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.

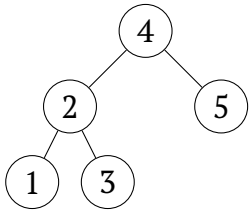
Пример.

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
```