Содержание

Про STL 3

Обязательные задачи 4

Задача 6A. Сила с тобой, Люк [2 sec, 256 mb] 4

Задача 6B. Миллиардеры [2 sec, 256 mb] 5

Задача 6C. Гонка с дозарядкой [2 sec, 256 mb] 7

Дополнительные задачи 8

Задача 6D. Game [2 sec, 256 mb] 8

Задача 6E. Самая дальняя [4 sec, 256 mb] 9

Во всех задачах можно использовать `std::sort`, `std::set`.
Во всех задачах **нельзя** использовать дерево отрезков, декартово дерево.

Пример работы с файлами.

Если вы не умеете читать/выводить данные, или открывать файлы, воспользуйтесь примерами. `http://acm.math.spbu.ru/~sk1/algo/sum/`

Пример работы с файлами.

В некоторых задачах большой ввод и вывод. Про ввод-вывод в C++:
`http://acm.math.spbu.ru/~sk1/algo/input-output/cpp_common.html`
Имеет смысл пользоваться супер быстрым вводом-выводом. Две версии:
`http://acm.math.spbu.ru/~sk1/algo/input-output/io_export.cpp.html`
`http://acm.math.spbu.ru/~sk1/algo/input-output/fread_write_export.cpp.html`

Выделение памяти.

В некоторых задачах нужен STL, который активно использует динамическую память (set-ы, map-ы) переопределение стандартного аллокатора ускорит вашу программу:
`http://acm.math.spbu.ru/~sk1/algo/memory.cpp.html`

Про STL

В сегодняшних задачах достаточно уметь пользоваться следующими структурами, алгоритмами из STL: `vector`, `sort`, `set`, `map`, `unordered_map`, `priority_queue`. Местами может быть полезно ускорить программу, использовав рукописную хеш-таблицу вместо `map`, `unordered_map`. Местами можно ускорить программу, использовав `priority_queue` вместо `set`. Если вы что-то не знаете про STL, например, как пользоваться `set`, всегда к вашим услугам `google: C++ set`, `google: C++ set example`

Возможно вам пригодятся следующие примеры:
`set<int> s;` – куча целых чисел, множество целых чисел
`multiset<int> s;` – куча целых чисел, допускающая одинаковые значения
`*s.begin(), *s.rbegin();` – минимум и максимум
`map<int,int> m;` – ассоциативный массив, сохраняющий порядок
`unordered_map<int,int> m;` – ассоциативный массив
`unordered_map<int,int> m(N);` – теперь память зарезервивована под N элементов
`map<int, vector<int>> m;` – для целого числа храним вектор целых чисел
`map<string, set<int>> m;` – для строки храним множество целых чисел
`priority_queue<int> q;` – куча, в корне максимум
`priority_queue<int, vector<int>, greater<int>> q;` – куча, в корне минимум

Не забудьте: переопределение стандартного аллокатора ускорит вашу программу:
`http://acm.math.spbu.ru/~sk1/algo/memory.cpp.html`
Это нужно добавить в начало программы.

Обязательные задачи

Задача 6A. Сила с тобой, Люк [2 sec, 256 mb]

Дан массив a из n чисел, нужно научиться обрабатывать запросы двух типов.

- `change (i, y)` — сделать a_i равным y.
- `int get ()` — вернуть индекс i такой, что a_i встречается в массиве наименьшее возможное число раз. Если таких i несколько, вернуть минимально возможный.

Все индексы, встречающиеся в задаче, нумеруются с нуля.

Формат входных данных

На первой строке размер массива n. На второй сам массив — n целых чисел от 0 до 10^9-1. На третей строке число запросов q. Следующие q строк содержат запросы в формате "?" (`get`) и "= i y" (`change`).

Ограничения: $n, q \leqslant 300\,000$.

Формат выходных данных

На каждый запрос `get` выведите на отдельной строке ответ.

Замечание

Поскольку 8.5 мегабайт нельзя прочитать из файла мгновенно (как и записать 1 мегабайт данных), используйте максимально быстрые ввод вывод.

Примеры

sforce.in	sforce.out
4 1 2 3 4 5 ? = 0 2 ? = 3 3 ?	0 2 0
6 1 1 2 1 1 0 3 ? = 5 2 ?	2 2

Задача 6B. Миллиардеры [2 sec, 256 mb]

Возможно, вы знаете, что из всех городов мира больше всего миллиардеров живёт в Москве. Но, поскольку работа миллиардера подразумевает частые перемещения по всему свету, в определённые дни какой-то другой город может занимать первую строчку в таком рейтинге. Ваши приятели из ФСБ, ФБР, MI5 и Шин Бет скинули вам списки перемещений всех миллиардеров за последнее время. Ваш работодатель просит посчитать, сколько дней в течение этого периода каждый из городов мира был первым по общей сумме денег миллиардеров, находящихся в нём.

Формат входных данных

В первой строке записано число n — количество миллиардеров ($1 \leqslant n \leqslant 10\,000$). Каждая из следующих n строк содержит данные на определённого человека: его имя, название города, где он находился в первый день данного периода, и размер состояния. В следующей строке записаны два числа: m — количество дней, о которых есть данные ($1 \leqslant m \leqslant 50\,000$), k — количество зарегистрированных перемещений миллиардеров ($0 \leqslant k \leqslant 50\,000$). Следующие k строк содержат список перемещений в формате: номер дня (от 1 до $m-1$), имя человека, название города назначения. Вы можете считать, что миллиардеры путешествуют не чаще одного раза в день, и что они отбывают поздно вечером и прибывают в город назначения рано утром следующего дня. Список упорядочен по возрастанию номера дня. Все имена и названия городов состоят не более чем из 20 латинских букв, регистр букв имеет значение. Состояния миллиардеров лежат в пределах от 1 до 100 миллиардов.

Формат выходных данных

В каждой строке должно содержаться название города и, через пробел, количество дней, в течение которых этот город лидировал по общему состоянию миллиардеров, находящихся в нём. Если таких дней не было, пропустите этот город. Города должны быть отсортированы по алфавиту (используйте обычный порядок символов: ABC...Zabc...z).

Примеры

milliarder.in	milliarder.out
5 Abramovich London 15000000000 Deripaska Moscow 10000000000 Potanin Moscow 5000000000 Berezovsky London 2500000000 Khodorkovsky Chita 1000000000 25 9 1 Abramovich Anadyr 5 Potanin Courchevel 10 Abramovich Moscow 11 Abramovich London 11 Deripaska StPetersburg 15 Potanin Norilsk 20 Berezovsky Tbilisi 21 Potanin StPetersburg 22 Berezovsky London	Anadyr 5 London 14 Moscow 1

Замечание

Если упорядочить события по времени, то

```
London : 1
Anadyr : 5
Moscow : 1
London : 13
```

что означает, что сперва один день Лондон был на первом месте и так далее...

Задача 6C. Гонка с дозарядкой [2 sec, 256 mb]

Есть $Y + 1$ гоночная трасса. i-я трасса – горизонтальный отрезок $(0, i)$ – (X, i). Есть n заправок. j-я заправка представляет собой вертикальный отрезок (x_j, y_{j1}) – (x_j, y_{j2}). Проезжая по i-й трассе, машина начинает в точке $(0, i)$ и двигается прямолинейно равномерно к точке (X, i), тратя на каждую единицу расстояния одну единицу бензина. Если в какой-то момент машина проезжает заправку (точка-машина лежит на отрезке-заправке), то бак машины мгновенно заполняется до максимума. Если в какой-то момент бензин закончился, а машина не находится в точке заправки или точке (X, i), трасса считается не пройденной. Для каждого i от 0 до Y определите, какой минимальный объём бака должна иметь машина, чтобы пройти i-ю трассу. Машина начинает с полным баком.

Формат входных данных

На первой строке целые числа n, Y, X $(1 \leqslant n, Y, X \leqslant 200\,000)$.

Следующие n строк содержат по три целых числа x_i, y_{i1}, y_{i2} – описания заправок $(0 < x_i < X,\ 0 \leqslant y_{i1} < y_{i2} \leqslant Y)$.

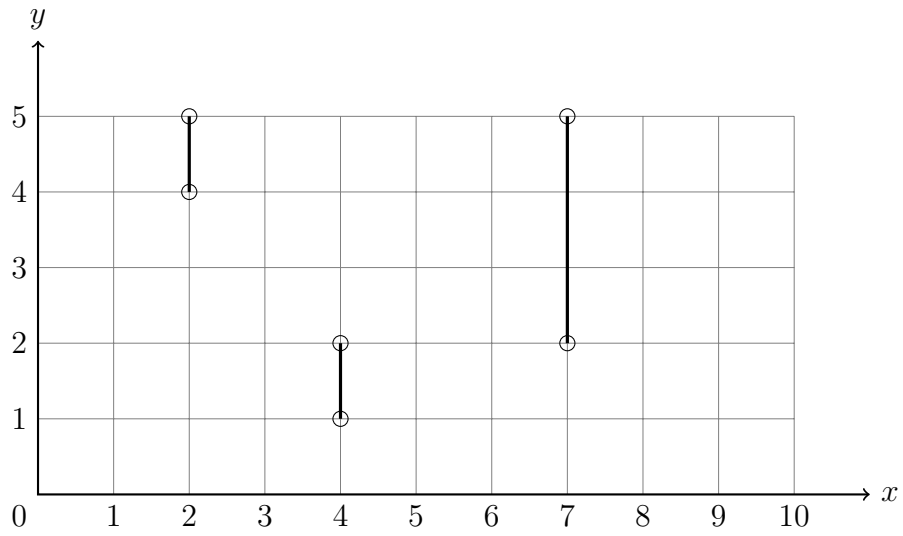
Формат выходных данных

Выведите $Y + 1$ целое число – ответы для всех трасс.

Примеры

segmentrace.in	segmentrace.out
3 5 10 4 1 2 7 2 5 2 4 5	10 6 4 7 5 5

Иллюстрация



Замечание

Зарпрещается пользоваться структурами данных не из STL. Есть простое решение, использующее только set<int>.

Это не простая задача. Главня идея для решения: сканирующая (заметающая) прямая и события. Заметьте, что сканировать события можно в разных направлениях: по x и по y.

Дополнительные задачи

Задача 6D. Game [2 sec, 256 mb]

Вася и Петя снова затеяли очень интересную игру! Петя выписывает на доску по порядку n чисел. Теперь мальчики, не советуясь, выбирают по 2 числа от 1 до n.

Пусть Петя выбрал числа l и r ($l \leqslant r$). Тогда очки, заработанные им, вычисляются по формуле $\sum_{k=l}^{r} b_k \cdot \min_{k=l..r} b_k$, где b_k — k-ое число на доске. Аналогично вычисляются очки, заработанные Васей. Затем мальчики говорят друг другу свои очки. У кого очков меньше, тот проиграл. Если очков одинаковое количество, то это считается ничьёй.

Представьте, что вы — Вася. И вы очень хотите хотя бы не проиграть. При этом очень не хотите быть пойманными на жульничестве (то есть хотите, чтобы названное вами число очков возможно было получить честным способом).

Напишите программу, которая находит отрезок, дающий максимальное количество очков.

Формат входных данных

В первой строке ввода задано натуральное число n ($1 \leqslant n \leqslant 2\,000\,000$) — количество чисел на доске. В следующей строке содержатся n целых чисел b_k через пробел в том же порядке, что они написаны на доске ($|b_k| \leqslant 100\,000$).

Формат выходных данных

Выведите число, которое должен назвать Вася, чтобы заведомо не проиграть и не быть пойманным на жульничестве. Во второй строке выведите границы отрезка, чтобы Вася смог доказать свою честность, если его начнут подозревать в жульничестве. Удачи!

Пример

game.in	game.out
3 1 2 3	10 2 3

Замечание

Эта задача не про STL =)

Принимаются **только** решения за линейное время.

Задача 6E. Самая дальняя [4 sec, 256 mb]

Даны N точек на плоскости, нужно уметь обрабатывать следующие запросы:

- `get a b` — возвращает максимум по всем точкам величины $ax + by$.
- `add x y` — добавить точку в множество.

Формат входных данных

Число N ($1 \leqslant N \leqslant 10^5$) и N точек. Далее число M ($1 \leqslant M \leqslant 10^5$ — количество запросов и собственно запросы. Формат запросов можно посмотреть в примере. Все координаты точек и числа `a`, `b` — целые числа, по модулю не превосходящие 10^9.

Формат выходных данных

На каждый запрос вида `get` выведите одно целое число — максимум величины $ax + by$.

Пример

mostfar.in	mostfar.out
3 0 0 1 0 0 1 10 get 1 1 get -1 -1 get 1 -1 get -1 1 add 2 2 add -2 -2 get 1 1 get -1 -1 get 1 -1 get -1 1	1 0 1 1 4 4 1 1

Замечание

Автор задачи уже написал некоторый код, который вы можете скачать по адресу
`http://acm.math.spbu.ru/~sk1/mm/au-download/statements/mostfar-lib.html`
Код можно скачать и использовать. Код на языке `C++`.

Написанная часть умеет делать две вещи:

- `Build(` множество точек на плоскости `)`.
- `GetMax(a, b)`.

Время работы: `Build` за $O(N \log N)$, `GetMax` за $O(\log N)$.