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

https://stepik.org/1547

Александр Куликов

24 апреля 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 16 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

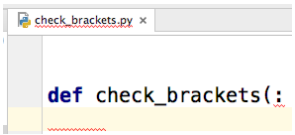
всех примерах из условия, а в вашем курсе оно не работает почему-то. Наверное, что-то не так с вашей проверяющей системой.”

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:

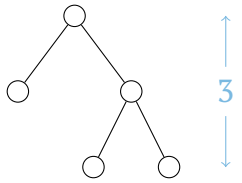
```
10
```

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. $1 \le n \le 10^5$.

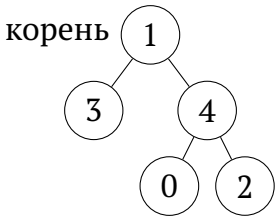
Пример.

Вход:

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

Выход:

```
3
```



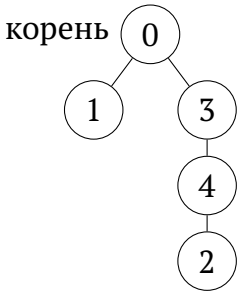
Пример.

Вход:

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

Выход:

```
4
```

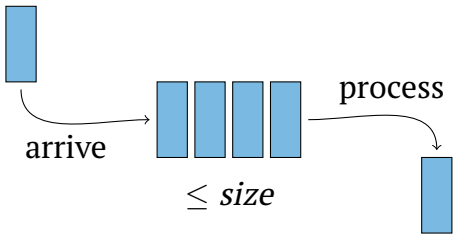


2.3 Обработка сетевых пакетов

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 пакевто 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:

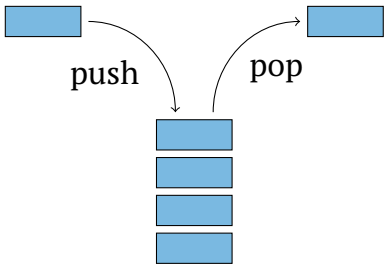
```
0
1
```

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. $1 \le q \le 400\,000$, $0 \le v \le 100\,000$.

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:

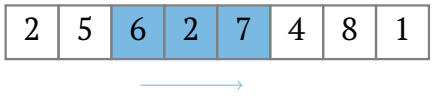
```
9
9
9
9
```

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
```

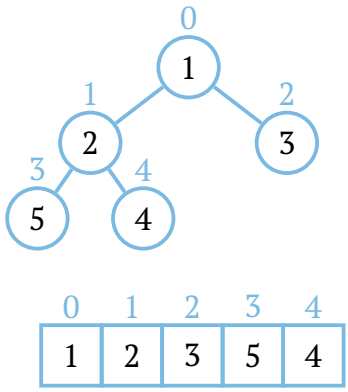
3 Очереди с приоритетами

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 $A[i] \le A[2i+1]$ и $A[i] \le A[2i+2]$ 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 $0 \le m \le 4n$. 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 входа. 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. $1 \leq n, m \leq 100\ 000$; $0 \leq r_i \leq 10\ 000$; $1 \leq destination_i, source_i \leq n$.

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 так или иначе 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

6 Деревья поиска