Задачи к курсу «Алгоритмы: теория и практика. Структуры данных»

https://stepik.org/1547

Александр Куликов

15 мая 2017 г.

Содержание

1 Предисловие

1.1 Добро пожаловать!

Добро пожаловать на курс по структурам данных! В курсе будут рассмотрены структуры данных, наиболее часто использующиеся на практике: массивы, списки, очереди, стеки, динамические массивы, очереди с приоритетами, системы непересекающихся множеств, хеш-таблицы, сбалансированные деревья. Вы узнаете, как такие структуры данных реализованы в разных языках программирования, и, конечно же, потренируетесь самостоятельно их реализовывать, применять и расширять.

Основная цель курса — узнать, как устроены основные структуры данных (чтобы не пользоваться их готовыми реализациями как чёрным ящиком, а точно знать, чего от реализации ожидать), и научиться выбирать подходящую структуру данных при решении заданной вычислительной задачи.

1.2 Требования

Для освоения курса вам понадобятся знание одного из распространённых языков программирования (C++, Java, Python, Octave, Haskell) на базовом уровне (циклы, массивы, списки, очереди) и базовые знания математики (доказательство от противного, доказательство по индукции, логарифм, экспонента).

Данный курс является продолжением курса “Алгоритмы: теория и практика. Методы”. Если вы не проходили его, мы настоятельно рекомендуем вам пройти хотя бы его первую неделю: там вводятся стандартные обозначения для оценки времени работы алгоритмов, которыми мы будем активно пользоваться, а также даются рекомендации по решению задач на программирование.

1.3 Задачи на программирование

Важная составляющая данного курса — закрепление изученного материала через решение задач на программирование. Про каждую задачу гарантируется, что она может быть решена на `C++`, `Java`,

`Python3` с хотя бы тройным запасом по времени и памяти. Мы верим, что на других языках программирования решения тоже есть, но не проверяли этого и, соответственно, не гарантируем этого.

Мы сознательно не раскрываем входные данные, на которых проверяющая система тестирует ваши программы. Ваша цель в данном курсе — потренироваться писать быстрые и надёжные программы, потренироваться тестировать их и отлаживать.

1.4 Сертификат

За каждую задачу на программирование даётся один балл, если она решена до мягкого дедлайна, и полбалла, если она решена после мягкого, но до жёсткого дедлайна.

Всего в курсе будет 17 задач на программирование. Для получения сертификата необходимо набрать хотя бы 9 баллов, для получения сертификата с отличием — хотя бы 12.

1.5 Форум

Пожалуйста, не размещайте решения задач на программирование на форуме (под стэпами), даже если оно не работает. Если ваше решение не принимается тестирующей системой и вы никак не можете понять, почему так происходит, задайте вопрос на форуме: «Я протестировал своё решение на всех примерах из условия задачи, протестировал его на больших входах, протестировал на таких-то краевых входах, а оно всё равно не принимается. Помогите, пожалуйста, придумать вход для отладки.» И наоборот: если вы видите такой вопрос на форуме, постарайтесь, пожалуйста, помочь тому, кто спрашивает. *В конце курса мы подарим экземпляр книги [С. Дасгупта, Х. Пападимитриу, У. Вазирани. Алгоритмы. МЦНМО. 2014.] самому хорошему, на наш субъективный взгляд, помощнику.*

После того, как вы решите задачу, у вас появится доступ ко вкладке «решения». Там вы можете разместить своё решение и посмотреть на решения других слушателей курса. Это отличный способ учиться друг у друга.

Пожалуйста, отнеситесь с пониманием к тому, что у преподавателя нет возможности отвечать на такие вопросы: «Скажите, пожалуйста, что не так с решением #56718239? Я проверил: оно работает на

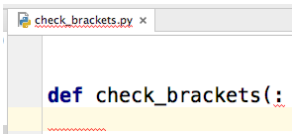
всех примерах из условия, а в вашем курсе оно не работает почему-то. Наверное, что-то не так с вашей проверяющей системой.»

2 Базовые структуры данных

2.1 Скобки в коде

Скобки в коде
Проверить, правильно ли расставлены скобки в данном коде.

Вход. Исходный код программы.
Выход. Проверить, верно ли расставлены скобки. Если нет, выдать индекс первой ошибки.



Вы разрабатываете текстовый редактор для программистов и хотите реализовать проверку корректности расстановки скобок. В коде могут встречаться скобки `[]{}()`. Из них скобки `[`, `{` и `(` считаются открывающими, а соответствующими им закрывающими скобками являются `]`, `}` и `)`.

В случае, если скобки расставлены неправильно, редактор должен также сообщить пользователю первое место, где обнаружена ошибка. В первую очередь необходимо найти закрывающую скобку, для которой либо нет соответствующей открывающей (например, скобка `]` в строке “`]()`”), либо же она закрывает не соответствующую ей открывающую скобку (пример: “`()[}`”). Если таких ошибок нет, необходимо найти первую открывающую скобку, для которой нет соответствующей закрывающей (пример: скобка `(` в строке “`{}([]`”).

Помимо скобок, исходный код может содержать символы латинского алфавита, цифры и знаки препинания.

Формат входа. Строка $s[1 \dots n]$, состоящая из заглавных и прописных букв латинского алфавита, цифр, знаков препинания и скобок из множества `[]{}()`.

Формат выхода. Если скобки в s расставлены правильно, выведите строку “Success". В противном случае выведите индекс (используя индексацию с единицы) первой закрывающей скобки, для которой нет соответствующей открывающей. Если такой нет, выведите индекс первой открывающей скобки, для которой нет соответствующей закрывающей.

Ограничения. $1 \leq n \leq 10^5$.

Пример.

Вход:

```
[]
```

Выход:

```
Success
```

Пример.

Вход:

```
{}[]
```

Выход:

```
Success
```

Пример.

Вход:

```
[()]
```

Выход:

```
Success
```

Пример.

Вход:

```
(())
```

Выход:

```
Success
```

Пример.

Вход:

```
{[]}()
```

Выход:

```
Success
```

Пример.

Вход:

```
{
```

Выход:

```
1
```

Пример.

Вход:

```
{[}
```

Выход:

```
3
```

Пример.

Вход:

```
foo(bar);
```

Выход:

```
Success
```

Пример.

Вход:

```
foo(bar[i);
```

Выход:

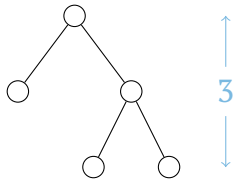
```
10
```

2.2 Высота дерева

Высота дерева
Вычислить высоту данного дерева.

> **Вход.** Корневое дерево с вершинами $\{0, \ldots, n-1\}$, заданное как последовательность $parent_0, \ldots, parent_{n-1}$, где $parent_i$ — родитель i-й вершины.
> **Выход.** Высота дерева.

Деревья имеют огромное количество применений в Computer Science. Они используются как для представления данных, так и во многих алгоритмах машинного обучения. Далее мы также узнаем, как сбалансированные деревья используются для реализации словарей и ассоциативных массивов. Данные структуры данных так или иначе используются во всех языках программирования и базах данных.



Ваша цель в данной задаче — научиться хранить и эффективно обрабатывать деревья, даже если в них сотни тысяч вершин.

Формата входа. Первая строка содержит натуральное число n. Вторая строка содержит n целых неотрицательных чисел $parent_0, \ldots, parent_{n-1}$. Для каждого $0 \le i \le n-1$, $parent_i$ — родитель вершины i; если $parent_i = -1$, то i является корнем. Гарантируется, что корень ровно один. Гарантируется, что данная последовательность задаёт дерево.

Формат выхода. Высота дерева.

Ограничения. $1 \le n \le 10^5$.

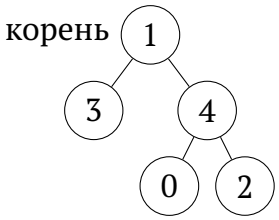
Пример.

Вход:

```
5
4 -1 4 1 1
```

Выход:

```
3
```



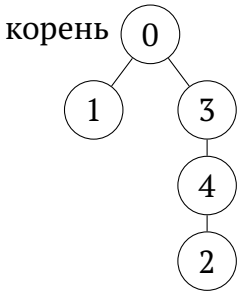
Пример.

Вход:

```
5
-1 0 4 0 3
```

Выход:

```
4
```

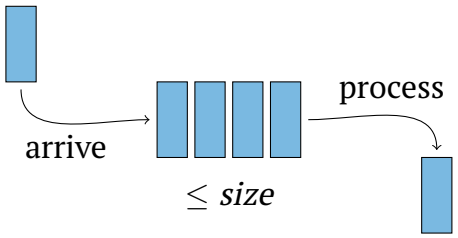


2.3 Обработка сетевых пакетов

Обработка сетевых пакетов
Реализовать обработчик сетевых пакетов.

> **Вход.** Размер буфера *size* и число пакетов n, а также две последовательности $\mathit{arrival}_1, \ldots, \mathit{arrival}_n$ и $\mathit{duration}_1, \ldots, \mathit{duration}_n$, обозначающих время поступления и длительность обработки n пакетов.
> **Выход.** Для каждого из данных n пакетов необходимо вывести время начала его обработки или -1, если пакет не был обработан (это происходит в случае, когда пакет поступает в момент, когда в буфере компьютера уже находится *size* пакетов).

Ваша цель — реализовать симулятор обработки сетевых пакетов. Для i-го пакета известно время его поступления $\mathit{arrival}_i$, а также время $\mathit{duration}_i$, необходимое на его обработку. В вашем распоряжении имеется один процессор, который обрабатывает пакеты в порядке их поступления. Если процессор начинает обрабатывать пакет i (что занимает время $\mathit{duration}_i$), он не прерывается и не останавливается до тех пор, пока не обработает пакет.



У компьютера, обрабатывающего пакеты, имеется сетевой буфер размера *size*. До начала обработки пакеты хранятся в буфере. Если буфер полностью заполнен в момент поступления пакета (есть *size* пакетов, поступивших ранее, которые до сих пор не обработаны), этот пакет отбрасывается и уже не будет обработан. Если несколько пакетов поступает в одно и то же время, они все будут сперва сохранены в буфер (несколько последних из них могут быть отброшены, если буфер заполнится).

Компьютер обрабатывает пакеты в порядке их поступления. Он начинает обрабатывать следующий пакет из буфера сразу после того, как обработает текущий пакет. Компьютер может простаивать, если

все пакеты уже обработаны и в буфере нет пакетов. Пакет освобождает место в буфере сразу же, как компьютер заканчивает его обработку.

Формат входа. Первая строка входа содержит размера буфера *size* и число пакетов n. Каждая из следующих n строк содержит два числа: время $\mathit{arrival}_i$ прибытия i-го пакета и время $\mathit{duration}_i$, необходимое на его обработку. Гарантируется, что $\mathit{arrival}_1 \leq \mathit{arrival}_2 \leq \cdots \leq \mathit{arrival}_n$. При этом может оказаться, что $\mathit{arrival}_{i-1} = \mathit{arrival}_i$. В таком случае считаем, что пакет $i-1$ поступил раньше пакета i.

Формата выхода. Для каждого из n пакетов выведите время, когда процессор начал его обрабатывать, или -1, если пакет был отброшен.

Ограничения. Все числа во входе целые. $1 \leq \mathit{size} \leq 10^5$; $1 \leq n \leq 10^5$; $0 \leq \mathit{arrival}_i \leq 10^6$; $0 \leq \mathit{duration}_i \leq 10^3$; $\mathit{arrival}_i \leq \mathit{arrival}_{i+1}$ для всех $1 \leq i \leq n-1$.

Пример.

Вход:

```
1 0
```

Выход:

Если пакетов нет, выводить ничего не нужно.

Пример.

Вход:

```
1 1
0 0
```

Выход:

```
0
```

Пакет поступил в момент времени 0, и компьютер тут же начал его обрабатывать.

Пример.

Вход:

```
1 2
0 1
0 1
```

Выход:

```
0
-1
```

Первый пакет поступил в момент времени 0, второй пакет поступил также в момент времени 0, но был отброшен, поскольку буфер в этот момент полностью заполнен (первым пакетом). Первый пакет начал обрабатываться в момент времени 0, второй был отброшен.

Пример.

Вход:

```
1 2
0 1
1 1
```

Выход:

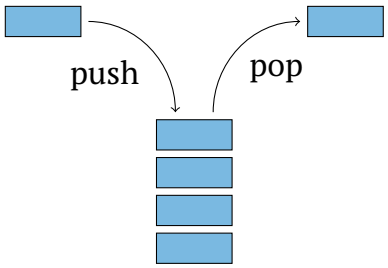
```
0
1
```

2.4 Стек с поддержкой максимума

Стек с поддержкой максимума
Реализовать стек с поддержкой операций `push`, `pop` *и* `max`.

> **Вход.** Последовательность запросов `push`, `pop` и `max` .
> **Выход.** Для каждого запроса `max` вывести максимальное число, находящее на стеке.

Стек — абстрактная структура данных, поддерживающая операции `push` и `pop`. Несложно реализовать стек так, чтобы обе эти операции работали за константное время. В данной задача ваша цель — расширить интерфейс стека так, чтобы он дополнительно поддерживал операцию `max` и при этом чтобы время работы всех операций по-прежнему было константным.



Формата входа. Первая строка содержит число запросов q. Каждая из последующих q строк задаёт запрос в одном из следующих форматов: `push v`, `pop`, or `max`.

Формат выхода. Для каждого запроса `max` выведите (в отдельной строке) текущий максимум на стеке.

Ограничения. $1 \leq q \leq 400\,000$, $0 \leq v \leq 100\,000$.

Пример.

Вход:

```
3
push 1
push 7
pop
```

Выход:

Выход пуст, потому что нет `max` запросов.

Пример.

Вход:

```
5
push 2
push 1
max
pop
max
```

Выход:

```
2
2
```

Пример.

Вход:

```
6
push 7
push 1
push 7
max
pop
max
```

Выход:

```
7
7
```

Пример.

Вход:

```
5
push 1
push 2
max
pop
max
```

Выход:

```
2
1
```

Пример.

Вход:

```
10
push 2
push 3
push 9
push 7
push 2
max
max
max
pop
max
```

Выход:

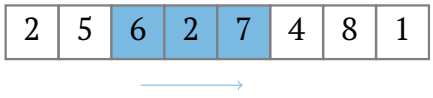
```
9
9
9
9
```

2.5 Максимум в скользящем окне

Максимум в скользящем окне
Найти максимум в каждом окне размера m *данного массива чисел* $A[1 \dots n]$.

> **Вход.** Массив чисел $A[1 \dots n]$ и число $1 \le m \le n$.
> **Выход.** Максимум подмассива $A[i \dots i + m - 1]$ для всех $1 \le i \le n - m + 1$.

Наивный способ решить данную задачу — честно просканировать каждое окно и найти в нём максимум. Время работы такого алгоритма — $O(nm)$. Ваша задача — реализовать алгоритм со временем работы $O(n)$.



Формат входа. Первая строка входа содержит число n, вторая — массив $A[1 \dots n]$, третья — число m.

Формат выхода. $n - m + 1$ максимумов, разделённых пробелами.

Ограничения. $1 \le n \le 10^5$, $1 \le m \le n$, $0 \le A[i] \le 10^5$ для всех $1 \le i \le n$.

Пример.

Вход:

```
8
2 7 3 1 5 2 6 2
4
```

Выход:

```
7 7 5 6 6
```

Пример.

Вход:

```
3
2 1 5
1
```

Выход:

```
2 1 5
```

Пример.

Вход:

```
3
2 3 9
3
```

Выход:

```
9
```

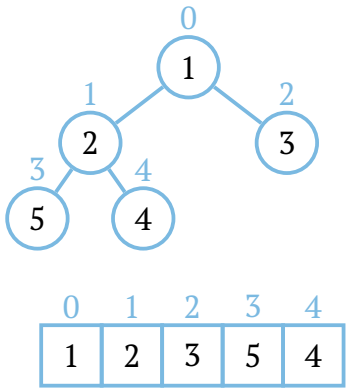
3 Очереди с приоритетами

3.1 Построение кучи

Построение кучи
Переставить элементы заданного массива чисел так, чтобы он удовлетворял свойству мин-кучи.

> **Вход.** Массив чисел $A[0 \dots n-1]$.
> **Выход.** Переставить элементы массива так, чтобы выполнялись неравенства $A[i] \leq A[2i+1]$ и $A[i] \leq A[2i+2]$ для всех i.

Построение кучи — ключевой шаг алгоритма сортировки кучей. Данный алгоритм имеет время работы $O(n \log n)$ в худшем случае в отличие от алгоритма быстрой сортировки, который гарантирует такую оценку только в среднем случае. Алгоритм быстрой сортировки чаще используют на практике, поскольку в большинстве случаев он работает быстрее, но алгоритм сортировки кучей используется для внешней сортировки данных, когда необходимо отсортировать данные огромного размера, не помещающиеся в память компьютера.



Чтобы превратить данный массив в кучу, необходимо произвести несколько обменов его элементов. Обменом мы называем базовую операцию, которая меняет местами элементы $A[i]$ и $A[j]$. Ваша цель в данной задаче — преобразовать заданный массив в кучу за линейное количество обменов.

Формат входа. Первая строка содержит число n. Следующая строка задаёт массив чисел $A[0], \dots, A[n-1]$.

Формат выхода. Первая строка выхода должна содержать число обменов m, которое должно удовлетворять неравенству $0 \leq m \leq 4n$. Каждая из последующих m строк должна задавать обмен двух

элементов массива A. Каждый обмен задаётся парой различных индексов $0 \le i \ne j \le n-1$. После применения всех обменов в указанном порядке массив должен превратиться в мин-кучу, то есть для всех $0 \le i \le n-1$ должны выполняться следующие два условия:

- если $2i+1 \le n-1$, то $A[i] < A[2i+1]$.
- если $2i+2 \le n-1$, то $A[i] < A[2i+2]$.

Ограничения. $1 \le n \le 10,5$; $0 \le A[i] \le 10^9$ для всех $0 \le i \le n-1$; все $A[i]$ попарно различны; $i \ne j$.

Пример.

Вход:

```
5
5 4 3 2 1
```

Выход:

```
3
1 4
0 1
1 3
```

Пример.

Вход:

```
5
1 2 3 4 5
```

Выход:

```
0
```

3.2 Параллельная обработка

Параллельная обработка
По данным n процессорам и m задач определите, для каждой из задач, каким процессором она будет обработана.

> **Вход.** Число процессоров n и последовательность чисел $t_0, \ldots, t_{m-1}$, где t_i — время, необходимое на обработку i-й задачи.
> **Выход.** Для каждой задачи определите, какой процессор и в какое время начнёт её обрабатывать, предполагая, что каждая задача поступает на обработку первому освободившемуся процессору.

В данной задаче ваша цель — реализовать симуляцию параллельной обработки списка задач. Такие обработчики (диспетчеры) есть во всех операционных системах.

У вас имеется n процессоров и последовательность из m задач. Для каждой задачи дано время, необходимое на её обработку. Очередная работа поступает к первому доступному процессору (то есть если доступных процессоров несколько, то доступный процессор с минимальным номером получает эту работу).

Формат входа. Первая строка входа содержит числа n и m. Вторая содержит числа $t_0, \ldots, t_{m-1}$, где t_i — время, необходимое на обработку i-й задачи. Считаем, что и процессоры, и задачи нумеруются с нуля.

Формат выхода. Выход должен содержать ровно m строк: i-я (считая с нуля) строка должна содержать номер процесса, который получит i-ю задачу на обработку, и время, когда это произойдёт.

Ограничения. $1 \leq n \leq 10^5$; $1 \leq m \leq 10^5$; $0 \leq t_i \leq 10^9$.

Пример.

Вход:

```
2 5
1 2 3 4 5
```

Выход:

```
0 0
1 0
0 1
1 2
0 4
```

Пример.

Вход:

```
4 20
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Выход:

```
0 0
1 0
2 0
3 0
0 1
1 1
2 1
3 1
0 2
1 2
2 2
3 2
0 3
1 3
2 3
3 3
0 4
1 4
2 4
3 4
```

4 Системы непересекающихся множеств

4.1 Объединение таблиц

Ваша цель в данной задаче — реализовать симуляцию объединения таблиц в базе данных.

В базе данных есть n таблиц, пронумерованных от 1 до n, над одним и тем же множеством столбцов (атрибутов). Каждая таблица содержит либо реальные записи в таблице, либо символьную ссылку на другую таблицу. Изначально все таблицы содержат реальные записи, и i-я таблица содержит r_i записей. Ваша цель — обработать m запросов типа $(\textit{destination}_i, \textit{source}_i)$:

1. Рассмотрим таблицу с номером $\textit{destination}_i$. Пройдясь по цепочке символьных ссылок, найдём номер реальной таблицы, на которую ссылается эта таблица:

   пока таблица $\textit{destination}_i$ содержит символическую ссылку:
   $\textit{destination}_i \leftarrow \texttt{symlink}(\textit{destination}_i)$

2. Сделаем то же самое с таблицей $\textit{source}_i$.

3. Теперь таблицы $\textit{destination}_i$ и $\textit{source}_i$ содержат реальные записи. Если $\textit{destination}_i \neq \textit{source}_i$, скопируем все записи из таблицы $\textit{source}_i$ в таблицу $\textit{destination}_i$, очистим таблицу $\textit{source}_i$ и пропишем в неё символическую ссылку на таблицу $\textit{destination}_i$.

4. Выведем максимальный размер среди всех n таблиц. Размером таблицы называется число строк в ней. Если таблица содержит символическую ссылку, считаем её размер равным нулю.

Формат входа. Первая строка содержит числа n и m — число таблиц и число запросов, соответственно. Вторая строка содержит n целых чисел $r_1, \ldots, r_n$ — размеры таблиц. Каждая из последующих m строк содержит два номера таблиц $destination_i$ и $source_i$, которые необходимо объединить.

Формат выхода. Для каждого из m запросов выведите максимальный размер таблицы после соответствующего объединения.

Ограничения. $1 \le n, m \le 100\,000$; $0 \le r_i \le 10\,000$;
$1 \le destination_i, source_i \le n$.

Пример.

Вход:

```
5 5
1 1 1 1 1
3 5
2 4
1 4
5 4
5 3
```

Выход:

```
2
2
3
5
5
```

Изначально каждая таблица содержит ровно одну строку.

1. После первой операции объединения все записи из таблицы 5 копируются в таблицу 3. Теперь таблица 5 является ссылкой на таблицу 3, а таблица 3 содержит две записи.

2. Вторая операция аналогичным образом переносит все записи из таблицы 2 в таблицу 4.

3. Третья операция пытается объединить таблицы 1 и 4, но таблица 4 ссылается на таблицу 2, поэтому все записи из таблицы 2 копируются в таблицу 1. Таблица 1 теперь содержит три строки.

4. Чтобы произвести четвёртую операцию, проследим пути из ссылок: $4 \to 2 \to 1$ и $5 \to 3$. Скопируем все записи из таблицы 1 в таблицу 3, после чего в таблице 3 будет пять записей.

5. После этого все таблицы ссылаются на таблицу 3, поэтому все оставшиеся запросы объединения ничего не меняют.

Пример.

Вход:

```
6 4
10 0 5 0 3 3
6 6
6 5
5 4
4 3
```

Выход:

```
10
10
10
11
```

1. Запрос объединения таблицы 6 с собой ничего не меняет, максимальным размером по-прежнему остаётся 10 (таблица 1).
2. Записи из таблицы 5 копируются в таблицу 6, размер таблицы 6 становится равным 6.
3. Записи из таблицы 4 копируются в таблицу 6, размер таблицы 6 становится равным 10.
4. Записи из таблицы 3 копируются в таблицу 6, размер таблицы 6 становится равным 11.

4.2 Автоматический анализ программ

При автоматическом анализе программ возникает такая задача.

Система равенств и неравенств
Проверить, можно ли присвоить переменным целые значения, чтобы выполнить заданные равенства вида $x_i = x_j$ и неравенства вида $x_p \neq x_q$.

> **Вход.** Число переменных n, а также список равенств вида $x_i = x_j$ и неравенства вида $x_p \neq x_q$.
> **Выход.** Проверить, выполнима ли данная система.

Формат входа. Первая строка содержит числа n, e, d. Каждая из следующих e строк содержит два числа i и j и задаёт равенство $x_i = x_j$. Каждая из следующих d строк содержит два числа i и j и задаёт неравенство $x_i \neq x_j$. Переменные индексируются с 1: $x_1, \ldots, x_n$.

Формат выхода. Выведите 1, если переменным $x_1, \ldots, x_n$ можно присвоить целые значения, чтобы все равенства и неравенства выполнились. В противном случае выведите 0.

Ограничения. $1 \leq n \leq 10^5$; $0 \leq e, d$; $e + d \leq 2 \cdot 10^5$; $1 \leq i, j \leq n$.

Пример.

Вход:

```
4 6 0
1 2
1 3
1 4
2 3
2 4
3 4
```

Выход:

```
1
```

Все переменные просто равны друг другу, поэтому система выполнима.

Пример.

Вход:

```
6 5 3
2 3
1 5
2 5
3 4
4 2
6 1
4 6
4 5
```

Выход:

```
0
```

$x_1 = x_2 = x_3 = x_4 = x_5$, но $x_4 \neq x_5$.

5 Хеш-таблицы

5.1 Телефонная книга

Телефонная книга
Реализовать структуру данных, эффективно обрабатывающую запросы вида `add number name`, `del number` *и* `find number`.

> **Вход.** Последовательность запросов вида `add number name`, `del number` и `find number`, где `number` — телефонный номер, содержащий не более семи знаков, а `name` — короткая строка.
> **Выход.** Для каждого запроса `find number` выведите соответствующее имя или сообщите, что такой записи нет.

Цель в данной задаче — реализовать простую телефонную книгу, поддерживающую три следующих типа запросов. С указанными ограничениями данная задача может быть решена с использованием таблицы с прямой адресацией.

- `add number name`: добавить запись с именем `name` и телефонным номером `number`. Если запись с таким телефонным номером уже есть, нужно заменить в ней имя на `name`.
- `del number`: удалить запись с соответствующим телефонным номером. Если такой записи нет, ничего не делать.
- `find number`: найти имя записи с телефонным номером `number`. Если запись с таким номером есть, вывести имя. В противном случае вывести «not found» (без кавычек).

Формат входа. Первая строка содержит число запросов n. Каждая из следующих n строк задаёт запрос в одном из трёх описанных выше форматов.

Формат выхода. Для каждого запроса `find` выведите в отдельной строке либо имя, либо «not found».

Ограничения. $1 \leq n \leq 10^5$. Телефонные номера содержат не более семи цифр и не содержат ведущих нулей. Имена содержат только буквы латинского алфавита, не являются пустыми строками и имеют длину не больше 15. Гарантируется, что среди имён не встречается строка «not found».

Пример.

Вход:

```
12
add 911 police
add 76213 Mom
add 17239 Bob
find 76213
find 910
find 911
del 910
del 911
find 911
find 76213
add 76213 daddy
find 76213
```

Выход:

```
Mom
not found
police
not found
Mom
daddy
```

Пример.

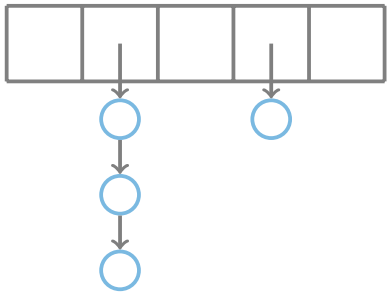
Вход:

```
8
find 3839442
add 123456 me
add 0 granny
find 0
find 123456
del 0
del 0
find 0
```

Вхыод:

```
not found
granny
me
not found
```

5.2 Хеширование цепочками



Хеширование цепочками — один из наиболее популярных методов реализации хеш-таблиц на практике. Ваша цель в данной задаче — реализовать такую схему, используя таблицу с m ячейками и полиномиальной хеш-функцией на строках

$$h(S) = \left(\sum_{i=0}^{|S|-1} S[i] x^i \bmod p \right) \bmod m \,,$$

где $S[i]$ — ASCII-код i-го символа строки S, $p = 1\,000\,000\,007$ — простое число, а $x = 263$. Ваша программа должна поддерживать следующие типы запросов:

- `add string`: добавить строку `string` в таблицу. Если такая строка уже есть, проигнорировать запрос;
- `del string`: удалить строку `string` из таблицы. Если такой строки нет, проигнорировать запрос;
- `find string`: вывести «yes» или «no» в зависимости от того, есть в таблице строка `string` или нет;
- `check i`: вывести i-й список (используя пробел в качестве разделителя); если i-й список пуст, вывести пустую строку.

При добавлении строки в цепочку, строка должна добавляться **в начало** цепочки.

Формат входа. Первая строка размер хеш-таблицы m. Следующая строка содержит количество запросов n. Каждая из последующих n строк содержит запрос одного из перечисленных выше четырёх типов.

Формат выхода. Для каждого из запросов типа `find` и `check` выведите результат в отдельной строке.

Ограничения. $1 \le n \le 10^5$; $\frac{n}{5} \le m \le n$. Все строки имеют длину от одного до пятнадцати и содержат только буквы латинского алфавита.

Пример.

Вход:

```
5
12
add world
add HellO
check 4
find World
find world
del world
check 4
del HellO
add luck
add GooD
check 2
del good
```

Выход:

```
HellO world
no
yes
HellO
GooD luck
```

ASCII коды букв 'w', 'o', 'r', 'l', 'd' равны 119, 111, 114, 108, 100, соответственно. Поэтому

$$h(\texttt{world}) = (119 + 111 \times 263 + 114 \times 263^2 + 108 \times 263^3 + 100 \times 263^4 \bmod 1\,000\,000\,007) \bmod 5 = 4\,.$$

Оказывается, что $h(\texttt{HellO})$ тоже равно четырём. Поскольку новые строки добавляются в начало списка, после второго запроса `add` список содержит строки `HellO` и `world` (именно в таком порядке). Строка `World` не находится, а `world` находится. После удаления строки `world` в цепочке 4 остаётся только строка `HellO`. И так далее.

Пример.

Вход:

```
4
8
add test
add test
find test
del test
find test
find Test
add Test
find Test
```

Выход:

```
yes
no
no
yes
```

Пример.

Вход:

```
3
12
check 0
find help
add help
add del
add add
find add
find del
del del
find del
check 0
check 1
check 2
```

Выход:

```
no
yes
yes
no

add help
```

Обратите внимание на то, что нужно выводить пустую строку в случае, если соответствующая цепочка пуста. Строки в запросах могут совпадать с названиями запросов.

Указания.

- Будьте осторожны с переполнением целого типа. Используйте `long long` в C++ и `long` в Java при необходимости. При вычислении значения многочлена по модулю p берите результат по модулю p после каждой арифметической операции.
- Будьте осторожны с отрицательными числами по модулю p. Во многих языках программирования $(-2)\%5 \neq 3\%5$. Один

из способов избежать этого — использовать $x \leftarrow ((a\%p) + p)\%p$ вместо $x \leftarrow a\%p$.

5.3 Поиск образца в тексте

Поиск образца в тексте
Найти все вхождения строки Pattern в строку Text.

> **Вход.** Строки *Pattern* и *Text*.
> **Выход.** Все индексы i строки *Text*, начиная с которых строка *Pattern* входит в *Text*:
> $Text[i..i + |Pattern| - 1] = Pattern$.


Реализуйте алгоритм Карпа–Рабина.

Формат входа. Образец *Pattern* и текст *Text*.

Формат выхода. Индексы вхождений строки *Pattern* в строку *Text* в возрастающем порядке, используя индексацию с нуля.

Ограничения. $1 \leq |Pattern| \leq |Text| \leq 5 \cdot 10^5$. Суммарная длина всех вхождений образца в текста не превосходит 10^8. Обе строки содержат буквы латинского алфавита.

Пример.

Вход:

```
aba
abacaba
```

Выход:

```
0 4
```

Образец `aba` входит в позициях 0 (`aba`caba) и 4 (abac`aba`) в текст `abacaba`.

Пример.

Вход:

```
Test
testTesttesT
```

Выход:

```
4
```

Пример.

Вход:

```
aaaaa
baaaaaaa
```

Выход:

```
1 2 3
```

Данный пример демонстрирует, что вхождения могут накладываться друг на друга.

Подсказки по реализации.

- Будьте осторожны с переполнением целого типа. Применяйте операцию взятия по модулю p после каждой арифметической операции.
- Будьте осторожны с взятием отрицательных чисел по модулю.
- В языке программирования `Python` используйте оператор `==` для сравнения строк (он работает быстро).
- В языке программирования `C++` будьте осторожны с методом `substr` класса `string`: он создаёт новую строку, на что тратится и время, и память. Если делать это в цикле, времени и памяти потратится много.
- А в языке программирование `Java` метод `substring` **не** создаёт новой строки. Не используйте `new String` там, где можно обойтись вызовом `substring`.

6 Деревья поиска

6.1 Обход двоичного дерева

Обход двоичного дерева
Построить in-order, pre-order и post-order обходы данного двоичного дерева.

> **Вход.** Двоичное дерево.
> **Выход.** Все его вершины в трёх разных порядках: in-order, pre-order и post-order.

In-order обход соответствует следующей рекурсивной процедуре, получающей на вход корень *v* текущего поддерева: произвести рекурсивный вызов для *v.left*, напечатать *v.key*, произвести рекурсивный вызов для *v.right*. Pre-order обход: напечатать *v.key*, произвести рекурсивный вызов для *v.left*, произвести рекурсивный вызов для *v.right*. Post-order: произвести рекурсивный вызов для *v.left*, произвести рекурсивный вызов для *v.right*, напечатать *v.key*.

Формат входа. Первая строка содержит число вершин n. Вершины дерева пронумерованы числами от 0 до $n-1$. Вершина 0 является корнем. Каждая из следующих n строк содержит информацию о вершинах $0, 1, \ldots, n-1$: i-я строка задаёт числа key_i, $left_i$ и $right_i$, где key_i — ключ вершины i, $left_i$ — индекс левого сына вершины i, а $right_i$ — индекс правого сына вершины i. Если у вершины i нет одного или обоих сыновей, соответствующее значение равно -1.

Формат выхода. Три строки: in-order, pre-order и post-order обходы.

Ограничения. $1 \le n \le 10^5$; $0 \le key_i \le 10^9$; $-1 \le left_i, right_i \le n-1$. Гарантируется, что вход задаёт корректное двоичное дерево: в частности, если $left_i \ne -1$ и $right_i \ne -1$, то $left_i \ne right_i$; никакая вершина не является сыном двух вершин; каждая вершина является потомком корня.

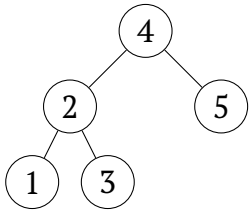
Пример.

Вход:

```
5
4 1 2
2 3 4
5 -1 -1
1 -1 -1
3 -1 -1
```

Выход:

```
1 2 3 4 5
4 2 1 3 5
1 3 2 5 4
```



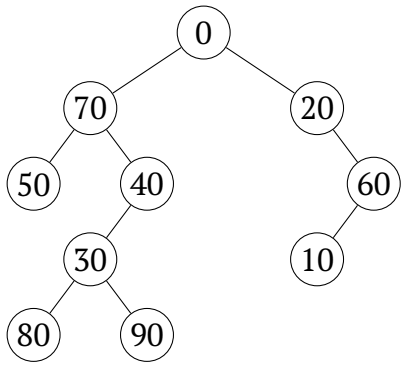
Пример.

Вход:

```
10
0 7 2
10 -1 -1
20 -1 6
30 8 9
40 3 -1
50 -1 -1
60 1 -1
70 5 4
80 -1 -1
90 -1 -1
```

Выход:

```
50 70 80 30 90 40 0 20 10 60
0 70 50 40 30 80 90 20 60 10
50 80 90 30 40 70 10 60 20 0
```



6.2 Проверка свойства дерева поиска

Проверка свойства дерева поиска
Проверить, является ли данное двоичное дерево деревом поиска.

> **Вход.** Двоичное дерево.
> **Выход.** Проверить, является ли оно корректным деревом поиска: верно ли, что для любой вершины дерева её ключ больше всех ключей в левом поддереве данной вершины и меньше всех ключей в правом поддереве.

Вы тестируете реализацию двоичного дерева поиска. У вас уже написан код, который ищет, добавляет и удаляет ключи, а также выводит внутреннее состояние структуры данных после каждой операции. Вам осталось проверить, что в каждый момент дерево остаётся корректным деревом поиска. Другими словами, вы хотите проверить, что для дерева корректно работает поиск, если ключ есть в дереве, то процедура поиска его обязательно найдёт, если ключа нет — то не найдёт.

Формат входа. Первая строка содержит число вершин n. Вершины дерева пронумерованы числами от 0 до $n-1$. Вершина 0 является корнем. Каждая из следующих n строк содержит информацию о вершинах $0, 1, \ldots, n-1$: i-я строка задаёт числа key_i, $left_i$ и $right_i$, где key_i — ключ вершины i, $left_i$ — индекс левого сына вершины i, а $right_i$ — индекс правого сына вершины i. Если у вершины i нет одного или обоих сыновей, соответствующее значение равно -1.

Формат выхода. Выведите «CORRECT», если дерево является корректным деревом поиска, и «INCORRECT» в противном случае.

Ограничения. $0 \le n \le 10^5$; $-2^{31} < key_i < 2^{31} - 1$; $-1 \le left_i, right_i \le n - 1$. Гарантируется, что вход задаёт корректное двоичное дерево: в частности, если $left_i \ne -1$ и $right_i \ne -1$, то $left_i \ne right_i$; никакая вершина не является сыном двух вершин; каждая вершина является потомком корня.

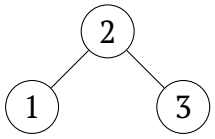
Пример.

Вход:

```
3
2 1 2
1 -1 -1
3 -1 -1
```

Выход:

```
CORRECT
```



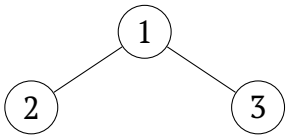
Пример.

Вход:

```
3
1 1 2
2 -1 -1
3 -1 -1
```

Выход:

```
INCORRECT
```



Пример.

Вход:

```
0
```

Выход:

```
CORRECT
```

Пустое дерево считается корректным.

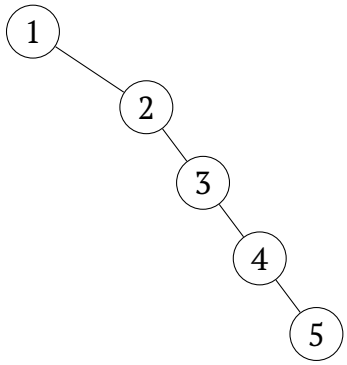
Пример.

Вход:

```
5
1 -1 1
2 -1 2
3 -1 3
4 -1 4
5 -1 -1
```

Выход:

```
CORRECT
```



Дерево не обязано быть сбалансированным.

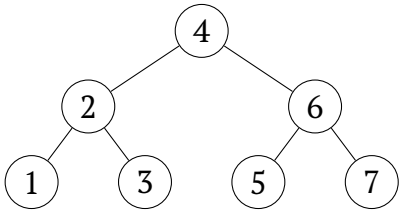
Пример.

Вход:

```
7
4 1 2
2 3 4
6 5 6
1 -1 -1
3 -1 -1
5 -1 -1
7 -1 -1
```

Выход:

```
CORRECT
```



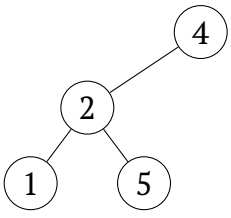
Пример.

Вход:

```
4
4 1 -1
2 2 3
1 -1 -1
5 -1 -1
```

Выход:

```
INCORRECT
```



6.3 Проверка более общего свойства дерева поиска

Данная задача полностью аналогична предыдущей, но проверять теперь нужно более общее свойство. Дереву разрешается содержать равные ключи, но они всегда должны находиться в правом поддереве. Формально, двоичное дерево называется деревом поиска, если для любой вершины её ключ больше всех ключей из её левого поддерева и **не меньше** всех ключей из правого поддерева.

Ограничения. $0 \le n \le 10^5$; $-2^{31} \le key_i \le 2^{31} - 1$ (таким образом, в качестве ключей допустимы минимальное и максимальное значение 32-битного целого типа, будьте осторожны с переполнением); $-1 \le left_i, right_i \le n - 1$. Гарантируется, что вход задаёт корректное двоичное дерево: в частности, если $left_i \ne -1$ и $right_i \ne -1$, то $left_i \ne right_i$; никакая вершина не является сыном двух вершин; каждая вершина является потомком корня.

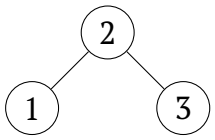
Пример.

Вход:

```
3
2 1 2
1 -1 -1
3 -1 -1
```

Выход:

```
CORRECT
```



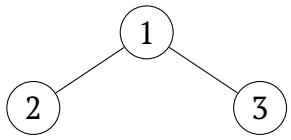
Пример.

Вход:

```
3
1 1 2
2 -1 -1
3 -1 -1
```

Выход:

```
INCORRECT
```



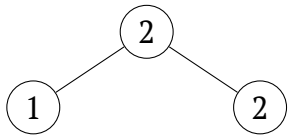
Пример.

Вход:

```
3
2 1 2
1 -1 -1
2 -1 -1
```

Выход:

```
CORRECT
```



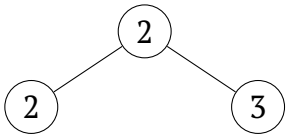
Пример.

Вход:

```
3
2 1 2
2 -1 -1
3 -1 -1
```

Выход:

```
INCORRECT
```



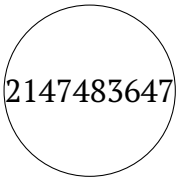
Пример.

Вход:

```
1
2147483647 -1 -1
```

Выход:

```
CORRECT
```



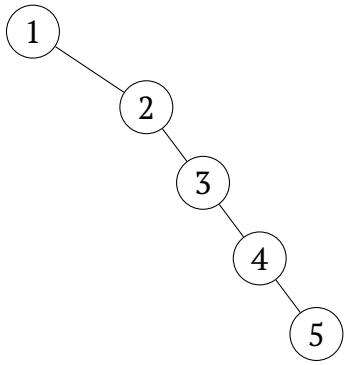
Пример.

Вход:

```
5
1 -1 1
2 -1 2
3 -1 3
4 -1 4
5 -1 -1
```

Выход:

```
CORRECT
```



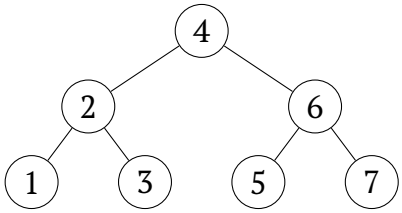
Пример.

Вход:

```
7
4 1 2
2 3 4
6 5 6
1 -1 -1
3 -1 -1
5 -1 -1
7 -1 -1
```

Выход:

```
CORRECT
```



6.4 Множество с запросами суммы на отрезке

Реализуйте структуру данных для хранения множества целых чисел, поддерживающую запросы добавления, удаления, поиска, а также суммы на отрезке. На вход в данной задаче будет дана последовательность таких запросов. Чтобы гарантировать, что ваша программа обрабатывает каждый запрос по мере поступления (то есть онлайн), каждый запрос будет зависеть от результата выполнения одного из предыдущих запросов. Если бы такой зависимости не было, задачу можно было бы решить оффлайн: сначала прочитать весь вход и сохранить все запросы в каком-нибудь виде, а потом прочитать вход ещё раз, параллельно отвечая на запросы.

Формат входа. Изначально множество пусто. Первая строка содержит число запросов n. Каждая из n следующих строк содержит запрос в одном из следующих четырёх форматов:

- `+ i`: добавить число $f(i)$ в множество (если оно уже есть, проигнорировать запрос);
- `- i`: удалить число $f(i)$ из множества (если его нет, проигнорировать запрос);
- `? i`: проверить принадлежность числа $f(i)$ множеству;
- `s l r`: посчитать сумму всех элементов множества, попадающих в отрезок $[f(l), f(r)]$.

Функция f определяется следующим образом. Пусть s — результат последнего запроса суммы на отрезке (если таких запросов ещё не было, то $s = 0$). Тогда

$$f(x) = (x + s) \bmod 1\,000\,000\,001\,.$$

Формат выхода. Для каждого запроса типа `? i` выведите «Found» или «Not found». Для каждого запроса суммы выведите сумму всех элементов множества, попадающих в отрезок $[f(l), f(r)]$. Гарантируется, что во всех тестах $f(l) \le f(r)$.

Ограничения. $1 \le n \le 10^5$; $0 \le i \le 10^9$.

Пример.

Вход:

```
15
? 1
+ 1
? 1
+ 2
s 1 2
+ 1000000000
? 1000000000
- 1000000000
? 1000000000
s 999999999 1000000000
- 2
? 2
- 0
+ 9
s 0 9
```

Выход:

```
Not found
Found
3
Found
Not found
1
Not found
10
```

Для первых пяти запросов $s = 0$, для следующих пяти — $s = 3$, для следующих пяти — $s = 1$. Заданные запросы разворачиваются в следующие: find(1), add(1), find(1), add(2), sum$(1, 2) \to 3$, add(2), find$(2) \to$ Found, del(2), find$(2) \to$ Not found, sum$(1, 2) \to 1$, del(3), find$(3) \to$ Not found, del(1), add(10), sum$(1, 10) \to 10$. Добавление элемента дважды не изменяет множество, как и попытки удалить элемент, которого в множестве нет.

Пример.

Вход:

```
5
? 0
+ 0
? 0
- 0
? 0
```

Выход:

```
Not found
Found
Not found
```

Пример.

Вход:

```
5
+ 491572259
? 491572259
? 899375874
s 310971296 877523306
+ 352411209
```

Выход:

```
Found
Not found
491572259
```

6.5 Rope

Ваша цель в данной задаче — реализовать структуру данных Rope. Данная структура данных хранит строку и позволяет эффективно вырезать кусок строки и переставить его в другое место.

Формат входа. Первая строка содержит исходную строку S, вторая — число запросов q. Каждая из последующих q строк задаёт запрос тройкой чисел i, j, k и означает следующее: вырезать подстроку $S[i..j]$ (где i и j индексируются с нуля) и вставить её после k-го символа оставшейся строки (где k индексируется с единицы), при этом если $k = 0$, то вставить вырезанный кусок надо в начало.

Формат выхода. Выведите полученную (после всех q запросов) строку.

Ограничения. S содержит только буквы латинского алфавита. $1 \le |S| \le 300\,000$; $1 \le q \le 100\,000$; $0 \le i \le j \le n-1$; $0 \le k \le n-(j-i+1)$.

Пример.

Вход:

```
hlelowrold
2
1 1 2
6 6 7
```

Выход:

```
helloworld
```

`hlelowrold` → `hellowrold` → `helloworld`

Пример.

Вход:

```
abcdef
2
0 1 1
4 5 0
```

Выход:

```
efcabd
```

`abcdef` → `cabdef` → `efcabd`