Структура данных - дерево

Артамонов Ю.Н.

Международный университет
природы, общества и человека "Дубна"
филиал Котельники

28 апреля 2016 г.

Поиск элемента в отсортированном списке

```
def find_sort(a,L):
if L==[]:
    return False
elif (L[len(L)-1]<a) | (L[0]>a):
    return False
else:
    n = len(L)//2
    if L[n-1]<a:
        return find_sort(a,L[n:len(L)])
    elif L[n-1] == a:
        return True
    else:
        return find_sort(a,L[0:n])
```

Добавление элемента в отсортированный список

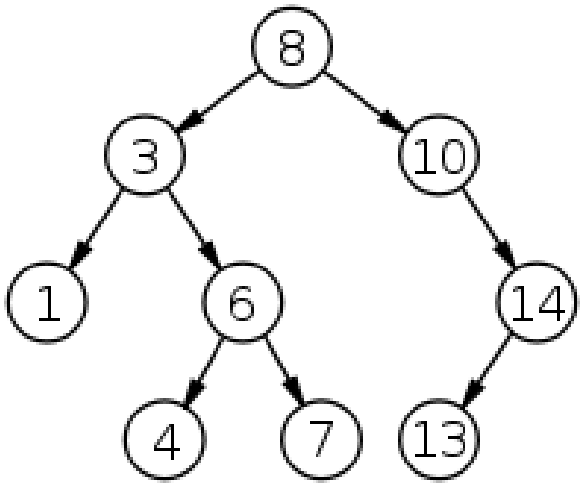
```
def add_sort(a,L):
    if L==[]:
        return [a]
    elif L[len(L)-1] < a:
        return L+[a]
    else:
        n = len(L)//2
        if L[n-1]<a:
            return L[0:n]+add_sort(a,L[n:len(L)])
        elif L[n-1] == a:
            return L[0:n]+[a]+L[n:len(L)]
        else:
            return add_sort(a,L[0:n])+L[n:len(L)]
```

Оценка вычислительной сложности

Каждая итерация уменьшает поик в 2 раза, поэтому, если последовательность длины $n = 2^x$, то потребуется всего x итераций.
В общем случае потребуется $x = log_2(n)$ операций.

Общее описание

- Дерево - это граф, который не содержит циклов
- Бинарное дерево - каждая вершина имеет не более двух детей
- Бинарное дерево поиска для последовательности: [8, 10, 3, 1, 14, 6, 13, 4, 7] (это альтернативный подход для ускорения поиска - отсортировать частично).



Основные операции с бинарным деревом поиска

- Формирование структуры узлов дерева
- Задание дерева
- Основные операции с деревом: добавление элемента, поиск элемента, удаление элемента, обход дерева, балансировка дерева

Формирование структуры узлов дерева

```
class node():
__parent = None
__left = None
__right = None
__info = None
def __init__(self, value):
    self.info = value
    self.__parent = None
    self.__left = None
    self.__right = None
```

Формирование структуры узлов дерева(продолжение)

```
    def get_parent(self):
        return self.__parent
    def get_left(self):
        return self.__left
    def get_right(self):
        return self.__right
    def get_info(self):
        return self.info
    def set_info(self,a):
        self.__info = a
```

Формирование структуры узлов дерева(продолжение)

```
def set_parent(self,m_node):
    if isinstance(m_node,node):
        self.__parent = m_node
def set_left(self,m_node):
    if isinstance(m_node,node):
        self.__left = m_node
def set_right(self,m_node):
    if isinstance(m_node,node):
        self.__right = m_node
```

Задание дерева

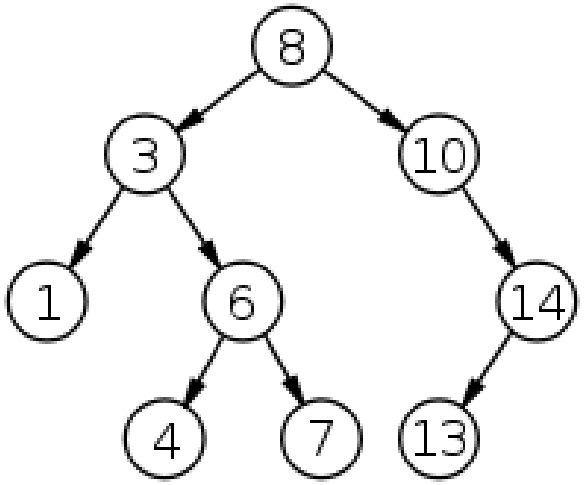
```
class tree():
__top = None

def __init__(self,value):
    self.__top = node(value)
    #return 'ok'

def get_top(self):
    return self.__top
```

Добавление элемента в дерево

```
def add_node(self, value,current_node):
    if current_node.get_info()>value:
        if current_node.get_left() == None:
            a = node(value)
            current_node.set_left(a)
            a.set_parent(current_node)
        else:
            current_node = current_node.get_left()
            self.add_node(value,current_node)
    else:
        if current_node.get_right() == None:
            a = node(value)
            current_node.set_right(a)
            a.set_parent(current_node)
        else:
            current_node = current_node.get_right()
            self.add_node(value,current_node)
def add_list(self,lst):
    for i in lst: self.add_node(i,self.get_top())
```

Визуализация дерева

Задание: выполнить сериализацию дерева:



В виде:
[8, [3, [1, [], []], [6, [4, [], []], [7, [], []]]], [10, [], [14, [13, [], []], []]]]

Визуализация дерева

```
    def serialization(self,current_node):
        if (current_node.get_left() == None) & (current_node.
get_right() == None):
            return [current_node.get_info(),[],[]]
        elif (current_node.get_left() == None) & (current_node.
get_right() != None):
            return [current_node.get_info(),[],self.serialization(
current_node.get_right())]
        elif (current_node.get_left() != None) & (current_node.
get_right() == None):
            return [current_node.get_info(),self.serialization(
current_node.get_left()),[]]
        else:
            return [current_node.get_info(),self.serialization(
current_node.get_left()), self.serialization(current_node.
get_right())]
```

Поиск элемента в дереве

Задание: выполнить поиск элемента в дереве

Поиск элемента в дереве

```
    def find(self,value,current_node):
        if current_node == None:
            return False
        if (current_node.get_left() == None) & (current_node.
get_right() == None):
            if value == current_node.get_info():
                return True
            else:
                return False
        elif current_node.get_info()==value:
            return True
        elif current_node.get_info()>value:
            return self.find(value,current_node.get_left())
        else:
            return self.find(value,current_node.get_right())
```

Подсчет количества вершин дерева

Задание: реализовать функцию подсчета узлов дерева

Подсчет узлов дерева

```
    def count_vertex(self,current_node):
        if (current_node.get_left() == None) & (current_node.
get_right() == None):
            return 1
        elif (current_node.get_left() != None) & (current_node.
get_right() == None):
            return 1+self.count_vertex(current_node.get_left())
        elif (current_node.get_left() == None) & (current_node.
get_right() != None):
            return 1+self.count_vertex(current_node.get_right())
        else:
            return 1+self.count_vertex(current_node.get_left())+self
.count_vertex(current_node.get_right())
```

Подсчет количества листьев дерева

Задание: реализовать функцию подсчета листьев дерева

Подсчет листьев дерева

```
    def count_leaves(self,current_node):
        if (current_node.get_left() == None) & (current_node.
get_right() == None):
            return 1
        elif (current_node.get_left() != None) & (current_node.
get_right() == None):
            return self.count_leaves(current_node.get_left())
        elif (current_node.get_left() == None) & (current_node.
get_right() != None):
            return self.count_leaves(current_node.get_right())
        else:
            return self.count_leaves(current_node.get_left())+self.
count_leaves(current_node.get_right())
```

Расчет глубины дерева

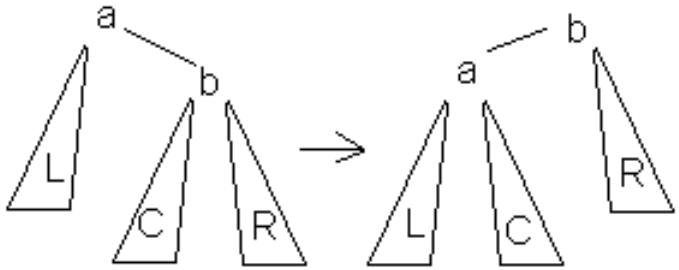
Задание: реализовать функцию определения глубины дерева

Подсчет глубины дерева

```
    def deep_tree(self,current_node):
        if (current_node.get_left() == None) & (current_node.
get_right() == None):
            return 1
        elif (current_node.get_left() != None) & (current_node.
get_right() == None):
            return 1+self.deep_tree(current_node.get_left())
        elif (current_node.get_left() == None) & (current_node.
get_right() != None):
            return 1+self.deep_tree(current_node.get_right())
        else:
            if self.deep_tree(current_node.get_right())>self.
deep_tree(current_node.get_left()):
                return 1+self.deep_tree(current_node.get_right())
            else:
                return 1+self.deep_tree(current_node.get_left())
```

Балансировка дерева

Всегда желательно, чтобы все пути в дереве от корня до листьев имели примерно одинаковую длину, то есть чтобы глубина и левого, и правого поддеревьев была примерно одинакова в любом узле. В противном случае теряется производительность. В вырожденном случае может оказаться, что все левое дерево пусто на каждом уровне, есть только правые деревья, и в таком случае дерево вырождается в список (идущий вправо). Поиск (а значит, и удаление и добавление) в таком дереве по скорости равен поиску в списке. Для балансировки дерева применяется операция «поворот дерева». Поворот налево выглядит так:



Задания

Задание: реализовать функции движения по дереву вверх, вправо, влево, используя понятие текущей вершины (current_node).
Задание: реализовать алгоритм балансировки дерева от заданной вершины поворотом влево (вправо).

Обходы дерева

- infix_traverse - обходят все дерево следуя порядку: левое поддерево, вершина, правое поддерево. В итоге получаются элементы, упорядоченные по возрастанию.
- postfix_traverse - обходят все дерево следуя порядку: правое поддерево, вершина, левое поддерево. В итоге получаются элементы, упорядоченные по убыванию
- prefix_traverse - обходят все дерево следуя порядку: вершина, левое поддерево, правое поддерево. В итоге получаются элементы, как они записаны в дереве

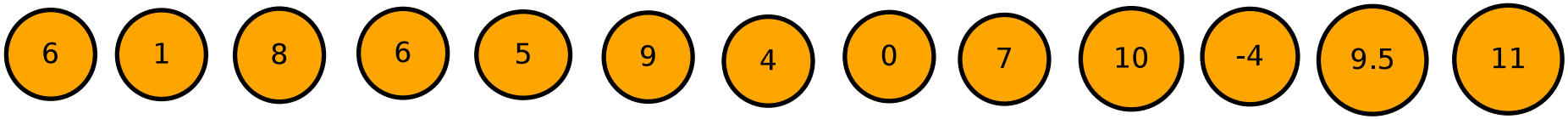
Обход дерева infix_traverse

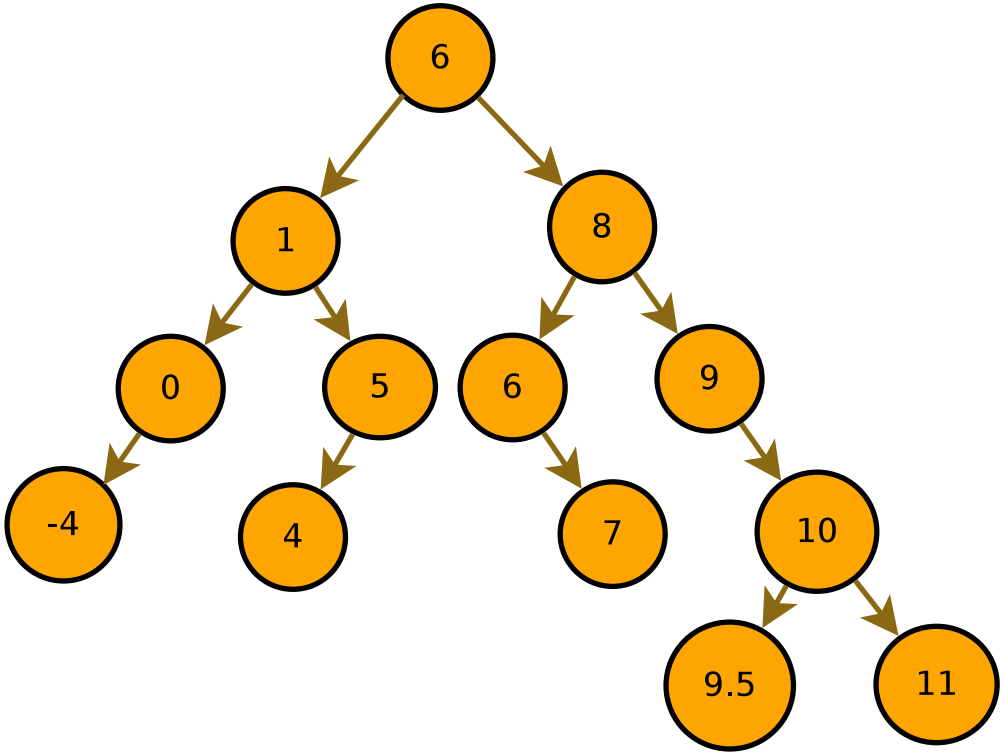
```
    def infix_traverse(self,  current_node):
        if current_node == None:
            return []
        elif (current_node.get_left()==[]) & (current_node.get_right
()==[]):
            return [current_node.get_info()]
        elif (current_node.get_left()!=[]) & (current_node.get_right
()==[]):
            return self.infix_traverse(current_node.get_left())+ [
current_node.get_info()]
        elif (current_node.get_left()==[]) & (current_node.get_right
()!=[]):
            return  [current_node.get_info()]+self.infix_traverse(
current_node.get_right())
        else:
            return  self.infix_traverse(current_node.get_left())+
                    [current_node.get_info()]+
                    self.infix_traverse(current_node.get_right())
```

Задания

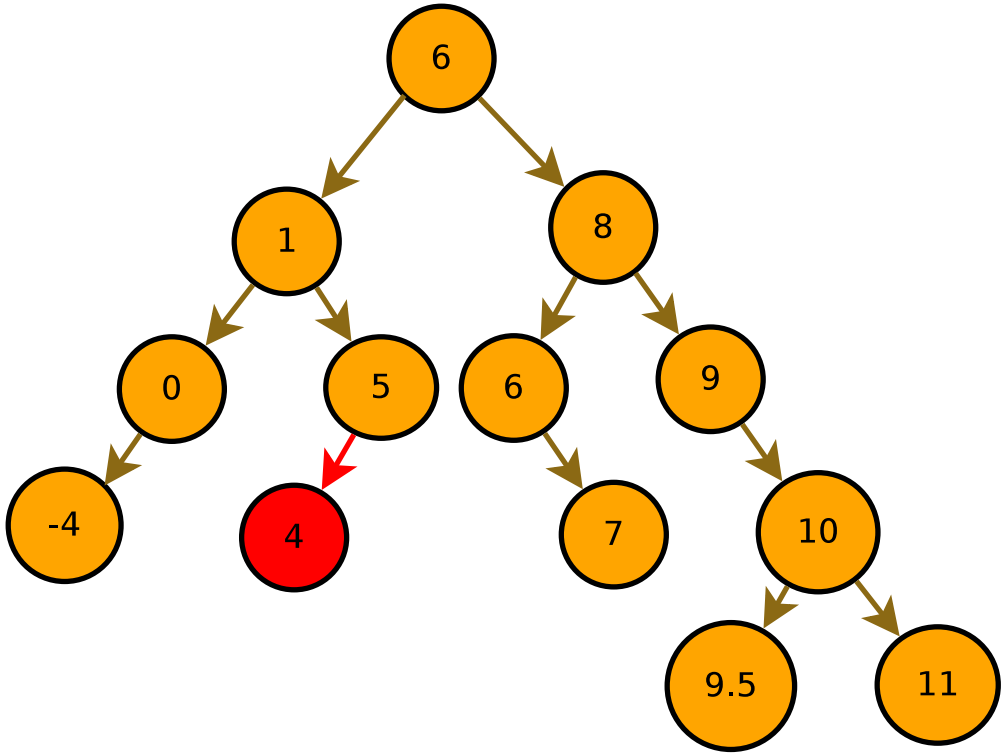
Задание: реализовать postfix_traverse.
Задание: реализовать prefix_traverse.

Удаление узла -исходное бинарное дерево

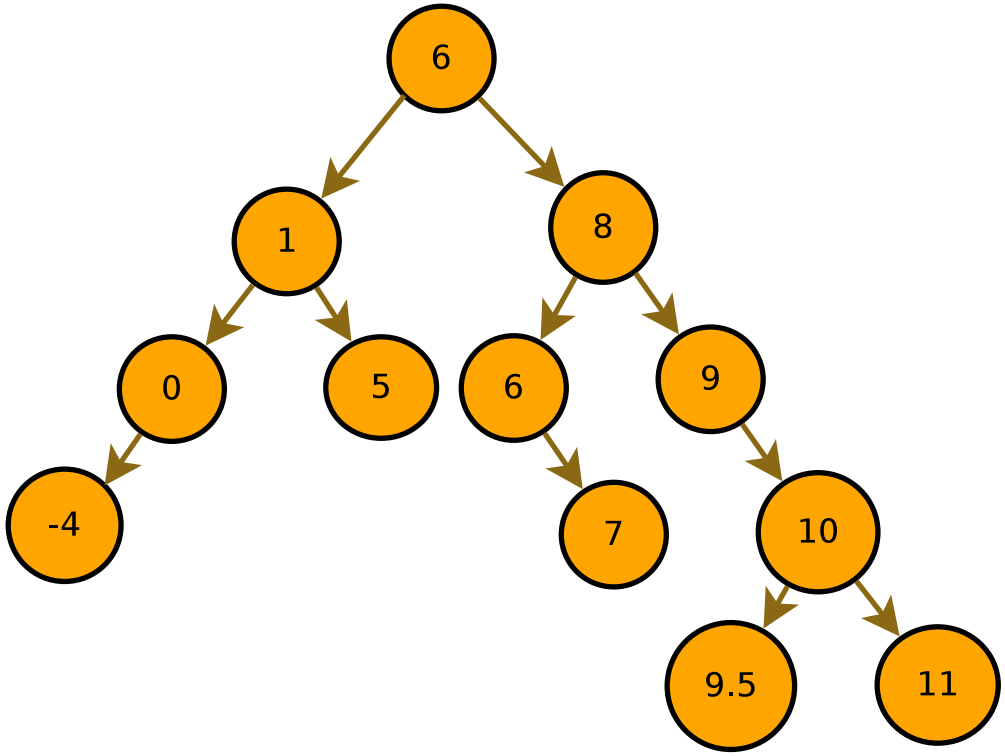




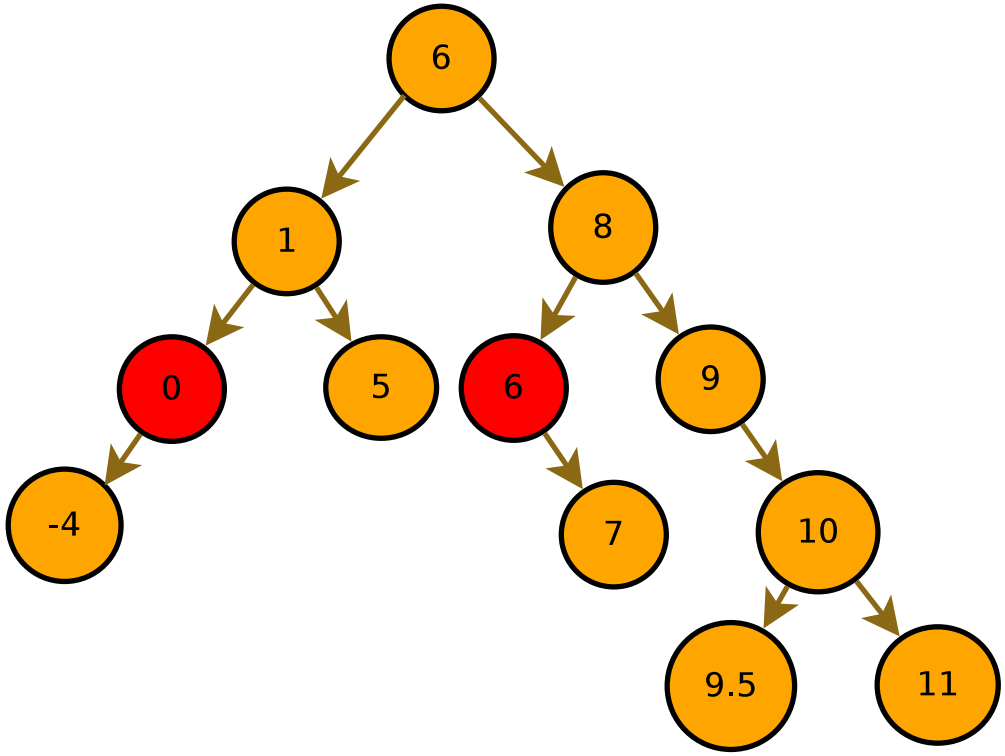
Удаление узла - случай, когда нет детей



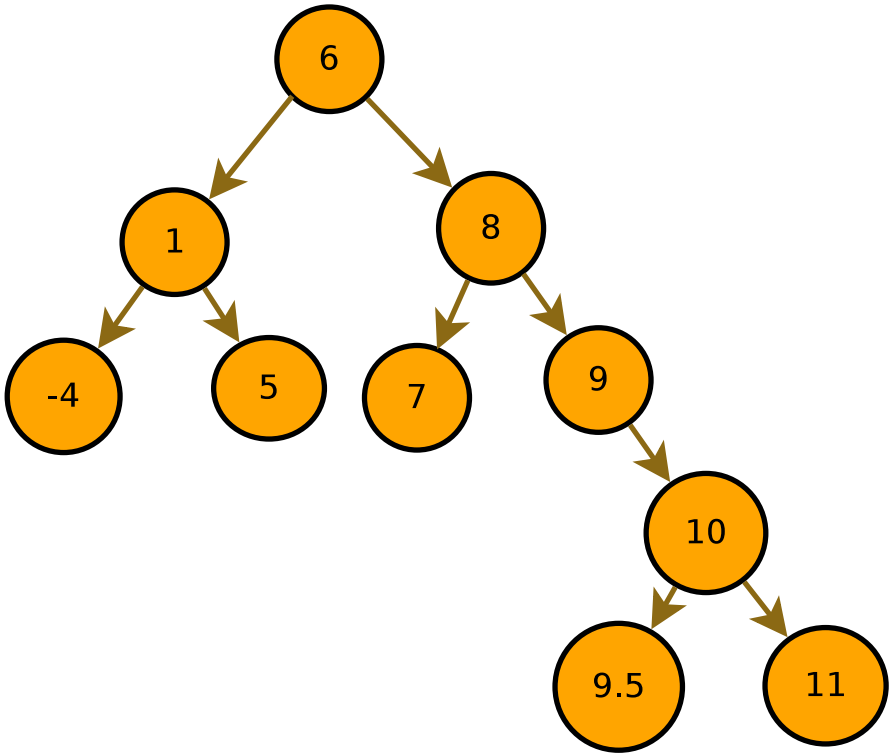
Удаление узла - результат удаления, когда нет детей



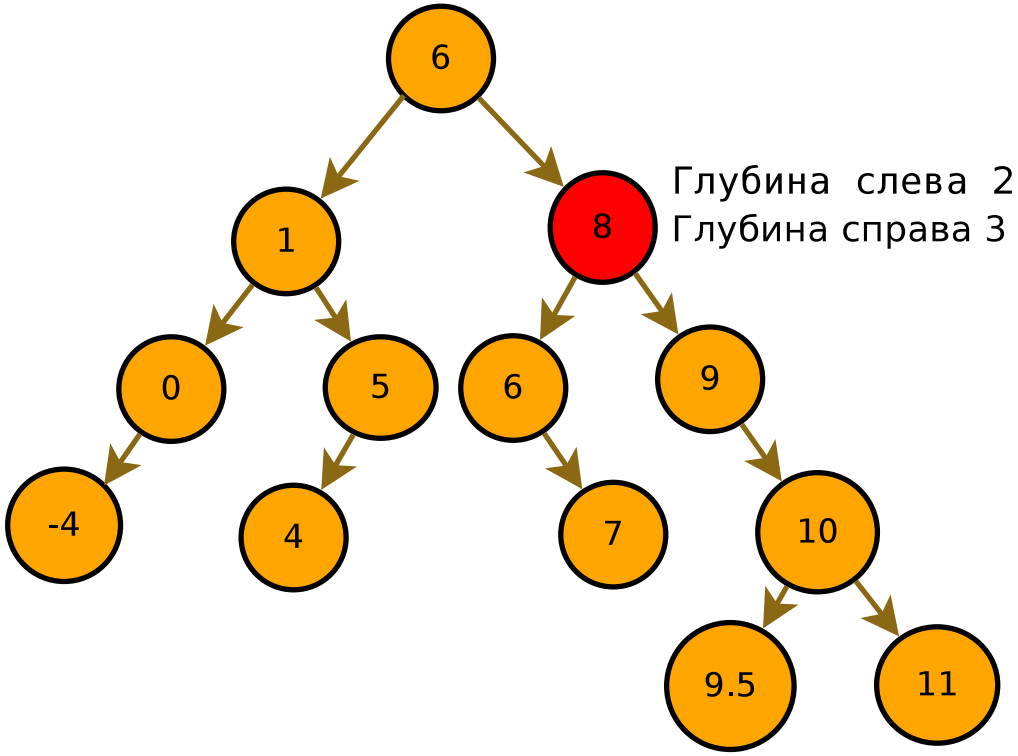
Удаление узла - случай, когда только один ребенок



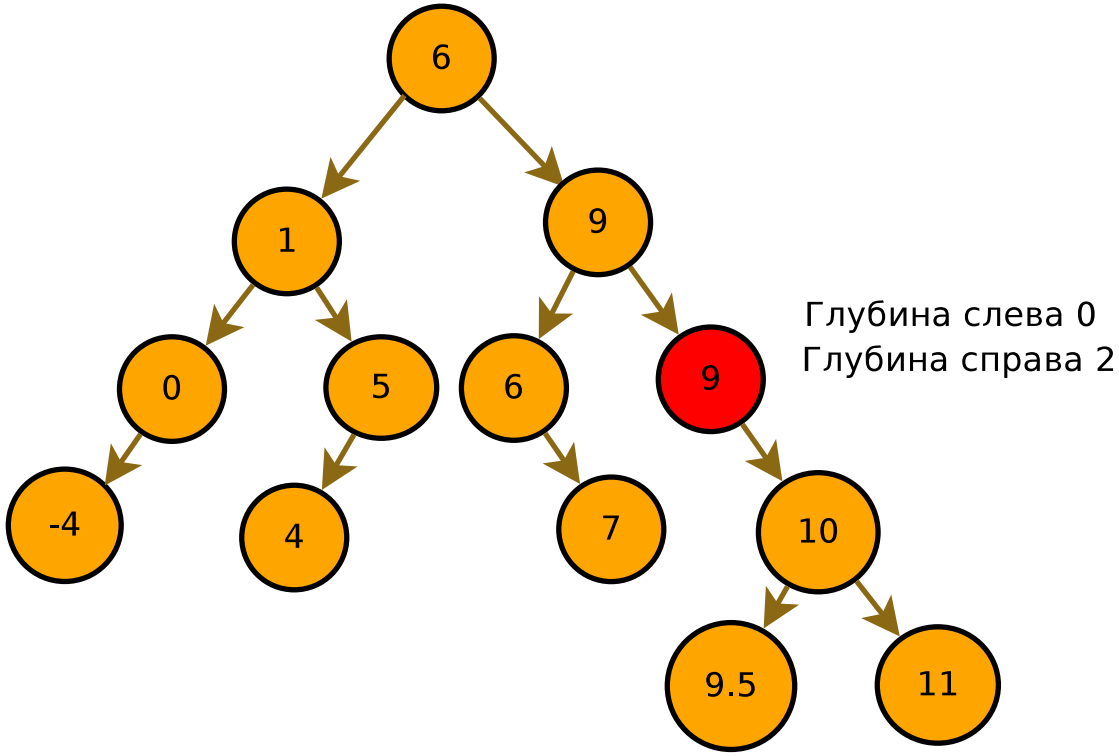
Удаление узла - результат удаления, когда только один ребенок



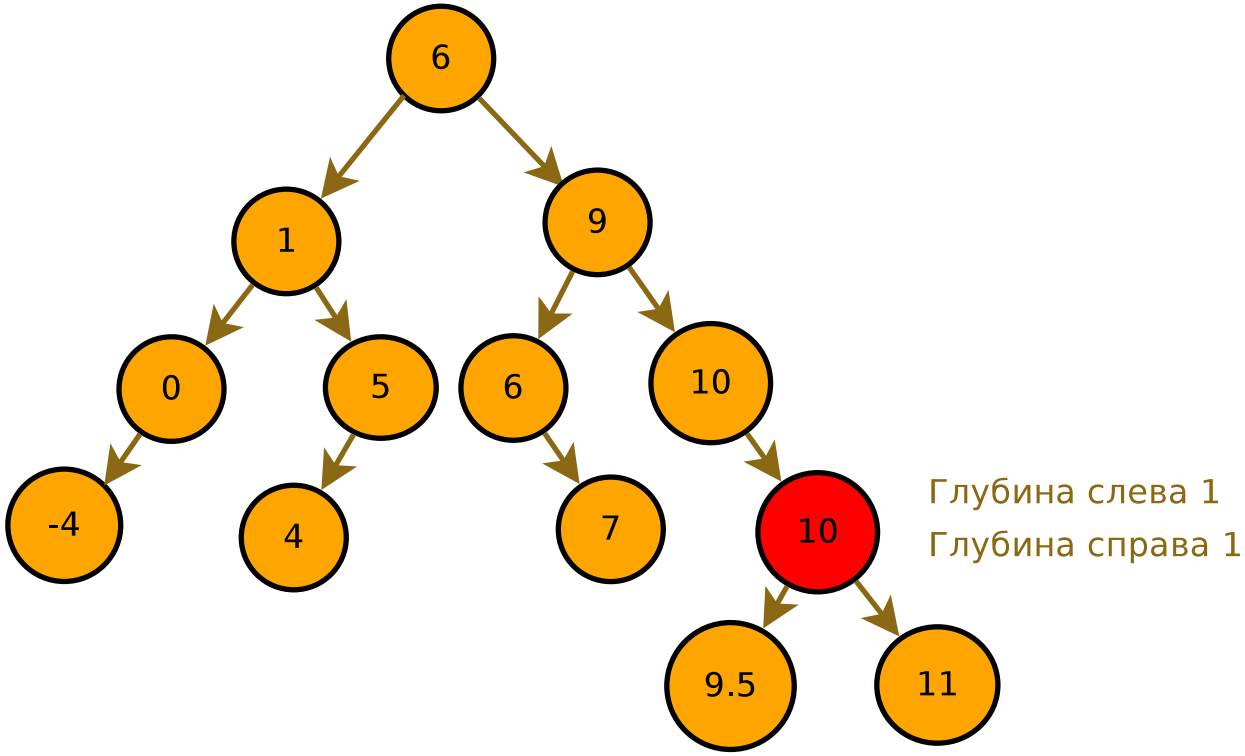
Удаление узла - случай удаления, когда оба ребенка



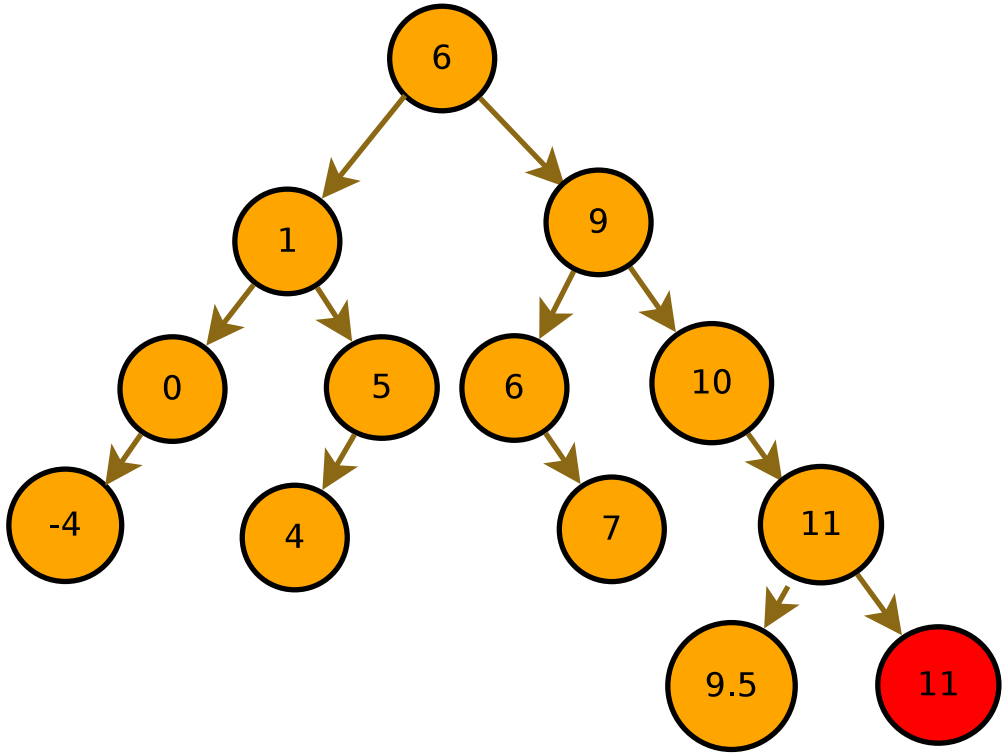
Удаление узла - случай удаления, когда оба ребенка



Удаление узла - случай удаления, когда оба ребенка

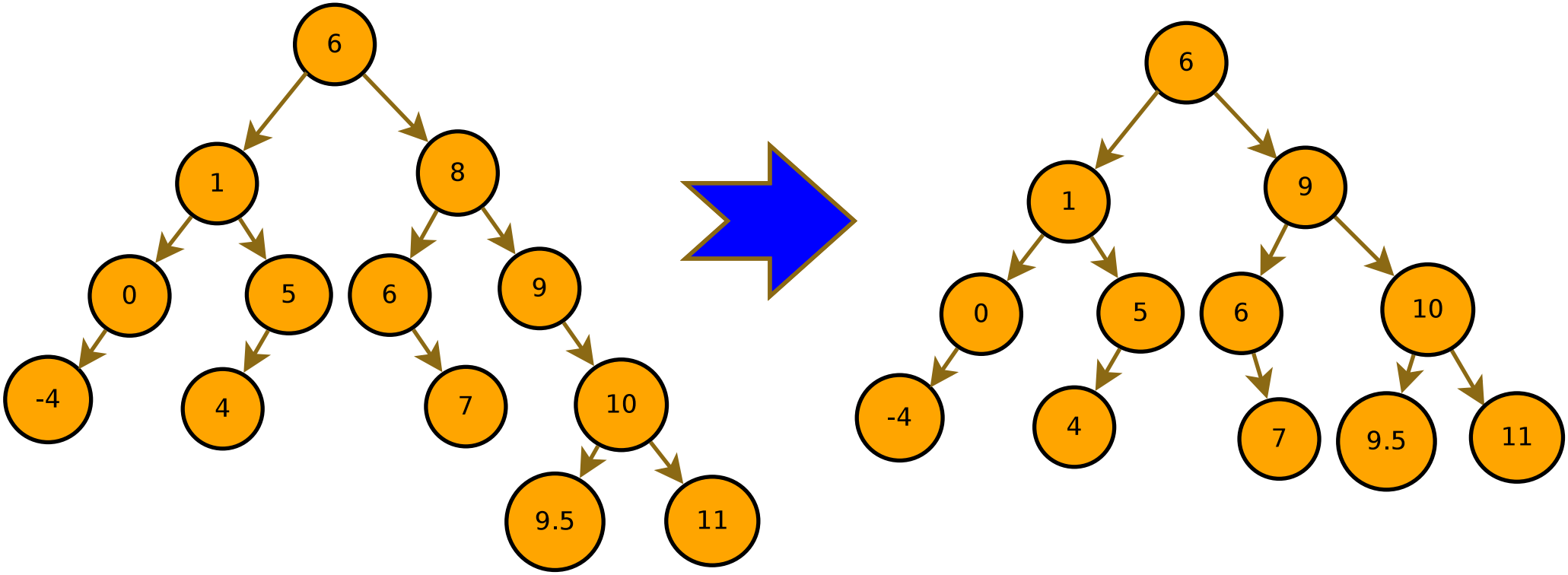


Удаление узла - случай удаления, когда оба ребенка



Глубина слева 0
Глубина справа 0

Удаление узла - результат удаления, когда оба ребенка



Задания

Задание: реализовать алгоритм удаления заданного узла
Задание: найти минимальный элемент дерева
Задание: найти максимальный элемент дерева
Задание: выполнить сортировку по возрастанию путем последовательного удаления минимальных элементов
Задание: выполнить сортировку по убыванию путем последовательного удаления максимальных элементов