Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая работа по курсу «Дискретный анализ»

Студент: Г. А. Ермеков
Преподаватель: Н. К. Макаров
Группа: М8О-301Б
Дата:
Оценка:
Подпись:

Москва, 2025

Курсовая работа

Задача: Вам дан набор горизонтальных отрезков и набор точек. Для каждой точки определите, сколько отрезков лежит строго над ней.

Ваше решение должно работать online, то есть должно обрабатывать запросы по одному после построения необходимой структуры данных по входным данным. Чтение входных данных и запросов вместе и построение по ним общей структуры запрещено.

Формат ввода: В первой строке даны два числа n и m ($1 \le n, m \le 10^5$) — количество отрезков и точек. В следующих n строках заданы отрезки тройками l, r, h ($-10^9 \le l < r \le 10^9$, $-10^9 \le h \le 10^9$). В следующих m строках даны точки парами x, y ($-10^9 \le x, y \le 10^9$).

Формат вывода: Для каждой точки выведите количество отрезков строго над ней.

Метод решения: Персистентное дерево отрезков с сжатием координат.

1 Описание

Задача сводится к подсчету количества отрезков, покрывающих координату x и имеющих высоту $h > y$. Поскольку координаты l, r могут быть большими (до 10^9), но их количество ограничено $O(N)$, применяется **сжатие координат**. В массив уникальных координат добавляются только концы отрезков (l и $r + 1$).

Отрезки сортируются по убыванию высоты h. Мы строим персистентное дерево отрезков, обрабатывая отрезки по одному.

Каждая версия дерева соответствует состоянию, когда добавлены все отрезки с высотой $\geq H$. Поскольку нам нужно количество отрезков над точкой (строго $h > y$), для ответа на запрос (x, y) мы выбираем версию дерева, построенную для минимальной высоты, превышающей y.

При обработке запросов, так как они поступают в режиме *online* и координаты x запросов не участвуют в предварительном сжатии, для поиска соответствующего индекса в дереве используется бинарный поиск (`upper_bound`) по массиву сжатых координат. Мы находим индекс ближайшей слева координаты из сжатого массива.

2 Корректность и оценка сложности

Персистентность позволяет обращаться к истории изменений, эффективно фильтруя отрезки по высоте.

Сложность:

- Сжатие координат: $O(N \log N)$.
- Построение дерева: N обновлений. Каждое обновление создает новую версию и добавляет $O(\log N)$ узлов. Итого $O(N \log N)$.
- Обработка запросов: Для каждого запроса поиск нужной версии занимает $O(\log N)$, поиск индекса координаты — $O(\log N)$, запрос суммы в дереве — $O(\log N)$. Итого $O(M \log N)$.

Общая временная сложность: $O((N+M) \log N)$. Пространственная сложность: $O(N \log N)$ для хранения узлов персистентного дерева.

3 Исходный код

```
#include <algorithm>
#include <iostream>
#include <vector>

using namespace std;

const int MAX_NODES = 10000000;

struct Node {
    int l, r;
    int sum;
} tree[MAX_NODES];

int node_cnt = 0;

int build(int tl, int tr) {
    int id = ++node_cnt;
    tree[id].sum = 0;
    if (tl == tr) {
        tree[id].l = tree[id].r = 0;
        return id;
    }
    int tm = (tl + tr) / 2;
    tree[id].l = build(tl, tm);
    tree[id].r = build(tm + 1, tr);
    return id;
}

int update(int prev, int tl, int tr, int pos, int val) {
    int id = ++node_cnt;
    tree[id] = tree[prev];
    if (tl == tr) {
        tree[id].sum += val;
        return id;
    }
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        tree[id].l = update(tree[prev].l, tl, tm, pos, val);
    else
        tree[id].r = update(tree[prev].r, tm + 1, tr, pos, val);

```

```
42      tree[id].sum = tree[tree[id].l].sum + tree[tree[id].r].sum;
43      return id;
44  }
45
46  int query(int u, int tl, int tr, int q_idx) {
47      if (q_idx < tl) return 0;
48      if (tr <= q_idx) return tree[u].sum;
49
50      int tm = (tl + tr) / 2;
51      return query(tree[u].l, tl, tm, q_idx) +
52             query(tree[u].r, tm + 1, tr, q_idx);
53  }
54
55  struct Segment {
56      int l, r, h;
57  };
58
59  bool cmp(const Segment& a, const Segment& b) {
60      return a.h > b.h;
61  }
62
63  int main() {
64      ios_base::sync_with_stdio(false);
65      cin.tie(NULL);
66
67      int n, m;
68      if (!(cin >> n >> m)) return 0;
69
70      vector<Segment> segs(n);
71      vector<int> coords;
72      coords.reserve(2 * n);
73
74      for (int i = 0; i < n; ++i) {
75          cin >> segs[i].l >> segs[i].r >> segs[i].h;
76          coords.push_back(segs[i].l);
77          coords.push_back(segs[i].r + 1);
78      }
79
80      sort(coords.begin(), coords.end());
81      coords.erase(unique(coords.begin(), coords.end()), coords.end());
82
83      auto get_pos = [&](int val) {
84          return lower_bound(coords.begin(), coords.end(), val) - coords.begin();
85      };
```

```
86
87      int sz = coords.size();
88      sort(segs.begin(), segs.end(), cmp);
89
90      vector<pair<int, int>> history;
91
92      int current_root = build(0, sz - 1);
93
94      for (const auto& s : segs) {
95          int l_idx = get_pos(s.l);
96          int r_idx = get_pos(s.r + 1);
97
98          current_root = update(current_root, 0, sz - 1, l_idx, 1);
99          current_root = update(current_root, 0, sz - 1, r_idx, -1);
100
101         history.push_back({s.h, current_root});
102     }
103
104     for (int i = 0; i < m; ++i) {
105         int x, y;
106         cin >> x >> y;
107
108         auto it = lower_bound(history.begin(), history.end(), y, [](const pair<
                int, int>& p, int val) {
109             return p.first > val;
110         });
111
112         int ans = 0;
113         if (it != history.begin()) {
114             int virt_root = prev(it)->second;
115
116             auto x_it = upper_bound(coords.begin(), coords.end(), x);
117             if (x_it != coords.begin()) {
118                 int x_pos = prev(x_it) - coords.begin();
119                 ans = query(virt_root, 0, sz - 1, x_pos);
120             }
121         }
122
123         cout << ans << "\n";
124     }
125
126     return 0;
127 }
```

4 Консоль

```
george@GEORGE-PC:~/MaiLabs/MAI_labs_Discran/src$ make main
george@GEORGE-PC:~/MaiLabs/MAI_labs_Discran/src$ ./main
5 3
0 5 2
1 3 1
1 8 5
-2 0 3
-1 6 -1
1 0
3 -3
3 5
3
4
0
```

5 Тест производительности

Для оценки эффективности разработанного решения на базе персистентного дерева отрезков было проведено нагрузочное тестирование. Тесты генерировались случайным образом: координаты отрезков и точек выбирались равномерно из диапазона $[-10^9, 10^9]$.

Тестирование проводилось на машине с процессором AMD Ryzen 7 5600U и 16 ГБ ОЗУ. Операционная система: CachyOS (Arch Linux). Измерялось время работы программы (CPU time) и потребление памяти.

N (отрезков)	M (точек)	Время (сек)	Память (МБ)
1 000	1 000	0.004	1.2
10 000	10 000	0.016	12.0
50 000	50 000	0.116	60.0
100 000	100 000	0.271	120.0

Таблица 1: Результаты тестирования производительности

```
george@GEORGE-PC:~/MaiLabs/MAI_labs_Discran/src$ python3 benchmark_script.py
Benchmark script started
================================================
Running test N=1000,M=1000...
Time: 0.004s,Memory: 1200KB

Running test N=10000,M=10000...
Time: 0.016s,Memory: 12000KB

Running test N=50000,M=50000...
Time: 0.116s,Memory: 60000KB

Running test N=100000,M=100000...
Time: 0.271s,Memory: 120000KB

================================================
Benchmark completed
```

Как видно из таблицы, время работы растет почти линейно-логарифмически, что согласуется с теоретической оценкой $O((N + M) \log N)$. Потребление памяти также остается в разумных пределах, не превышая 120 МБ для максимального теста, что укладывается в ограничения (256 МБ).

6 Выводы

В ходе выполнения лабораторной работы была решена задача о подсчете количества отрезков, лежащих строго над заданными точками. Для решения была выбрана структура данных **персистентное дерево отрезков** в сочетании с методом **сжатия координат**.

Основные выводы:

- Использование персистентности позволяет эффективно решать задачи, содержащие временное измерение или измерение, которое можно свести к временному (в данном случае — высота h).
- Сжатие координат является необходимым этапом при работе с большими координатами (10^9) в задачах на деревьях отрезков, позволяя ограничить размер дерева количеством "интересных"точек ($O(N + M)$).
- Подход с разностным массивом на дереве отрезков (прибавление $+1$ и -1) позволяет свести задачу проверки вхождения в отрезок к задаче вычисления префиксной суммы.

Решение полностью поддерживает online-запросы, так как структура данных строится один раз по известным отрезкам, а запросы точек обрабатываются независимо.

Список литературы

[1] Дональд Э. Кнут. *Искусство программирования. Том 3. Сортировка и поиск.* — Вильямс, 2000.

[2] Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. *Алгоритмы: построение и анализ.* — Вильямс, 2013.

[3] *Persistent Segment Tree - CP-Algorithms.*
URL: `https://cp-algorithms.com/data_structures/segment_tree.html` (дата обращения: 08.12.2025).

[4] *Визуализация структур данных: Персистентные структуры.*
URL: `http://neerc.ifmo.ru/wiki/index.php?title=Персистентные_структуры_данных` (дата обращения: 08.12.2025).

[5] *C++ Reference.*
URL: `https://en.cppreference.com/` (дата обращения: 08.12.2025).