Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ»

Студент: А. А. Терво
Преподаватель: С. А. Сорокин
Группа: М8О-307Б

Дата:
Оценка:
Подпись:

Москва, 2022

Курсовой проект

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

В первой строке вам даны два числа n и m — количество отрезков и количество точек соответственно. В следующих n строках вам заданы отрезки, в виде троек чисел l, r и h — координаты x левой и правой границ отрезка и координата y отрезка соответственно. В следующих m строках вам даны пары чисел — координаты точек.

Для каждой точки выведите количество отрезков 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.

1 Описание

Требуется реализовать поиск количество отрезков, лежащих строго над точкой, используя персистентные структуры данных.

Задачу я буду решать при помощи алгоритма заметающей прямой.

Суть решения заключается в следующем: координатная плоскость разбивается на вертикальные полосы таким образом, чтобы каждый отрезок либо пересекал полосу полностью, либо не пересекал её вообще; таким образом, каждая полоса будет отличаться от предыдущей (левее неё) на один отрезок (в новой полосе либо начинается новый отрезок, либо кончается старый); для каждой полосы хранятся отрезки, пересекающие её в отсортированном виде; при осуществлении запроса сначала производится поиск полосы методом бинарного поиска (сложность $O(logn)$, где n — количество полос), а затем осуществляется поиск в сортированной структуре координаты y точки, что также выполняется за $O(logn)$; и, наконец, подсчитывается количество искомых отрезков.

В качестве отсортированной структуры буду использовать бинарное дерево, но сделаю его персистентным. При добавлении или удалении вершины будет создаваться новый корень, по которому можно будет получить доступ к нужной версии дерева.

Персистентность реализуется следующим образом: при добавлении или удалении вершины создаётся новая ветвь дерева до этой вершины, остальные ветви остаются нетронутыми, в новое дерево переносятся только указатели на их узлы.

2 Исходный код

Опишу класс $TNode$ узла дерева и класс $TTree$ самого дерева.

В классе дерева корни каждого состояния хранятся в векторе. Реализованы функции добавления и удаления узла, отладочные методы печати дерева на экран, проверки его на корректность добавления и удаления узлов, а также вспомогательные методы для поиска и печати.

В `main.cpp` описаны структуры полосы и отрезка, реализованы функции вставки в вектор, добавления в отсортированную очередь с сохранением порядка и функция подсчёта элементов всех поддеревьев. В самой функции `main()` сначала считываются отрезки, затем по ним строится структура данных, далее в цикле считываются точки и выполняются запросы.

```
  struct TNode {
    TNode* Left;
    TNode* Right;
    double Height;
    TNode(double h) {
      Height = h;
      Left = nullptr;
      Right = nullptr;
    }
    ~TNode() {}
  };

  class TTree {
  public:
    std::vector<TNode*> Roots;
    TTree() {}
    ~TTree() {}

    void PrintTree(int);
    TNode* AddNode(double);
    TNode* RemNode(double);
    bool Check(double);
    bool Check(double, TNode*);
    void FullCheck();
    void FullCheck(TNode*);
    std::vector<double> Values;

  private:
    bool RecFind(TNode*, double &);
    void RecPrintTree(TNode*, int);
  };

}
```

3 Консоль

```
[alext@alext-pc solution]$ cat test10.txt
10 10
9552 9753 1268
8813 9028 3090
2353 2459 6112
8335 8660 2324
4764 5612 2174
823 1050 2027
6267 6592 9861
3665 4378 6732
6551 7493 8248
1892 2490 3770
6636 7150
5971 4256
2315 8539
7347 4668
5997 3459
9356 5322
5783 4120
8171 4309
4944 1750
2688 7563
[alext@alext-pc solution]$ ./solution < test10.txt
0
1
0
0
0
1
0
1
0
0
0
```

4 Тест производительности

Сравнивать производительность буду с супер наивной реализацией, в ней все отрезки складываются в вектор, затем при выполнении запроса происходит обход всего вектора и подсчёт количества отрезков. Тесты производительности буду проводить с помощью различных наборов данных:

1. 10 отрезков и 10 запросов
2. 50 отрезков и 50 запросов
3. 200 отрезков и 200 запросов
4. 20 отрезков и 200 запросов
5. 200 отрезков и 20 запросов

```
alext@alext-pc solution]$ ./naive <test10.txt
Super naive time: 37us
[alext@alext-pc solution]$ ./solution <test10.txt
Persistent time: 48us
alext@alext-pc solution]$ ./naive <test50.txt
Super naive time: 155us
[alext@alext-pc solution]$ ./solution <test50.txt
Persistent time: 263us
alext@alext-pc solution]$ ./naive <test200.txt
Super naive time: 155895us
[alext@alext-pc solution]$ ./solution <test200.txt
Persistent time: 22673us
alext@alext-pc solution]$ ./naive <test20-200.txt
Super naive time: 2197us
[alext@alext-pc solution]$ ./solution <test20-200.txt
Persistent time: 547us
alext@alext-pc solution]$ ./naive <test200-20.txt
Super naive time: 149us
alext@alext-pc solution]$ ./solution <test200-20.txt
Persistent time: 635us
```

Как видим, решение с использованием персистентных структур данных эффективнее на достаточно больших наборах особенно тогда, когда количество запросов превышает количество данных.

5 Выводы

При выполнении курсового проекта я изучил определение и виды персистентных структур данных.

Самым сложным было принять, что программа на 40 строк нас не устраивает, и надо писать прогамму на 400.

Единственным минусом данной реализации, помимо сложности, является утечка памяти из-за особенностей реализации перстистентной версии бинарного дерева поиска. Эту проблему можно решить при помощи умных указателей.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Сортировка подсчётом — Викиконспекты.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Сортировка_подсчётом` (дата обращения: 09.10.2020).

[3] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008