МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра №806 «Вычислительна математика и программирование»

Курсовая работа по курсу “Дискретный анализ”

Персистентные структуры данных

Студент: Кочкожаров Иван Вячеславович

Группа: М8О-308Б-22

Преподаватель: Макаров Никита Константинович

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2024

Содержание

Репозиторий

`https://github.com/kochkozharov/da-labs`

Постановка задачи

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 вам даны два числа n и m ($1 <= n, m <= 10^5$) — количество отрезков и количество точек соответственно. В следующих n строках вам заданы отрезки, в виде троек чисел l, r и h ($-10^9 <= l < r <= 10^9$, $-10^9 <= h <= 10^9$) — координаты x левой и правой границ отрезка и координата y отрезка соответственно. В следующих m строках вам даны пары чисел x, y ($-10^9 <= x, y <= 10^9$) — координаты точек.

Формат вывода

Для каждой точки выведите количество отрезков над ней

Метод решения

Мы имеем набор горизонтальных отрезков, которые характеризуются координатами l, r, h, где l и r задают горизонтальные границы, а h — вертикальную позицию. Задача заключается в том, чтобы для каждой точки (x,y) подсчитать количество отрезков, которые находятся строго выше неё. Задача усложняется тем, что точки поступают нам online.

Входные данные в данной задаче могут быть очень большими (до 10^5 отрезков и точек), и координаты могут варьироваться от -10^9 до 10^9. Поэтому наивные решения, которые перебирают все отрезки для каждой точки, будут слишком медленными O(n * m), особенно если данные содержат большие диапазоны чисел.

Основой решения является персистентное дерево отрезков. Оно позволяет за лагорифмическую сложность находить количество отрезков, которые «покрывают» определенную точку по оси x.

Дерево отрезков — это структура данных, используемая для работы с запросами диапазонов. Оно позволяет быстро выполнять следующие операции:

- Поиск суммы или минимума/максимума элементов в поддиапазоне (и любой другой ассоциативной функции)
- Обновление значения элемента с индексом i

Дерево отрезков строится на основе набора чисел, деля его на промежутки (или отрезки) и храня значения для каждого отрезка в вершинах дерева. Каждый внутренний узел дерева хранит информацию о сумме (или другой агрегированной информации) на некотором подмассиве. Дерево каждый раз делит весь массив на две половины и объединяет их результаты, что позволяет обрабатывать запросы диапазонов за $O(log(n))$, поскольку высота дерева ограничена $O(log(n))$.

Персистентное дерево отрезков — это дерево отрезков, которое сохраняет все предыдущие версии дерева при обновлении. Это значит,что каждый раз, когда мы обновляем дерево (добавляем отрезок или изменяем значение), создается новая версия дерева, в которой обновленный узел будет ссылаться на новую копию, а остальные узлы будут ссылаться на старые версии. Дерево отрезков хранит все версии дерева, а каждый запрос (в этом случае суммирование и обновление значения) будет обращаться к соответствующей версии дерева, не затрагивая предыдущие.

Чтобы эффективно работать с деревом отрезков, мы не можем позволить себе использовать прямые значения координат по оси y, так как они могут быть очень большими ($-10^9 <= h <= 10^9$) и разреженными. Вместо того чтобы напрямую оперировать с большими числами, мы можем поставить их в соответствие натуральным числам, где каждое уникальное значение будет отображаться на индекс в некотором компактном диапазоне, например, от 0 до k - 1, где k — это количество уникальных значений.

Так как запросы и обновления приходят по мере обработки данных, нам нужно найти версию дерева, которая отражает состояние данных на момент запроса. Для этого используется ассоциативный массив на основе rb-дерева на std::map.

Описание алгоритма

Для каждого отрезка, заданного координатами l (левая граница), r (правая граница) и высотой h, мы создаём два события:

1. Событие начала отрезка — оно происходит в точке l, где отрезок начинается. Это событие увеличивает количество "активных"отрезков на высоте h, то есть, увеличивает количество перекрывающих сегментов на данной высоте.

2. Событие окончания отрезка — оно происходит в точке r+1, где отрезок заканчивается. Это событие уменьшает количество "активных"отрезков на высоте h, так как отрезок больше не перекрывает эту точку.

Для оптимизации работы с высотами сегментов, мы используем сжатие координат. Вместо того чтобы работать с исходными значениями высот h отрезков, мы собираем уникальные высоты в множество. Множество устраняет повторяющиеся значения и сортирует их в порядке возрастания.

Затем мы создаём отображение (map) uniqueHeights, которое каждому уникальному значению высоты h присваивает уникальный индекс. Этот индекс будет использоваться для работы с сегментным деревом, что позволяет значительно сократить размер данных.

Строим персистентное дерево отрезков с максимальным числом индексов maxY, равным количеству уникальных высот.

Все события сегментов (начало и конец отрезков) сортируются по координате x.

Обработка каждого события:

- Когда мы обрабатываем событие начала отрезка (с координатой l), это означает, что сегмент с высотой h начинает перекрывать пространство на отрезке. Мы обновляем дерево отрезков, увеличив количество отрезков, перекрывающих высоту h на этом интервале.

- Когда мы обрабатываем событие окончания отрезка (с координатой r+1), это означает, что сегмент перестал перекрывать пространство в этой точке. Мы обновляем дерево, уменьшив количество отрезков, перекрывающих высоту h на этом интервале.

Для каждого события сегмента мы обновляем дерево и сохраняем новую версию, отражающую изменения в количестве перекрывающих сегментов на всех высотах.

Точки, для которых нужно найти количество перекрывающих их отрезков, также сортируются по координате x.

Для каждой точки (x,y), заданной в запросах, мы ищем минимальную высоту, которая больше или равна значению y, чтобы выполнить запрос в дереве найденной версии. Это делается с помощью метода upper_bound для uniqueHeights(который содержит отсортированные уникальные высоты).

Для того чтобы выполнить запрос на нужной версии дерева, мы используем отображение versionMap, которая отображает кординату отрезка в момент времени, когда деревья обновлялись. Для нахождения самой поздней версии дерева, которая актуальна для данной точки x:

- Мы ищем в отображении versionMap индекс, равный значению x (координата точки, приходящей online) или меньше.
- Если значение x не найдено, возвращаем индекс предыдущей версии дерева, которая актуальна для точки x.

После того как мы нашли нужную версию дерева, выполняем запрос к дереву, чтобы подсчитать количество сегментов, которые перекрывают точку.

После того как все запросы обработаны, выводим количество перекрывающих отрезков для каждой точки.

Описание программы

Программа содержит структуры Segments, которая хранит информацию о каждом отрезке: координата левой границы, высота и тип события (начало или конец отрезка), и Points, которая хранит информацию о каждой точке: координаты точки.

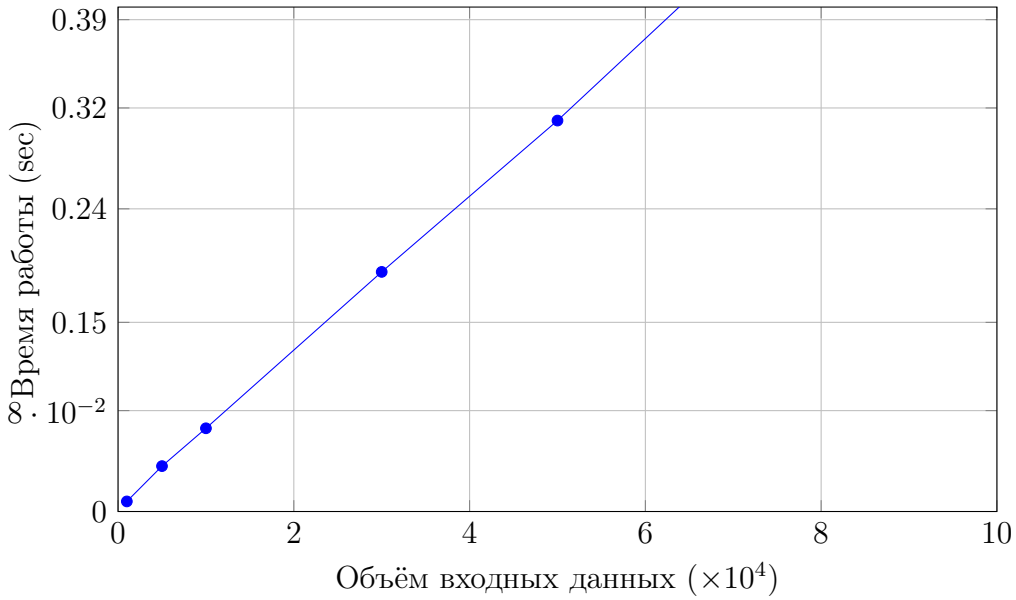
Также в программе реализован класс персистентного дерева отрезков - PersistentSegmentTree, который поддерживает несколько версий дерева для каждого события (начало или конец отрезка). Он содержит методы:

- std::shared_ptr<Node> Build(int start, int end) - создание нового персистеного узла
- PersistentSegmentTree(int size) - построение дерева для заданного количества элементов
- void Set(int version, int index, int heightIndex, int value) - обновление дерева, добавляя или удаляя отрезки
- int Sum(int version, int left, int right) constint Sum(int version, int left, int right) const - запрос на сумму на отрезке
- int Versions() const - получить количество версий дерева

Тест производительности

Для измерения производительсти создадим тесты с различными размерами входных данных. Подсчет времени будем производить с помощью библиотеки chrono, которая позволяет фиксировать время в начале и конце работы программы. Для каждого теста будем задавать размеры n и m: для первого теста (1000, 1000) для второго (5000, 5000), для третьего (10000, 10000), для четвертого (50000, 50000) и для пятого (100000, 100000), и генерировать данные для n отрезков и m точек с различными размерами

На графике представлена зависимость времени выполнения подсчета количества отрезков над точками от объёма входных данных.



Время работы каждого обновления в дереве — $O(log(n))$, и так как для каждого отрезка происходит два обновления (при старте и завершении), общая сложность на обработку отрезков составляет $O(n * log(n))$.

Для каждой точки также выполняется запрос $O(log(n))$. Таким образом, общее время работы алгоритма: $O((n + m) * log(n))$.

Выводы

В ходе данной курсовой работы были изучены персистентные структуры данных, которые позволяют более эффективно решать задачи, связанные с обработкой запросов и изменениями в данных.

В частности, была разработано и реализовано персистентное дерево отрезков, которое позволяет отслеживать изменения в отрезках и отвечать на запросы о количестве отрезков, лежащих строго над точками. Такой подход обеспечивает возможность обработки запросов в реальном времени, без необходимости повторного анализа всех данных при каждом новом запросе. Также в ходе реализации алгоритма была использована компрессия координат по оси y для эффективного использования памяти и ускорения работы структуры данных.

Использование персистентных структур данных позволило значительно улучшить производительность и качество решения задачи, особенно в условиях, требующих обработки запросов в реальном времени, а так же улучшило мои навыки в олимпиадном программировании.