Teaching of Data Structures

Submission deadline:	**2018-11-18 23:59:59**
Evaluation:	**3.7500**
Max. assessment:	**2.0000** (Without bonus points)
Submissions:	24 / 25
Advices:	0 / 0

Data structures are taught in many different courses at our faculty -- for example in AG1/2, PA1/2, and so on. This is no surprise, as data structures are a very important tool to effectively handle large amounts of data. However, it is not convenient to start teaching data structures starting from the most advanced ones, as students might get lost very quickly. Because of that, it is better to first demonstrate basic principles of easier and more basic variant of a particular data structure, show its disadvantages and imperfections, and slowly start to incorporate enhancements. This way, students get to understand the data structure much better.

The problem is when a basic variant of a data structure is to be used to handle large amounts of data -- due to its simplicity, there are natural limitations to the size of the input data. For example a basic implementation of a binary search tree can, for some sequence of operations, degenerate in a way all subsequent operations start to require linear time to complete. Teachers aren't satisfied with this situation and so they would like to visualize elementary data structures to students in a reasonable time. Because of that, you were called to help. Your first visualized data structure will be the mentioned binary search tree.

You task is to simulate behavior of a binary search tree (BST), **exactly** according to AG1 lecture, so it can be used in an external visualization program. Slides of lecture with the referential description of a BST can be found **at this link**. Visualization program is going to communicate with your implementation via a given interface. The input for your data structure will be standard BST operations. As the output, the visualization program will require your implementation to answer queries about the parent of a vertex represented by a given key. In order to ensure tolerable latency of the visualization program for large amount of data (tests on large datasets), your implementation is required to simulate behavior of a BST efficiently enough -- even in such case your implementation still has to be able to answer queries about the structure of visualized BST. Because this implementation is only a prototype, the current visualization program is rather aged and does not support communication with programs using external libraries. For that reason, it is not possible to use STL library to solve this task.

Input and Output Format:

- Input consists of several lines, where one line encodes one command (see the commands in quotes below):

  `"1 X"`
  : where `X` is a positive integer, $1 \leq X \leq 10^9$, requires your program to insert a key of value `X` to the BST.

  `"2 X"`
  : where `X` is a positive integer, $1 \leq X \leq 10^9$, requires your program to delete a key of value `X` from the BST.

  `"3 X"`
  : where `X` is a positive integer, $1 \leq X \leq 10^9$, requires your program to output the BST parent of a vertex represented by a key of value `X`; should the vertex has no parent in the BST, output a single line with text `"noparent"` (without quotes).

  `"4 X"`
  : where `X` is a positive integer, $1 \leq X \leq 10^9$, requires your program to output a successor in the BST of a key of value `X`; should the key `X` have no successor in the BST, output a single line with text `"nosuccessor"` (without quotes).

  `"5 X Y"`

where `X, Y` are positive integers, $1 \leq X \leq 10^9$, $1 \leq Y \leq 2$, requires your program to perform a simple rotation at a BST vertex representing a key of value `X` (for `Y = 1` perform a left rotation, for `Y = 2` a right rotation); should it be impossible to perform such rotation (the vertex is missing a left child in the case of a right rotation, or vice versa), output a single line with text `"norotate"` (without quotes).

`"6"`

encodes the end of the input to the BST, so there will be no more commands on input; it is guaranteed this is the last command on input.

- You can assume all inputs are valid.

Moreover:

- Should the key `X` given in a command of type 3, 4, or 5 not be present in the BST, output a single line with text `"notfound"` (without quotes).
- It is possible, that your program will be given a command to delete a key not present in the BST. If this happens, such command should be ignored.
- On the contrary, no two keys of the same value will be present in the BST at the same time. However, after a deletion of a certain key, it is possible this key will be inserted to the BST again.

Classification Conditions:

1. To pass the basic compulsory tests, your solution has to return correct answers for inputs of size at most 1 000 commands. Issued commands are only of types 1, 2, 3 and 6.
2. To pass the test with extended operations, your solution has to output correct answers within the time limit for inputs of size at most 1 000 commands. All types of commands may be issued.
3. To pass the test on large dataset #1, your solution has to output correct answers within the time limit for inputs of size at most 1 000 000 commands. Issued commands are only of types 1, 3, 4, 5 and 6.
4. To pass the test on large dataset #2, your solution has to output correct answers within the time limit for inputs of size at most 1 000 000 commands. All types of commands may be issued.

- Tests based on the examples are divided into two different tests because of the limitations on the types of commands. First example test examines the submitted program on the Input Examples 1 and 2 and is mandatory. The second example test checks the program on the Input Examples 3 and 4 and is not mandatory.

Examples:

Input Example 1:

```
1 30
1 10
1 20
1 25
3 10
3 20
3 30
3 25
3 1000
6
```

Output Example 1:

```
30
10
noparent
```

```
20
notfound
```

Input Example 2:

```
1 10
1 20
1 15
1 12
1 17
3 10
3 20
3 15
3 12
3 17
2 17
3 10
3 20
3 15
3 12
3 17
2 15
3 10
3 20
3 15
3 12
3 17
6
```

Output Example 2:

```
noparent
10
20
15
15
noparent
10
20
15
notfound
noparent
10
notfound
20
notfound
```
