

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

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»
Тема: Хэширование. Хэш-таблицы
Вариант 1

Выполнил:
Гайдук А. С.
К3241

Проверила:
Ромакина О. М.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета.....	2
Задачи по варианту	3
Задача №1. Множество.....	3
Задача №2. Телефонная книга	6
Задача №5. Выборы в США.....	10
Дополнительные задачи	14
Задача №4. Прошитый ассоциативный массив	14
Задача №6. Фибоначчи возвращается.....	19
Вывод	23

Задачи по варианту

Задача №1. Множество

Реализуйте множество с операциями «добавление ключа», «удаление ключа», «проверка существования ключа».

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

class HashSet:
    def __init__(self, size):
        self.size = size
        self.table = [[] for _ in range(size)]

    def _hash(self, key):
        return key % self.size

    def add(self, key):
        index = self._hash(key)
        if key not in self.table[index]:
            self.table[index].append(key)

    def delete(self, key):
        index = self._hash(key)
        self.table[index] = [k for k in self.table[index] if k != key]

    def find(self, key):
        index = self._hash(key)
        return key in self.table[index]

def set_ops(operations):
    hash_set = HashSet(size=100003)
    result = []

    for operation in operations:
        parts = operation.split()
        command = parts[0]
        x = int(parts[1])

        if command == 'A':
            hash_set.add(x)
        elif command == 'D':
            hash_set.delete(x)
        elif command == '?':
            result.append('Y' if hash_set.find(x) else 'N')
    return result

with open('input.txt', 'r') as f:
    n = int(f.readline().strip())
    operations = [f.readline().strip() for _ in range(n)]

start_time = time.perf_counter()
```

```

results = set_ops(operations)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as f:
    f.write('\n'.join(results) + '\n')

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Я создала класс HashSet, который реализует хэш-таблицу для операций с множеством. Здесь у меня инициализация таблицы заданного размера, сама хэш-таблица представлена списком смежности, т. е. каждая ячейка является списком, храня все значения, которые хэш-функцию поместит в одну ячейку; метод `_hash`, который распределяет ключи по таблице (возвращает индекс для ключа через взятие остатка от деления), а также сами методы `add`, `delete`, `find`.

Функция `set_ops` вызывает методы класса HashSet в зависимости от команды в строке с операцией.

Результат работы кода на примерах из текста задачи:

task1.py		input.txt	output.txt
1		8	
2		A 2	
3		A 5	
4		A 3	
5		? 2	
6		? 4	
7		A 2	
8		D 2	
9		? 2	

```
task1.py  input.txt  output.txt x
1 |
2 N
3 N
```

Результат работы кода на максимальных и минимальных значениях:

```
task1.py  input.txt x  output.txt
1 | 1
2 ? 3
```

```
task1.py  input.txt  output.txt x
1 | N
```

```
task1.py  input.txt x  output.txt
⚠ The file size (11,19 MB) exceeds the configured
1 | 500000
2 | A 523392024686160706
3 | A 19417741862140460
4 | D -304055257890162827
5 | ? 837363634587837394
```

```
task1.py  input.txt  output.txt x
1 | N
2 | N
3 | N
4 | N
5 | N
6 | N
7 | N
8 | N
9 | N
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.070468 сек	6.423695 МВ
Пример из задачи	0.069870 сек	6.424235 МВ
Верхняя граница диапазона значений входных данных из текста задачи	1.684723 сек	50.724193 МВ

Вывод по задаче:

В этой задаче я ознакомилась с реализацией хэш-таблицы, узнала о таком методе цепочек для решения коллизий, смогла реализовать множество с операциями «добавление», «удаление», «проверка существования ключа» с использованием собственной хэш-таблицы.

Задача №2. Телефонная книга

В этой задаче ваша цель - реализовать простой менеджер телефонной книги. Он должен уметь обрабатывать следующие типы пользовательских запросов:

- `add number name` – это команда означает, что пользователь добавляет в телефонную книгу человека с именем `name` и номером телефона `number`. Если пользователь с таким номером уже существует, то ваш менеджер должен перезаписать соответствующее имя.
- `del number` – означает, что менеджер должен удалить человека с номером из телефонной книги. Если такого человека нет, то он должен просто игнорировать запрос.
- `find number` – означает, что пользователь ищет человека с номером телефона `number`. Менеджер должен ответить соответствующим именем или строкой «not found» (без кавычек), если такого человека в книге нет.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = [[] for _ in range(size)]

    def _hash(self, key):
        return int(key) % self.size

    def add(self, key, value):
        index = self._hash(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index][i] = (key, value)
                return
        self.table[index].append((key, value))

    def delete(self, key):
        index = self._hash(key)
        self.table[index] = [(k, v) for k, v in self.table[index] if k !=
key]

    def find(self, key):
        index = self._hash(key)
        for k, v in self.table[index]:
            if k == key:
                return v
        return "not found"

def phone_manager(operations):
    hash_table = HashTable(size=100003)
    result = []
    for operation in operations:
        parts = operation.split()
        command = parts[0]
        number = int(parts[1])

        if command == "add":
            name = parts[2]
            hash_table.add(number, name)
        if command == "del":
            hash_table.delete(number)
        if command == "find":
            name = hash_table.find(number)
            result.append(name)
    return result

with open('input.txt', 'r') as f:
    n = int(f.readline())
    operations = [f.readline().strip() for _ in range(n)]

start_time = time.perf_counter()

result = phone_manager(operations)
```

```
end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

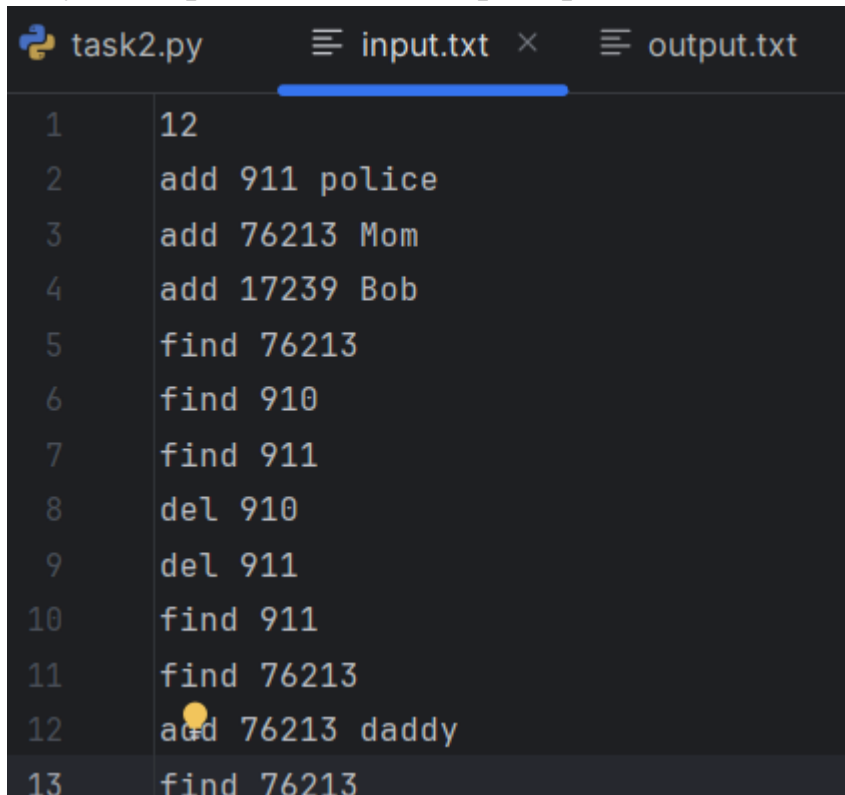
with open('output.txt', 'w') as f:
    f.write('\n'.join(result) + '\n')

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

Текстовое объяснение решения:

Для решения этого задания я создала хэш-таблицу, преобразовав таблицу из задачи №1 так, чтобы она могла хранить не только ключи, но и связанные с ними значения (в данном случае имена). Метод `phone_manager` создает объект класса `NashTable` и вызывает его методы в зависимости от типа команды в строке с операцией.

Результат работы кода на примерах из текста задачи:



The screenshot shows a code editor with three tabs: 'task2.py', 'input.txt', and 'output.txt'. The 'input.txt' tab is active and displays a list of 13 commands. The 'output.txt' tab is also visible but empty. The commands in 'input.txt' are:

Line	Command
1	12
2	add 911 police
3	add 76213 Mom
4	add 17239 Bob
5	find 76213
6	find 910
7	find 911
8	del 910
9	del 911
10	find 911
11	find 76213
12	add 76213 daddy
13	find 76213


```
task2.py  input.txt  output.txt x
1 Mom
2 not found
3 police
4 not found
5 Mom
6 daddy
```

```
task2.py  input.txt x  output.txt
1 8
2 find 3839442
3 add 123456 me
4 add 0 granny
5 find 0
6 find 123456
7 del 0
8 del 0
9 find 0
```

```
task2.py  input.txt  output.txt x
1 not found
2 granny
3 me
4 not found
```

Результат работы кода на максимальных и минимальных значениях:

```
task2.py  input.txt x  output.txt
1 1
2 find 911
```

```
task2.py  input.txt  output.txt x
1 not found
```

```
task2.py x input.txt x output.txt
1 100000
2 del 24974
3 add 65 fc
4 del 45
5 del 8
```

```
task2.py input.txt output.txt x
1 not found
2 not found
3 not found
4 not found
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.070555 сек	6.423881 МВ
Пример из задачи	0.070219 сек	6.425293 МВ
Пример из задачи	0.071239 сек	6.424663 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.520793 сек	15.680498 МВ

Вывод по задаче:

Я научилась реализовывать хэш-таблицу, которая может хранить не только ключи, но и значения, соответствующие ключам.

Задача №5. Выборы в США

Как известно, в США президент выбирается не прямым голосованием, а путем двухуровневого голосования. Сначала проводятся выборы в каждом

штате и определяется победитель выборов в данном штате. Затем проводятся государственные выборы: на этих выборах каждый штат имеет определенное число голосов — число выборщиков от этого штата. На практике все выборщики от штата голосуют в соответствии с результатами голосования внутри штата, то есть на заключительной стадии выборов в голосовании участвуют штаты, имеющие различное число голосов. Вам известно за кого проголосовал каждый штат и сколько голосов было отдано данным штатом. Подведите итоги выборов: для каждого из участника голосования определите число отданных за него голосов.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = [[] for _ in range(size)]

    def _hash(self, key):
        return hash(key) % self.size

    def add(self, key, value):
        index = self._hash(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index][i] = (key, v + value)
                return
        self.table[index].append((key, value))

    def get_all(self):
        candidates = {}
        for chain in self.table:
            for k, v in chain:
                candidates[k] = v
        return candidates

def elections(lines):
    hash_table = HashTable(size=100003)
    for line in lines:
        parts = line.strip().split()
        candidate = parts[0]
        votes = int(parts[1])
        hash_table.add(candidate, votes)
    result = hash_table.get_all()
    final_result = sorted(result.items())
    return final_result

with open('input.txt', 'r') as file:
    lines = file.readlines()
```

```

start_time = time.perf_counter()

result = elections(lines)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as file:
    for candidate, votes in result:
        file.write(f"{candidate} {votes}\n")

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Для решения этой задачи я создала хэш-таблицу с методом `get_all`, где создается пустой словарь для итоговых данных. Проходя по всем элементам всех цепочек хэш-таблицы, пары ключ-значение (где ключ – фамилия кандидата, а значение – количество голосов) добавляются в словарь кандидатов. В методе для реализации самих «выборов» кандидаты и их голоса поочередно добавляются в хэш-таблицу, а затем вызывается метод `get_all()` для всей таблицы.

Результат работы кода на примерах из текста задачи:

task5.py	input.txt	output.txt
1	McCain 10	
2	McCain 5	
3	Obama 9	
4	Obama 8	
5	McCain 1	

task5.py	input.txt	output.txt
1	McCain 16	
2	Obama 17	

```
task5.py  input.txt  output.txt
1  ivanov 100
2  ivanov 500
3  ivanov 300
4  petr 70
5  tourist 1
6  tourist 2
```

```
task5.py  input.txt  output.txt
1  ivanov 900
2  petr 70
3  tourist 3
```

```
task5.py  input.txt  output.txt
1  bur 1
```

```
task5.py  input.txt  output.txt
1  bur 1
```

	Время выполнения	Затраты памяти
Пример из задачи	0.115143 сек	6.424335 МВ
Пример из задачи	0.102441 сек	6.424569 МВ
Пример из задачи	0.152090 сек	6.423781 МВ

Вывод по задаче:

В ходе выполнения этой задачи я реализовала алгоритм подсчета голосов за каждого кандидата в выборах и вывела результаты в лексикографическом порядке посредством сортировки.

Дополнительные задачи

Задача №4. Прошитый ассоциативный массив

Реализуйте прошитый ассоциативный массив. Ваш алгоритм должен поддерживать следующие типы операций:

- `get x` – если ключ `x` есть в множестве, выведите соответствующее ему значение, если нет, то выведите `<none>`.
- `prev x` – вывести значение, соответствующее ключу, находящемуся в ассоциативном массиве, который был вставлен позже всех, но до `x`, или `<none>`, если такого нет или в массиве нет `x`.
- `next x` – вывести значение, соответствующее ключу, находящемуся в ассоциативном массиве, который был вставлен раньше всех, но после `x`, или `<none>`, если такого нет или в массиве нет `x`.
- `put x y` – поставить в соответствие ключу `x` значение `y`. При этом следует учесть, что
 - если, независимо от предыстории, этого ключа на момент вставки в массиве не было, то он считается только что вставленным и оказывается самым последним среди добавленных элементов – то есть, вызов `next` с этим же ключом сразу после выполнения текущей операции `put` должен вернуть `<none>`;
 - если этот ключ уже есть в массиве, то значение необходимо изменить, и в этом случае ключ не считается вставленным еще раз, то есть, не меняет своего положения в порядке добавленных элементов.
- `delete x` – удалить ключ `x`. Если ключа в ассоциативном массиве нет, то ничего делать не надо.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

class HashMap:
```

```

def __init__(self, size=100003):
    self.size = size
    self.table = [[] for _ in range(size)]
    self.order = []

def _hash(self, key):
    return hash(key) % self.size

def put(self, key, value):
    index = self._hash(key)
    table = self.table[index]
    for i, (k, v) in enumerate(table):
        if k == key:
            table[i] = (key, value)
            return
    table.append((key, value))
    self.order.append(key)

def get(self, key):
    index = self._hash(key)
    table = self.table[index]
    for k, v in table:
        if k == key:
            return v
    return "<none>"

def delete(self, key):
    index = self._hash(key)
    table = self.table[index]
    for i, (k, v) in enumerate(table):
        if k == key:
            del table[i]
            self.order.remove(key)
            return

def prev(self, key):
    if key not in self.order:
        return "<none>"
    index = self.order.index(key)
    if index > 0:
        prev_key = self.order[index - 1]
        return self.get(prev_key)
    return "<none>"

def next(self, key):
    if key not in self.order:
        return "<none>"
    index = self.order.index(key)
    if index < len(self.order) - 1:
        next_key = self.order[index + 1]
        return self.get(next_key)
    return "<none>"

def op_manager(operations):
    result = []
    hash_map = HashMap()
    for operation in operations:
        parts = operation.strip().split()
        command = parts[0]

```

```

        if command == "put":
            key, value = parts[1], parts[2]
            hash_map.put(key, value)
        elif command == "get":
            key = parts[1]
            result.append(hash_map.get(key))
        elif command == "delete":
            key = parts[1]
            hash_map.delete(key)
        elif command == "prev":
            key = parts[1]
            result.append(hash_map.prev(key))
        elif command == "next":
            key = parts[1]
            result.append(hash_map.next(key))

    return result

with open('input.txt', 'r') as f:
    n = int(f.readline())
    operations = [f.readline().strip() for _ in range(n)]

start_time = time.perf_counter()

result = op_manager(operations)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as f:
    f.write('\n'.join(result) + '\n')

print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое  
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Я создала класс HashMap, который реализует ассоциативный массив с использованием списка смежности и отслеживает порядок добавления ключей. Метод put аналогичен методу add, который я реализовывала в предыдущих задачах, однако теперь ключ добавляется в список self.order, чтобы отслеживать порядок вставки. В методе get мы находим индекс подписка в списке смежности, получаем его и проверяем все элементы подписка. Если ключ найден, то возвращаем значение, иначе <none>. В методе delete если ключ найден среди элементов подписка, то он удаляется в том числе из списка с порядком. Методы prev и next основаны на проверке, есть ли у ключа предыдущий/следующий ключ, если да, то возвращается его значение, иначе <none>. Метод op_manager создает объект класса

HashMap, проходит циклом по всем операциям и вызывает методы класса, исходя из типа операции.

Результат работы кода на примерах из текста задачи:

```
task4.py  input.txt  output.txt
1 14
2 put zero a
3 put one b
4 put two c
5 put three d
6 put four e
7 get two
8 prev two
9 next two
10 delete one
11 delete three
12 get two
13 prev two
14 next two
15 next four
```

```
task4.py  input.txt  output.txt
1 c
2 b
3 d
4 c
5 a
6 e
7 <none>
```

Результат работы кода на максимальных и минимальных значениях:

task4.pyinput.txtoutput.txt

The file size (18,25 MB) exceeds the configured limit (2

1

500000

2

put aaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaa

3

get aaaaaaaaaaaaaaaaaaaaaa

4

put aaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaa

5

get aaaaaaaaaaaaaaaaaaaaaa

task4.pyinput.txtoutput.txt

The file size (5,5 MB) exceeds the configured limit

1

aaaaaaaaaaaaaaaaaaaaaa

2

aaaaaaaaaaaaaaaaaaaaaa

3

aaaaaaaaaaaaaaaaaaaaaa

4

aaaaaaaaaaaaaaaaaaaaaa

task4.pyinput.txtoutput.txt

1

1

2

get a

task4.pyinput.txtoutput.txt

1

<none>

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.069730 сек	6.424201 MB
Пример из задачи	0.066277 сек	6.425986 MB
Верхняя граница диапазона значений входных данных из	1.340868 сек	65.660285 MB

Вывод по задаче:

В ходе выполнения этой задачи я узнала, что такое ассоциативный массив, реализовала алгоритм, поддерживающий различные операции.

Задача №6. Фибоначчи возвращается

Вам дается последовательность чисел. Для каждого числа определите, является ли оно числом Фибоначчи. Напомним, что числа Фибоначчи определяются, например, так:

$$F_0 = F_1 = 1$$

$$F_i = F_{i-1} + F_{i-2} \text{ для } i \geq 2$$

Листинг кода:

```
import tracemalloc
import time
import sys

sys.set_int_max_str_digits(5000)

tracemalloc.start()

def is_fibonacci(n, fib_set):
    return n in fib_set

def generate_fibonacci_up_to_limit(limit):
    fib_set = set()
    a, b = 1, 1
    fib_set.add(a)
    fib_set.add(b)
    while b <= limit:
        a, b = b, a + b
        fib_set.add(b)
    return fib_set

with open('input.txt', 'r') as f:
    n = int(f.readline())
    numbers = [int(f.readline().strip()) for _ in range(n)]

start_time = time.perf_counter()

max_limit = 10 ** 5000
fib_set = generate_fibonacci_up_to_limit(max_limit)
result = ["Yes" if is_fibonacci(num, fib_set) else "No" for num in
numbers]

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()
```

```

with open('output.txt', 'w') as f:
    f.write("\n".join(result))

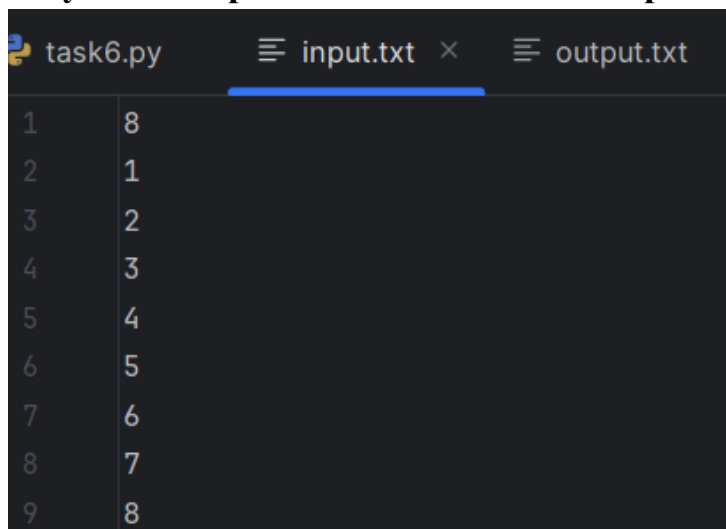
print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

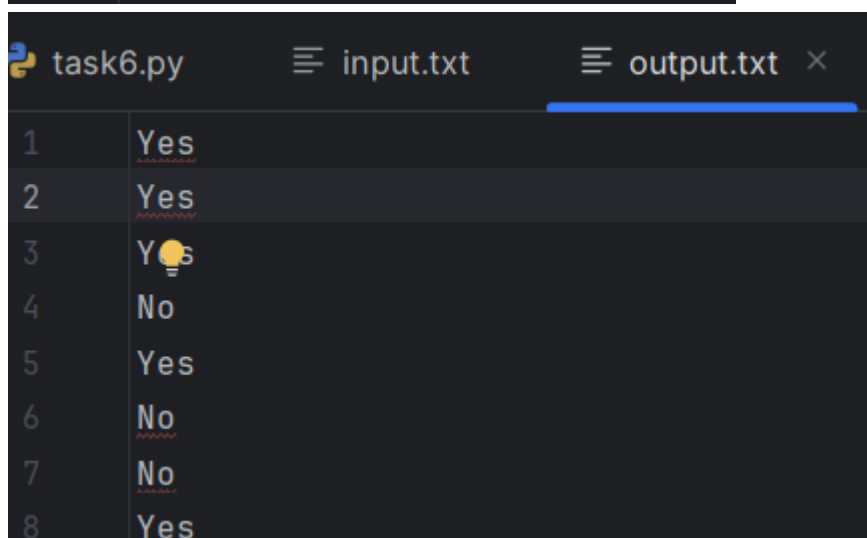
Текстовое объяснение решения:

Я создала метод, который генерирует числа Фибоначчи до определенного лимита (в нашем случае $10^{*}5000$, так как число в десятичной форме должно состоять не более чем из 5000 цифр). Здесь все числа Фибоначчи сохраняются в множество. Метод `is_fibonacci` проверяет, находится ли указанное число в множестве, найденном методом `generate_fibonacci_up_to_limit`.

Результат работы кода на примерах из текста задачи:



task6.py	input.txt	output.txt
1	8	
2	1	
3	2	
4	3	
5	4	
6	5	
7	6	
8	7	
9	8	



task6.py	input.txt	output.txt
1		Yes
2		Yes
3		Yes
4		No
5		Yes
6		No
7		No
8		Yes

Результат работы кода на максимальных и минимальных значениях:

```
task6.py  input.txt  output.txt
1 282
2 77672415689604263875460251380930
3 92749120955585779701932982061626
4 66957013849555455018299698988552
5 92755283261525968701558462674419
6 67450113899957575082475424710683
```

```
task6.py  input.txt  output.txt
1 Yes
2 No
3 Yes
4 No
5 Yes
```

```
task6.py  input.txt  output.txt
1 1
2 1
```

```
task6.py  input.txt  output.txt
1 Yes
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.039436 сек	29.330795 MB
Пример из задачи	0.039383 сек	29.330859 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.049550 сек	29.807395 MB

Вывод по задаче:

В ходе выполнения этой задачи я смогла реализовать эффективный алгоритм для определения чисел Фибоначчи согласно условиям.

Вывод

В ходе выполнения данной лабораторной работы я ознакомилась с такими понятиями, как множество, словари, хеш-таблицы и хеш-функции, а также смогла реализовать алгоритмы, основанные на принципах их работы.