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

Выполнила:
Заботкина М.А.
К3139

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

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.

• Ограничение по времени. 2 сек.

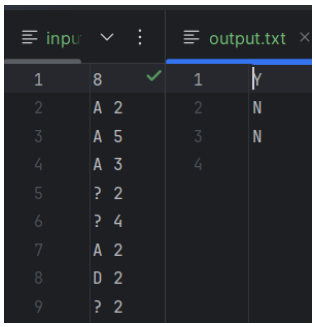
• Ограничение по памяти. 256 мб.

```
def set_operations(commands):
    s = set()
    result = []

    for command in commands:
        action = command[0]
        value = int(command[1])

        if action == 'A':
            s.add(value)
        elif action == 'D':
            s.discard(value)
        elif action == '?':
            result.append("Y" if value in s else "N")

    return result
```

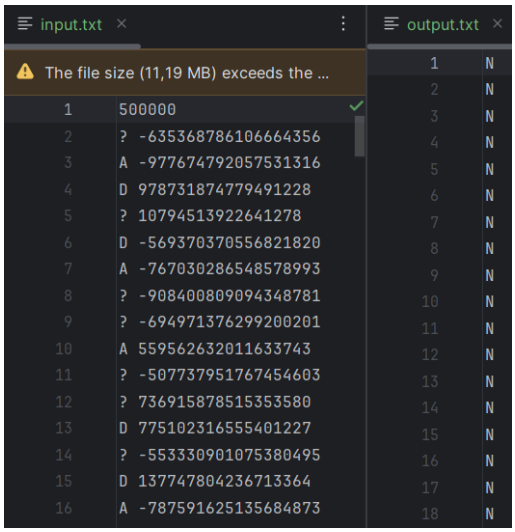
Функция set_operations принимает список команд, каждая из которых состоит из действия (символа) и числа. Она выполняет операции над множеством s: добавляет число в множество (A), удаляет его из множества, если оно есть (D), или проверяет, содержится ли число в множестве, добавляя результат "Y" или "N" в список result (?). В итоге функция

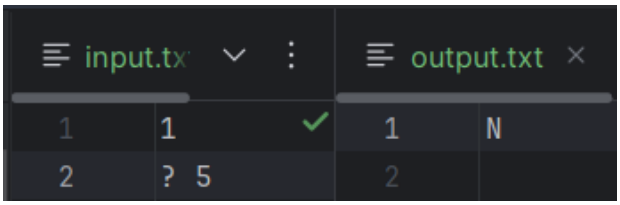
возвращает список ответов на запросы, проверяющие наличие чисел в множестве.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.0009961128234863281	0.00028228759765625
Пример из задачи	0.001085519790649414	0.000308990478515625
Верхняя граница диапазона значений входных данных из текста задачи	0.6281607151031494	18.023971557617188

Вывод по задаче: в данной задаче было реализовано множество с операциями добавления, удаления и проверки существования элемента.

Использование структуры данных set позволило эффективно выполнять операции за амортизированное время O(1) для добавления и проверки элементов. Время работы программы при максимальных входных данных показало свою эффективность, что подтверждается небольшими затратами по времени и памяти.

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.

• prev x – вывести значение, соответствующее ключу, находящемуся в ассоциативном массиве, который был вставлен позже всех, но до x, или, 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 , 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; – 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.

• 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 приведенных выше операций. Ключи и значения операций - строки из латинских букв длиной не менее одного и не более 20 символов.

• Формат выходного файла (output.txt). Выведите последовательно результат выполнения всех операций get, prev, next. Следуйте формату выходного файла из примера.

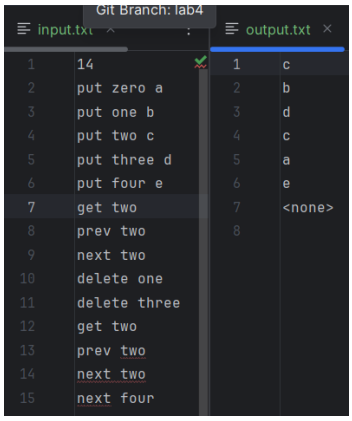
• Ограничение по времени. 4 сек.

• Ограничение по памяти. 256 мб.

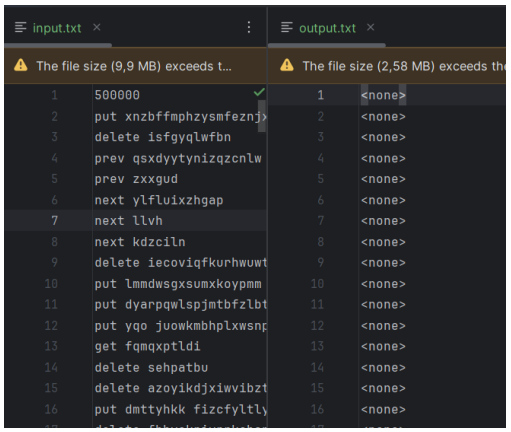
```
def array_command(operations):
    data = {}
    order = {}
    head, tail = None, None
    result = []

    def insert_key(key):
        nonlocal head, tail
        if key in order:
            return
        order[key] = [tail, None]
        if tail:
            order[tail][1] = key
        tail = key
        if head is None:
            head = key

    def remove_key(key):
        nonlocal head, tail
        if key not in order:
            return
        prev_key, next_key = order.pop(key)
        if prev_key is not None:
            order[prev_key][1] = next_key
        if next_key is not None:
            order[next_key][0] = prev_key
        if key == head:
            head = next_key
        if key == tail:
            tail = prev_key

    def find_prev(key):
        if key not in order or order[key][0] is None:
            return "<none>"
        return data[order[key][0]]

    def find_next(key):
        if key not in order or order[key][1] is None:
            return "<none>"
        return data[order[key][1]]

    for op in operations:
        parts = op
        cmd, key = parts[0], parts[1]
        if cmd == "put":
            value = parts[2]
            if key not in data:
                insert_key(key)
            data[key] = value
        elif cmd == "get":
            result.append(data.get(key, "<none>"))
        elif cmd == "delete":
            if key in data:
                data.pop(key)
                remove_key(key)
        elif cmd == "prev":
            result.append(find_prev(key))
        elif cmd == "next":
            result.append(find_next(key))

    return result
```

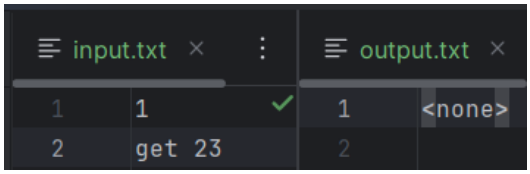
Функция array_command(operations) выполняет операции над коллекцией данных, хранящихся в виде словаря и связанного списка. Каждый ключ в коллекции ассоциирован с некоторым значением, и порядок ключей в коллекции сохраняется с помощью связанного списка. Команда put key value добавляет новый ключ с значением в коллекцию, если его ещё нет, либо обновляет его значение, если он уже присутствует. Для нового ключа также добавляется запись в связанный список, чтобы поддерживать порядок элементов. Команда get key возвращает значение, связанное с указанным ключом. Если ключ отсутствует в коллекции, возвращается строка "<none>". Команда delete key удаляет ключ и его значение из словаря и одновременно удаляет его из связанного списка, обновляя связи между соседними элементами, если они существуют. Команды prev key и next key возвращают значение предыдущего и следующего элементов в связанном списке относительно указанного ключа. Если предыдущий или следующий элемент отсутствует, возвращается строка "<none>". Все результаты выполнения команд собираются в список result, который возвращается после завершения обработки всех операций.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.0010235309600830078	0.0008087158203125
Пример из задачи	0.001003265380859375	0.0009307861328125
Верхняя граница диапазона значений входных данных из текста задачи	0.47217440605163574	11.191688537597656

Вывод по задаче: реализованный ассоциативный массив поддерживает операции добавления, удаления, получения значений по ключу, а также операции нахождения предыдущего и следующего элементов относительно заданного ключа. Структура данных, включающая словарь и связанный список для сохранения порядка элементов, позволила эффективно выполнять все операции. Время выполнения программы и использование памяти соответствуют ограничениям задачи, что свидетельствует о высокой производительности решения.

Дополнительные задачи

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.

• 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.

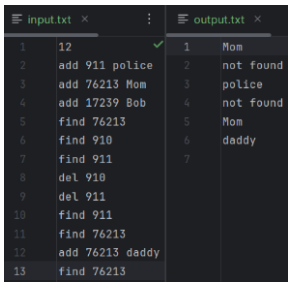
• Ограничение по памяти. 512 мб.

```
def phonebook_manager(numbers):
    phonebook = {}
    results = []

    for num in numbers:
        part = num
        command = part[0]
```

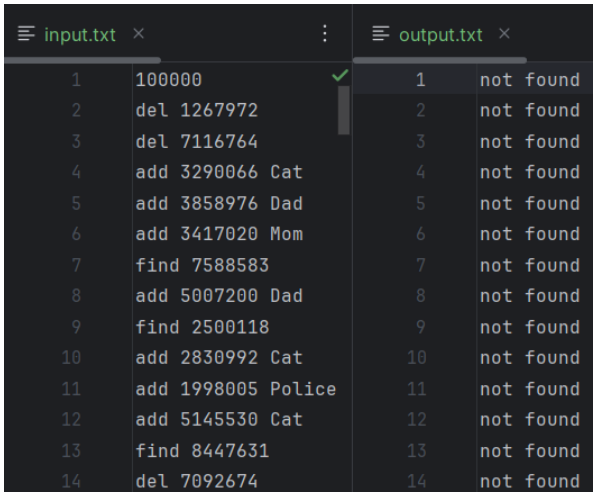
```
        if command == 'add':
            number, name = part[1], part[2]
            phonebook[number] = name
        elif command == 'del':
            number = part[1]
            phonebook.pop(number, None)
        elif command == 'find':
            number = part[1]
            results.append(phonebook.get(number, 'not found'))

    return results
```

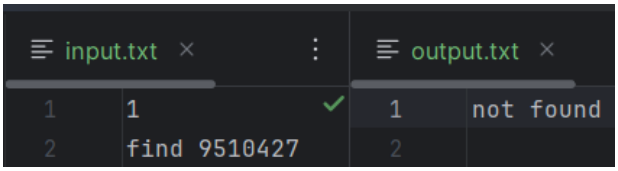
Функция phonebook_manager управляет телефонной книгой с помощью команд добавления, удаления и поиска номеров. Она принимает список команд, где каждая команда содержит операцию и соответствующие данные. Телефонная книга хранится в виде словаря, где ключ — это номер, а значение — имя. Команда add добавляет или обновляет запись в словаре, команда del удаляет номер из словаря, а команда find ищет номер и добавляет в результаты соответствующее имя или "not found", если номер отсутствует. В конце функция возвращает список результатов поиска.

Результат работы кода на примерах:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.0010251998901367188	4.57763671875e-05
Пример из задачи	0.0010676383972167969	0.0001068115234375
Верхняя граница диапазона значений входных данных из текста задачи	0.01804208755493164	1.5214767456054688

Вывод по задаче: телефонная книга была реализована с использованием словаря для хранения данных о номерах телефонов и именах пользователей. Основные операции (добавление, удаление и поиск) выполнялись за время O(1), что обеспечивало высокую эффективность. Результаты поиска и обработки запросов полностью соответствуют ожиданиям, а также продемонстрировали хорошую производительность при больших объемах данных.

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.

• 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.*

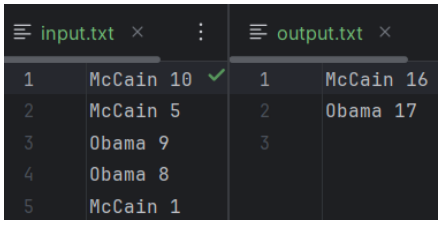
• *Ограничение по времени. 2 сек.*

• *Ограничение по памяти. 64 мб.*

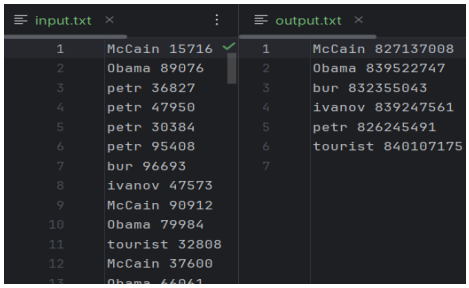
```
def vote(data):
    votes = {}
    for line in data:
        candidate, vote_count = line
        vote_count = int(vote_count)
        if candidate in votes:
            votes[candidate] += vote_count
        else:
            votes[candidate] = vote_count
    sorted_candidates = sorted(votes.items())

    return sorted_candidates
```

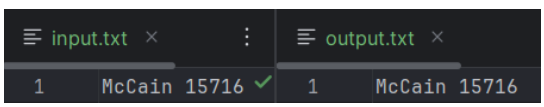
Функция vote подсчитывает общее количество голосов для каждого кандидата на основе переданных данных. На вход функция принимает список, где каждая строка содержит имя кандидата и количество голосов за него. Сначала создаётся пустой словарь votes, который используется для хранения суммы голосов по каждому кандидату. Для каждой строки из входных данных функция проверяет, есть ли кандидат уже в словаре: если есть, то добавляет к нему количество голосов; если нет — создаёт новую запись. После подсчёта всех голосов кандидаты сортируются в алфавитном порядке, и функция возвращает отсортированный список пар (имя кандидата, общее количество голосов).

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.0009312629699707031	0.000148773193359375
Пример из задачи	0.0009405612945556641	0.0001220703125
Верхняя граница диапазона значений входных данных из текста задачи	0.1551196575164795	0.000949859619140625

Вывод по задаче: в этой задаче было реализовано подведение итогов выборов с учетом голосов каждого штата за кандидатов. Использование словаря для подсчета голосов и сортировка кандидатов в алфавитном порядке обеспечили правильный вывод результатов. Время выполнения и использование памяти также соответствуют ограничениям задачи, и решение показало хорошую производительность при обработке данных.

Вывод

Лабораторная работа позволила глубже понять принципы хеширования и их практическое применение в задачах, требующих эффективного хранения и обработки данных. Все задачи, выполненные в рамках работы, продемонстрировали высокую эффективность хеш-таблиц, их способность быстро выполнять операции и эффективно использовать память.