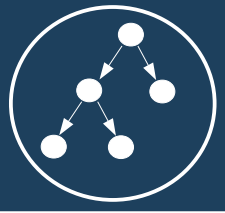


Data Structures and Algorithms

Структуры данных.
Бинарное дерево поиска

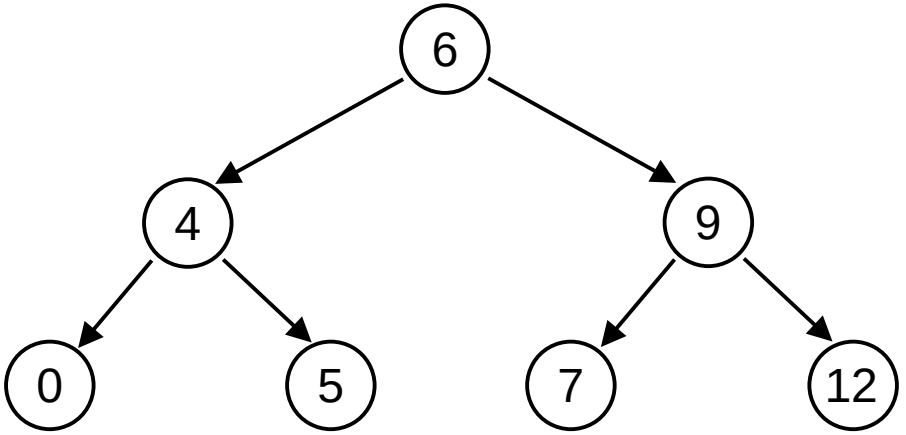
Бинарное дерево поиска

Бинарное дерево поиска (binary search tree, BST) — частный случай упорядоченного дерева. Его особенностями являются следующие:

1) Узел имеет не более двух дочерних узлов (отсюда название бинарное)
2) Оба поддерева (левое и правое) также являются бинарными деревьями поиска
3) У всех узлов левого поддерева произвольного узла значения ключей меньше, чем значение ключа самого узла
4) У всех узлов правого поддерева произвольного узла значения ключей больше, чем значение ключа самого узла

Из определения бинарного дерева следует, что данные в каждом узле должны обладать ключами, на которых определена операция сравнения.

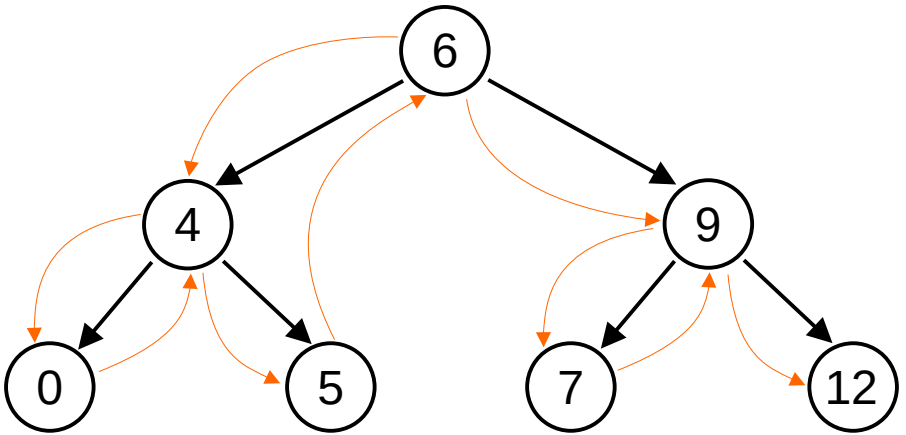
Графическое представление бинарного дерева



Способы обхода дерева

Так как дерево, это частный вид графа, то для обхода дерева реализуется теми же способами. Дерево можно обходить как в глубину, так и в ширину. В качестве стартовой вершины всегда выбирается корневой узел.

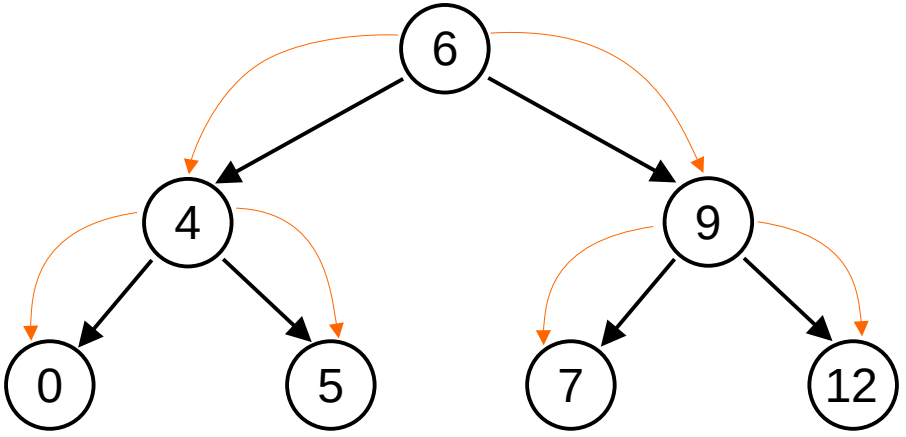
Обход дерева в глубину



При обходе дерева в глубину очень эффективным оказывается следующий рекурсивный подход. Сначала обрабатывается дочерняя вершина слева, текущая вершина, потом дочерняя вершина справа. При таком обходе ключи дерева будут перебираться в возрастающем порядке.

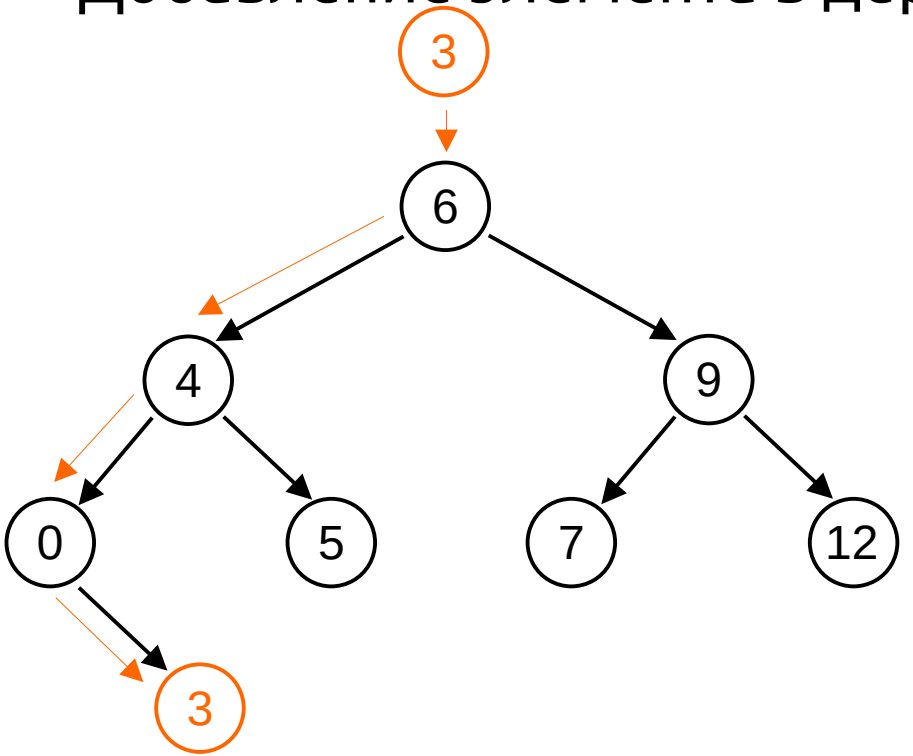
Существуют и другие порядки обхода: левая, правая, узел и узел, левая, правая.

Обход дерева в ширину



Обход дерева в ширину принципиально не отличается от обхода графа в ширину. Единственным отличием будет факт постоянства выбора стартового узла (корневой узел).

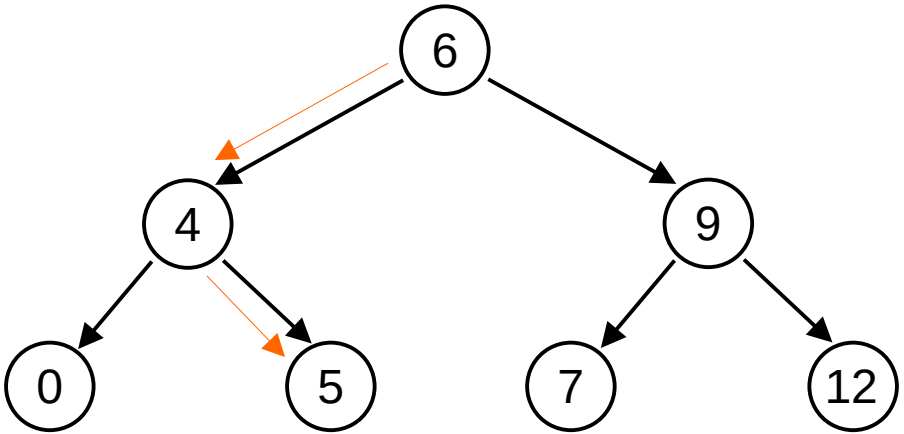
Добавление элемента в дерево



Проще всего реализовать добавление элемента используя следующий рекурсивный подход. Если узла по ссылке нет, то формируем новый узел, добавляем и заканчиваем. Если есть, то сравниваем ключ узла с ключом добавляемого. Если ключ добавляемого меньше то выполняем переход по левому ребру текущего, если больше по правому.

Поиск элемента в дереве

Ищем значение 5.



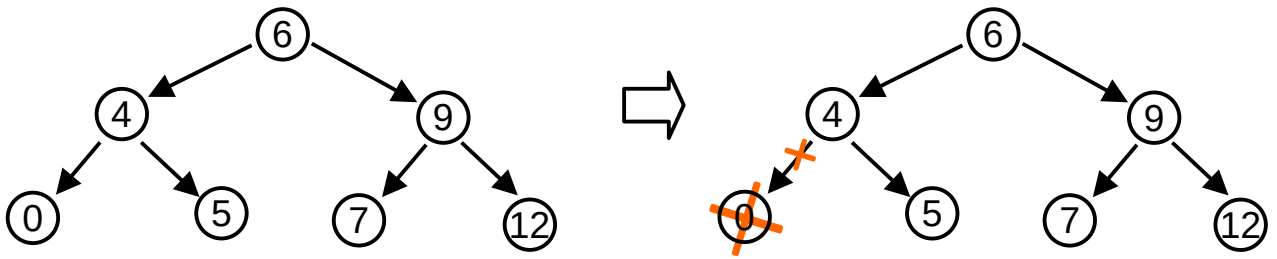
Наиболее оптимальным алгоритмом поиска является следующий рекурсивный подход — если узла по ссылке нет, то заканчиваем (поиск неудачен). В противном случае проверяем значение ключа текущего узла, если равен искомому то возвращаем данные (поиск успешен), если искомый ключ меньше ключа узла то переходим по левому ребру, в противном случае по правому ребру.

Удаление узла

При удалении узла следует рассматривать несколько случаев:

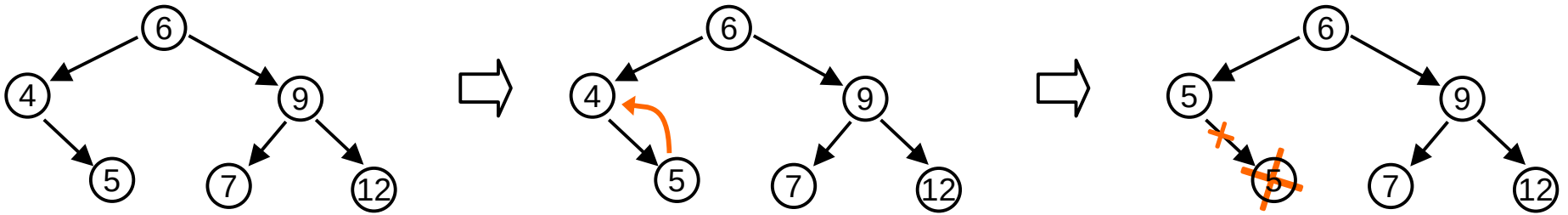
1) У удаляемого узла нет дочерних узлов. В таком случае просто удаляем узел (не забывая удалить ребро у родительского).
2) У удаляемого узла только один дочерний узел. Заменяем удаляемый узел на дочерний.
3) У удаляемого узла два дочерних узла. В правом поддереве удаляемого узла ищем узел с минимальным значением ключа. Заменяем данные удаляемого ключа на данные найденного. Найденный узел удаляем.

Удаление узла



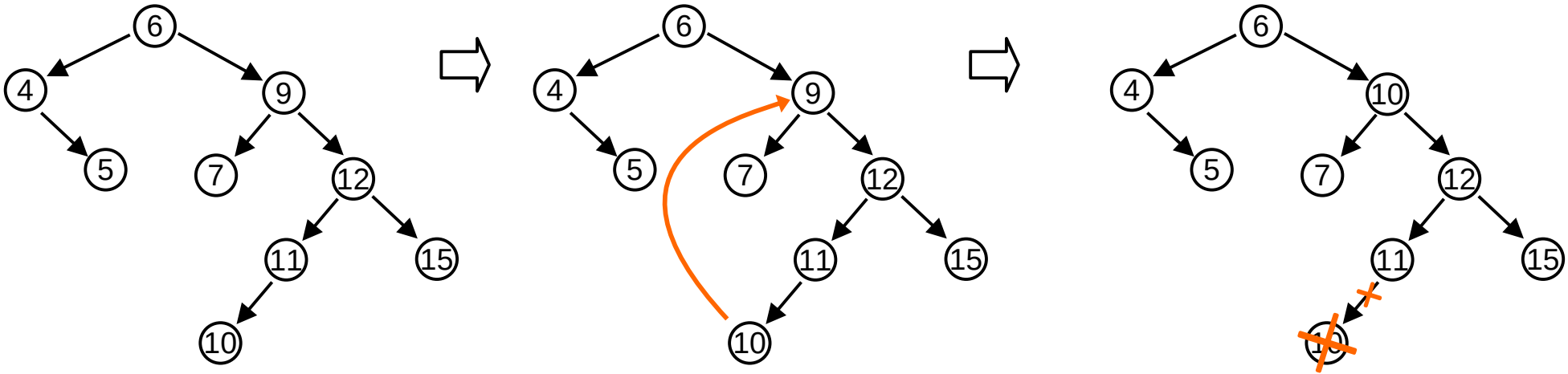
У удаляемого узла нет дочерних узлов. В таком случае просто удаляем узел (не забывая удалить ребро у родительского). Предположим, что нужно удалить узел с ключом **0**. У этого узла нет дочерних узлом поэтому просто удаляем его.

Удаление узла



У удаляемого узла только один дочерний узел. Заменяем удаляемый узел на дочерний. Дочерний удаляем. Предположим нужно удалить узел со значением **4**.

Удаление узла



У удаляемого узла два дочерних узла. В правом поддереве удаляемого узла ищем узел с минимальным значением ключа. Заменяем данные удаляемого ключа на данные найденного. Найденный узел удаляем. Предположим что удаляется узел с ключом **9**.

Представление узла

Наиболее простым способом представление узла является использование структуры где хранится данные, ключ и ссылку (или указатель) на левый и правый узел.

Node

- **key**
- **data**
- **left_node**
- **right_node**

Реализация на Python

Описание структуры узла и дерева

```
class BinaryTree:
    class Node:
        def __init__(self, key, data=None, left_node=None, right_node=None):
            self.key = key
            self.data = data
            self.left_node = left_node
            self.right_node = right_node

        def __str__(self):
            return str(self.key)

    def __init__(self):
        self.root = None
```

Метод добавления узла

```
def add(self, key, data=None):

    def _add(key, data, node):
        if node is None:
            return BinaryTree.Node(key, data)
        if node.key == key:
            node.data = data
            return node
        if node.key > key:
            node.left_node = _add(key, data, node.left_node)
        else:
            node.right_node = _add(key, data, node.right_node)
        return node

    self.root = _add(key, data, self.root)
```

Метод удаления узла

```
def remove(self, key):

    def find_smallest_node(node):
        if node.left_node is None:
            return node
        else:
            return find_smallest_node(node.left_node)

    def _remove(key, node):
        if node is None:
            return node
        if node.key == key:
            if node.left_node is None and node.right_node is None:
                return None
            if node.left_node is None:
                return node.right_node
            if node.right_node is None:
                return node.left_node
            if node.left_node is not None and node.right_node is not None:
                smallest_node = find_smallest_node(node.right_node)
                node.key = smallest_node.key
                node.data = smallest_node.data
                node.right_node = _remove(smallest_node.key, node.right_node)
                return node
        if node.key > key:
            node.left_node = _remove(key, node.left_node)
            return node
        if node.key < key:
            node.right_node = _remove(key, node.right_node)
            return node

    self.root = _remove(key, self.root)
```

Метод получения значения по ключу

```
def get(self, key):
    def _get(key, node):
        if node is None:
            return None
        if node.key == key:
            return node.data
        if node.key > key:
            return _get(key, node.left_node)
        else:
            return _get(key, node.right_node)
    return _get(key, self.root)
```

Метод получения размеров дерева

```
def size(self):
    def _size(node):
        if node is None:
            return 0
        return _size(node.left_node) + 1 + _size(node.right_node)
    return _size(self.root)
```

Реализация на Java

Описание представления узла и дерева

```
class BinaryTree {
    private class Node {
        int key;
        Object data;
        Node leftNode;
        Node rightNode;

        public Node(int key, Object data) {
            this.key = key;
            this.data = data;
        }

        @Override
        public String toString() {
            return "Node [key=" + key + ", data=" + data + "]";
        }
    }
    private Node root;
```

Методы добавления узла

```
public void addNode(int key, Object data) {
         root = addNodeRecursive(root, key, data);
    }
private Node addNodeRecursive(Node node, int key, Object data) {
    if (node == null) {
         return new Node(key, data);
    }
    if (key == node.key) {
         node.data = data;
         return node;
    }
    if (key < node.key) {
         node.leftNode = addNodeRecursive(node.leftNode, key, data);
    } else {
         node.rightNode = addNodeRecursive(node.rightNode, key, data);
    }
    return node;
}
```

Методы поиска по ключу узла

```
public Object findByKey(int key) {
            return findByKeyRecursive(root, key);
      }

private Object findByKeyRecursive(Node node, int key) {
      if (node == null) {
            return null;
      }
      if (node.key == key) {
            return node.data;
      }
      return (key < node.key) ? findByKeyRecursive(node.leftNode, key) : findByKeyRecursive(node.rightNode, key);
}
```

Методы удаления узла

```java
public void deleteNode(int key) {
        root = deleteNodeRewcursive(root, key);
}

private Node deleteNodeRewcursive(Node node, int key) {
        if (node == null) {
                return node;
        }
        if (key == node.key) {
                // node has no child node
                if (node.leftNode == null && node.rightNode == null) {
                        return null;
                }
                // node has exactly one child
                if (node.leftNode == null) {
                        return node.rightNode;
                }
                if (node.rightNode == null) {
                        return node.leftNode;
                }
                // node has two child node
                if (node.rightNode != null && node.leftNode != null) {
                        Node smallestNode = findSmallestValue(node.rightNode);
                        node.key = smallestNode.key;
                        node.data = smallestNode.data;
                        node.rightNode = deleteNodeRewcursive(node.rightNode, smallestNode.key);
                        return node;
                }
        }
        if (key < node.key) {
                node.leftNode = deleteNodeRewcursive(node.leftNode, key);
                return node;
        } else {
                node.rightNode = deleteNodeRewcursive(node.rightNode, key);
                return node;
        }
}

private Node findSmallestValue(Node node) {
        return node.leftNode == null ? node : findSmallestValue(node.leftNode);
}
```

Методы для получения размера дерева

```
public int size() {
    return sizeRecursive(root);
}

private int sizeRecursive(Node node) {
    if (node == null) {
        return 0;
    }
    return 1 + sizeRecursive(node.leftNode) + sizeRecursive(node.rightNode);
}
```

Реализация на Fortran

Описание узла и дерева

```
type Node
    integer::node_key
    character(len = 20)::node_data
    type(Node), pointer::node_left
    type(Node), pointer::node_right
end type Node

type BinaryTree
    type(Node), pointer::root

    contains
        procedure,pass::init
        procedure,pass::add
        procedure,pass::get
        procedure,pass::remove
        procedure,pass::show_tree
        procedure,pass::tree_size
        procedure,pass::destroy_tree
        procedure,nopass,private::r_add
        procedure,nopass,private::r_show_tree
        procedure,nopass,private::r_get
        procedure,nopass,private::r_remove
        procedure,nopass,private::r_tree_size
        procedure,nopass,private::find_smallest_node
        procedure,nopass,private::r_destroy_tree
end type BinaryTree
```

Процедуры для добавления узла

```
recursive subroutine r_add(node_key, node_data, add_node)
    integer, intent(in)::node_key
    character(len=*), intent(in)::node_data
    type(Node),pointer,intent(inout)::add_node
    if (.not. associated(add_node)) then
        allocate(add_node)
        add_node%node_key = node_key
        add_node%node_data = node_data
        add_node%node_left => null()
        add_node%node_right => null()
        return
    end if
    if (add_node%node_key == node_key) then
        add_node%node_data = node_data
        return
    end if
    if(add_node%node_key > node_key) then
        call r_add(node_key, node_data, add_node%node_left)
    else
        call r_add(node_key, node_data, add_node%node_right)
    end if
end subroutine r_add

recursive subroutine add(this, node_key, node_data)
    class(BinaryTree)::this
    integer, intent(in)::node_key
    character(len=*), intent(in)::node_data
    call r_add(node_key, node_data, this%root)
end subroutine add
```

Процедуры для получения значения по ключу

```
recursive subroutine r_get(node_key,r_node, result_data, operation_result)
    integer, intent(in)::node_key
    type(Node), pointer::r_node
    character(len=*), intent(inout)::result_data
    logical, intent(inout)::operation_result
    if (.not. associated(r_node)) then
        operation_result = .false.
        return
    end if
    if(r_node%node_key == node_key) then
        result_data = r_node%node_data
        operation_result = .true.
        return
    end if
    if(r_node%node_key > node_key) then
        call r_get(node_key, r_node%node_left, result_data, operation_result)
    else
        call r_get(node_key, r_node%node_right, result_data, operation_result)
    end if
end subroutine r_get

subroutine get(this, node_key, result_data, operation_result)
    class(BinaryTree)::this
    integer, intent(in)::node_key
    character(len=*), intent(inout)::result_data
    logical, intent(inout)::operation_result
    call r_get(node_key,this%root, result_data, operation_result)
end subroutine get
```

Рекурсивная процедура для удаления узла по ключу

```
recursive subroutine r_remove(node_key ,r_node, operation_result)
    integer, intent(in)::node_key
    type(Node), pointer::r_node, temp_node
    logical, intent(inout)::operation_result

    if(.not. associated(r_node)) then
        operation_result = .false.
        return
    end if

    if (r_node%node_key == node_key) then
        if(.not. associated (r_node%node_left) .and. .not. associated(r_node%node_right)) then
            deallocate(r_node)
            r_node => null()
            operation_result = .true.
            return
        end if

        if (.not.associated (r_node%node_left)) then
            temp_node => r_node%node_right
            deallocate(r_node)
            r_node => temp_node
            operation_result = .true.
            return
        end if

        if (.not.associated (r_node%node_right)) then
            temp_node => r_node%node_left
            deallocate(r_node)
            r_node => temp_node
            operation_result = .true.
            return
        end if

        if(associated (r_node%node_left) .and. associated(r_node%node_right)) then
            call find_smallest_node(r_node%node_right, temp_node)
            r_node%node_key = temp_node%node_key
            r_node%node_data = temp_node%node_data
            call r_remove(temp_node%node_key,r_node%node_right,operation_result)
            return
        end if
    end if

    if(r_node%node_key > node_key) then
        call r_remove(node_key ,r_node%node_left, operation_result)
    else
        call r_remove(node_key ,r_node%node_right, operation_result)
    end if
end subroutine r_remove
```

Процедуры для удаления узал по ключу

```
recursive subroutine find_smallest_node(r_node, temp_node)
    type(Node), pointer, intent(inout)::r_node, temp_node

    if(.not. associated(r_node%node_left)) then
        temp_node => r_node
        return
    else
        call find_smallest_node (r_node%node_left, temp_node)
    end if
end subroutine find_smallest_node


subroutine remove(this, node_key, operation_result)
    class(BinaryTree)::this
    integer, intent(in)::node_key
    logical, intent(inout)::operation_result

    call this%r_remove(node_key ,this%root, operation_result)

end subroutine remove
```

Процедура для получения размера дерева

```
recursive subroutine r_tree_size(r_node, r_size)
    type(Node), pointer::r_node
    integer, intent(inout)::r_size
    if (.not. associated(r_node)) then
        return
    else
        r_size = r_size + 1
        call r_tree_size(r_node%node_left, r_size)
        call r_tree_size(r_node%node_right, r_size)
    end if
end subroutine r_tree_size

subroutine tree_size(this,r_size)
    class(BinaryTree)::this
    integer, intent(inout)::r_size
    r_size = 0
    call r_tree_size(this%root,r_size)
end subroutine tree_size
```

Процедура для очистки дерева

```
recursive subroutine r_destroy_tree(r_node)
    type(Node), pointer, intent(inout)::r_node
    if(.not. associated (r_node)) then
        return
    end if
    call r_destroy_tree(r_node%node_left)
    call r_destroy_tree(r_node%node_right)
    if(.not. associated (r_node%node_left) .and. .not. associated(r_node%node_right)) then
        deallocate(r_node)
        r_node => null()
        return
    end if
end subroutine r_destroy_tree

subroutine destroy_tree(this)
    class(BinaryTree)::this
    call r_destroy_tree(this%root)
end subroutine destroy_tree
```

Список литературы

1) Джеймс А. Андерсон «Дискретная математика и комбинаторика». Издательский дом «Вильямс», 2004.
2) Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн // Алгоритмы: построение и анализ 3-е издание. — М.: «Вильямс», 2013. — С. 1328. ISBN 978-5-8459-1794-2
3) Роберт Седжвик, Кевин Уэйн «Алгоритмы на java 4-е издание» Пер. с англ. - М. : ООО "И.Д. Вильямс", 2013. ISBN 978-5-8459-1781-2.