    return res;
  }
};

struct DSU {
  int n;
  SegTree seg;
  DSU(int _n) : n(_n), seg(n) {}
  int get(int p, int x) { return seg.query(p, 0, n, x, x + 1); }
  int set(int p, int x, int v) { return seg.modify(p, 0, n, x, v); }
  int find(int p, int x) {
    int parent = get(p, x);
    if (parent < 0) return x;
    return find(p, parent);
  }
  int is_same(int p, int x, int y) { return find(p, x) == find(p, y); }
  int merge(int p, int x, int y) {
    int rx = find(p, x), ry = find(p, y);
    if (rx == ry) return -1;
    int rank_x = -get(p, rx), rank_y = -get(p, ry);
    if (rank_x < rank_y) {
      p = set(p, rx, ry);
    } else if (rank_x > rank_y) {
      p = set(p, ry, rx);
    } else {
      p = set(p, ry, rx);
      p = set(p, rx, -rx - 1);
    }
    return p;
  }
};
```

Fenwick Tree

- askd version

```
template <typename T> struct FenwickTree {
  int size = 1, high_bit = 1;
  vector<T> tree;
  FenwickTree(int _size) : size(_size) {
    tree.resize(size + 1);
    while((high_bit << 1) <= size) high_bit <<= 1;
  }
  FenwickTree(vector<T>& arr) : FenwickTree(arr.size()) {
    for(int i = 0; i < size; i++) update(i, arr[i]);
  }
  int lower_bound(T x) {
    int res = 0; T cur = 0;
    for(int bit = high_bit; bit > 0; bit >>= 1) {
      if((res|bit) <= size && cur + tree[res|bit] < x) {
        res |= bit; cur += tree[res];
      }
    }
    return res;
  }
  T prefix_sum(int i) {
    T ret = 0;
    for(i++; i > 0; i -= (i & -i)) ret += tree[i];
    return ret;
  }
  T range_sum(int l, int r) { return (l > r) ? 0 : prefix_sum(r) - prefix_sum(l - 1); }
  void update(int i, T delta) { for(i++; i <= size; i += (i & -i)) tree[i] += delta; }
};
```

- Nea1 version

```cpp
template <typename T>
struct Fenwick {
  const int n;
  vector<T> a;
  Fenwick(int n) : n(n), a(n) {}
  void add(int x, T v) {
    for (int i = x + 1; i <= n; i += i & -i) {
      a[i - 1] += v;
    }
  }
  T sum(int x) {
    T ans = 0;
    for (int i = x; i > 0; i -= i & -i) {
      ans += a[i - 1];
    }
    return ans;
  }
  T rangeSum(int l, int r) { return sum(r) - sum(l); }
};
```

PBDS

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
using namespace std;
using namespace __gnu_pbds;
template<typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
template<typename T, typename X>
using ordered_map = tree<T, X, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
template<typename T, typename X>
using fast_map = cc_hash_table<T, X>;
template<typename T, typename X>
using ht = gp_hash_table<T, X>;
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

struct splitmix64 {
    size_t operator()(size_t x) const {
        static const size_t fixed = chrono::steady_clock::now().time_since_epoch().count();
        x += 0x9e3779b97f4a7c15 + fixed;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
};
```

Treap

- (No rotation version)

```cpp
struct Node {
  Node *l, *r;
  int s, sz;
  // int t = 0, a = 0, g = 0; // for lazy propagation
  ll w;

  Node(int _s) : l(nullptr), r(nullptr), s(_s), sz(1), w(rng()) {}
  void apply(int vt, int vg) {
    // for lazy propagation
    // s -= vt;
    // t += vt, a += vg, g += vg;
  }
  void push() {
    // for lazy propagation
    // if (l != nullptr) l->apply(t, g);
    // if (r != nullptr) r->apply(t, g);
    // t = g = 0;
  }
  void pull() { sz = 1 + (l ? l->sz : 0) + (r ? r->sz : 0); }
};
```

```cpp

std::pair<Node *, Node *> split(Node *t, int v) {
  if (t == nullptr) return {nullptr, nullptr};
  t->push();
  if (t->s < v) {
    auto [x, y] = split(t->r, v);
    t->r = x;
    t->pull();
    return {t, y};
  } else {
    auto [x, y] = split(t->l, v);
    t->l = y;
    t->pull();
    return {x, t};
  }
}

Node *merge(Node *p, Node *q) {
  if (p == nullptr) return q;
  if (q == nullptr) return p;
  if (p->w < q->w) swap(p, q);
  auto [x, y] = split(q, p->s + rng() % 2);
  p->push();
  p->l = merge(p->l, x);
  p->r = merge(p->r, y);
  p->pull();
  return p;
}

Node *insert(Node *t, int v) {
  auto [x, y] = split(t, v);
  return merge(merge(x, new Node(v)), y);
}

Node *erase(Node *t, int v) {
  auto [x, y] = split(t, v);
  auto [p, q] = split(y, v + 1);
  return merge(merge(x, merge(p->l, p->r)), q);
}

int get_rank(Node *&t, int v) {
  auto [x, y] = split(t, v);
  int res = (x ? x->sz : 0) + 1;
  t = merge(x, y);
  return res;
}

Node *kth(Node *t, int k) {
  k--;
  while (true) {
    int left_sz = t->l ? t->l->sz : 0;
    if (k < left_sz) {
      t = t->l;
    } else if (k == left_sz) {
      return t;
    } else {
      k -= left_sz + 1, t = t->r;
    }
  }
}

Node *get_prev(Node *&t, int v) {
  auto [x, y] = split(t, v);
  Node *res = kth(x, x->sz);
  t = merge(x, y);
  return res;
}

Node *get_next(Node *&t, int v) {
  auto [x, y] = split(t, v + 1);
  Node *res = kth(y, 1);
```

```
  t = merge(x, y);
  return res;
}
```

- USAGE

```
int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n;
  cin >> n;
  Node *t = nullptr;
  for (int op, x; n--;) {
    cin >> op >> x;
    if (op == 1) {
      t = insert(t, x);
    } else if (op == 2) {
      t = erase(t, x);
    } else if (op == 3) {
      cout << get_rank(t, x) << "\n";
    } else if (op == 4) {
      cout << kth(t, x)->s << "\n";
    } else if (op == 5) {
      cout << get_prev(t, x)->s << "\n";
    } else {
      cout << get_next(t, x)->s << "\n";
    }
  }
}
```

Implicit treap

- Split by size

```
struct Node {
  Node *l, *r;
  int s, sz;
  // int lazy = 0;
  ll w;

  Node(int _s) : l(nullptr), r(nullptr), s(_s), sz(1), w(rnd()) {}
  void apply() {
    // for lazy propagation
    // lazy ^= 1;
  }
  void push() {
    // for lazy propagation
    // if (lazy) {
    //   swap(l, r);
    //   if (l != nullptr) l->apply();
    //   if (r != nullptr) r->apply();
    //   lazy = 0;
    // }
  }
  void pull() { sz = 1 + (l ? l->sz : 0) + (r ? r->sz : 0); }
};

std::pair<Node *, Node *> split(Node *t, int v) {
  // first->sz == v
  if (t == nullptr) return {nullptr, nullptr};
  t->push();
  int left_sz = t->l ? t->l->sz : 0;
  if (left_sz < v) {
    auto [x, y] = split(t->r, v - left_sz - 1);
    t->r = x;
    t->pull();
    return {t, y};
  } else {
    auto [x, y] = split(t->l, v);
    t->l = y;
    t->pull();
    return {x, t};
```

```
  }
}

Node *merge(Node *p, Node *q) {
  if (p == nullptr) return q;
  if (q == nullptr) return p;
  if (p->w < q->w) {
    p->push();
    p->r = merge(p->r, q);
    p->pull();
    return p;
  } else {
    q->push();
    q->l = merge(p, q->l);
    q->pull();
    return q;
  }
}
```

Persistent implicit treap

```
pair<Node *, Node *> split(Node *t, int v) {
  // first->sz == v
  if (t == nullptr) return {nullptr, nullptr};
  t->push();
  int left_sz = t->l ? t->l->sz : 0;
  t = new Node(*t);
  if (left_sz < v) {
    auto [x, y] = split(t->r, v - left_sz - 1);
    t->r = x;
    t->pull();
    return {t, y};
  } else {
    auto [x, y] = split(t->l, v);
    t->l = y;
    t->pull();
    return {x, t};
  }
}

Node *merge(Node *p, Node *q) {
  if (p == nullptr) return new Node(*q);
  if (q == nullptr) return new Node(*p);
  if (p->w < q->w) {
    p = new Node(*p);
    p->push();
    p->r = merge(p->r, q);
    p->pull();
    return p;
  } else {
    q = new Node(*q);
    q->push();
    q->l = merge(p, q->l);
    q->pull();
    return q;
  }
}
```

2D Sparse Table

- Sorry that this sucks - askd

```
template <class T, class Compare = less<T>>
struct SparseTable2d {
  int n = 0, m = 0;
  T**** table;
  int* log;
  inline T choose(T x, T y) {
    return Compare()(x, y) ? x : y;
  }
```

```
  SparseTable2d(vector<vector<T>>& grid) {
    if(grid.empty() || grid[0].empty()) return;
    n = grid.size(); m = grid[0].size();
    log = new int[max(n, m) + 1];
    log[1] = 0;
    for(int i = 2; i <= max(n, m); i++)
      log[i] = log[i - 1] + ((i ^ (i - 1)) > i);
    table = new T***[n];
    for(int i = n - 1; i >= 0; i--) {
      table[i] = new T**[m];
      for(int j = m - 1; j >= 0; j--) {
        table[i][j] = new T*[log[n - i] + 1];
        for(int k = 0; k <= log[n - i]; k++) {
          table[i][j][k] = new T[log[m - j] + 1];
          if(!k) table[i][j][k][0] = grid[i][j];
          else table[i][j][k][0] = choose(table[i][j][k-1][0], table[i+(1<<(k-1))][j][k-1][0]);
          for(int l = 1; l <= log[m - j]; l++)
            table[i][j][k][l] = choose(table[i][j][k][l-1], table[i][j+(1<<(l-1))][k][l-1]);
        }
      }
    }
  }
  T query(int r1, int r2, int c1, int c2) {
    assert(r1 >= 0 && r2 < n && r1 <= r2);
    assert(c1 >= 0 && c2 < m && c1 <= c2);
    int rl = log[r2 - r1 + 1], cl = log[c2 - c1 + 1];
    T ca1 = choose(table[r1][c1][rl][cl], table[r2-(1<<rl)+1][c1][rl][cl]);
    T ca2 = choose(table[r1][c2-(1<<cl)+1][rl][cl], table[r2-(1<<rl)+1][c2-(1<<cl)+1][rl][cl]);
    return choose(ca1, ca2);
  }
};
```

- USAGE

```
vector<vector<int>> test =  {
  {1, 2, 3, 4}, {2, 3, 4, 5}, {9, 9, 9, 9}, {-1, -1, -1, -1}
};

SparseTable2d<int> st(test);                // Range min query
SparseTable2d<int,greater<int>> st2(test);  // Range max query
```