

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

Отчет по лабораторной работе №6
по курсу «Алгоритмы и структуры данных»

Тема: Хеширование. Хеш-таблицы.

Вариант 12

Выполнил:

Колпаков А.С.

К3139

Проверил:

Афанасьев А.В

Санкт-Петербург

2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Множество	3
Задача №2. Телефонная книга	7
Дополнительные задачи	11
Задача №5. Выбора в США	11
Задача №6. Фибоначчи возвращается	14
Задача №8. Почти интерактивная хэш-таблица	19
Вывод	22

Задачи по варианту

Задача №1. Множество

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

- **Формат входного файла (input.txt).** В первой строке входного файла находится строго положительное целое число операций N , не превышающее $5 \cdot 10^5$. В каждой из последующих N строк находится одна из следующих операций:
 - $A\ x$ – добавить элемент x в множество. Если элемент уже есть в множестве, то ничего делать не надо.
 - $D\ x$ – удалить элемент x . Если элемента x нет, то ничего делать не надо.
 - $?\ x$ – если ключ x есть в множестве, выведите «Y», если нет, то выведите «N».

Аргументы указанных выше операций – **целые числа**, не превышающие по модулю 10^{18} .

- **Формат выходного файла (output.txt).** Выведите последовательно результат выполнения всех операций «?». Следуйте формату выходного файла из примера.

Листинг кода:

```
import utils

def process_operations(n, data):
    result = []
    s = set()

    for i in range(n):
        line = data[i]
        operation = line[0]
        x = int(line[1])

        if operation == 'A':
```

```

        s.add(x)
    elif operation == 'D':
        s.discard(x)
    elif operation == '?':
        if x in s:
            result.append('Y\n')
        else:
            result.append('N\n')

    return result


if __name__ == '__main__':
    data =
utils.read_data('lab6/task1/textf/input.txt')
    res = process_operations(data[0], data[1:])
    utils.print_task_data(6, 1, data, res)
    utils.write_file("lab6/task1/textf/output.txt",
res)

```

Текстовое объяснение решения:

1. Проходимся по заданным значениям циклом.
2. Если команда равна A добавляем элемент, если D удаляем элемент, если команда равна ?, добавляем в массив результатов Y, если значение есть в массиве s, N, если значения нет в массиве N.

Результат работы кода на примерах из текста задачи:

 input.txt U

itmo > algo > lab6 > task1

1 8

2 A 2

3 A 5

4 A 3

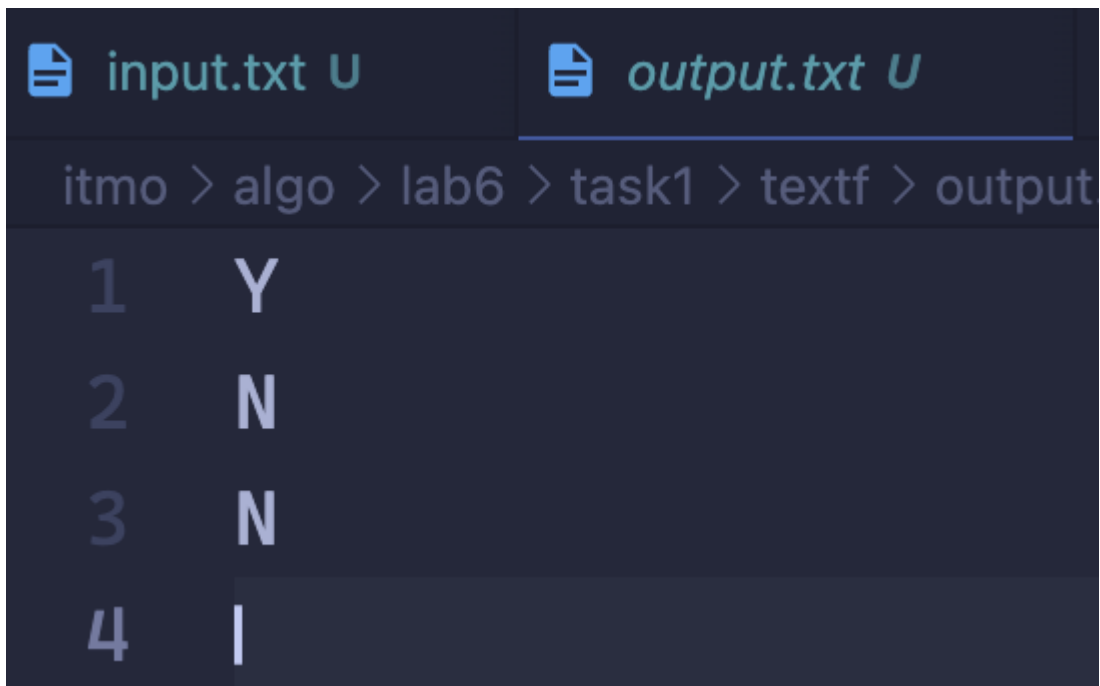
5 ? 2

6 ? 4

7 A 2

8 D 2

9 ? 2



```
lab 6 task 1
input: [8, ['A', '2'], ['A', '5'], ['A', '3'], ['?', '2'], ['?', '4'], ['A', '2'], ['D', '2'], ['?', '2']]
output: ['Y\n', 'N\n', 'N\n']
```

	Время выполнения	Затраты памяти
Пример из задачи	0.000506292009958997 4 секунд	0.005377769470214844 МБ

Вывод по задаче:

Программа успешно реализует множество с операциями «добавление ключа», «удаление ключа», «проверка существования ключа» .

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 utils

def manage_phonebook(data):
    phonebook = {}
    result = []

    for query in data:
        command = query[0]

        if command == 'add':
            number = query[1]
            name = query[2]
            phonebook[number] = name

        elif command == 'del':
            number = query[1]
            if number in phonebook:
                del phonebook[number]
```

```

        elif command == 'find':
            number = query[1]
            if number in phonebook:
                result.append(phonebook[number] +
'\n')

            else:
                result.append('not found\n')

        return result


if __name__ == '__main__':
    data =
utils.read_data('lab6/task2/textf/input.txt')
    res = manage_phonebook(data[1:])
    utils.print_task_data(6, 2, data, res)
    utils.write_file("lab6/task2/textf/output.txt",
res)

```

Текстовое объяснение решения:

1. Задаем пустой словарь, в котором будут добавлены значения телефонных номеров.
2. Если команда = 'add', добавляем ключ телефона и значение имени человека в словарь.
3. Если команда = 'del', удаляем значение из словаря.
4. Если команда = 'find', то добавляем в результат значение имени, найденное по ключу в словаре.

Результат работы кода на примерах из текста задачи:

 *input.txt* U

itmo > algo > lab6 > task2 > textf > input.tx

```
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
```

```
output.txt U
itmo > algo > lab6 > task2 > t
1 Mom
2 not found
3 police
4 not found
5 Mom
6 daddy
7
```

```
lab 6 task 2
input: [12, ['add', '911', 'police'], ['add', '76213', 'Mom'], ['add', '17239', 'Bob'], ['find', '76213'], ['find', '910'], ['find', '911'], ['del', '910'], ['del', '911'], ['find', '911'], ['find', '76213'], ['add', '76213', 'daddy'], ['find', '76213']]
output: ['Mom\n', 'not found\n', 'police\n', 'not found\n', 'Mom\n', 'daddy\n']
```

	Время выполнения	Затраты памяти
Пример из задачи	0.000299833016470074 65 секунд	0.005731582641601562 5 МБ

Вывод по задаче:

Программа эффективно реализует алгоритм работы простого менеджера телефонной книги.

Дополнительные задачи

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 utils

def process_elections(data):
    votes = {}

    for line in data:
        candidate, vote_count = line

        if candidate in votes:
            votes[candidate] += int(vote_count)
        else:
            votes[candidate] = int(vote_count)

    sorted_candidates = sorted(votes.items())

    res = []
    for candidate, votes in sorted_candidates:
        res.append(f'{candidate} {votes}\n')
```


```
    return res

if __name__ == '__main__':
    data =
utils.read_data('lab6/task5/textf/input.txt')
    res = process_elections(data)
    utils.print_task_data(6, 5, data, res)
    utils.write_file("lab6/task5/textf/output.txt",
res)
```

Текстовое объяснение решения:

1. Создаем пустой словарь для подсчета количества голосов.
2. Проходим циклом по заданным данным и добавляем значение голосов по ключу кандидата в словарь.
3. В конце сортируем по алфавиту.

Результат работы кода:

 *input.txt* U

itmo > algo > lab6 > task5 > te

1 **ivanov 100**


2 **ivanov 500**

3 **ivanov 300**

4 **petr 70**

5 **tourist 1**

6 **tourist 2**

 *output.txt* U

itmo > algo > lab6 > task5 > textf

1 **ivanov 900**

2 **petr 70**

3 **tourist 3**

//

```
lab 6 task 5
input: [['ivanov', '100'], ['ivanov', '500'], ['ivanov', '300'], ['petr', '70'], ['tourist', '1'], ['tourist', '2']]
output: ['ivanov 900\n', 'petr 70\n', 'tourist 3\n']
```

	Время выполнения	Затраты памяти
Пример из задачи	0.0006839580019004643 секунд	0.0052337646484375 МБ

Вывод по задаче:

Программа успешно проводит подсчет количества отданных голосов за каждого кандидата.

Задача №6. Фибоначчи возвращается

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

$$\begin{aligned} F_0 &= F_1 = 1 \\ F_i &= F_{i-1} + F_{i-2} \text{ для } i \geq 2. \end{aligned} \quad (1)$$

- **Формат ввода / входного файла (input.txt).** Первая строка содержит одно число N ($1 \leq N \leq 10^6$) - количество запросов. Следующие N строк содержат по одному целому числу. При этом соблюдаются следующие ограничения при проверке:
 1. Размер каждого числа не превосходит 5000 цифр в десятичном представлении.
 2. Размер входа не превышает 1 Мб.
- **Формат вывода / выходного файла (output.txt).** Для каждого числа, данного во входном файле, выведите «Yes», если оно является числом Фибоначчи, и «No» в противном случае.

Листинг кода:

```
import utils

def is_fibonacci(num):
    x1 = 5 * num ** 2 + 4
    x2 = 5 * num ** 2 - 4
```

```
    return is_perfect_square(x1) or  
is_perfect_square(x2)
```

```
def is_perfect_square(x):
```

```
    if x < 0:
```

```
        return False
```

```
    left, right = 0, x
```

```
    while left <= right:
```

```
        mid = (left + right) // 2
```

```
        square = mid * mid
```

```
        if square == x:
```

```
            return True
```

```
        elif square < x:
```

```
            left = mid + 1
```

```
        else:
```

```
            right = mid - 1
```

```
    return False
```

```
def process_fibonacci(data):
```

```
    results = []
```

```
    for num in data:
```

```
        if is_fibonacci(num):
```

```
            results.append("Yes\n")
```

```
        else:
```

```
        results.append("No\n")


    return results

if __name__ == '__main__':
    data =
utils.read_data('lab6/task6/textf/input.txt')
    res = process_fibonacci(data[1:])
    utils.print_task_data(6, 6, data, res)
    utils.write_file("lab6/task6/textf/output.txt",
res)
```

Текстовое объяснение решения:

1. Задаем функцию `is_fibonacci`, в которой проверяем, является ли число числом фибоначчи. Для этого какое-либо из чисел x_1 x_2 должно являться полным квадратом.
2. Следующей функцией задаем проверку числа на полный квадрат.
3. После чего задаем функцию-обработчик входных данных, которая проходит циклом по заданному массиву данных и выявляет, является ли число числом фибоначчи, при этом записывая результаты в массив.

Результат работы кода на примерах из текста задачи:

 *input.txt* U

itmo > algo > lab6 > task6 >

1 8

2 1

3 2

4 3

5 4

6 5

7 6

8 7

9 8

```
output.txt U
itmo > algo > lab6 > task6 >
1  Yes
2  Yes
3  Yes
4  No
5  Yes
6  No
7  No
8  Yes
```

```
lab 6 task 6
input: [8, 1, 2, 3, 4, 5, 6, 7, 8]
output: ['Yes\n', 'Yes\n', 'Yes\n', 'No\n', 'Yes\n', 'No\n', 'No\n', 'Yes\n']
```

	Время выполнения	Затраты памяти
Пример из задачи	0.000903249994735233 5 секунд	0.0052032470703125 МБ

Вывод по задаче:
Программа эффективно выполняет алгоритм выявления, является ли заданное число числом фибоначчи.

Задача №8. Почти интерактивная хеш-таблица

В данной задаче у Вас не будет проблем ни с вводом, ни с выводом. Просто реализуйте быструю хеш-таблицу.

В этой хеш-таблице будут храниться целые числа из диапазона $[0; 10^{15} - 1]$. Требуется поддерживать добавление числа x и проверку того, есть ли в таблице число x . Числа, с которыми будет работать таблица, генерируются следующим образом. Пусть имеется четыре целых числа N, X, A, B такие что:

- $1 \leq N \leq 10^7$
- $1 \leq X \leq 10^{15}$
- $1 \leq A \leq 10^3$
- $1 \leq B \leq 10^{15}$

Требуется N раз выполнить следующую последовательность операций:

- Если X содержится в таблице, то установить $A \leftarrow (A + A_C) \bmod 10^3, B \leftarrow (B + B_C) \bmod 10^{15}$.
- Если X не содержится в таблице, то добавить X в таблицу и установить $A \leftarrow (A + A_D) \bmod 10^3, B \leftarrow (B + B_D) \bmod 10^{15}$.
- Установить $X \leftarrow (X \cdot A + B) \bmod 10^{15}$.

Начальные значения X, A и B , а также N, A_C, B_C, A_D и B_D даны во входном файле. Выведите значения X, A и B после окончания работы.

Листинг кода:

```
import utils

def solve_hash(data):
    N, X, A, B = data[0]
    AC, BC, AD, BD = data[1]

    hash_table = set()

    for _ in range(N):
        if X in hash_table:
            A = (A + AC) % 10**3
            B = (B + BC) % 10**15
```

```

        else:
            hash_table.add(X)
            A = (A + AD) % 10**3
            B = (B + BD) % 10**15

            X = (X * A + B) % 10**15

        return f'{X} {A} {B}'

if __name__ == '__main__':
    data =
utils.read_data('lab6/task8/textf/input.txt')
    res = solve_hash(data)
    utils.print_task_data(6, 8, data, res)
    utils.write_file("lab6/task8/textf/output.txt",
[res])

```

Текстовое объяснение решения:

1. Обрабатываем входящие данные и задаем необходимые переменные.
2. Далее проходим циклом по всем значениям, если X есть в хэш таблице, то обновляем A и B, в зависимости от As и Bs.
3. Если X нет в таблице, то добавляем его в таблицу и обновляем A и B, в зависимости от Ad и Bd.
4. В конце обновляем X и возвращаем значения X A B.

Результат работы кода на примерах из текста задачи:

```
input.txt U
itmo > algo > lab6 > task
1 4 0 0 0
2 1 1 0 0
```

```
output.txt U
itmo > algo > lab6 >
1 3 1 1
```

```
lab 6 task 8
input: [[4, 0, 0, 0], [1, 1, 0, 0]]
output: 3 1 1
```

	Время выполнения	Затраты памяти
Пример из задачи	0.000903249994735233 5 секунд	0.0052032470703125 МБ

Вывод по задаче:

Программа эффективно выполняет алгоритм быстрой хэш-таблицы.

Вывод

В ходе лабораторной работы были изучены и реализованы базовые операции с такими структурами данных, как множества и словари. Также была реализована работа с хеш-таблицами и хеш-функциями, что позволило ознакомиться с их основными принципами, эффективностью и применением. Полученные знания закреплены на практике через выполнение задач, требующих использования данных структур для решения алгоритмических проблем.