Упражнение 9.4: Binary Search Tree

20 Punkte

Search tree operations, traversal

На предпоследнем листе упражнения вы реализовали отсортированный набор в виде двусвязного списка. В этом упражнении мы хотим сделать это снова, но на этот раз с помощью двоичного дерева поиска, поскольку это специализированная и эффективная структура данных для этой задачи. логарифмическую сложность для типичных операций, таких как вставка и удаление.

В этом упражнении вы должны реализовать данный BSTInterface в отдельном классе BinarySearchTree на основе его документации. Дерево двоичного поиска должно иметь возможность искать значения `Integer`.

Предупреждение: Не используйте внутренние методы из Java API, за исключением следующих классов и интерфейсов: `List<E>`, `ArrayList<E>`, `NoSuchElementException` и `String`. Вы также можете использовать Random для тестирования.

1. Создайте новый проект `09-BinarySearchTree`.
2. Внутренний класс TreeNode (узел дерева) должен иметь конструктор с параметром, который должен хранить переданное значение `int` в атрибуте значения `int value`. Кроме того, узел должен хранить ссылки на два своих дочерних узла (`left`, `right`) и родительский узел (`parent`).
3. Методы `insert()` и `exists()` должны быть реализованы рекурсивно. Вы уже знаете итеративные варианты из слайдов упражнений
4. Если вставляемый элемент уже существует в дереве, должно быть выброшено исключение ElementExistsException. Для этого вам нужно написать свой собственный класс `Exception`, наследуемый от java.lang.Exception. Составьте содержательное и со смыслом сообщение об ошибке!
5. Как вы реализуете `remove(int value)` - решать вам. Однако вы значительно упростите свою работу, если вы структурируете свой код в соответствии с тремя возможными случаями и создадите несколько вспомогательных методов:
   - Определение *количества детей (дочерних узлов)* узла для различения случаев (↝ `int numChildren(TreeNode node)`).
   - *Найдите* узел со значением, которое вы ищете, если таковой имеется (↝ `TreeNode find(TreeNode node, int value)`).
   - *Удалить* **узел**. Здесь имеет смысл реализовать большую часть логики. (↝ `void remove(TreeNode node)`).
   - *Замена* ссылки на узел в родительском узле (особый случай: корень). (↝ `void replaceInParent(TreeNode node, TreeNode newNode)`).

   Внимание: не забывайте согласовывать ссылки на родительские узлы!

6. Наконец, создайте основной метод для тестирования ваших реализаций. Протестируйте и сравните, что происходит, когда вы проходите по дереву в последовательности *inorder, preorder и postorder* ↝ Methoden `inOrderList`, `preOrderList` und `postOrderList`!).

   Примечание. Для этой задачи также доступна библиотека Java **TreeViever** для графического отображения дерева. Когда вы реализовали основы работы с деревом, вы можете интегрировать файл **tree-viewer.jar** в свой проект, а затем вызвать метод `TreeViewer.displayTree(BinarySearchTree tree)`.

   Удалите все варианты использования TreeViewer (включая импорт!) Или закомментируйте их перед отправкой. В противном случае EST не сможет обработать вашу заявку, и вы получите 0 баллов.

7. Сдайте классы **BinarySearchTree.java** и **ElementExistsException.java.**