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 20

Выполнил:

Смирнов Георгий Валерьевич

К3139

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

## 

# 

# 

[**Задачи по вариантам**](#_2eum5q9t62jj) **3**

[Задач №6. Опознание двоичного дерева поиска.](#_uzl6cklx62gu) 4

[Задача №8. Высота дерева возвращается](#_1fob9te) 8

[Задача №17. Множество с суммой](#_3znysh7) 12

[**Вывод**](#_2et92p0) **19**

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

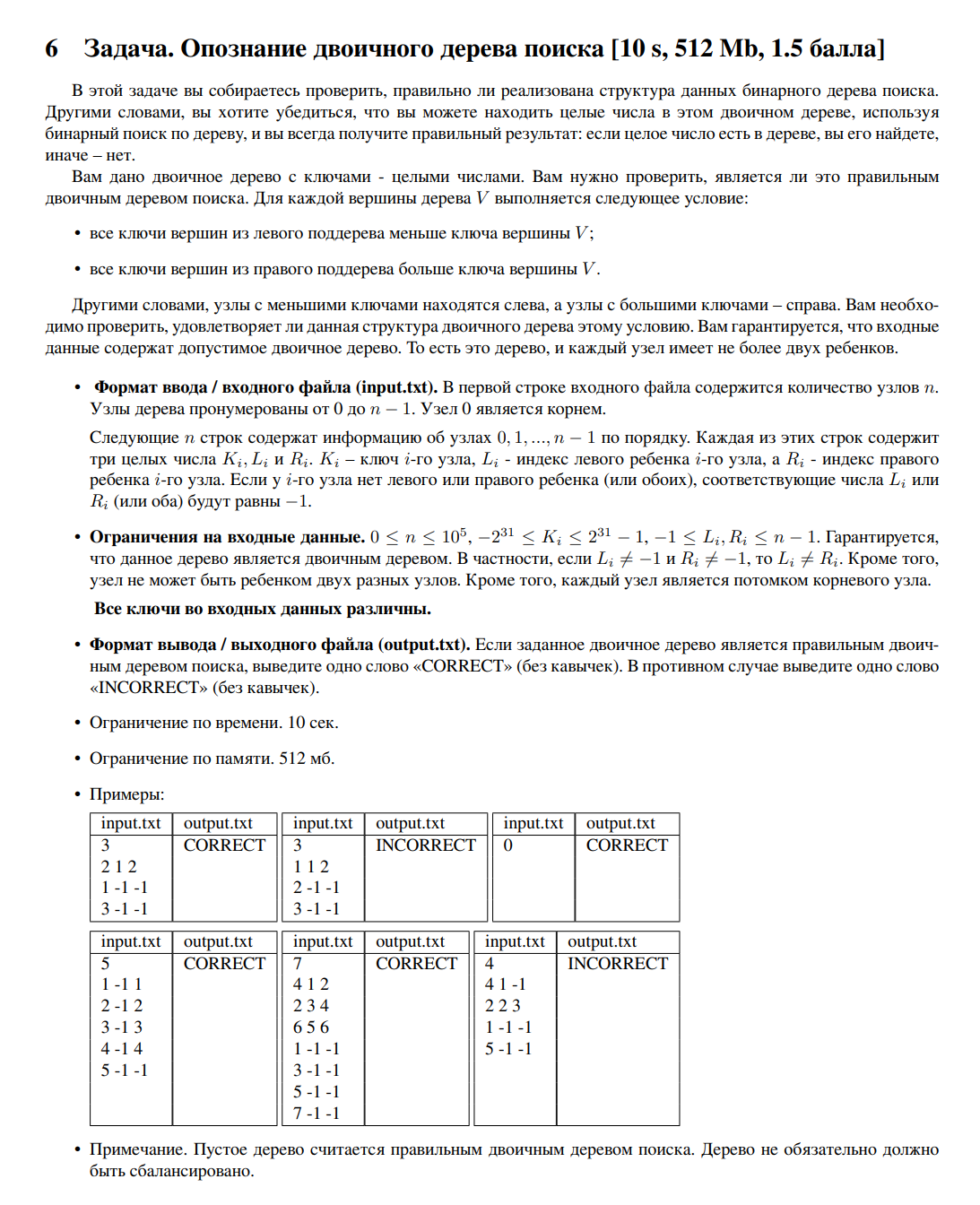
# 

# 

# 

# Задачи по вариантам

## Задач №6. Опознание двоичного дерева поиска.



**Код программы**

def is\_bst(tree, node, min\_key, max\_key):

if node == -1:

return True

key, left, right = tree[node]

if not (min\_key < key < max\_key):

return False

return is\_bst(tree, left, min\_key, key) and is\_bst(tree, right, key, max\_key)

def main():

with open('input.txt', 'r') as f:

data = f.read().split()

n = int(data[0])

if n == 0:

with open('output.txt', 'w') as f:

f.write("CORRECT\n")

return

tree = []

index = 1

for i in range(n):

key = int(data[index])

left = int(data[index + 1])

right = int(data[index + 2])

tree.append((key, left, right))

index += 3

result = "CORRECT" if is\_bst(tree, 0, float('-inf'), float('inf')) else "INCORRECT"

with open('output.txt', 'w') as f:

f.write(result + "\n")

if \_\_name\_\_ == "\_\_main\_\_":

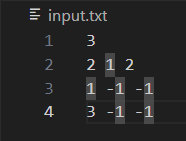
main()

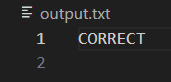
**Текстовое объяснение решения**

Если дерево пустое, программа записывает "CORRECT" в файл output.txt и завершает выполнение. В противном случае, программа строит дерево из считанных данных, где каждый узел представлен ключом и индексами левого и правого дочерних узлов. Затем функция is\_bst рекурсивно проверяет, что все узлы дерева удовлетворяют свойствам бинарного дерева поиска, используя минимальные и максимальные допустимые значения для ключей узлов. Результат проверки ("CORRECT" или "INCORRECT") записывается в файл

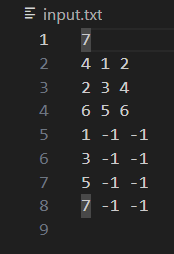
Результат работы кода на примерах из текста задачи:

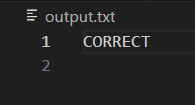
1)





2)

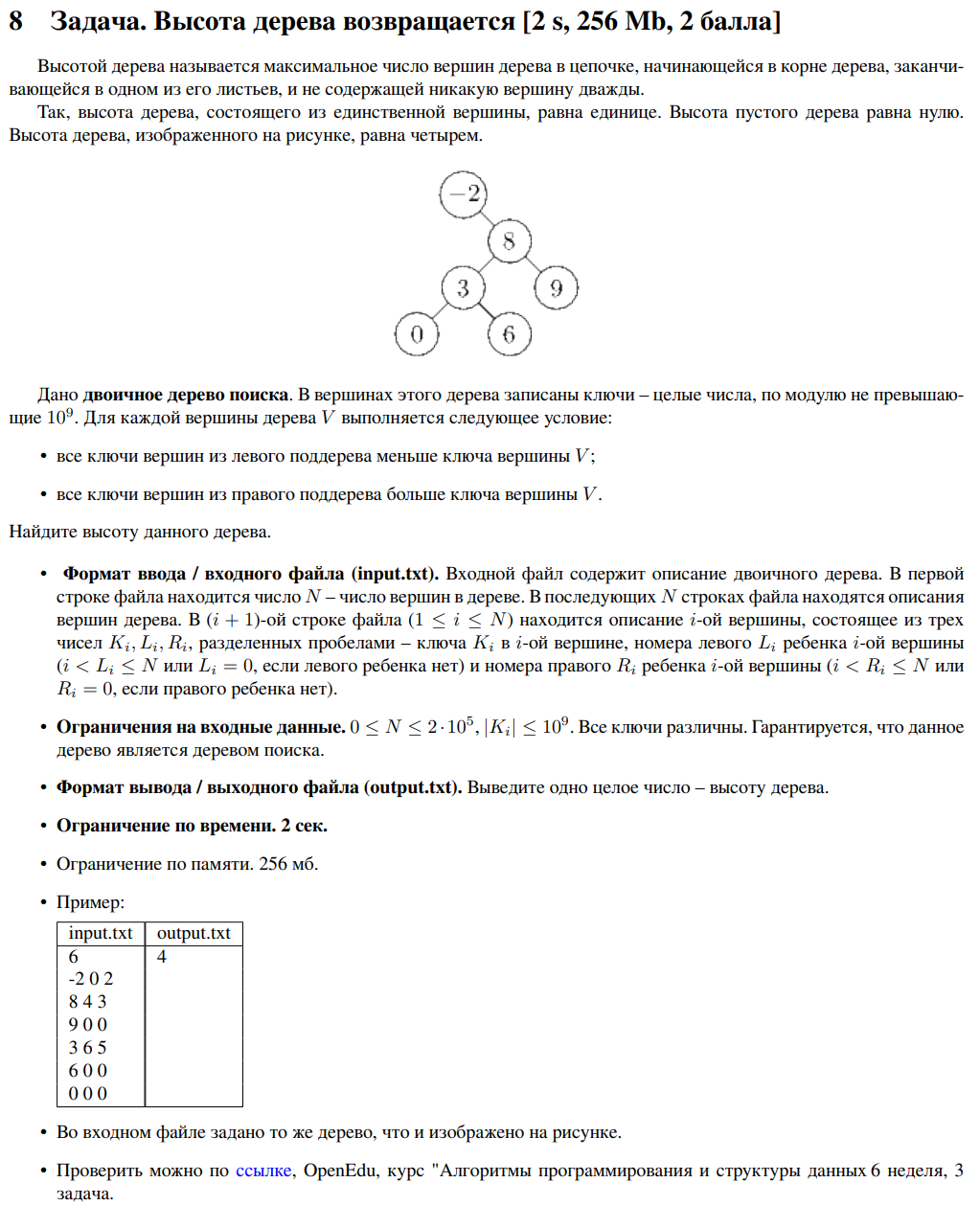




|  | Время выполнения, с | Затраты памяти, МБ |
| --- | --- | --- |
| Пример из задачи | 0.0010037422180175781 | 0.02048 |
| Пример из задачи | 0.0009994506835937 | 0.02048 |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

## Задача №8. Высота дерева возвращается



**Код программы**

import psutil

import time

def tree\_height(node, nodes):

if node == 0:

return 0

key, left, right = nodes[node - 1]

left\_height = tree\_height(left, nodes)

right\_height = tree\_height(right, nodes)

return max(left\_height, right\_height) + 1

def main():

start\_time = time.time()

process = psutil.Process()

with open('input.txt', 'r') as f:

n = int(f.readline().strip())

if n < 0 or n > 2 \* 10\*\*5:

raise ValueError("Количество узлов должно быть в диапазоне от 0 до 200000.")

if n == 0:

with open('output.txt', 'w') as f\_out:

f\_out.write("0")

return

nodes = []

for \_ in range(n):

key, left, right = map(int, f.readline().strip().split())

if not (-10\*\*9 <= key <= 10\*\*9):

raise ValueError("Ключи должны быть в диапазоне от -10^9 до 10^9.")

if not (0 <= left <= n) or not (0 <= right <= n):

raise ValueError("Индексы детей должны быть в диапазоне от 0 до n.")

nodes.append((key, left, right))

height = tree\_height(1, nodes)

with open('output.txt', 'w') as f\_out:

f\_out.write(f"{height}\n")

end\_time = time.time()

memory\_info = process.memory\_info()

print(f"Время выполнения: {end\_time - start\_time:.6f} секунд")

print(f"Использование памяти: {memory\_info.rss / (1024 \* 1024):.6f} MB")

if \_\_name\_\_ == "\_\_main\_\_":

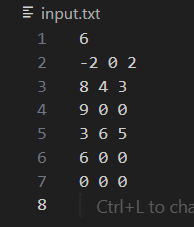
main()

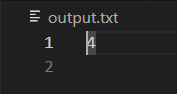
**Текстовое объяснение решения**

Сначала программа считывает данные из файла input.txt. Если количество узлов (n) равно нулю, программа записывает "0" в файл output.txt и завершает выполнение. В противном случае, программа считывает информацию о каждом узле, проверяя корректность значений ключей и индексов дочерних узлов. Затем программа вычисляет высоту дерева, начиная с корневого узла, используя рекурсивную функцию tree\_height, которая возвращает максимальную высоту между левым и правым поддеревьями плюс один.

Результат работы кода на примерах из текста задачи:

1)

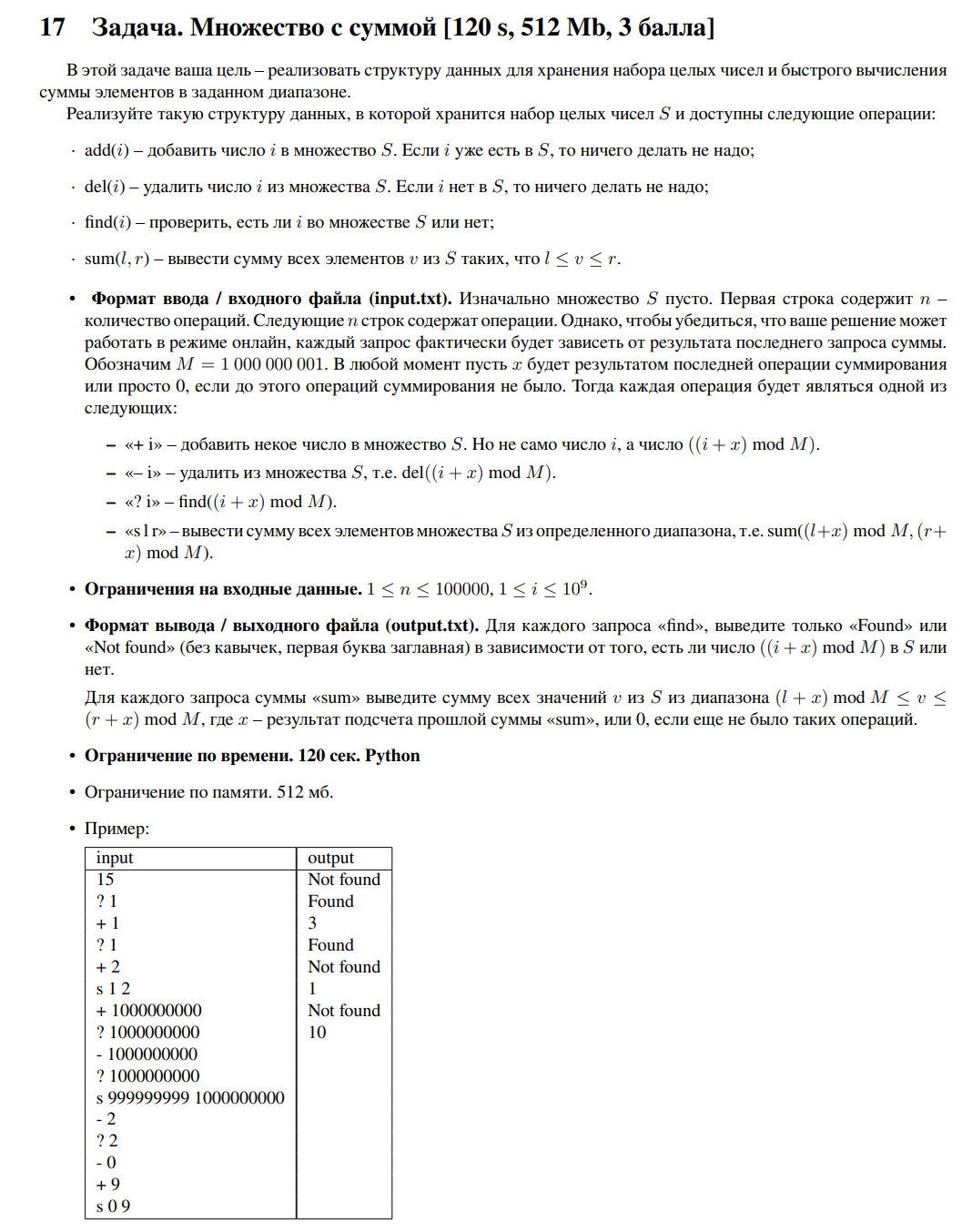




|  | Время выполнения, с | Затраты памяти, МБ |
| --- | --- | --- |
| Пример из задачи | 0.001005 секунд | 14.835938 MB |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

## Задача №17. Множество с суммой



**Код программы**

import sys

import time

import tracemalloc

class Node:

def \_\_init\_\_(self, key):

self.key = key

self.left = None

self.right = None

self.parent = None

self.subtree\_sum = key

class SplayTree:

def \_\_init\_\_(self):

self.root = None

def \_update(self, node):

if node:

node.subtree\_sum = node.key

if node.left:

node.subtree\_sum += node.left.subtree\_sum

if node.right:

node.subtree\_sum += node.right.subtree\_sum

def \_rotate(self, x):

p = x.parent

g = p.parent

if p.left == x:

p.left = x.right

if x.right:

x.right.parent = p

x.right = p

else:

p.right = x.left

if x.left:

x.left.parent = p

x.left = p

p.parent = x

x.parent = g

if g:

if g.left == p:

g.left = x

else:

g.right = x

else:

self.root = x

self.\_update(p)

self.\_update(x)

def \_splay(self, x):

while x.parent:

p = x.parent

g = p.parent

if g:

if (g.left == p) == (p.left == x):

self.\_rotate(p)

else:

self.\_rotate(x)

self.\_rotate(x)

def \_find(self, key):

node = self.root

while node:

if key == node.key:

self.\_splay(node)

return node

elif key < node.key:

if not node.left:

self.\_splay(node)

return None

node = node.left

else:

if not node.right:

self.\_splay(node)

return None

node = node.right

return None

def add(self, key):

if not self.root:

self.root = Node(key)

return

node = self.root

while True:

if key == node.key:

self.\_splay(node)

return

elif key < node.key:

if not node.left:

node.left = Node(key)

node.left.parent = node

self.\_splay(node.left)

return

node = node.left

else:

if not node.right:

node.right = Node(key)

node.right.parent = node

self.\_splay(node.right)

return

node = node.right

def delete(self, key):

node = self.\_find(key)

if not node:

return

self.\_splay(node)

if not node.left:

self.\_replace(node, node.right)

elif not node.right:

self.\_replace(node, node.left)

else:

min\_node = self.\_subtree\_min(node.right)

if min\_node.parent != node:

self.\_replace(min\_node, min\_node.right)

min\_node.right = node.right

min\_node.right.parent = min\_node

self.\_replace(node, min\_node)

min\_node.left = node.left

min\_node.left.parent = min\_node

self.\_update(self.root)

def \_replace(self, u, v):

if not u.parent:

self.root = v

elif u == u.parent.left:

u.parent.left = v

else:

u.parent.right = v

if v:

v.parent = u.parent

def \_subtree\_min(self, node):

while node.left:

node = node.left

return node

def find(self, key):

return self.\_find(key) is not None

def sum\_range(self, l, r):

if not self.root:

return 0

self.\_find(l)

if self.root.key < l:

if not self.root.right:

return 0

self.root = self.root.right

self.root.parent = None

self.\_find(r)

if self.root.key > r:

if not self.root.left:

return 0

self.root = self.root.left

self.root.parent = None

return self.\_subtree\_sum(self.root, l, r)

def \_subtree\_sum(self, node, l, r):

if not node:

return 0

if node.key < l:

return self.\_subtree\_sum(node.right, l, r)

if node.key > r:

return self.\_subtree\_sum(node.left, l, r)

return node.key + self.\_subtree\_sum(node.left, l, r) + self.\_subtree\_sum(node.right, l, r)

def read\_input(file\_path):

with open(file\_path, 'r') as file:

n = int(file.readline().strip())

operations = [file.readline().strip() for \_ in range(n)]

return operations

def write\_output(file\_path, results):

with open(file\_path, 'w') as file:

for result in results:

file.write(result + "\n")

def main():

tracemalloc.start()

start\_time = time.time()

operations = read\_input('input.txt')

tree = SplayTree()

last\_sum = 0

results = []

for operation in operations:

if operation.startswith('+'):

value = (int(operation[1:]) + last\_sum) % 1000000000

tree.add(value)

elif operation.startswith('-'):

value = (int(operation[1:]) + last\_sum) % 1000000000

tree.delete(value)

elif operation.startswith('?'):

value = (int(operation[1:]) + last\_sum) % 1000000000

if tree.find(value):

results.append("Found")

else:

results.append("Not found")

elif operation.startswith('s'):

l, r = map(int, operation[1:].split())

l = (l + last\_sum) % 1000000000

r = (r + last\_sum) % 1000000000

last\_sum = tree.sum\_range(min(l, r), max(l, r))

results.append(str(last\_sum))

last\_sum = (last\_sum + 1000000000) % 1000000000

write\_output('output.txt', results)

end\_time = time.time()

current, peak = tracemalloc.get\_traced\_memory()

tracemalloc.stop()

print(f"Время выполнения: {end\_time - start\_time} секунд")

print(f"Использование памяти: текущее = {current / 10\*\*6} MB, пик = {peak / 10\*\*6} MB")

if \_\_name\_\_ == "\_\_main\_\_":

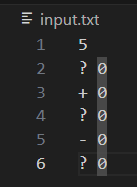
main()

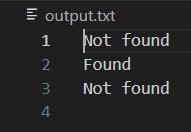
**Текстовое объяснение решения**

После считывания данных сортируем заявки по времени окончания. После этого выбираем максимальное количество непересекающихся заявок. Для этого проходим по отсортированному списку заявок и добавляем заявку в выбор, если её начало не пересекается с концом последней выбранной заявки.

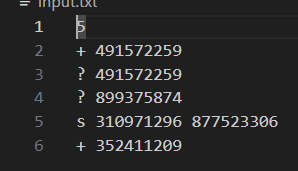
Результат работы кода на примерах из текста задачи:

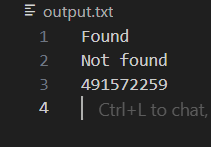
1)





2)





|  | Время выполнения, с | Затраты памяти, МБ |
| --- | --- | --- |
| Пример из задачи | 0.004002809524536133 | 0.018301 |
| Пример из задачи | 0.0010004043579101562 | 0.018401 |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

раничения по времени и памяти

# Вывод

В ходе данной лабораторной работы я научился решать задачи. Написанные программы были протестированы, а также были измерены потребляемый ими объём памяти и время работы. Все программы работаю корректно и укладываются в установленные ограничения по времени и памяти на примерах из задач.