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 3

Выполнила:
Блинова П. В.
К3139 (номер группы)

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №1. Множество

Текст задачи.

1 задача. 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.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
8	Y
A 2	N
A 5	N
A 3	
? 2	
? 4	
A 2	
D 2	
? 2	

Листинг кода.

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

    for i in range(n):
        line = data[i].split()
        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')
            else:
                result.append('N')

    return result
```

Текстовое объяснение решения.

Функция fibonacci_check моделирует работу с множеством (set) для обработки команд добавления, удаления и запроса элементов. Она принимает на вход: • n: количество команд. • data: список строк, где каждая строка представляет команду в формате "Операция Значение". "Операция" может быть 'A' (добавить), 'D' (удалить) или '?' (запрос), а "Значение" — целое число. Функция использует множество s для хранения элементов и список result для результатов запросов. Цикл перебирает команды: • 'A': добавляет значение в множество s. • 'D': удаляет значение из множества s. • '?': проверяет наличие значения в множестве s и добавляет 'Y' (есть) или 'N' (нет) в result. Функция возвращает список result.

Результат работы кода на примерах из текста задачи:

```
Входные данные:
8
A 2
A 5
A 3
? 2
? 4
A 2
D 2
? 2
Результат:
Y
N
N
Время работы: 3.900006413459778e-05
Память: 0.00045013427734375 Мб
```

Вывод по задаче:

В ходе работы над задачей мной была изучена работа с множествами.

Задача №2. Телефонная книга

Текст задачи.

2 задача. 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.
- **Формат ввода / входного файла (input.txt).** В первой строке входного файла содержится число N ($1 \leq N \leq 10^5$) - количество запросов. Далее следуют N строк, каждая из которых содержит один запрос в формате, описанном выше.

  Все номера телефонов состоят из десятичных цифр, в них нет нулей в начале номера, и каждый состоит не более чем из 7 цифр. Все имена представляют собой непустые строки из латинских букв, каждая из которых имеет длину не более 15. Гарантируется при проверке, что не будет человека с именем «not found».
- **Формат вывода / выходного файла (output.txt).** Выведите результат каждого поискового запроса `find` – имя, соответствующее номеру телефона, или «not found» (без кавычек), если в телефонной книге нет человека с таким номером телефона. Выведите по одному результату в каждой строке в том же порядке, как были заданы запросы типа `find` во входных данных.
- Ограничение по времени. 6 сек.
- Ограничение по памяти. 512 мб.
- Примеры:

Листинг кода.

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

    for query in data:
        command = query.split()[0]

        if command == 'add':
            phonebook[query.split()[1]] = query.split()[2]
```

```
        elif command == 'del':
            number = query.split()[1]
            if number in phonebook:
                del phonebook[number]

        elif command == 'find':
            number = query.split()[1]
            if number in phonebook:
                result.append(phonebook[number])
            else:
                result.append('not found')

    return result
```

Текстовое объяснение решения.

Функция manage_phonebook моделирует работу телефонной книги с использованием словаря Python. Она принимает на вход список команд (data), где каждая команда представляет собой строку одного из трёх типов: • add имя номер: добавить запись в телефонную книгу. • del имя: удалить запись из телефонной книги. • find имя: найти номер по имени. Функция использует словарь phonebook для хранения записей (ключ — имя, значение — номер) и список result для хранения результатов запросов find. Цикл перебирает команды: • add: добавляет новую запись в phonebook. • del: удаляет запись из phonebook, если имя существует. • find: ищет имя в phonebook. Если имя найдено, в result добавляется соответствующий номер; иначе — строка 'not found'. Функция возвращает список result.

Результат работы кода на примерах из текста задачи:

```
Входные данные:
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
Результат:
Mom
not found
police
not found
Mom
daddy
Время работы: 5.91999851167202e-05
Память: 0.0007581710815429688 Мб
```

Вывод по задаче:

В ходе работы над задачей мной была алгоритм написания задач в телефонной книге.

Задача №5. Выборы США

Текст задачи.

5 задача. 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.

- **Формат ввода / входного файла (input.txt).** Каждая строка входного файла содержит фамилию кандидата, за которого отдают голоса выборщики этого штата, затем через пробел идет количество выборщиков, отдавших голоса за этого кандидата.
- **Формат вывода / выходного файла (output.txt).** Выведите фамилии всех кандидатов в ***лексикографическом*** порядке, затем, через пробел, количество отданных за них голосов.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 64 мб.
- Примеры:

Листинг кода.

```
def process_elections(candidates_input):
    votes = defaultdict(int)

    for line in candidates_input:
        candidate, vote_count = line.split()
        votes[candidate] += int(vote_count)

    sorted_candidates = sorted(votes.items())

    result = []
    for candidate, votes in sorted_candidates:
        result.append(f'{candidate} {votes}')

    return result
```

Текстовое объяснение решения.

Функция process_elections принимает на вход список строк, где каждая строка содержит имя кандидата и количество голосов, полученных им. Функция выполняет следующие шаги: 1. Инициализация: Создается словарь votes с помощью defaultdict(int), который будет использоваться для хранения кандидатов и их голосов. defaultdict позволяет автоматически инициализировать значения для новых ключей, в данном случае — с нулем. 2. Сбор голосов: Для каждой строки входных данных: - Строка делится на две части: имя кандидата и количество голосов с помощью метода split(). - Количество голосов преобразуется в целое число и добавляется к текущему счетчику голосов кандидата в словаре votes. 3. Сортировка кандидатов: Кандидаты сортируются в алфавитном порядке с помощью функции sorted(), которая принимает items() словаря votes. Это возвращает список кортежей, где каждый кортеж состоит из имени кандидата и соответствующего количества голосов. 4. Формирование результата: Создается пустой список result, в который добавляются строки, содержащие имя кандидата и количество голосов в формате 'имя количество_голосов'. Это достигается с помощью цикла for. 5. Возврат результата: Функция возвращает список result, который содержит информацию о кандидатах и полученных ими голосах.
Результат работы кода на примерах из текста задачи:

```
Входные данные:
McCain 10
McCain 5
Obama 9
Obama 8
McCain 1
Результат:
McCain 16
Obama 17
Время работы: 6.9299945607781 41e-05
Память: 0.0004138946533203125 Мб
```

Вывод по задаче:
В ходе работы над задачей мной была алгоритм голосования.

Задача №6. Фибоначчи возвращается

Текст задачи.

6 задача. 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.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 128 мб. ***Внимание:*** *есть вероятность превышения по памяти, т.к. сами по себе числа Фибоначчи большие. Делайте проверку на память!*
- Примеры:

Листинг кода.

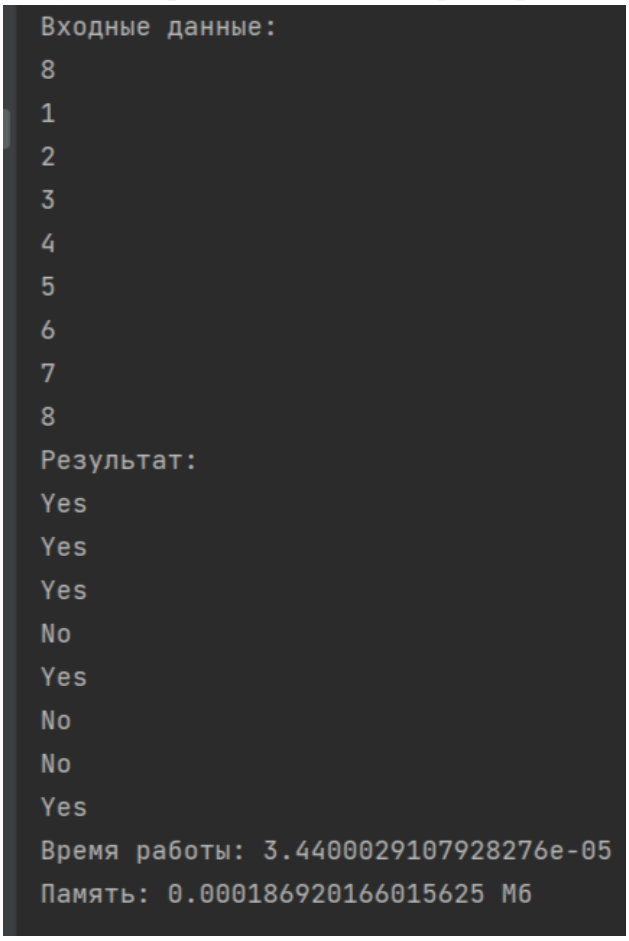
```
def is_fibonacci_number(number):
    x = int(number)
    f1 = 5 * x ** 2 + 4
    f2 = 5 * x ** 2 - 4
    return math.isqrt(f1) ** 2 == f1 or math.isqrt(f2) ** 2 == f2


def fibonacci_check(arr):
    result = []
    for number in arr:
        result.append("Yes" if is_fibonacci_number(number) else "No")
    return result
```

Текстовое объяснение решения.

Функция is_fibonacci_number эффективно проверяет, является ли данное число числом Фибоначчи, используя указанное математическое свойство. Функция fibonacci_check обрабатывает массив чисел, применяя

is_fibonacci_number к каждому элементу и возвращая список строк "Yes" или "No".

Результат работы кода на примерах из текста задачи:



Вывод по задаче:

В ходе работы над задачей мной был изучен алгоритм определения числа Фибоначчи.

Вывод

В ходе лабораторной работы был изучен метод работы со следующим: множество, словари, хеш-таблицы и хеш-функции.