

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

Отчет по лабораторной работе № 2-2

По дисциплине Алгоритмы и структуры данных

Обучающийся Овсянкин Даниил Витальевич

Преподаватель: Ромакина О.М

Факультет Инфокоммуникационных технологий

Группа К3244

Направление подготовки 45.03.04 Интеллектуальные системы в гуманитарной сфере

Образовательная программа Интеллектуальные системы в гуманитарной сфере

Санкт-Петербург

2025 г.

1 Задача. Обход двоичного дерева [5 s, 512 Мб, 1 балл]

В этой задаче вы реализуете три основных способа обхода двоичного дерева «в глубину»: центрированный (in-order), прямой (pre-order) и обратный (post-order). Очень полезно попрактиковаться в их реализации, чтобы лучше понять бинарные деревья поиска.

Вам дано корневое двоичное дерево. Выведите центрированный (in-order), прямой (pre-order) и обратный (post-order) обходы в глубину.

```
def inorder(keys, left, right, root=0): 1 usage
    res = []
    stack = []
    v = root
    while v != -1 or stack:
        while v != -1:
            stack.append(v)
            v = left[v]
        v = stack.pop()
        res.append(str(keys[v]))
        v = right[v]
    return res

def preorder(keys, left, right, root=0): 1 usage
    res = []
    stack = [root]
    while stack:
        v = stack.pop()
        if v == -1:
            continue
        res.append(str(keys[v]))
        stack.append(right[v])
        stack.append(left[v])
    return res

def postorder(keys, left, right, root=0): 1 usage
    res = []
    stack = [(root, False)]
    while stack:
        v, seen = stack.pop()
        if v == -1:
            continue
        if seen:
            res.append(str(keys[v]))
        else:
            stack.append((v, True))
            stack.append((right[v], False))
            stack.append((left[v], False))
    return res

def solve(): 1 usage
    with open('input.txt', 'r', encoding='utf-8') as fin:
        n_line = fin.readline().strip()
        n = int(n_line)
        keys = [0] * n
        left = [0] * n
        right = [0] * n
        for i in range(n):
            k, l, r = map(int, fin.readline().split())
            keys[i], left[i], right[i] = k, l, r

    in_ord = inorder(keys, left, right, root=0)
    pre_ord = preorder(keys, left, right, root=0)
    post_ord = postorder(keys, left, right, root=0)

    with open('output.txt', 'w', encoding='utf-8') as fout:
        fout.write(" ".join(in_ord) + "\n")
        fout.write(" ".join(pre_ord) + "\n")
        fout.write(" ".join(post_ord) + "\n")

if __name__ == "__main__":
    solve()
```

Текстовое объяснение решения:

В задаче было реализовано три углублённых обхода бинарного дерева без рекурсии. Узлы читаются в массивы ключей и индексов левого/правого ребёнка, где отсутствие ребёнка кодируется значением -1, корень — индекс 0. Для центрированного обхода (in-order) используется классический «спуск по левому краю» со стеком; для прямого (pre-order) — стек с добавлением правого, затем левого ребёнка; для обратного (post-order) — стек состояний с флагом «посещён после детей». Такой подход имеет линейную сложность $O(n)$ и не зависит от глубины дерева, поэтому надёжно работает при $n \leq 10^5$

Результат на примере:

Пример №1

```
input.txt
5
4 1 2
2 3 4
5 -1 -1
1 -1 -1
3 -1 -1

output.txt
1 2 3 4 5
4 2 1 3 5
1 3 2 5 4
```

Пример №2

```
input.txt
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

output.txt
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
```

Вывод: Задача решалась итеративными DFS для трёх вариантов обхода; решение линейное по времени и памяти и устойчиво к глубоким, вырожденным деревьям.

12 Задача. Проверка сбалансированности [2 s, 256 Мб, 2 балла]

АВЛ-дерево является сбалансированным в следующем смысле: для любой вершины высота ее левого поддерева отличается от высоты ее правого поддерева не больше, чем на единицу.

Введем понятие баланса вершины: для вершины дерева V ее баланс $B(V)$ равен разности высоты правого поддерева и высоты левого поддерева. Таким образом, свойство АВЛ-дерева, приведенное выше, можно сформулировать следующим образом: для любой ее вершины V выполняется следующее неравенство:

$$-1 \leq B(V) \leq 1$$

Обратите внимание, что, по историческим причинам, определение баланса в этой и последующих задачах этой недели «зеркально отражено» по сравнению с определением баланса в лекциях! Надеемся, что этот факт не доставит Вам неудобств. В литературе по алгоритмам – как российской, так и мировой – ситуация, как правило, примерно та же.

Дано двоичное дерево поиска. Для каждой его вершины требуется определить ее баланс.

```
def solve(): 1 usage
    with open('input.txt', 'r', encoding='utf-8') as fin:
        line = fin.readline()
        if not line:
            n = 0
        else:
            n = int(line.strip())

    if n == 0:
        with open('output.txt', 'w', encoding='utf-8') as fout:
            fout.write("")
        return

    K = [0] * (n + 1)
    L = [0] * (n + 1)
    R = [0] * (n + 1)
    parent = [0] * (n + 1)

    for i in range(1, n + 1):
        k, l, r = map(int, fin.readline().split())
        K[i], L[i], R[i] = k, l, r
        if l != 0:
```

```

        parent[l] = i
        if r != 0:
            parent[r] = i

    root = 1
    while root <= n and parent[root] != 0:
        root += 1
    if root > n:
        root = 1

    height = [0] * (n + 1)
    balance = [0] * (n + 1)

    stack = [(root, 0)]
    while stack:
        v, phase = stack.pop()
        if v == 0:
            continue
        if phase == 0:

```

```

            stack.append((v, 1))
            stack.append((R[v], 0))
            stack.append((L[v], 0))
        else:
            hl = height[L[v]] if L[v] != 0 else 0
            hr = height[R[v]] if R[v] != 0 else 0
            height[v] = (hl if hl > hr else hr) + 1
            balance[v] = hr - hl

    with open('output.txt', 'w', encoding='utf-8') as fout:
        fout.write("\n".join(str(balance[i]) for i in range(1, n + 1)))

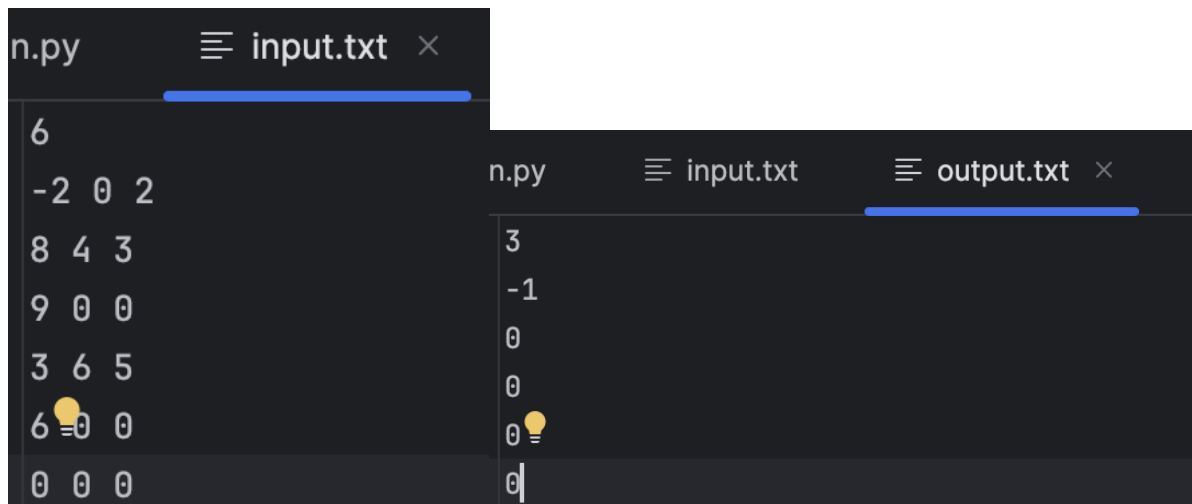
if __name__ == "__main__":
    solve()

```

Текстовое объяснение решения:

В задаче было реализовано вычисление баланса каждой вершины двоичного дерева через итеративный пост-обход. Сначала находится корень по массиву parent. Затем стеком выполняется обход «дети -> вершина», для каждой вершины вычисляются высоты левого и правого поддеревьев (пустое - 0) и баланс как их разность $hr - hl$. Такой подход имеет линейную сложность $O(N)$, не использует глубокую рекурсию.

Результат на примере:



```
n.py
6
-2 0 2
8 4 3
9 0 0
3 6 5
6 0 0
0 0 0

input.txt
3
-1
0
0
0
0

output.txt
3
-1
0
0
0
0
```

Вывод:

В задаче было реализовано итеративное вычисление баланса узлов через пост-обход: для каждого узла за один линейный проход считаются высоты левого и правого поддеревьев и печатается разность $hr - hl$. Решение работает за $O(N)$ по времени и $O(N)$ по памяти и устойчиво к глубоким деревьям.

17 Задача. Множество с суммой [120 s, 512 Mb, 3 балла]

В этой задаче ваша цель – реализовать структуру данных для хранения набора целых чисел и быстрого вычисления суммы элементов в заданном диапазоне.

Реализуйте такую структуру данных, в которой хранится набор целых чисел S и доступны следующие операции:

- $add(i)$ – добавить число i в множество S . Если i уже есть в S , то ничего делать не надо;
- $del(i)$ – удалить число i из множества S . Если i нет в S , то ничего делать не надо;
- $find(i)$ – проверить, есть ли i во множестве S или нет;
- $sum(l, r)$ – вывести сумму всех элементов v из S таких, что $l \leq v \leq r$.

Код задачи получился очень объемным, так что его можно посмотреть на Github

https://github.com/ovsyankaboi/Algorithms_2_sem/blob/main/lab2/task17/code/main.py

Текстовое объяснение решения:

В задаче было реализовано динамическое множество на декартовом дереве по ключу, где каждая вершина хранит сумму своего поддерева. Операции добавления, удаления и поиска работают за $O(\log n)$ за счёт split/merge. Запрос суммы по диапазону сводится к разности двух префиксных сумм $sum_leq(R) - sum_leq(L-1)$. Формат онлайн поддерживается сдвигом аргументов на x -

результат прошлой суммы - по модулю $M=10^9+1$, в запросе s полученный диапазон упорядочивается по возрастанию.

Результат на примере:

Пример №1

```
input.txt  x
5
+ 491572259
? 491572259
? 899375874
s 310971296 877523306
+ 352411209

output.txt  x
Found
Not found
491572259
```

Пример №2

```
input.txt  x
5
? 0
+ 0
? 0
- 0
? 0

output.txt  x
Not found
Found
Not found
```

Пример №3

```
input.txt  x
15
? 1
+ 1
? 1
+ 2
s 1 2
+ 1000000000
? 1000000000
- 1000000000
? 1000000000
s 999999999 1000000000
- 2
? 2
- 0
+ 9
s 0 9

output.txt  x
Not found
Found
3
Found
Not found
1
Not found
10
```

Вывод: в задаче было реализовано декартово дерево с агрегированием суммы в поддеревьях; все операции выполняются за $O(\log n)$, а онлайн сдвиг параметров и модуль корректно учтены при обработке запросов.

18 Задача. Вережка [120 s, 512 Мб, 5 баллов]

В этой задаче вы реализуете Вережку (или Rope) – структуру данных, которая может хранить строку и эффективно вырезать часть (подстроку) этой строки и вставлять ее в другое место. Эту структуру данных можно улучшить, чтобы она стала персистентной, то есть чтобы разрешить доступ к предыдущим версиям строки. Эти свойства делают ее подходящим выбором для хранения текста в текстовых редакторах.

Это очень сложная задача, более сложная, чем почти все предыдущие сложные задачи этого курса.

Вам дана строка S , и вы должны обработать n запросов. Каждый запрос описывается тремя целыми числами i, j, k и означает вырезание подстроки $S[i...j]$ (здесь индексы i и j в строке считаются от 0) из строки и вставка ее после k -го символа оставшейся строки (как бы символы в оставшейся строке нумеруются с 1). Если $k = 0$, $S[i...j]$ вставляется в начало. Дополнительные пояснения смотрите в примерах.

Код задачи:

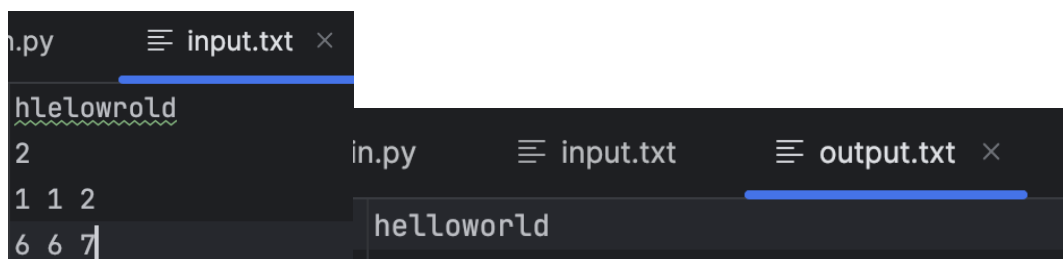
https://github.com/ovsyankaboi/Algorithms_2_sem/blob/main/lab2/task18/code/main.py

Текстовое объяснение решения:

В задаче было реализовано Rope на декартовом дереве по неявному ключу — позиции. Узлы хранят **чанки** исходной строки и суммарные размеры поддеревьев. Операция запроса (i, j, k) выполняется двумя `split` для выделения куска $S[i..j]$, затем дерево без этого куска собирается `merge(left, right)`. После этого по условию индексы считаются в оставшейся строке, поэтому делается `split` по позиции k и вставка: `merge(merge(L, mid), R)`. Получение ответа - симметричный обход, склеивающий чанки. Каждое действие занимает $O(\log N)$, а хранение чанков ускоряет реализацию.

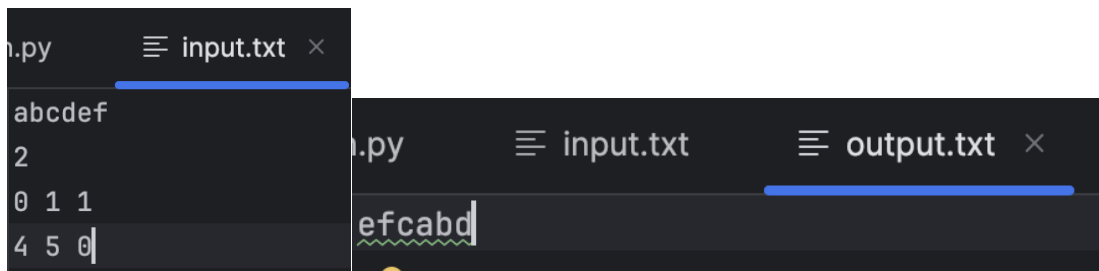
Результат на примере:

Пример №1



```
helloworld
2
1 1 2
6 6 7
helloworld
```

Пример №2



Вывод: задача решалась с помощью Rope на имплицитном treap'е: вырезания и вставки подстрок сводятся к последовательностям split/merge, что обеспечивает эффективность и корректность при больших размерах строки и числе запросов.

16 Задача. K -й максимум [2 s, 512 Mb, 3 балла]

Напишите программу, реализующую структуру данных, позволяющую добавлять и удалять элементы, а также находить k -й максимум.

- **Формат ввода / входного файла (input.txt).** Первая строка входного файла содержит натуральное число n – количество команд. Последующие n строк содержат по одной команде каждая. Команда записывается в виде двух чисел c_i и k_i – тип и аргумент команды соответственно. Поддерживаемые команды:

- +1 (или просто 1): Добавить элемент с ключом k_i .
- 0 : Найти и вывести k_i -й максимум.
- -1 : Удалить элемент с ключом k_i .

Гарантируется, что в процессе работы в структуре не требуется хранить элементы с равными ключами или удалять несуществующие элементы. Также гарантируется, что при запросе k_i -го макс-симула, он существует.

Код задачи:

https://github.com/ovsyankaboi/Algorithms_2_sem/blob/main/lab2/task16/code/main.py

Текстовое объяснение решения:

В задаче было реализовано множество на декартовом дереве по ключу: в каждом узле хранится размер поддерева, что позволяет за $O(\log n)$ вставлять, удалять и находить элемент заданного ранга. Запрос 0 k трактуется как k -й максимум при 0-индексации: это $(\text{size}-1-k)$ -й элемент по возрастанию, который извлекается спуском по дереву с учётом размеров левых поддеревьев. Вставка/удаление выполняются через стандартные split/merge

Результат на примере:

The screenshot shows a code editor with two files: `input.txt` and `output.txt`. The `input.txt` file contains a sequence of operations on a treap: an initial value of 11, followed by adding 5, 3, and 7, then deleting 1, 2, and 3, then adding 10, and finally deleting 1, 2, and 3. The `output.txt` file shows the resulting values after each operation: 7, 5, 3, 10, 7, and 3.

```
input.txt
11
+1 5
+1 3
+1 7
0 1
0 2
0 3
-1 5
+1 10
0 1
0 2
0 3

output.txt
7
5
3
10
7
3
```

Вывод: в задаче было реализовано дерево порядка (treap) с размером поддеревя; все операции выполняются за $O(\log n)$, k-й максимум выдаётся как элемент нужного ранга, что соответствует формату входа и ограничениям.

Вывод по лабораторной работе

Практиковался и научился решать задачи на двоичные деревья поиска и сбалансированные деревья поиска