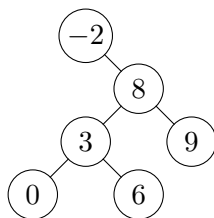


## Задача А. Высота дерева (!) (1 балл)

Имя входного файла: `height.in`  
Имя выходного файла: `height.out`  
Ограничение по времени: 2 секунды  
Ограничение по памяти: 256 мегабайт

Высотой дерева называется максимальное число вершин дерева в цепочке, начинающейся в корне дерева, заканчивающейся в одном из его листьев, и не содержащей никакую вершину дважды.

Так, высота дерева, состоящего из единственной вершины, равна единице. Высота пустого дерева (да, бывает и такое!) равна нулю. Высота дерева, изображенного на рисунке, равна четырем.



Дано двоичное дерево поиска. В вершинах этого дерева записаны ключи — целые числа, по абсолютному значению не превышающие  $10^9$ . Для каждой вершины дерева  $V$  выполняется следующее условие:

- все ключи вершин из левого поддерева меньше ключа вершины  $V$ ;
- все ключи вершин из правого поддерева больше ключа вершины  $V$ .

Найдите высоту данного дерева.

### Формат входного файла

Входной файл содержит описание двоичного дерева. В первой строке файла находится число  $N$  ( $0 \leq N \leq 200000$ ) — число вершин в дереве. В последующих  $N$  строках файла находятся описания вершин дерева. В  $(i + 1)$ -ой строке файла ( $1 \leq i \leq N$ ) находится описание  $i$ -ой вершины, состоящее из трех чисел  $K_i, L_i, R_i$ , разделенных пробелами — ключа в  $i$ -ой вершине ( $|K_i| \leq 10^9$ ), номера левого ребенка  $i$ -ой вершины ( $i < L_i \leq N$  или  $L_i = 0$ , если левого ребенка нет) и номера правого ребенка  $i$ -ой вершины ( $i < R_i \leq N$  или  $R_i = 0$ , если правого ребенка нет).

Все ключи различны. Гарантируется, что данное дерево является деревом поиска.

### Формат выходного файла

Выведите одно целое число — высоту дерева.

### Пример

<code>height.in</code>	<code>height.out</code>
6	4
-2 0 2	
8 4 3	
9 0 0	
3 5 6	
0 0 0	
6 0 0	

### Примечание

Во входном файле задано то же дерево, что и изображено на рисунке.

## Задача В. Проверка корректности (1 балл)

Имя входного файла: `check.in`  
Имя выходного файла: `check.out`  
Ограничение по времени: 2 секунды  
Ограничение по памяти: 256 мегабайт

Свойство двоичного дерева поиска можно сформулировать следующим образом: для каждой вершины дерева  $V$  выполняется следующее условие:

- все ключи вершин из левого поддерева меньше ключа вершины  $V$ ;
- все ключи вершин из правого поддерева больше ключа вершины  $V$ .

Дано двоичное дерево. Проверьте, выполняется ли для него свойство двоичного дерева поиска.

### Формат входного файла

Входной файл содержит описание двоичного дерева. В первой строке файла находится число  $N$  ( $0 \leq N \leq 200000$ ) — число вершин в дереве. В последующих  $N$  строках файла находятся описания вершин дерева. В  $(i + 1)$ -ой строке файла ( $1 \leq i \leq N$ ) находится описание  $i$ -ой вершины, состоящее из трех чисел  $K_i, L_i, R_i$ , разделенных пробелами — ключа в  $i$ -ой вершине ( $|K_i| \leq 10^9$ ), номера левого ребенка  $i$ -ой вершины ( $i < L_i \leq N$  или  $L_i = 0$ , если левого ребенка нет) и номера правого ребенка  $i$ -ой вершины ( $i < R_i \leq N$  или  $R_i = 0$ , если правого ребенка нет).

### Формат выходного файла

Выведите «YES», если данное на входе дерево является двоичным деревом поиска, и «NO», если не является.

### Примеры

<code>check.in</code>	<code>check.out</code>
6 -2 0 2 8 4 3 9 0 0 3 5 6 0 0 0 6 0 0	YES
0	YES
3 5 2 3 6 0 0 4 0 0	NO

### Примечание

Решение данной задачи поможет Вам в дальнейшем проверять корректность построенных деревьев при отладке более сложных алгоритмов.

## Задача С. Простое двоичное дерево поиска (2 балла)

Имя входного файла: `bstsimple.in`  
Имя выходного файла: `bstsimple.out`  
Ограничение по времени: 2 секунды  
Ограничение по памяти: 256 мегабайт

Реализуйте двоичное дерево поиска.

### Формат входного файла

Входной файл содержит описание операций с деревом, их количество не превышает 100. В каждой строке находится одна из следующих операций:

- `insert x` — добавить в дерево ключ  $x$
- `delete x` — удалить из дерева ключ  $x$ . Если ключа  $x$  в дереве нет, то ничего делать не надо
- `exists x` — если ключ  $x$  есть в дереве выведите «true», если нет «false»
- `next x` — выведите минимальный элемент в дереве, строго больший  $x$ , или «none» если такого нет
- `prev x` — выведите максимальный элемент в дереве, строго меньший  $x$ , или «none» если такого нет

В дерево помещаются и извлекаются только целые числа, не превышающие по модулю  $10^9$ .

### Формат выходного файла

Выведите последовательно результат выполнения всех операций `exists`, `next`, `prev`. Следуйте формату выходного файла из примера.

### Пример

<code>bstsimple.in</code>	<code>bstsimple.out</code>
<code>insert 2</code>	<code>true</code>
<code>insert 5</code>	<code>false</code>
<code>insert 3</code>	<code>5</code>
<code>exists 2</code>	<code>3</code>
<code>exists 4</code>	<code>none</code>
<code>next 4</code>	<code>3</code>
<code>prev 4</code>	
<code>delete 5</code>	
<code>next 4</code>	
<code>prev 4</code>	

### Примечание

Необходимо написать свое бинарное дерево поиска.

## Задача D. Интерпретатор языка Quack (3 балла)

Имя входного файла: quack.in  
Имя выходного файла: quack.out  
Ограничение по времени: 2 секунды  
Ограничение по памяти: 64 мегабайта

Язык Quack — забавный язык, который фигурирует в задаче G с Internet Problem Solving Contest 2004 (<http://ipsc.ksp.sk/2004/real/problems/g.html>). В этой лабораторной работе вам требуется написать интерпретатор языка Quack.

Виртуальная машина, на которой исполняется программа на языке Quack, имеет внутри себя очередь, содержащую целые числа по модулю 65536 (то есть, числа от 0 до 65535, или беззнаковое 16-битное целое). Слово `get` в описании операций означает извлечение из очереди, `put` — добавление в очередь. Кроме того, у виртуальной машины есть 26 регистров, которые обозначаются буквами от `a` до `z`. Изначально все регистры хранят нулевое значение. В языке Quack существуют следующие команды:

<code>+</code>	Сложение: <code>get x, get y, put (x+y) modulo 65536</code>
<code>-</code>	Вычитание: <code>get x, get y, put (x-y) modulo 65536</code> .
<code>*</code>	Умножение: <code>get x, get y, put (x*y) modulo 65536</code> .
<code>/</code>	Целочисленное деление: <code>get x, get y, put x div y</code> . (будем считать, что $x \div 0 = 0$ )
<code>%</code>	Взятие по модулю: <code>get x, get y, put x modulo y</code> . (будем считать, что $x \bmod 0 = 0$ )
<code>&gt;[register]</code>	Положить в регистр: <code>get x, установить значение [register] в x</code> .
<code>&lt;[register]</code>	Взять из регистра: <code>put значение [register]</code> .
<code>P</code>	Напечатать: <code>get x, вывести x в стандартный поток вывода и перевести строку</code> .
<code>P[register]</code>	Вывести значение регистра <code>[register]</code> в стандартный поток вывода и перевести строку.
<code>C</code>	Вывести как символ: <code>get x, вывести символ с ASCII кодом <math>x \bmod 256</math> в стандартный поток вывода</code> .
<code>C[register]</code>	Вывести регистр как символ: вывести символ с ASCII кодом $x \bmod 256$ (где $x$ — значение регистра <code>[register]</code> ) в стандартный поток вывода.
<code>:[label]</code>	Метка: эта строка программы имеет метку <code>[label]</code> .
<code>J[label]</code>	Переход на строку с меткой <code>[label]</code> .
<code>Z[register][label]</code>	Переход если 0: если значение регистра <code>[register]</code> равно нулю, выполнение программы продолжается с метки <code>[label]</code> .
<code>E[register1][register2][label]</code>	Переход если равны: если значения регистров <code>[register1]</code> и <code>[register2]</code> равны, исполнение программы продолжается с метки <code>[label]</code> .
<code>G[register1][register2][label]</code>	Переход если больше: если значение регистра <code>[register1]</code> больше, чем значение регистра <code>[register2]</code> , исполнение программы продолжается с метки <code>[label]</code> .
<code>Q</code>	Завершить работу программы. Работа также завершается, если выполнение доходит до конца программы.
<code>[number]</code>	Просто число во входном файле — <code>put</code> это число.

### Формат входного файла

Входной файл содержит корректную синтаксически программу на языке Quack. Известно, что программа завершает работу не более чем за 100 000 шагов.

## Формат выходного файла

Выведите содержимого стандартного потока вывода виртуальной машины в выходной файл.

## Пример

quack.in	quack.out
100 0 :start >a Zaend <a <a 1 + - >b <b Jstart :end P	5050

Второй пример подразумевает UNIX-переводы строки (один символ с кодом 10).

Лабораторная работа для групп М31хх  
Двоичное дерево поиска. Интерпретатор языка Quack. 202х год.

quack.in	quack.out
58	58
49	49
10	10
62	62
97	97
10	10
80	80
97	97
10	10
90	90
97	97
50	50
10	10
60	60
97	97
10	10
74	74
49	49
10	10
58	58
50	50
10	10
48	48
10	10
58	58
51	51
10	10
62	62
97	97
10	10
90	90
97	97
52	52
10	10
67	67
97	97
10	10
74	74
51	51
10	10
58	58
52	52
10	10
0	0
:1	:1
>a	>a
Pa	Pa
Za2	Za2
<a	<a
J1	J1
:2	:2
0	0
:3	:3
>a	>a
Za4	Za4
Ca	Ca
J3	J3
:4	:4