Содержание

Must have 2

Задача 11A. Persistent Array [3 sec, 256 mb] 2

Задача 11B. Персистентная очередь [1 sec, 256 mb] 3

Обязательные задачи 4

Задача 11C. CHM [0.4 sec, 256 mb] 4

Задача 11D. Persistent List [4 sec, 768 mb] 5

Задача 11E. Вперёд! [0.7 sec, 256 mb] 6

Дополнительные задачи 7

Задача 11F. Вставка ключевых значений [1 sec, 256 mb] 7

Задача 11G. Эх, дороги... [0.5 sec, 256 mb] 8

Обратите внимание, входные данные лежат в **стандартном потоке ввода** (он же stdin), вывести ответ нужно в **стандартный поток вывода** (он же stdout).

В некоторых задачах большой ввод и вывод. Пользуйтесь быстрым вводом-выводом.

В некоторых задачах нужен STL, который активно использует динамическую память (set-ы, map-ы) переопределение стандартного аллокатора ускорит вашу программу.

Обратите внимание на `GNU C++` компиляторы с суффиксом `inc`, они позволяют пользоваться дополнительной библиотекой. Под ними можно сдать вот это.

Must have

Задача 11A. Persistent Array [3 sec, 256 mb]

Дан массив (вернее, первая, начальная его версия).

Нужно уметь в online отвечать на два запроса:

- `create i j x` — создать из i-й версии новую, в которой j-й элемент равен x, а остальные элементы такие же, как в i-й версии.
- `get i j` — сказать, чему равен j-й элемент в i-й версии.

Это интерактивная задача. Запросы нужно обрабатывать в online, результат каждого запроса `flush`-ить до чтения следующего.

Формат входных данных

Количество чисел в массиве N ($1 \leqslant N \leqslant 10^5$) и N элементов массива. Далее количество запросов M ($1 \leqslant M \leqslant 10^5$) и M запросов. Формат описания запросов можно посмотреть в примере. Если уже существует K версий, новая версия получает номер $K + 1$. И исходные, и новые элементы массива — целые числа от 0 до 10^9. Элементы в массиве нумеруются числами от 1 до N.

Формат выходных данных

На каждый запрос типа `get` вывести соответствующий элемент нужного массива.

Пример

stdin	stdout
6 1 2 3 4 5 6 11 create 1 6 10 create 2 5 8 create 1 5 30 get 1 6 get 1 5 get 2 6 get 2 5 get 3 6 get 3 5 get 4 6 get 4 5	6 5 10 5 10 8 6 30

Замечание

Буфферизированные способы ввода, например, `readInt`, не подойдут, так как пытаются читать сразу "до конца файла". Пример корректного кода:

```
string query;
for (int i = 0; i < queryCnt; i++) {
    getline(cin, query);
    process(query);
    cout << flush; // cout << endl; тоже делает flush
}
```

Задача 11B. Персистентная очередь [1 sec, 256 mb]

Реализуйте персистентную очередь.

Формат входных данных

Первая строка содержит количество действий n ($1 \leqslant n \leqslant 200\,000$).
В строке номер $i + 1$ содержится описание действия i:

- `1 t m` — добавить в конец очереди номер t ($0 \leqslant t < i$) число m;
- `-1 t` — удалить из очереди номер t ($0 \leqslant t < i$) первый элемент.

В результате действия i, описанного в строке $i + 1$ создается очередь номер i.
Изначально имеется пустая очередь с номером ноль.
Все числа во входном файле целые, и помещаются в знаковый 32-битный тип.

Формат выходных данных

Для каждой операции удаления выведите удаленный элемент на отдельной строке.

Примеры

stdin	stdout
10 1 0 1 1 1 2 1 2 3 1 2 4 -1 3 -1 5 -1 6 -1 4 -1 8 -1 9	1 2 3 1 2 4

Подсказка по решению

Все запросы в offline ⇒ вы простое и быстрое решение за $\mathcal{O}(n)$ с деревом версий.
Ещё можно сдать *pairing* идею с лекции.
Переиспользовать онлайн-персистентность из задачи A тоже можно попробовать, но может не пройти по времени.

Обязательные задачи

Задача 11C. CHM [0.4 sec, 256 mb]

Ваша задача — реализовать *Persistent Disjoint-Set-Union*.

Disjoint-Set-Union. Изначально у вас есть n элементов, каждый в отдельном множестве. Нужно научиться отвечать на 2 типа запросов:

- `+ a b` — объединить множества, в которых лежат элементы a и b.
- `? a b` — сказать, лежат ли элементы a и b сейчас в одном множестве.

Persistent Disjoint-Set-Union. Теперь у нас будет несколько копий (версий) структуры данных *Disjoint-Set-Union*. Запросы будут выглядеть так:

- `+ i a b` — запрос к i-й структуре, объединить множества, в которых лежат элементы a и b. При этом i-я структура остается неизменной, создается новая версия, ей присваивается новый номер (какой? читайте дальше).
- `? i a b` — запрос к i-й структуре, сказать, лежат ли элементы a и b сейчас в одном множестве.

Формат входных данных

На первой строке 2 числа N ($1 \leqslant N \leqslant 10^5$) и K ($0 \leqslant K \leqslant 10^5$) — число элементов и число запросов. Изначально все элементы находятся в различных множествах. Эта изначальная копия (версия) структуры имеет номер 0.

Далее следуют K строк, на каждой описание очередного запроса. Формат запросов описан выше. Запросы нумеруются целыми числами от 1 до K.

Пусть j-й из K запросов имеет вид «`+ i a b`». Тогда новая версия получит номер j. Запросы вида «`? i a b`» не порождают новых структур.

Формат выходных данных

Для каждого запроса вида `? i a b` на отдельной строке нужно вывести `YES` или `NO`.

Пример

stdin	stdout
4 7 + 0 1 2 ? 0 1 2 ? 1 1 2 + 1 2 3 ? 4 3 1 ? 0 4 4 ? 4 1 4	NO YES YES YES NO

Подсказка по решению

Эту задачу Вы умеете решать и в offline, и в online. Offline точно пройдёт по времени.

Задача 11D. Persistent List [4 sec, 768 mb]

Даны N списков. Каждый состоит из одного элемента.
Нужно научитсья совершать следующие операции:

- `merge` — взять два каких-то уже существующих списка и породить новый, равный их конкатенации.
- `head` — взять какой-то уже существующий список L и породить два новых, в одном первый элемент L, во втором весь L кроме первого элемента.
- `tail` — взять какой-то уже существующий список L и породить два новых, в одном весь L кроме последнего элемента, во втором последний элемент L.

Для свежесозданных списков нужно говорить сумму элементов в них по модулю $10^9 + 7$.

Формат входных данных

Число N ($1 \leqslant N \leqslant 10^5$). Далее N целых чисел от 1 до 10^9 — элементы списков.
Исходные списки имеют номера — $1, 2, \cdots, N$.
Затем число M ($1 \leqslant M \leqslant 10^5$) — количество операций.
Далее даны операции в следующем формате:

- `merge i j`
- `head i`
- `tail i`

Здесь `i` и `j` — номера уже существующих списков.
Если в текущий момент имеется K списков, новый список получает номер $K + 1$.
Для операций `head` и `tail` считается, что сперва порождается левая часть, затем правая (см. пример). Также вам гарантируется, что никогда не будут порождаться пустые списки.

Формат выходных данных

Для каждого нового списка нужно вывести сумму элементов по модулю $10^9 + 7$.

Пример

stdin	stdout
4 1 2 3 4 7 merge 1 2 merge 3 4 merge 6 5 head 7 tail 9 merge 2 3 merge 1 1	3 7 10 3 7 5 2 5 2

Подсказка по решению

Какая глубина может быть после n операций `merge` с самим собой?
Обязательно ли хранить все элементы?

Задача 11E. Вперёд! [0.7 sec, 256 mb]

Капрал Дукар любит раздавать приказы своей роте. Самый любимый его приказ — "Вперёд!". Капрал строит солдат в ряд и отдаёт некоторое количество приказов, каждый из них звучит так: "Рядовые с l_i по l_j — вперёд!"

Перед тем, как Дукар отдал первый приказ, солдаты были пронумерованы от 1 до n, слева направо. Услышав приказ "Рядовые с l_i по l_j — вперёд!", солдаты, стоящие на местах с l_i по l_j включительно, продвигаются в начало ряда, в том же порядке, в котором были.

Например, если в какой-то момент солдаты стоят в порядке $1, 3, 6, 2, 5, 4$, то после приказа "Рядовые с 2 по 3 — вперёд!", порядок будет таким: $3, 6, 1, 2, 5, 4$. А если потом Капрал вышлет вперёд солдат с 3 по 4, то порядок будет уже таким: $1, 2, 3, 6, 5, 4$.

Вам дана последовательность из приказов Капрала. Найдите порядок, в котором будут стоять солдаты после исполнения всех приказов.

Формат входных данных

В первой строке входного файла указаны числа n и m ($2 \leqslant n \leqslant 100\,000$, $1 \leqslant m \leqslant 100\,000$) — число солдат и число приказов. Следующие m строк содержат приказы в виде двух целых чисел: l_i и r_i ($1 \leqslant l_i \leqslant r_i \leqslant n$).

Формат выходных данных

Выведите в выходной файл n целых чисел — порядок, в котором будут стоять солдаты после исполнения всех приказов.

Пример

stdin	stdout
6 3 2 4 3 5 2 2	1 4 5 2 3 6

Подсказка по решению

Обычная задача про дерево по неявному ключу со split.

Проще (короче) всего пишется Декартово Дерево.

Дополнительные задачи

Задача 11F. Вставка ключевых значений [1 sec, 256 mb]

Вас наняла на работу компания MacroHard, чтобы вы разработали новую структуру данных для хранения целых ключевых значений.

Эта структура выглядит как массив A бесконечной длины, ячейки которого нумеруются с единицы. Изначально все ячейки пусты. Единственная операция, которую необходимо поддерживать — это операция `Insert(L,K)`, где L — положение в массиве, а K — некоторое положительное целое ключевое значение. Операция выполняется следующим образом:

- Если ячейка `A[L]` пуста, то присвоить `A[L] := K`.
- Иначе выполнить `Insert(L+1,A[L])`, а затем присвоить `A[L] := K`.

По заданной последовательности из N целых чисел $L_1, L_2, \ldots, L_N$ вам необходимо вывести содержимое этого массива после выполнения следующей последовательности операций:

```
Insert(L_1, 1)
Insert(L_2, 2)
...
Insert(L_N, N)
```

Формат входных данных

В первой строке входного файла содержится N (число операций `Insert`) и M (максимальный номер позиции, которую можно использовать в операции `Insert`). ($1 \leqslant N, M \leqslant 131\,072$).

В следующей строке даны N целых чисел L_i ($1 \leqslant L_i \leqslant M$).

Формат выходных данных

Выведите содержимое массива после выполнения данной последовательности операций `Insert`. На первой строке выведите W — номер последней несвободной позиции в массиве. Далее выведите W целых чисел — $A[1], A[2], \ldots, A[W]$. Для пустых ячеек выводите нули.

Пример

stdin	stdout
5 4 3 3 4 1 3	6 4 0 5 2 3 1

Подсказка по решению

У жюри есть простое решение одним декартовым деревом.

Задача 11G. Эх, дороги... [0.5 sec, 256 mb]

В многострадальном Тридесятом государстве опять готовится дорожная реформа. Впрочем, надо признать, дороги в этом государстве находятся в довольно плачевном состоянии. Так что реформа не повредит. Одна проблема — дорожникам не развернуться, поскольку в стране действует жесткий закон — из каждого города должно вести не более двух дорог. Все дороги в государстве двусторонние, то есть по ним разрешено движение в обоих направлениях (разумеется, разметка отсутствует). В результате реформы некоторые дороги будут строиться, а некоторые другие закрываться на бессрочный ремонт.

Петя работает диспетчером в службе грузоперевозок на дальние расстояния. В связи с предстоящими реформами, ему необходимо оперативно определять оптимальные маршруты между городами в условиях постоянно меняющейся дорожной ситуации. В силу большого количества пробок и сотрудников дорожной полиции в городах, критерием оптимальности маршрута считается количество промежуточных городов, которые необходимо проехать.

Помогите Пете по заданной последовательности сообщений об изменении структуры дорог и запросам об оптимальном способе проезда из одного города в другой, оперативно отвечать на запросы.

Формат входных данных

В первой строке входного файла заданы числа n — количество городов, m — количество дорог в начале реформы и q — количество сообщений об изменении дорожной структуры и запросов ($1 \leqslant n \leqslant 100\,000$, $0 \leqslant m \leqslant 100\,000$, $0 \leqslant q \leqslant 200\,000$). Следующие m строк содержат по два целых числа каждая — пары городов, соединенных дорогами перед реформой. Следующие q строк содержат по три элемента, разделенных пробелами. «`+` i j» означает строительство дороги от города i до города j, «`-` i j» означает закрытие дороги от города i до города j, «`?` i j» означает запрос об оптимальном пути между городами i и j.

Гарантируется, что в начале и после каждого изменения никакие два города не соединены более чем одной дорогой, и из каждого города выходит не более двух дорог. Никакой город не соединяется дорогой сам с собой.

Формат выходных данных

На каждый запрос вида «`?` i j» выведите одно число — минимальное количество промежуточных городов на маршруте из города i в город j. Если проехать из i в j невозможно, выведите -1.

Пример

stdin	stdout
5 4 6 1 2 2 3 1 3 4 5 ? 1 2 ? 1 5 - 2 3 ? 2 3 + 2 4 ? 1 5	0 -1 1 2