**Лабораторная работа №2**

**«Двоичные деревья»**

**Задание №1.**

Измените программу tree.java (см. листинг 8.1) так, чтобы она создавала двоичное дерево по вводимой пользователем цепочке символов (например, A, B и т. д.). Каждая буква отображается в собственном узле. Дерево должно строиться так, чтобы все узлы, содержащие буквы, были листовыми. Родительские узлы могут содержать какой-нибудь символ, не являющийся буквой, например +. Проследите за тем, чтобы каждый родительский узел имел ровно двух потомков. Неважно, если дерево получится несбалансированным. Обратите внимание: созданное дерево не является деревом поиска; в нем не существует быстрого способа найти заданный узел. Дерево, которое у вас получится, выглядит именно так:

****

Например, можно начать с создания массива деревьев. (Группа несвязанных деревьев называется лесом.) Каждая буква, вводимая пользователем, сохраняется в узле. Узел помещается в дерево, корнем которого он становится. Далее все эти деревья, состоящие из одного узла, помещаются в массив. Начните с создания нового дерева с корнем + и двумя деревьями из одного узла в качестве потомков. Затем продолжайте включать в это дерево одноузловые деревья из массива. Не беспокойтесь, если дерево получится несбалансированным. Промежуточное дерево можно сохранить в массиве поверх дерева, содержимое которого уже было добавлено ранее.

Методы find(), insert() и delete(), относящиеся только к деревьям поиска, можно удалить. Оставьте метод displayTree() и методы обхода, потому что они работают с любыми двоичными деревьями.

Листинг 8.1. Программа tree.java

*// tree.java*

*// Работа с двоичным деревом*

*// Запуск программы: C>java TreeApp*

**import** java.io.\*;

**import** java.util.\*; *// Для использования класса Stack*

*////////////////////////////////////////////////////////////////*

**class** Node

{

**public** **int** iData; *// Данные, используемые в качестве ключа*

**public** **double** dData; *// Другие данные*

**public** Node leftChild; *// Левый потомок узла*

**public** Node rightChild; *// Правый потомок узла*

**public** **void** displayNode() *// Вывод узла*

{

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print('{');

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(iData);

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(", ");

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(dData);

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("} ");

}

} *// Конец класса Node*

*////////////////////////////////////////////////////////////////*

**class** Tree

{

**private** Node root; *// first node of tree*

*// -------------------------------------------------------------*

**public** Tree() *// Конструктор*

{ root = **null**; } *// Пока нет ни одного узла*

*// -------------------------------------------------------------*

**public** Node find(**int** key) *// Поиск узла с заданным ключом*

{ *// (предполагается, что дерево не пустое)*

Node current = root; *// Начать с корневого узла*

**while**(current.iData != key) *// Пока не найдено совпадение*

{

**if**(key < current.iData) *// Двигаться налево?*

current = current.leftChild;

**else** *// Или направо?*

current = current.rightChild;

**if**(current == **null**) *// Если потомка нет,*

**return** **null**; *// поиск завершился неудачей*

}

**return** current; *// Элемент найден*

}

*// -------------------------------------------------------------*

**public** **void** insert(**int** id, **double** dd)

{

Node newNode = **new** Node(); *// Создание нового узла*

newNode.iData = id; *// Вставка данных*

newNode.dData = dd;

**if**(root==**null**) *// Корневой узел не существует*

root = newNode;

**else** *// Корневой узел занят*

{

Node current = root; *// Начать с корневого узла*

Node parent;

**while**(**true**) *// (внутренний выход из цикла)*

{

parent = current;

**if**(id < current.iData) *// Двигаться налево?*

{

current = current.leftChild;

**if**(current == **null**) *// Если достигнут конец цепочки,*

{ *// вставить слева*

parent.leftChild = newNode;

**return**;

}

}

**else** *// Или направо?*

{

current = current.rightChild;

**if**(current == **null**) *// Если достигнут конец цепочки,*

{ *// вставить справа*

parent.rightChild = newNode;

**return**;

}

}

}

}

}

*// -------------------------------------------------------------*

**public** **boolean** delete(**int** key) *// Удаление узла с заданным ключом*

{ *// (предполагается, что дерево не пусто)*

Node current = root;

Node parent = root;

**boolean** isLeftChild = **true**;

**while**(current.iData != key) *// Поиск узла*

{

parent = current;

**if**(key < current.iData) *// Двигаться налево?*

{

isLeftChild = **true**;

current = current.leftChild;

}

**else** *// Или направо?*

{

isLeftChild = **false**;

current = current.rightChild;

}

**if**(current == **null**) *// Конец цепочки*

**return** **false**; *// Узел не найден*

}

*// Удаляемый узел найден*

*// Если узел не имеет потомков, он просто удаляется.*

**if**(current.leftChild==**null** &&

current.rightChild==**null**)

{

**if**(current == root) *// Если узел является корневым,*

root = **null**; *// дерево очищается*

**else** **if**(isLeftChild)

parent.leftChild = **null**; *// Узел отсоединяется*

**else** *// от родителя*

parent.rightChild = **null**;

}

*// Если нет правого потомка, узел заменяется левым поддеревом*

**else** **if**(current.rightChild==**null**)

**if**(current == root)

root = current.leftChild;

**else** **if**(isLeftChild)

parent.leftChild = current.leftChild;

**else**

parent.rightChild = current.leftChild;

*// Если нет левого потомка, узел заменяется правым поддеревом*

**else** **if**(current.leftChild==**null**)

**if**(current == root)

root = current.rightChild;

**else** **if**(isLeftChild)

parent.leftChild = current.rightChild;

**else**

parent.rightChild = current.rightChild;

**else** *// Два потомка, узел заменяется преемником*

{

*// Поиск преемника для удаляемого узла (current)*

Node successor = getSuccessor(current);

*// Родитель current связывается с посредником*

**if**(current == root)

root = successor;

**else** **if**(isLeftChild)

parent.leftChild = successor;

**else**

parent.rightChild = successor;

*// Преемник связывается с левым потомком current*

**return** **true**; *// Признак успешного завершения*

}

*// -------------------------------------------------------------*

*// Метод возвращает узел со следующим значением после delNode.*

*// Для этого он сначала переходит к правому потомку, а затем*

*// отслеживает цепочку левых потомков этого узла.*

**private** Node getSuccessor(Node delNode)

{

Node successorParent = delNode;

Node successor = delNode;

Node current = delNode.rightChild; *// Переход к правому потомку*

**while**(current != **null**) *// Пока остаются левые потомки*

{

successorParent = successor;

successor = current;

current = current.leftChild; *// Переход к левому потомку*

}

*// Если преемник не является*

**if**(successor != delNode.rightChild) *// правым потомком,*

{ *// создать связи между узлами*

successorParent.leftChild = successor.rightChild;

successor.rightChild = delNode.rightChild;

}

**return** successor;

}

*// -------------------------------------------------------------*

**public** **void** traverse(**int** traverseType)

{

**switch**(traverseType)

{

**case** 1: [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("**\n**Preorder traversal: ");

preOrder(root);

**break**;

**case** 2: [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("**\n**Inorder traversal: ");

inOrder(root);

**break**;

**case** 3: [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("**\n**Postorder traversal: ");

postOrder(root);

**break**;

}

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println();

}

*// -------------------------------------------------------------*

**private** **void** preOrder(Node localRoot)

{

**if**(localRoot != **null**)

{

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(localRoot.iData + " ");

preOrder(localRoot.leftChild);

preOrder(localRoot.rightChild);

}

}

*// -------------------------------------------------------------*

**private** **void** inOrder(Node localRoot)

{

**if**(localRoot != **null**)

{

inOrder(localRoot.leftChild);

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(localRoot.iData + " ");

inOrder(localRoot.rightChild);

}

}

*// -------------------------------------------------------------*

**private** **void** postOrder(Node localRoot)

{

**if**(localRoot != **null**)

{

postOrder(localRoot.leftChild);

postOrder(localRoot.rightChild);

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(localRoot.iData + " ");

}

}

*// -------------------------------------------------------------*

**public** **void** displayTree()

{

[Stack](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+stack) globalStack = **new** [Stack](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+stack)();

globalStack.push(root);

**int** nBlanks = 32;

**boolean** isRowEmpty = **false**;

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(

"......................................................");

**while**(isRowEmpty==**false**)

{

[Stack](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+stack) localStack = **new** [Stack](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+stack)();

isRowEmpty = **true**;

**for**(**int** j=0; j<nBlanks; j++)

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(' ');

**while**(globalStack.isEmpty()==**false**)

{

Node temp = (Node)globalStack.pop();

**if**(temp != **null**)

{

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(temp.iData);

localStack.push(temp.leftChild);

localStack.push(temp.rightChild);

**if**(temp.leftChild != **null** ||

temp.rightChild != **null**)

isRowEmpty = **false**;

}

**else**

{

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("--");

localStack.push(**null**);

localStack.push(**null**);

}

**for**(**int** j=0; j<nBlanks\*2-2; j++)

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print(' ');

}

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println();

nBlanks /= 2;

**while**(localStack.isEmpty()==**false**)

globalStack.push( localStack.pop() );

}

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(

"......................................................");

}

*// -------------------------------------------------------------*

} *// Конец класса Tree*

*////////////////////////////////////////////////////////////////*

**class** TreeApp

{

**public** **static** **void** main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)[] args) **throws** [IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception)

{

**int** value;

Tree theTree = **new** Tree();

theTree.insert(50, 1.5);

theTree.insert(25, 1.2);

theTree.insert(75, 1.7);

theTree.insert(12, 1.5);

theTree.insert(37, 1.2);

theTree.insert(43, 1.7);

theTree.insert(30, 1.5);

theTree.insert(33, 1.2);

theTree.insert(87, 1.7);

theTree.insert(93, 1.5);

theTree.insert(97, 1.5);

**while**(**true**)

{

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("Enter first letter of show, ");

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("insert, find, delete, or traverse: ");

**int** choice = getChar();

**switch**(choice)

{

**case** 's':

theTree.displayTree();

**break**;

**case** 'i':

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("Enter value to insert: ");

value = getInt();

theTree.insert(value, value + 0.9);

**break**;

**case** 'f':

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("Enter value to find: ");

value = getInt();

Node found = theTree.find(value);

**if**(found != **null**)

{

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("Found: ");

found.displayNode();

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.print("**\n**");

}

*// -------------------------------------------------------------*

**public** **static** [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) getString() **throws** [IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception)

{

[InputStreamReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+inputstreamreader) isr = **new** [InputStreamReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+inputstreamreader)([System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).in);

[BufferedReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bufferedreader) br = **new** [BufferedReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bufferedreader)(isr);

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) s = br.readLine();

**return** s;

}

*// -------------------------------------------------------------*

**public** **static** **char** getChar() **throws** [IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception)

{

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) s = getString();

**return** s.charAt(0);

}

*//-------------------------------------------------------------*

**public** **static** **int** getInt() **throws** [IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception)

{

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) s = getString();

**return** [Integer](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+integer).parseInt(s);

}

*// -------------------------------------------------------------*

} *// Конец класса TreeApp*

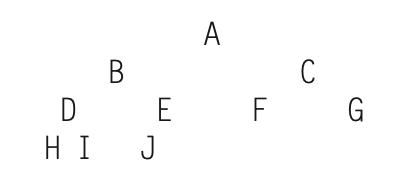
*////////////////////////////////////////////////////////////////*

**Задание №2.**

Усовершенствуйте программу из п. 8.1, чтобы она создавала сбалансированное дерево. Одно из возможных решений — проследить за тем, чтобы на нижнем уровне было как можно больше листовых узлов. Для начала создайте трехузловое дерево из каждой пары одноузловых деревьев, с новым корневым узлом +. В результате создается лес трехузловых деревьев. Затем каждая пара трехузловых деревьев объединяется для создания леса семиузловых деревьев. С ростом количества узлов в каждом дереве количество деревьев уменьшается, пока не останется только одно дерево.

**Задание №3.**

Снова начните с программы tree.java и постройте дерево из символов, вводимых пользователем. На этот раз должно создаваться полное дерево, то есть дерево, содержащее все возможные узлы на всех уровнях (кроме правого края последнего ряда). Символы должны быть упорядочены сверху вниз и слева направо в каждом ряду. Таким образом, строка ABCDEFGHIJ будет упорядочена в виде



Например, такое дерево можно строить сверху вниз (а не снизу вверх, как в двух предыдущих программных проектах). Начните с создания узла, который стане корнем итогового дерева. Если представить, что узлы нумеруются в порядке следования букв (корневому узлу присваивается номер 1), то у любого узла с номером n левый потомок имеет номер 2\*n, а правый — 2\*n + 1. В решении можно использовать рекурсивный метод, который создает двух потомков, а затем вызывает себя для каждого потомка. Узлы не обязательно создавать в порядке их размещения на дереве. Как и в предыдущих проектах, функции для работы с деревом поиска можно удалить из класса Tree.