САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №2

по курсу «Алгоритмы и структуры данных»

Тема: Двоичные деревья поиска

Вариант 22

Выполнил:

Федюкин М. В.

К3244

Проверила:

Артамонова В.Е.

Санкт-Петербург

2023 г.

**Содержание отчета**

Содержание отчета 1

Задачи по варианту 2

Обязательные задачи

Задача №2 3

Задача №7 4

Задача №16 5

Дополнительные задачи

Задача №1 7

Задача №3 19

Задача №5 11

Задача №6 13

Задача №8 15

Задача №9 17

Задача №10 20

Задача №14 22

Задача №15 23

Вывод 25

**Задачи по варианту**

Задание к лабораторной работе № 2-2: https://drive.google.com/file/d/1zMekiOINg47dnGyEPcBDrxUvtnxKymx6/view?usp=drive\_link

Мой вариант – 22.  
  
Обязательные задачи: 2, 7, 16.   
  
Дополнительные задачи: 1, 3, 5, 6, 8, 9, 10, 14, 15.

**Обязательные задачи**

**Задача 2**

Известно, что hi = (hi-1 + hi+1) / 2 – 1, откуда hi +1 = 2\*hi - hi-1 + 2. Получается, зная положение первых двух элементов, можно вычислить положение всех элементов. Найдем закономерность:

h0 = A

h1 = ?

h2 = 2\* h1 – h0 + 2

h3 = 3\* h1 – 2\* h0 + 6

h4 = 4\* h1 – 3\* h0 + 12

…

hn = n\* h1 – (n-1)\* h0 + n\*(n-1)

Нам нужно подобрать такое значение h1, чтобы значение полученной функции в точке n, равной количеству элементов, было минимальным и значения всех элементов оказались не ниже нуля. Искомое значение лежит в диапазоне от нуля до h0. Чтобы найти его, воспользуемся бинарным поиском.

*def* get\_h(n, h1, h0):  
 *return* n\*h1 - (n-1)\*h0 + n\*(n-1)  
  
  
*with* open('input.txt', 'r', encoding='utf-8') *as* file:  
 number, h0 = file.readline().split()  
number, h0 = int(number), float(h0)  
mn = 10\*\*9  
left, right = 0, h0  
mid = (left + right) / 2  
*while* mid != left *and* mid != right:  
 h1 = mid  
 flag = *True  
 for* n *in* range(2, number):  
 *if* get\_h(n, h1, h0) < 0:  
 flag = *False  
 break  
 if* flag:  
 right = mid  
 curr = get\_h(number-1, h1, h0)  
 *if* curr < mn:  
 mn = curr  
 *else*:  
 left = mid  
 mid = (left + right) / 2  
*with* open('output.txt', 'w', encoding='utf-8') *as* file:  
 print(mn, file=file)

**Задача 7**

Реализация этой задачи отличается от реализации шестой задачи единственным знаком равно.

*from* tree\_struct *import* Node, Tree  
*from* test *import* time\_memory  
*import* threading  
  
  
*def* main():  
 *with* open('input.txt', 'r') *as* f:  
 n = int(f.readline())  
 nodes = []  
 *for* i *in* range(n):  
 key, left, right = map(int, f.readline().split())  
 *if* left == -1:  
 left = *None  
 if* right == -1:  
 right = *None* nodes.append(Node(key, left, right))  
 *with* open('output.txt', 'w') *as* f:  
 tree = Tree(nodes=nodes)  
 tree.construct\_tree\_from\_indexes(index\_of\_root=0)  
 print("CORRECT" *if* tree.is\_correct\_bst(tree.root, with\_equal=*True*) *else* "INCORRECT", file=f)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 thread = threading.Thread(target=time\_memory(main))  
 thread.start()

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  2 1 2  2 -1 -1  3 -1 -1  3 -1 -1 | INCORRECT |
| 7  4 1 2  2 3 4  6 5 6  1 -1 -1  3 -1 -1  5 -1 -1  7 -1 -1 | CORRECT |
| 1  2147483647 -1 -1 | CORRECT |

**Задача 16**

Реализация с помощью AVL-дерева.

*from* tree\_struct\_AVL *import* AVL  
*from* test *import* time\_memory  
*import* threading  
  
  
*def* main():  
 tree = AVL()  
 *with* open('input.txt', 'r') *as* file:  
 *with* open('output.txt', 'w') *as* out\_file:  
 n = int(file.readline())  
 *for* \_ *in* range(n):  
 command, number = file.readline().split()  
 *if* command == '+1' *or* command == '1':  
 tree.insert(int(number))  
 *elif* command == '-1':  
 tree.delete(int(number))  
 *elif* command == '0':  
 number = int(number)  
 print(tree.get\_k\_elem(len(tree.nodes) - number + 1, tree.root).key, file=out\_file)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 thread = threading.Thread(target=time\_memory(main))  
 thread.start()

|  |  |
| --- | --- |
| **Input** | **Output** |
| 11  +1 5  +1 3  +1 7  0 1  0 2  0 3  -1 5  +1 10  0 1  0 2  0 3 | 7 5 3 10 7 3 |
| 9 +1 7 +1 8 +1 2 0 1 0 2 0 3 -1 2 +1 4 0 3 | 8 7 2 4 |
| 9 +1 798765 +1 8123456789 +1 2677 0 1 0 2 0 3 -1 234567 +1 412345678 0 3 | 8123456789 798765 2677 798765 |

**Дополнительные задачи**

**Задача 1**

Узлы, из которых состоит дерево, содержат поля «ключ», «родитель», «правый ребенок» и «левый ребенок». Само дерево содержит список узлов и указатель на корень дерева.

Изначально вводимые данные помещаю в массив, который передаю дереву как список узлов, имеющих только значения и индексы детей вместо самих детей. В процессе инициализации дерева (construct\_tree\_from\_indexes) вместо индексов детей помещаются сами дети, а также присваиваются родители. Гарантируется, что, обойдя всех детей, мы получим дерево без дыр.

*from* tree\_struct *import* Node, Tree  
*from* test *import* time\_memory  
*import* sys  
*import* threading  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 *with* open('input.txt', 'r') *as* file:  
 n = int(file.readline())  
 nodes = []  
 *for* i *in* range(n):  
 key, left, right = map(int, file.readline().split())  
 *if* left == -1:  
 left = *None  
 if* right == -1:  
 right = *None* nodes.append(Node(key, left, right))  
 tree = Tree(nodes=nodes)  
 tree.construct\_tree\_from\_indexes(index\_of\_root=0)  
 tree.in\_order\_traversal(tree.root)  
 print()  
 tree.pre\_order\_traversal(tree.root)  
 print()  
 tree.post\_order\_traversal(tree.root)  
 print()

*if* \_\_name\_\_ == '\_\_main\_\_':  
 thread = threading.Thread(target=time\_memory(main))  
 thread.start()

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  4 1 2  2 3 4  5 -1 -1  1 -1 -1  3 -1 -1 | 1 2 3 4 5  4 2 1 3 5  1 3 2 5 4 |
| 10  0 7 2  10 -1 -1  20 -1 6  30 8 9  40 3 -1  50 -1 -1  60 1 -1  70 5 4  80 -1 -1  90 -1 -1 | 50 70 80 30 90 40 0 20 10 60  0 70 50 40 30 80 90 20 60 10  50 80 90 30 40 70 10 60 20 0 |
| 6 2 -1 1 8 2 3 3 4 5 9 -1 -1 0 -1 -1 6 -1 -1 | 2 0 3 6 8 9  2 8 3 0 6 9  0 6 3 9 8 2 |

**Задача 3**

В данной задаче используется двоичное дерево поиска со следующим функционалом: поиск минимального элемента, большего k, вставка. Вставка опирается на функцию поиска элемента в дереве, которая реализована так, что она возвращает узел и статус – ok, если узел найден (соответственно, ничего вставлять не надо) или «возьми левого/правого ребенка» и узел, ребенком которого должен стать вставляемый элемент.

*from* tree\_struct *import* BST  
*from* test *import* time\_memory  
*import* threading  
  
  
*def* main():  
 bst = BST()  
 *with* open('input.txt', 'r', encoding='utf-8') *as* f:  
 text = f.readlines()  
 *with* open('output.txt', 'w', encoding='utf-8') *as* f:  
 *for* command *in* text:  
 ind, num = command.split()  
 num = int(num)  
 *if* ind == "+":  
 bst.insert(num)  
 *elif* ind == ">":  
 found = bst.next(num)  
 print(found.key *if* found *is not None else* 0, file=f)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 thread = threading.Thread(target=time\_memory(main))  
 thread.start()

|  |  |
| --- | --- |
| **Input** | **Output** |
| + 1  + 3  + 3  > 1  > 2  > 3  + 2  > 1 | 3 3 0 2 |
| + 7  > 9  + 4  + 13  > 8  + 1  + 6  > 1  + 10  + 15  > 13  + 5  > 4 | 0 13 4 15 5 |
| + 100000  > 6  > 1000  + 822  > 1  + 71  > 80 | 100000 100000 822 822 |

**Задача 5**

В данной задаче операции поиска и вставки реализованы так же, как в предыдущих.

*from* tree\_struct *import* BST  
*from* test *import* time\_memory  
*import* threading  
  
  
*def* main():  
 bst = BST()  
 *with* open('input.txt', 'r', encoding='utf-8') *as* f:  
 text = f.readlines()  
 *with* open('output.txt', 'w', encoding='utf-8') *as* f:  
 *for* command *in* text:  
 ind, num = command.split()  
 num = int(num)  
 *if* ind == "insert":  
 bst.insert(num)  
 *elif* ind == "delete":  
 bst.delete(num)  
 *elif* ind == "exists":  
 print("true" *if* bst.exists(num) *else* "false", file=f)  
 *elif* ind == "prev":  
 found = bst.prev(num)  
 print(found.key *if* found *is not None else* "none", file=f)  
 *elif* ind == "next":  
 found = bst.next(num)  
 print(found.key *if* found *is not None else* "none", file=f)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 thread = threading.Thread(target=time\_memory(main))  
 thread.start()

|  |  |
| --- | --- |
| **Input** | **Output** |
| insert 2  insert 5  insert 3  exists 2  exists 4  next 4  prev 4  delete 5  next 4  prev 4 | true false 5 3 none 3 |
| insert 7 insert 4 exists 0 prev 7 insert 15 next 4 insert 6 insert 1 next 1 exists 6 | false 4 7 4 true |
| insert 10102002 prev 999999999 insert 7182781 next 10 exists 3 insert 3 exists 3 prev 19 | 10102002 7182781 false true 3 |

**Задача 6**

Все значения левого поддерева должны быть меньше значения вершины, а все значения правого – больше. И это должно выполняться для каждого поддерева. Будем идти по дереву, для каждого шага сохраняя текущий минимум и максимум (если элемент меньше текущего минимума, а текущий минимум меньше предыдущего минимума, то элемент меньше предыдущего минимума и аналогично для максимума).

*from* tree\_struct *import* Node, Tree  
*from* test *import* time\_memory  
*import* threading  
  
  
*def* main():  
 *with* open('input.txt', 'r') *as* f:  
 n = int(f.readline())  
 nodes = []  
 *for* i *in* range(n):  
 key, left, right = map(int, f.readline().split())  
 *if* left == -1:  
 left = *None  
 if* right == -1:  
 right = *None* nodes.append(Node(key, left, right))  
 *with* open('output.txt', 'w') *as* f:  
 tree = Tree(nodes=nodes)  
 tree.construct\_tree\_from\_indexes(index\_of\_root=0)  
 print("CORRECT" *if* tree.is\_correct\_bst(tree.root) *else* "INCORRECT", file=f)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 thread = threading.Thread(target=time\_memory(main))  
 thread.start()

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  2 1 2  1 -1 -1  3 -1 -1  3 -1 -1 | CORRECT |
| 3  1 1 2  2 -1 -1  3 -1 -1 | INCORRECT |
| 7  4 1 2  2 3 4  6 5 6  1 -1 -1  3 -1 -1  5 -1 -1  7 -1 -1 | CORRECT |

**Задача 8**

В данной задаче к предыдущим полям узлов добавляется их высота (максимум из высот правого и левого поддеревьев + 1). Логика следующая: я пробегаюсь по списку имеющихся в дереве узлов. Если найденный узел – корень, то его высота пускай будет один. Иначе – высота каждого ребенка будет равна высоте родителя +1. Если у родителя нет высоты, надо сначала подняться до того, у которого есть, а потом спуститься.

*from* test *import* time\_memory  
*import* threading  
  
  
*class* Node:  
 *def* \_\_init\_\_(*self*, key=*None*, left=*None*, right=*None*, parent=*None*):  
 *self*.key = key  
 *self*.left = left  
 *self*.right = right  
 *self*.parent = parent  
 *self*.height = *None  
  
  
class* Tree:  
 *def* \_\_init\_\_(*self*, root=*None*, nodes=*None*):  
 *self*.root = root  
 *self*.nodes = nodes  
 *self*.height\_of\_tree = 0  
  
 *def* construct\_tree\_from\_indexes(*self*, index\_of\_root=0):  
 *for* index, node *in* enumerate(*self*.nodes):  
 *if* index == index\_of\_root:  
 node.height = 1  
 *self*.root = node  
 *if* node.left *is not None*:  
 *self*.nodes[node.left].parent = node  
 node.left = *self*.nodes[node.left]  
 *if* node.right *is not None*:  
 *self*.nodes[node.right].parent = node  
 node.right = *self*.nodes[node.right]  
  
 *def* mx\_height(*self*):  
 mx = 0  
 *for* node *in self*.nodes:  
 *if* node.parent *is None*:  
 node.height = 1  
 *if* node.height > mx:  
 mx = node.height  
 *else*:  
 *if* node.parent.height *is not None*:  
 node.height = node.parent.height + 1  
 *if* node.height > mx:  
 mx = node.height  
 *else*:  
 path = [node]  
 *while* node.parent.height *is None*:  
 node = node.parent  
 path.append(node)  
 *for* node *in* path:  
 node.height = node.parent.height  
 *if* node.height > mx:  
 mx = node.height  
 *return* mx  
  
  
*def* main():  
 *with* open('input.txt', 'r', encoding='utf-8') *as* f:  
 n = int(f.readline())  
 nodes = []  
 *for* i *in* range(n):  
 key, left, right = map(int, f.readline().split())  
 left -= 1  
 right -= 1  
 *if* left == -1:  
 left = *None  
 if* right == -1:  
 right = *None* nodes.append(Node(key, left, right))  
 tree = Tree(nodes=nodes)  
 tree.construct\_tree\_from\_indexes(index\_of\_root=0)  
 *with* open('output.txt', 'w', encoding='utf-8') *as* f:  
 print(tree.mx\_height(), file=f)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 thread = threading.Thread(target=time\_memory(main))  
 thread.start()

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  -2 0 2  8 4 3  9 0 0  3 6 5  6 0 0  0 0 0 | 4 |

**Задача 9**

К стандартным параметрам узла прибавляется размер. Так как гарантируется, что корень дерева никогда не будет удален, есть три случая: дерево пустое, тогда ничего делать не надо; в дереве нет искомого ключа, тогда тоже ничего делать не надо; и в дереве есть удаляемый ключ. Если ключ есть, необходимо уменьшить размер всех родительских узлов на размер удаляемого узла, а потом «отцепить» от дерева удаляемый узел.

*from* test *import* time\_memory  
*import* threading  
  
  
*class* Node:  
 *def* \_\_init\_\_(*self*, key=*None*, left=*None*, right=*None*, parent=*None*):  
 *self*.key = key  
 *self*.left = left  
 *self*.right = right  
 *self*.parent = parent  
 *self*.size = 1  
  
  
*class* Tree:  
 *def* \_\_init\_\_(*self*, root=*None*, nodes=*None*):  
 *self*.root = root  
 *self*.nodes = nodes  
  
 *def* construct\_tree\_from\_indexes(*self*, index\_of\_root=0):  
 *for* index, node *in* enumerate(*self*.nodes):  
 *if* index == index\_of\_root:  
 *self*.root = node  
 *if* node.left *is not None*:  
 *self*.nodes[node.left].parent = node  
 node.left = *self*.nodes[node.left]  
 *if* node.right *is not None*:  
 *self*.nodes[node.right].parent = node  
 node.right = *self*.nodes[node.right]  
 *self*.get\_sizes()  
  
 *def* get\_sizes(*self*):  
 *for* node *in self*.nodes:  
 *while* node.parent *is not None*:  
 node.parent.size += 1  
 node = node.parent  
  
 *def* find(*self*, k, root=*None*):  
 *if* root *is None or* root.key == k:  
 *return* root, "ok"  
 *elif* k < root.key:  
 *if* root.left *is None*:  
 *return* root, "take\_left\_kid"  
 *return self*.find(k, root.left)  
 *else*:  
 *if* root.right *is None*:  
 *return* root, "take\_right\_kid"  
 *return self*.find(k, root.right)  
  
 *def* delete\_altogether(*self*, key):  
 found = *self*.find(key, root=*self*.root)  
 *if* found[1] == "ok" *and* found[0] *is not None*:  
 node = found[0]  
 *if* node.parent *is not None*:  
 *if* node.parent.left == node:  
 node.parent.left = *None  
 else*:  
 node.parent.right = *None* curr\_node = node  
 *while* curr\_node.parent *is not None*:  
 curr\_node.parent.size -= node.size  
 curr\_node = curr\_node.parent  
 node.parent = *None  
 return self*.root.size  
  
  
*def* main():  
 *with* open('input.txt', 'r') *as* file:  
 n = int(file.readline())  
 nodes = []  
 *for* i *in* range(n):  
 key, left, right = map(int, file.readline().split())  
 left, right = left-1, right-1  
 *if* left == -1:  
 left = *None  
 if* right == -1:  
 right = *None* nodes.append(Node(key, left, right))  
 tree = Tree(nodes=nodes)  
 tree.construct\_tree\_from\_indexes(index\_of\_root=0)  
 m = int(file.readline())  
 to\_del = map(int, file.readline().split())  
 *with* open('output.txt', 'w') *as* file:  
 *for* key *in* to\_del:  
 print(tree.delete\_altogether(key), file=file)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 thread = threading.Thread(target=time\_memory(main))  
 thread.start()

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  -2 0 2  8 4 3  9 0 0  3 6 5  6 0 0  0 0 0  4  6 9 7 8 | 5  4  4  1 |

**Задача 10**

Решение этой задачи совпадает с 6-й с парой правок.

*from* tree\_struct *import* Node, Tree  
*from* test *import* time\_memory  
*import* threading  
  
  
*def* main():  
 *with* open('input.txt', 'r') *as* f:  
 n = int(f.readline())  
 nodes = []  
 *for* i *in* range(n):  
 key, left, right = map(int, f.readline().split())  
 left, right = left - 1, right - 1  
 *if* left == -1:  
 left = *None  
 if* right == -1:  
 right = *None* nodes.append(Node(key, left, right))  
 *with* open('output.txt', 'w') *as* f:  
 tree = Tree(nodes=nodes)  
 tree.construct\_tree\_from\_indexes(index\_of\_root=0)  
 print("YES" *if* tree.is\_correct\_bst(tree.root) *else* "NO", file=f)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 thread = threading.Thread(target=time\_memory(main))  
 thread.start()

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  -2 0 2  8 4 3  9 0 0  3 6 5  6 0 0  0 0 0 | YES |
| 0 | YES |
| 3  NO  5 2 3  6 0 0  4 0 0 | NO |

**Задача 11**

Реализация данной задачи опирается на 14 (вставка) и 15 (удаление).

*from* tree\_struct\_AVL *import* AVL  
*from* test *import* time\_memory  
*import* threading  
  
  
*def* main():  
 avl = AVL()  
 *with* open('input.txt', 'r', encoding='utf-8') *as* f:  
 text = f.readlines()  
 *with* open('output.txt', 'w', encoding='utf-8') *as* f:  
 *for* command *in* text:  
 ind, num = command.split()  
 num = int(num)  
 *if* ind == "insert":  
 avl.insert(num)  
 *elif* ind == "delete":  
 avl.delete(num)  
 *elif* ind == "exists":  
 print("true" *if* avl.exists(num) *else* "false", file=f)  
 *elif* ind == "prev":  
 found = avl.prev(num)  
 print(found.key *if* found *is not None else* "none", file=f)  
 *elif* ind == "next":  
 found = avl.next(num)  
 print(found.key *if* found *is not None else* "none", file=f)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 thread = threading.Thread(target=time\_memory(main))  
 thread.start()

|  |  |
| --- | --- |
| **Input** | **Output** |
| insert 2  insert 5  insert 3  exists 2  exists 4  next 4  prev 4  delete 5  next 4  prev 4 | true  false  5  3  none  3 |

**Задача 14**

*from* tree\_struct\_AVL *import* Node, AVL  
*from* test *import* time\_memory  
*import* threading  
  
  
*def* main():  
 *with* open('input.txt', 'r', encoding='utf-8') *as* f:  
 n = int(f.readline())  
 nodes = []  
 *for* i *in* range(n):  
 key, left, right = map(int, f.readline().split())  
 left -= 1  
 right -= 1  
 *if* left == -1:  
 left = *None  
 if* right == -1:  
 right = *None* nodes.append(Node(key, left, right))  
 to\_insert = int(f.readline())  
 tree = AVL(nodes=nodes)  
 tree.construct\_tree\_from\_indexes()  
 tree.insert(to\_insert)  
 dictionary = tree.prepare()  
 *with* open('output.txt', 'w', encoding='utf-8') *as* f:  
 print(len(tree.nodes), file=f)  
 *for* i, node *in* enumerate(tree.levels):  
 left = node.left.key *if* node.left *is not None else None* right = node.right.key *if* node.right *is not None else None* left\_i = dictionary[left] *if* left *is not None else* 0  
 right\_i = dictionary[right] *if* right *is not None else* 0  
 print(node.key, left\_i, right\_i, file=f)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 thread = threading.Thread(target=time\_memory(main))  
 thread.start()

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  3 0 2  4 0 0  5 | 3  4 2 3  3 0 0  5 0 0 |

**Задача 15**

Операция удаления отличается от операции удаления из обычного дерева тем же кодом, что указан для 14 задачи. Единственное, в зависимости от того, какого типа удаление мы делаем, будет изменяться то, какой узел нужно брать для дальнейшей ребалонсировки.

*from* tree\_struct\_AVL *import* Node, AVL  
*from* test *import* time\_memory  
*import* threading  
  
  
*def* main():  
 *with* open('input.txt', 'r', encoding='utf-8') *as* f:  
 n = int(f.readline())  
 nodes = []  
 *for* i *in* range(n):  
 key, left, right = map(int, f.readline().split())  
 left -= 1  
 right -= 1  
 *if* left == -1:  
 left = *None  
 if* right == -1:  
 right = *None* nodes.append(Node(key, left, right))  
 to\_delete = int(f.readline())  
 tree = AVL(nodes=nodes)  
 tree.construct\_tree\_from\_indexes()  
 tree.delete(to\_delete)  
 dictionary = tree.prepare()  
 *with* open('output.txt', 'w', encoding='utf-8') *as* f:  
 print(len(tree.nodes), file=f)  
 *for* i, node *in* enumerate(tree.levels):  
 left = node.left.key *if* node.left *is not None else None* right = node.right.key *if* node.right *is not None else None* left\_i = dictionary[left] *if* left *is not None else* 0  
 right\_i = dictionary[right] *if* right *is not None else* 0  
 print(node.key, left\_i, right\_i, file=f)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 thread = threading.Thread(target=time\_memory(main))  
 thread.start()

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  4 2 3  3 0 0  5 0 0  4 | 2  3 0 2  5 0 0 |

**Вывод**

Мне очень нравятся деревья. Это интересная структура данных. Её интересно реализовывать и работать с ней в дальнейшем. Создавать свои (условно) собственные ячейки в большой работующей системе. Чувствую себя прям-таки программистом начинающим. Хоть и не без ужасных затупов. И не без помощи и гайдов (а как же), я справился с работой после большого перерыва.