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

Выполнила:
Мкртчян.К.Г.
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Оглавление

Задачи по варианту

Задача №1. Множество. [N баллов]

Текст задачи:

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 · 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.

Решение:

```
from lab6.src.utils import File
from llist import sllist


class HashTable:
    def __init__(self, array):
        self.table = {}
        self.answer = []
        self.array = array

    def add_key(self, key):
        h = hash(key)
        if h in self.table.keys():
            self.table[h].append(key)
        else:
            self.table[h] = sllist(key)

    def delete_key(self, key):
        h = hash(key)
        if h in self.table.keys():
            self.table.pop(h)
        else:
            pass

    def check_key(self, key):
        h = hash(key)
        if h in self.table.keys() and key in self.table[h]:
            self.answer += ['Y']
        else:
            self.answer += ['N']
```

```
    def actions(self):
        for a in self.array:
            command, key = a.split(" ")
            if command == "arr_a":
                self.add_key(key)
            elif command == "D":
                self.delete_key(key)
            elif command == "?":
                self.check_key(key)
            else:
                pass
        return self.answer
```

Объяснение решения:

Класс HashTable реализует хэш-таблицу с использованием связанных списков через библиотеку llist. Вот основные компоненты и функциональность класса:

Структура класса:

1) Инициализация (__init__):

self.table: создается пустой словарь, который будет использоваться для хранения хэш-значений и связанных с ними ключей.

self.answer: список, который будет хранить ответ на запросы о наличии ключей.

self.array: массив, содержащий команды и ключи, с которыми будет работать хэш-таблица.

2) Добавление ключа (add_key):

Метод принимает key, вычисляет его хэш через встроенную функцию hash(). Если хэш уже существует в словаре self.table, метод добавляет ключ к связанному списку, который ассоциирован с этим хэшем (используя метод append связанного списка). Если хэша нет, создается новый связанный список, и ключ добавляется в него.

3) Удаление ключа (delete_key):

Этот метод также вычисляет хэш ключа. Если хэш существует в словаре, он удаляется вместе с ключом (но фактически удаляется только сама запись по хэшу, без удаления конкретного ключа из связанного списка); реализация удаления конкретного ключа из списка не предусмотрена.
Если хэш не найден, не происходит никаких действий.

4) Проверка ключа (check_key):

Этот метод проверяет, существует ли хэш для данного ключа в словаре, и содержит ли связанный список этот ключ. Если ключ найден, в self.answer добавляется 'Y', в противном случае добавляется 'N'.

5) Обработка команд (actions):

Этот метод перебирает массив команд/ключей (self.array), разбивает каждую строку на команду и ключ, и в зависимости от команды вызывает соответствующий метод: добавляет, удаляет или проверяет ключ. В конце метод возвращает список ответов.

Время выполнения программы, затраты памяти и результат:

Lab #6 | Task #1 - Test Table

	данные	время, сек.	память, МБ	результат
Значения из примера	6	0.0	31.58	N
	arr_a 5			Y
	arr_a 3			
	D 2			
	D 3			
	? 3			
	? 5			
Значения из примера	13	0.0	31.6	N
	arr_a 5			Y
	arr_a 3			Y
	D 2			N
	D 3			N
	? 3			
	? 5			
	arr_a 8			
	arr_a 8			
	? 8			
	D 8			
	? 8			
	D 8			
	? 8			
Значения из примера	500000	1.18	72.46	N
	D 6913365326963498			N
	D 751788980556308591			N
	arr_a -609493958545722312			N
	D -497495063204917783			

Вывод по задаче: Мы узнали, как работают хэш-таблицы, включая использование хэш-функций для преобразования ключей в индексы, что позволяет эффективно хранить и извлекать данные.

Дополнительные задачи

Задача №2. Телефонная книга [N баллов]

Текст задачи:

- 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.

Решение:

```
class PhoneBook:
    def __init__(self, actions: list[str]) -> None:
        self.book = {}
        self.answer = []
        self.actions = actions

    def add_key(self, phone: int, name: str) -> None:
        h = hash(phone)
        self.book[h] = name


    def delete_key(self, phone: int) -> None:
        h = hash(phone)
        if h in self.book.keys():
            self.book.pop(h)
        else:
            pass

    def check_key(self, phone: int) -> None:
        h = hash(phone)
        if h in self.book.keys():
            self.answer += [self.book[h]]
        else:
            self.answer += ['not found']

    def distribute_actions(self) -> list[str]:
        for a in self.actions:
            arguments = a.split(" ")
            command, phone = arguments[0], int(arguments[1])
```

```
            if command == "add":
                name = arguments[2]
                self.add_key(phone, name)
            elif command == "del":
                self.delete_key(phone)
            elif command == "find":
                self.check_key(phone)
            else:
                pass
        return self.answer
```

Объяснение решения:

Класс PhoneBook реализует простую телефонную книгу и предоставляет методы для добавления, удаления и поиска записей по номерам телефонов. Давайте проанализируем его структуру и функциональность.

Структура класса

1) Инициализация (__init__):
   self.book: это словарь, который будет хранить записи телефонной книги. Ключом служит хэш номера телефона, а значением — имя абонента.
   self.answer: список, который будет хранить ответы на запросы (например, результат поиска).
   self.actions: список действий, которые будут выполняться с телефонной книгой.
2) Метод добавления записи (add_key):
   Принимает номер телефона и имя.
   Вычисляет хэш от номера телефона и добавляет запись в словарь self.book, где ключ — это хэш, а значение — имя абонента.
3) Метод удаления записи (delete_key):
   Принимает номер телефона. Вычисляет хэш и, если он существует в словаре, удаляет запись.
4) Метод поиска записи (check_key):
   Принимает номер телефона. Вычисляет хэш и проверяет, существует ли запись в словаре. Если запись найдена, добавляет имя в self.answer, иначе добавляет строку 'not found'.
5) Метод распределения действий (distribute_actions):
   Перебирает все действия (команды) из self.actions. Для каждой команды разбивает строку на части и определяет, какое действие (добавление, удаление, поиск) нужно выполнить. Вызывает соответствующий метод для выполнения действия. Возвращает список ответов после обработки всех действий.

Время выполнения, затраты памяти и результат:

Lab #6 | Task #2 - Test Table

	данные	время, сек.	память, МБ	результат
Значения из примера	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	0.0	31.73	Mom not found police not found Mom daddy
Значения из примера	8 find 3839442 add 123456 me add 0 granny find 0 find 123456 del 0 del 0 find 0	0.0	31.75	not found granny me not found
Максимальные значения	100000 add 5601123 ablurjxqlanrdpc add 3358420 oaoifltvtrshwgs add 4853746 gdzqkngyiehamih add 8962735 wncxjcypohemdnq add 3668965 tmomcfbiseaadlr add 9066544 tfcdmpvugjixuuu	0.03	46.21	gdzqkngyiehamih wuvukdmxbgfyzeo zkjecsocuraniwn mfyhtwxtbtfzbeu zbugkrssoyfjeuf sgkoqhrqgejdled bvqzmrcpqdgnkws

Вывод по задаче: в результате выполнения этой задачи мы изучили основные принципы работы с хэш-таблицами, освоили методы добавления, удаления и поиска элементов, а также увидели, как организовать и обрабатывать действия с использованием классов в Python.

Задача №5. Выборы в США. [N баллов]

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

Решение:

```
def elections(states_result: list[str]):
    voices_sum = {}
    for data in states_result:
        name, score = data.split(" ")
        if name in voices_sum.keys():
            voices_sum[name] += int(score)
        else:
            voices_sum[name] = int(score)

    sorted_v_sum = sorted(voices_sum.items())
    str_sorted_v_sum = [f'{s[0]} {s[1]}' for s in sorted_v_sum]
    return str_sorted_v_sum
```

Объяснение решения:

Функция "elections" обрабатывает результаты голосования из различных штатов и возвращает список строк, представляющих имена кандидатов и количество полученных голосов, отсортированных по именам кандидатов. Давайте разберем ее по шагам.

Структура функции

Инициализация словаря:

В начале создается пустой словарь, который будет использоваться для хранения суммы голосов для каждого кандидата. Ключами будут имена кандидатов, а значениями — суммированные голоса.

Обработка входных данных:

Функция проходит по каждому элементу списка результатов голосования. Каждый элемент представляет собой строку, содержащую имя кандидата и количество голосов, разделенные пробелом. Метод разбивает строку на две части: имя кандидата и количество голосов.

Суммирование голосов:

Если имя кандидата уже есть в словаре, значение увеличивается на количество голосов. При этом количество голосов конвертируется в целое число. Если имя кандидата отсутствует в словаре, то добавляется новая запись с его именем и количеством голосов.

Сортировка и форматирование результата:

Словарь превращается в список пар (ключ, значение), который сортируется по ключу (т.е. по имени кандидата). Затем с помощью спискового включения формируется новый список, где для каждого кортежа из отсортированного списка создается строка формата "имя голоса".
Возврат результата: в конце функция возвращает отсортированный список строк, представляющих имена кандидатов и количество их голосов.

Время выполнения, затраты памяти и результат:

Lab #6 | Task #5 - Test Table

	данные	время, сек.	память, МБ	результат
Значения из примера	ivanov 100 ivanov 500 ivanov 300 petr 70 tourist 1 tourist 2	0.0	31.83	ivanov 900 petr 70 tourist 3
Значения из примера	McCain 10 McCain 5 Obama 9 Obama 8 McCain 1	0.0	31.84	McCain 16 Obama 17
Значения из примера	bur 1	0.0	31.84	bur 1
Максимальные значения	jcs 56665 oqk 17940 cnh 88318 yvc 76896 djt 97681 yya 61950 kxn 68221 jfd 23515 upr 51184 jwk 81452	0.04	43.05	aaa 274306 aab 467984 aac 253448 aad 268158 aae 339491 aaf 319341 aag 218056 aah 359110 aai 177640 aaj 128078

Вывод по задаче: Мы узнали, как использовать словари (реализация которых в Python схожа с хэш-таблицей) для хранения и агрегации данных. Словари позволяют эффективно хранить пары "ключ-значение", что удобно для суммирования голосов по кандидатам.

Задача №6. Фибоначчи возвращается. [N баллов]

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

$F0 = F1 = 1$

$Fi = Fi-1 + Fi-2$ для $i \geq 2$.

- 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.

Решение:

```
def is_perfect_square(x):
    s = int(math.isqrt(x))
    return s * s == x


def is_fib(n: int) -> str:
    if is_perfect_square(5 * n * n + 4) or is_perfect_square(5 * n * n - 4):
        return "Yes"
    else:
        return "No"
```

Объяснение решения:

Функции is_perfect_square и is_fib работают следующим образом:

Функция is_perfect_square(x):

- Принимает целое число x в качестве аргумента.
- Использует функцию math.isqrt(x), чтобы вычислить целочисленный квадратный корень из x. Это означает, что функция возвращает наибольшее целое число s, такое что s * s не превышает x.
- Затем функция проверяет, является ли квадрат этого целочисленного корня равным x. Если да, то x является совершенным квадратом, и функция возвращает True. В противном случае возвращает False.

Функция is_fib(n: int) -> str:

- Принимает целое число n в качестве аргумента.
- Для проверки, является ли n числом Фибоначчи, функция использует два условия, основанных на свойствах чисел Фибоначчи. Она проверяет, является ли одно из двух выражений $(5\ n^2 + 4)$ или $(5\ n^2 - 4)$ совершенным квадратом, вызывая функцию is_perfect_square.

- Если одно из этих выражений является совершенным квадратом, то n является числом Фибоначчи, и функция возвращает строку "Yes". В противном случае возвращает строку "No".

Время выполнения, затраты памяти и результат:

Lab #6 | Task #6 - Test Table

	данные	время, сек.	память, МБ	результат
Значения из примера	5	0.0	31.54	Yes
Значения из примера	89	0.0	31.55	Yes
Значения из примера	610	0.0	31.56	Yes
Значения из примера	6100	0.0	31.56	No
Максимальные значения	100000...	0.0	31.56	No

Вывод по задаче: мы познакомились с функцией целочисленного квадратного корня math.isqrt и узнали еще об одном свойстве чисел Фибоначчи.

Вывод по лабораторной работе

В ходе лабораторной работы мы узнали, как работают хэш-функции и хэш-таблицы, мы реализовали их и применили для решения различных задач. Также рассмотрели и реализовали разные способы борьбы с коллизиями.