

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №2.2
по курсу «Алгоритмы и структуры данных»
Тема: Двоичные деревья поиска
Вариант 2

Выполнил:
Вилис З.М. (фамилия имя)
К3144 (номер группы)

Проверила:
Артамонова В.Е.

Санкт-Петербург
2023 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задание №1. Обход двоичного дерева[N баллов]	3
Задание №7. Оpozнание двоичного дерева поиска, усложненная версия[N баллов]	7
Задание №16. Задача K-ый максимум[N баллов]	11
Задание №6. Оpozнание двоичного дерева поиска[N баллов]	14
Дополнительные задачи	17
Задание №11. Сбалансированное двоичное дерево поиска[N баллов]	17
Вывод по лабораторной работе	18

Ошибка! Закладка не определена.

Задачи по варианту

Задача №2. Гирлянда [N баллов]

Текст задачи. Гирлянда состоит из n лампочек на общем проводе. Один её конец закреплён на заданной высоте A мм ($h_1 = A$).

Благодаря силе тяжести гирлянда прогибается: высота каждой неконцевой лампы на 1 мм меньше, чем средняя высота

ближайших соседей ($h_i =$

$$h_{i-1} + h_{i+1}$$

2

– 1 для $1 < i < N$).

Требуется найти минимальное значение высоты второго конца B ($B = h_n$), такое что для любого $\epsilon > 0$ при высоте

второго конца $B + \epsilon$ для всех лампочек выполняется условие $h_i > 0$. Обратите внимание на то, что при данном

значении высоты либо ровно одна, либо две соседних лампочки будут иметь нулевую высоту.

Подсказка: для решения этой задачи можно использовать двоичный поиск.

Листинг кода.

```
def is_valid(n, A, B):
    heights = [0] * n
    heights[0] = A
    heights[-1] = B

    for i in range(1, n - 1):
        heights[i] = (heights[i - 1] + heights[i + 1]) / 2 - 1

    zero_count = sum(1 for h in heights if h <= 0)

    return zero_count <= 2

def find_min_b(n, A):
    left, right = 0, A + n
    best_b = right

    while right - left > 1e-7:
        mid = (left + right) / 2
        if is_valid(n, A, mid):
            best_b = mid
            right = mid
        else:
            left = mid

    return best_b
```

```

with open('input.txt') as f:
    n, A = map(float, f.readline().strip().split())
    n = int(n)

result = find_min_b(n, A)

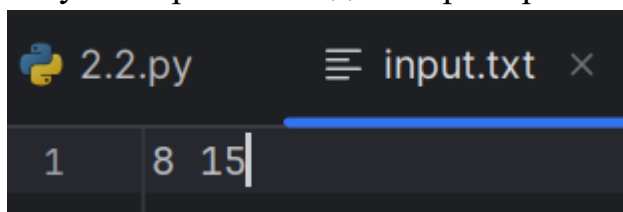
```

Текстовое объяснение решения. Функция `is_valid` проверяет, является ли данная высота конца гирлянды B допустимой, то есть сохраняет ли она хотя бы одну или две лампочки на нулевой высоте и остальные выше нуля.

Функция `find_min_b` выполняет двоичный поиск для нахождения минимальной возможной высоты B , используя функцию `is_valid` для проверки на каждом шаге.

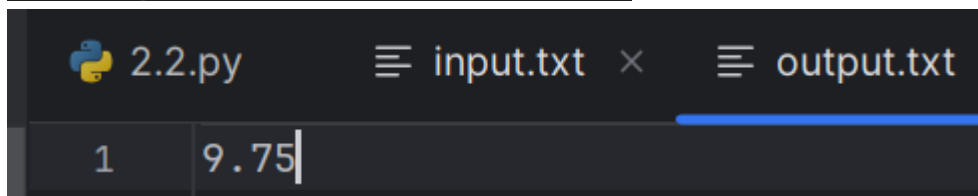
Чтение и запись данных осуществляется из входного файла `input.txt` и в выходной файл `output.txt`.

Результат работы кода на примерах из текста задачи:



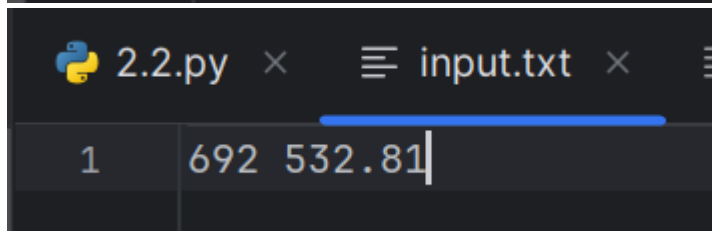
2.2.py input.txt ×

1	8 15
---	------



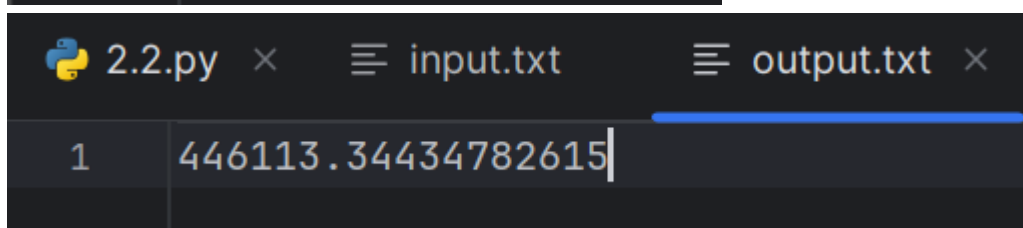
2.2.py input.txt × output.txt

1	9.75
---	------



2.2.py × input.txt ×

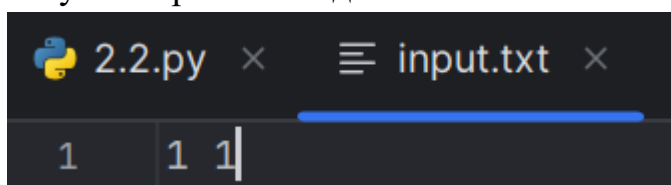
1	692 532.81
---	------------



2.2.py × input.txt × output.txt ×

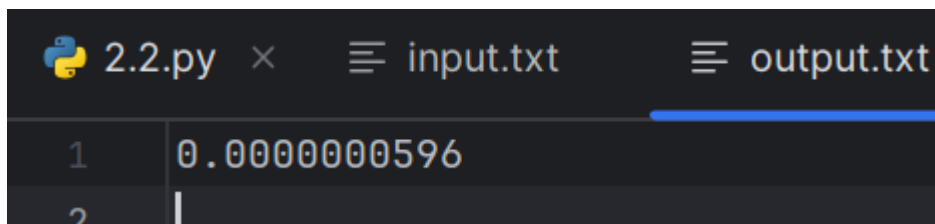
1	446113.34434782615
---	--------------------

Результат работы кода на максимальных и минимальных значениях:



2.2.py × input.txt ×

1	1 1
---	-----



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.005273754	22.8
Пример из задачи	0.004562635	23.2
Пример из задачи	0.004253618	23.2

Вывод по задаче: познакомилась с новыми методами поиска.

Задание №7. Опознавание двоичного дерева поиска, усложненная версия[N баллов]

Эта задача отличается от предыдущей тем, что двоичное дерева поиска может содержать равные ключи.

Вам дано двоичное дерево с ключами - целыми числами, которые могут повторяться. Вам нужно проверить, является ли это правильным двоичным деревом поиска. Теперь, для каждой вершины дерева V выполняется следующее условие:

- все ключи вершин из левого поддеревья меньше ключа вершины V ;
- все ключи вершин из правого поддеревья **больше или равны** ключу вершины V .

Другими словами, узлы с меньшими ключами находятся слева, а узлы с большими ключами – справа, дубликаты всегда справа. Вам необходимо проверить, удовлетворяет ли данная структура двоичного дерева этому условию.

```
f = open("input1.txt")
n = int(f.readline())
tree = []
for i in range(n):
```

```

a = f.readline().split()
x = [int(i) for i in a]
tree.append(x)

def if_correct(tr, size):
    for j in range(size):
        if tr[j][1] != -1 and tr[tr[j][1]][0] >= tr[j][0]:
            return False
        if tr[j][2] != -1 and tr[tr[j][2]][0] < tr[j][0]:
            return False
    return True

z = open("output.txt", "w")
fact = if_correct(tree, n)
if fact:
    z.write("CORRECT")
else:
    z.write("INCORRECT")

```

Текстовое объяснение решения:

Функция `if_correct` проверяет если правый ребенок меньше родителя или левый больше родителя, то возвращает `False`. Иначе возвращает `True`.

Результат работы кода на примерах из задачи:

3	CORRECT
2 1 2	
1 -1 -1	
3 -1 -1	
3	INCORRECT
1 1 2	
2 -1 -1	
3 -1 -1	

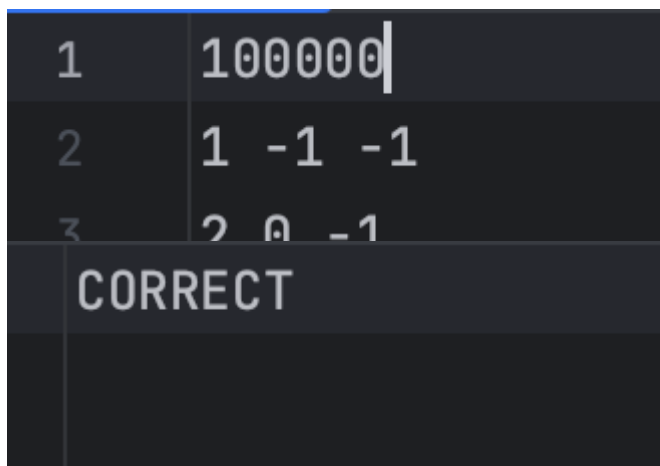
3	CORRECT
2 1 2	
1 -1 -1	
2 -1 -1	
3	INCORRECT
2 1 2	
2 -1 -1	
3 -1 -1	
5	CORRECT
1 -1 1	
2 -1 2	
3 -1 3	
4 -1 4	
5 -1 -1	

```
7 CORRECT
4 1 2
2 3 4
6 5 6
1 -1 -1
3 -1 -1
5 -1 -1
7 -1 -1
1
2147483647 -1 -1
CORRECT
```

Результат работы кода на минимальных значениях:

```
1
2147483647 -1 -1
CORRECT
```

Результат работы кода на максимальных значениях:



	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.033782738927383	22.8
Пример из задачи	0.002388237827398723	23.2
Верхняя граница диапазона значений входных данных из текста задачи(10000 элементов)	1.02399839028392303	23.2

Задание №16. Задача К-ый максимум[N баллов]

Напишите программу, реализующую структуру данных, позволяющую добавлять и удалять элементы, а также

находить k-й максимум.

```
f = open("input1.txt")
n = int(f.readline())
rez = []
z = open("output.txt", "w")
for i in range(n):
    s = f.readline().split()
    if s[0] == "+1":
        rez.append(int(s[1]))
        rez.sort()
    elif s[0] == "0":
        z.write(str(rez[len(rez)-int(s[1])]) + "\n")
    elif s[0] == "-1":
        rez.remove(int(s[1]))
```

Текстовое объяснение решения:

Если от программы требуется добавить элемент, то используется метод `append`. Если требуется найти K-ый максимум, он ищется вычитанием k из длины отсортированного списка. Если необходимо удалить, то используется `remove`.

Результат работы кода на примерах из задачи:

11	7
+1 5	
+1 3	5
+1 7	3
0 1	
0 2	10
0 3	7
-1 5	
+1 10	3
0 1	
0 2	
0 3	

Результат работы кода на минимальных значениях:

$$\begin{array}{r} 1 \\ + 1 \end{array}$$

$$|$$

Результат работы кода на максимальных значениях:

1	100000
2	+1 -497008111
3	0 1
4	0 1
5	-1 -497008111
6	+1 622138296
1	-497008111
2	-497008111
3	622138296
4	-999584751
5	-742651138
6	622138296

	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.00532635726332833	22.8
Пример из задачи	0.003662536725367762	23.2
Верхняя граница диапазона значений входных данных из текста задачи(10000 элементов)	0.046723726387263873	23.2

Вывод по задаче: Задача проходит по времени.

Задание №6. Оpozнание двоичного дерева поиска[N баллов]

В этой задаче вы собираетесь проверить, правильно ли реализована структура данных бинарного дерева поиска. Другими словами, вы хотите убедиться, что вы можете находить целые числа в этом двоичном дереве, используя бинарный поиск по дереву, и вы всегда получите правильный результат: если целое число есть в дереве, вы его найдете, иначе – нет.

Вам дано двоичное дерево с ключами - целыми числами. Вам нужно проверить, является ли это правильным двоичным деревом поиска. Для каждой вершины дерева V выполняется следующее условие:

- все ключи вершин из левого поддерева меньше ключа вершины V ;
- все ключи вершин из правого поддерева больше ключа вершины V .

Другими словами, узлы с меньшими ключами находятся слева, а узлы с большими ключами – справа. Вам необходимо проверить, удовлетворяет ли данная структура двоичного дерева этому условию. Вам гарантируется, что входные данные содержат допустимое двоичное дерево. То есть это дерево, и каждый узел имеет не более двух ребенков.

```
f = open("input1.txt")
n = int(f.readline())
tree = []
for i in range(n):
    a = f.readline().split()
    x = [int(i) for i in a]
    tree.append(x)

def if_correct(tr, size):
    for j in range(size):
        if tr[j][1] != -1 and tr[tr[j][1]][0] > tr[j][0]:
            return False
        if tr[j][2] != -1 and tr[tr[j][2]][0] < tr[j][0]:
            return False
    return True

z = open("output.txt", "w")
fact = if_correct(tree, n)
if fact:
    z.write("CORRECT")
else:
    z.write("INCORRECT")
```

Текстовое объяснение решения:

Функция `if_correct` проверяет если правый ребенок меньше родителя или левый больше родителя, то возвращает `False`. Иначе возвращает `True`.

Результат работы кода на примерах из задачи:

3	CORRECT
2 1 2	
1 -1 -1	
3 -1 -1	
3	INCORRECT
1 1 2	
2 -1 -1	
3 -1 -1	
0	CORRECT
5	CORRECT
1 -1 1	
2 -1 2	
3 -1 3	
4 -1 4	
5 -1 -1	

		CORRECT
1	100000	
2	1 -1 -1	
3	2 0 -1	
4	3 1 -1	
5	4 2 -1	
6	5 3 -1	
7	6 4 -1	
8	7 5 -1	
9	8 6 -1	

	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.05637347328443334	22.8
Пример из задачи	0.06237647364783463	23.2
Верхняя граница диапазона значений входных данных из текста задачи(10000 элеметов)	0.37346734983478394	23.2

Вывод по задаче: Задача очень похожа на задачу 7.

Дополнительные задачи

Задание №11. Сбалансированное двоичное дерево поиска[2 балла]

Реализуйте сбалансированное двоичное дерево поиска.

```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

class BinarySearchTree:
    def __init__(self):
        self.root = None

    def insert(self, key):
        self.root = self._insert(self.root, key)
```

```

def _insert(self, node, key):
    if not node:
        return Node(key)

    if key < node.key:
        node.left = self._insert(node.left, key)
    elif key > node.key:
        node.right = self._insert(node.right, key)

    return node

def delete(self, key):
    self.root = self._delete(self.root, key)

def _delete(self, node, key):
    if not node:
        return node

    if key < node.key:
        node.left = self._delete(node.left, key)
    elif key > node.key:
        node.right = self._delete(node.right, key)
    else:
        if not node.left:
            return node.right
        elif not node.right:
            return node.left
        else:
            node.key = self._min_value(node.right)
            node.right = self._delete(node.right, node.key)

    return node

def _min_value(self, node):
    current = node
    while current.left:
        current = current.left
    return current.key

def exists(self, key):
    return self._exists(self.root, key)

def _exists(self, node, key):
    if not node:
        return False

    if node.key == key:
        return True
    elif key < node.key:
        return self._exists(node.left, key)
    else:
        return self._exists(node.right, key)

def next(self, key):
    return self._next(self.root, key)

def _next(self, node, key):
    if not node:
        return "none"

    if key < node.key:

```



```

        left_result = self._next(node.left, key)
        if left_result == "none" or left_result <= key:
            return node.key
        else:
            return left_result
    else:
        return self._next(node.right, key)

def prev(self, key):
    return self._prev(self.root, key)

def _prev(self, node, key):
    if not node:
        return "none"

    if key > node.key:
        right_result = self._prev(node.right, key)
        if right_result == "none" or right_result >= key:
            return node.key
        else:
            return right_result
    else:
        return self._prev(node.left, key)

bst = BinarySearchTree()

z = open('output.txt', 'w')
with open('input11.txt', 'r') as file:
    for line in file:
        operation, key = line.split()
        key = int(key)

        if operation == 'insert':
            bst.insert(key)
        elif operation == 'delete':
            bst.delete(key)
        elif operation == 'exists':
            z.write(str(bst.exists(key)) + '\n')
        elif operation == 'next':
            z.write(str(bst.next(key)) + '\n')
        elif operation == 'prev':
            z.write(str(bst.prev(key)) + '\n')

```

Текстовое объяснение решения:

Класс BinarySearchTree обладает следующими функциями: insert, вставляющая в дерево новый элемент. Если элемент меньше чем текущий то оно добавляется влево. Иначе вправо. delete позволяет удалить элемент из дерева. min_value позволяет найти самый левый элемент дерева. exists позволяет удостовериться в существовании ключа пробегаая по дереву проверяя левое и правое в зависимости от того больше искомый ключ или меньше. next и prev позволяют находить минимальный строго больший и минимальный строго меньший.

```

insert 2      True
insert 5
insert 3      False
exists 2      5
exists 4|     3
next 4        3
prev 4
delete 5      none
next 4        3
prev 4

insert 2068   none
delete 2068   none
insert 840     none
next 840      none
prev 840      none
prev 840      True
prev 840      1908
prev 840      1908

```

	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи(1 элемент)	0.1873984273984732	10.5
Пример из задачи	0.26487837974923743	10.6
Верхняя граница диапазона значений входных данных из текста задачи(10000 элеметов)	0.4637826347638246783	11.6

Вывод по задаче: Задача реализует сбалансированное бинарное дерево.

Вывод по лабораторной работе

Лабораторная работа посвящена двоичным деревьям поиска. В ее ходе я научилась работе с ними.