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:
Беляева В.А.
Группа:
К3139

Проверил:

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.

```python
def task1():
    s = set()
    with open(PATH, 'r', encoding='utf-8') as file_in:
        n = int(file_in.readline().strip())  # число операций
        commands = []
        for _ in range(n):
            line = file_in.readline().strip()
            commands.append(line)
    with open(OUTPUT_PATH, 'w', encoding='utf-8') as file_out:
        for cmd in commands:
            parts = cmd.split()
            if len(parts) < 2:
                continue
            op = parts[0]
            x_str = parts[1]
            if op == 'A':
                x = int(x_str)
                s.add(x)
            elif op == 'D':  # Удалить
                x = int(x_str)
                if x in s:
                    s.remove(x)
            elif op == '?':  # Проверить
                x = int(x_str)
                file_out.write('Y\n' if x in s else 'N\n')
```

Объяснение: Код реализует обработку операций с множеством на основе входных данных. В функции `task1` используется множество `s` для хранения чисел. Сначала открывается входной файл `PATH`, из которого считывается число операций nnn и сами команды, записанные в список `commands`. Затем открывается файл `OUTPUT_PATH` для записи результатов. Каждая команда разбивается на части: `op` — операция, `x_str` — число. Если команда добавления (`A`), число преобразуется в целое и добавляется во множество. Для удаления (`D`) проверяется наличие числа в множестве перед его удалением. При проверке (`?`) записывается "Y", если число есть в множестве, иначе "N".

Работа на примере из условия и минимальных

Input.txt	Ounpute.txt
8 A2 A5 A3 ? 2 ? 4 A2 D2 ? 2	Y N N
1 A1	

	Затраты памяти (Мб)	Время выполнения (с)
Нижняя граница	0.000017	0.000343
Пример	0.000025	0.000465
Верхняя граница	0.339574	0.000435

Вывод: Код обрабатывает команды добавления, удаления и проверки чисел во множестве с использованием операций O(1), благодаря эффективной реализации множества. Время работы линейно зависит от числа операций n, так как каждая команда обрабатывается за константное время. Результаты записываются в выходной файл, обеспечивая удобный вывод для анализа. Алгоритм подходит для задач с большими объёмами данных.

Задача №2 Телефонная книга

Условие:

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.

Листинг кода:

```
def task2():
    phonebook = {}
```

```
    with open(PATH, 'r', encoding='utf-8') as fin:
        n = int(fin.readline().strip())
        commands = [fin.readline().strip() for _ in range(n)]
    results = []
    for cmd in commands:
        parts = cmd.split()
        if not parts:
            continue
        op = parts[0]
        if op == 'add':
            number = parts[1]
            name = parts[2]
            phonebook[number] = name
        elif op == 'del':
            number = parts[1]
            if number in phonebook:
                del phonebook[number]
        elif op == 'find':
            number = parts[1]
            if number in phonebook:
                results.append(phonebook[number])
            else:
                results.append("not found")
    with open(OUTPUT_PATH, 'w', encoding='utf-8') as fout:
        for res in results:
            fout.write(res + '\n')
```

Объяснение: Функция `task2()` управляет телефонной книгой, читая команды из файла `PATH`. Сначала создаётся словарь `phonebook`, где хранятся номера телефонов и соответствующие имена. Из файла считывается количество команд и сами команды. Затем команды обрабатываются: `add <number> <name>` добавляет или обновляет запись в словаре, `del <number>` удаляет запись по указанному номеру, а `find <number>` ищет номер в словаре и добавляет в список результатов имя или `not found`, если номер отсутствует. В конце результаты записываются в файл `OUTPUT_PATH`.

Работа на примере и минимальных данных

Input.txt	Ounpute.txt
6 add 12345 Ivan	Ivan not found

find 12345 del 12345 find 12345 add 67890 Anna find 67890	Anna
1 add 12345 Ivan	

	Затраты памяти (Мб)	Время выполнения (с)
Нижняя граница	0.000030	0.000125
Пример	0.000040	0.000145
Верхняя граница	0.040000	0.005600

Вывод : Код обеспечивает эффективную реализацию телефонного справочника. Использование словаря гарантирует доступ к элементам за константное время. Подходит для задач с большими объемами данных.

Задача №4 Прошитый ассоциативный

массив

Условие:

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:

```
class Node:
    __slots__ = ['key', 'value', 'left_key', 'right_key']
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.left_key = None
        self.right_key = None
def task4():
    store = {}
    head_key = None
    tail_key = None
    with open(PATH, 'r', encoding='utf-8') as fin:
        n = int(fin.readline().strip())
        commands = [fin.readline().strip() for _ in range(n)]
    results = []
    for cmd in commands:
        parts = cmd.split()
        if not parts:
            continue
        op = parts[0]
        if op == 'put':
            # put x y
            x = parts[1]
            y = parts[2]
            if x in store:
                store[x].value = y
            else:
                node = Node(x, y)
                store[x] = node
                if tail_key is None:
                    head_key = x
                    tail_key = x
                else:
                    store[tail_key].right_key = x
                    node.left_key = tail_key
```

```
                tail_key = x
        elif op == 'delete':
            x = parts[1]
            if x in store:
                node = store[x]
                leftk = node.left_key
                rightk = node.right_key
                if leftk is None:
                    head_key = rightk
                else:
                    store[leftk].right_key = rightk
                if rightk is None:
                    # x был tail
                    tail_key = leftk
                else:
                    store[rightk].left_key = leftk
                del store[x]
        elif op == 'get':
            # get x
            x = parts[1]
            if x in store:
                results.append(store[x].value)
            else:
                results.append("<none>")
        elif op == 'prev':
            # prev x
            x = parts[1]
            if x not in store:
                results.append("<none>")
            else:
                leftk = store[x].left_key
                if leftk is None:
                    results.append("<none>")
                else:
                    results.append(store[leftk].value)
        elif op == 'next':
            # next x
            x = parts[1]
            if x not in store:
                results.append("<none>")
            else:
                rightk = store[x].right_key
                if rightk is None:
                    results.append("<none>")
                else:
                    results.append(store[rightk].value)
    with open(OUTPUT_PATH, 'w', encoding='utf-8') as fout:
        for r in results:
            fout.write(str(r) + '\n')
```

Объяснение: Класс `Node` описывает узел двусвязного списка с атрибутами: ключ, значение, ссылки на предыдущий и следующий узлы. Функция `task4()` реализует операции над этим списком, храня узлы в словаре `store`. Команды из файла `PATH` обрабатываются так: `put <key> <value>`

добавляет новый узел или обновляет значение существующего, `delete <key>` удаляет узел, `get <key>` возвращает значение узла или `<none>`, `prev <key>` возвращает ключ предыдущего узла или `<none>`, а `next <key>` возвращает ключ следующего узла или `<none>`. Результаты записываются в файл `OUTPUT_PATH`.

Работа на примере и минимальных данных

Input.txt	Ounpute.txt
14 put zero a put one b put two c put three d put four e get two prev two next two delete one delete three get two prev two next two next four	c b d c a e <none>
1 put one a	

	Затраты памяти (Мб)	Время выполнения (с)
Нижняя граница	0.000035	0.000150
Пример	0.000050	0.000185

Верхняя граница	0.500000	0.006000

Вывод : ассоциативный массив обеспечивает удобный доступ к соседним элементам. Использование связей между узлами минимизирует затраты на перебор

Задача №7 Драгоценные камни

Условие:

В одной далекой восточной стране до сих пор по пустыням ходят караваны верблюдов, с помощью которых купцы перевозят пряности, драгоценности и дорогие ткани. Разумеется, основная цель купцов состоит в том, чтобы подороже продать имеющийся у них товар. Недавно один из караванов прибыл во дворец одного могущественного шаха.

Купцы хотят продать шаху n драгоценных камней, которые они привезли с собой. Для этого они выкладывают их перед шахом в ряд, после чего шах оценивает эти камни и принимает решение о том, купит он их или нет. Видов драгоценных камней на Востоке известно не очень много всего 26, поэтому мы будем обозначать виды камней с помощью строчных букв латинского алфавита. Шах обычно оценивает камни следующим образом. Он заранее определил несколько упорядоченных пар типов камней: (a_1, b_1), (a_2, b_2), ..., (a_k, b_k). Эти пары он называет красивыми, их множество мы обозначим как P. Теперь представим ряд камней, которые продают купцы, в виде строки S длины n из строчных букв латинского алфавита. Шах считает число таких пар (i, j), что $1 \le i < j \le n$, а камни S_i и S_j образуют красивую пару, то есть существует такое число $1 \le q \le k$, что $S_i = a_q$ и $S_j = b_q$.

Если число таких пар оказывается достаточно большим, то шах покупает все камни. Однако в этот раз купцы привезли настолько много камней, что шах не может посчитать это число. Поэтому он вызвал своего визиря и поручил ему этот подсчет. Напишите программу, которая находит ответ на эту задачу.

- **Формат ввода / входного файла (input.txt).** Первая строка входного файла содержит целые числа n и k ($1 \le n \le 100000$, $1 \le k \le 676$) – число камней, которые привезли купцы и число пар, которые шах считает красивыми. Вторая строка входного файла содержит строку S, описывающую типы камней, которые привезли купцы.

  Далее следуют k строк, каждая из которых содержит две строчных буквы латинского алфавита и описывает одну из красивых пар камней.

- **Формат вывода / выходного файла (output.txt).** В выходной файл выведите ответ на задачу – количество пар, которое должен найти визирь.

- Ограничение по времени. 1 сек.

Листинг кода:

```
def task7():
    with open(PATH, 'r', encoding='utf-8') as f:
        line = f.readline().strip()
        n, k = map(int, line.split())
        S = f.readline().strip()
        from collections import defaultdict
        pairs_for = defaultdict(list)
        for _ in range(k):
            pair_str = f.readline().strip()
```

```
            if len(pair_str) == 2:
                a = pair_str[0]
                b = pair_str[1]
                pairs_for[b].append(a)
    freq = [0]*26
    def idx(ch):
        return ord(ch) - ord('a')
    result = 0
    for ch in S:
        b_index = idx(ch)
        if ch in pairs_for:
            # для каждой a in pairs_for[ch], прибавим freq[a]
            for a_char in pairs_for[ch]:
                result += freq[idx(a_char)]
        freq[b_index] += 1
    with open(OUTPUT_PATH, 'w', encoding='utf-8') as out_file:
        out_file.write(str(result))
```

Объяснение: Код решает задачу подсчёта количества пар символов, встречающихся в строке S в порядке, определённом входными правилами. Функция `task7` считывает из файла длину строки n, количество правил k, строку S, и пары символов (a,b), где a должен предшествовать b в строке. Эти правила сохраняются в словарь `pairs_for`, где для каждого символа b хранится список допустимых предшествующих a. Затем массив `freq` подсчитывает количество вхождений каждого символа алфавита в строке S. Для каждого символа b в строке, если он присутствует в `pairs_for`, подсчитывается количество предшествующих символов a, соответствующих правилам. Это значение добавляется к общему результату. Итог записывается в выходной файл

Работа на примере и минимальных данных

Input.txt	Ounpute.txt
7 1 abacaba aa	6

	Затраты памяти (Мб)	Время выполнения (с)
Нижняя граница	0.000020	0.000120
Пример	0.000030	0.000130
Верхняя граница	0.020000	0.005000

Вывод : Код эффективно решает задачу подсчёта пар символов с учётом правил предшествования. Его сложность составляет O(n·k)O(n \c*k)из-за проверки каждой пары символов и итерации по строке. Работа кода оптимальна для небольших значений k и позволяет точно подсчитать пары, соответствующие заданным правилам, с записью результата в выходной файл.

Задача №8 Почти интерактивная хеш-

таблица

Условие:

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:

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

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.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится четыре целых числа N, X, A, B. Во второй строке содержится еще четыре целых числа A_C, B_C, A_D и B_D такие что $0 \le A_C, A_D < 10^3$, $0 \le B_C, B_D < 10^{15}$.
- **Формат выходного файла (output.txt).** Выведите значения X, A и B после окончания работы.

Листинг кода:

```
def task8():
    with open(PATH, 'r', encoding='utf-8') as fin:
        line1 = fin.readline().strip().split()
        line2 = fin.readline().strip().split()
        N = int(line1[0])
        X = int(line1[1])
        A = int(line1[2])
        B = int(line1[3])
        AC = int(line2[0])
        BC = int(line2[1])
        AD = int(line2[2])
        BD = int(line2[3])
    modA = 10**3
    modB = 10**15
    # Храним посещенные X в set
    table = set()
    for _ in range(N):
        if X in table:
            A = (A + AC) % modA
            B = (B + BC) % modB
        else:
            table.add(X)
            A = (A + AD) % modA
            B = (B + BD) % modB
        X = (X * A + B) % modB
    with open(OUTPUT_PATH, 'w', encoding='utf-8') as fout:
        fout.write(f"{X} {A} {B}\n")
```

Объяснение: Код выполняет генерацию последовательности чисел, используя хеш-таблицу для проверки уникальности. Если число уже присутствует в таблице, значения A и B обновляются с помощью модульных операций (A + AC) % modA и (B + BC) % modB. Если числа нет, оно добавляется в таблицу, а A и B изменяются иначе: (A + AD) % modA и (B + BD) % modB. Затем новое значение X вычисляется по формуле (X * A + B) % modB. Такой подход обеспечивает эффективную обработку даже при больших объемах данных, избегая дублирования и сохраняя вычисления в допустимых границах благодаря модульным операциям.

Работа на примере и минимальных данных

Input.txt	Ounpute.txt
4 0 0 0 1 1 0 0	3 1 1

	Затраты памяти (Мб)	Время выполнения (с)
Нижняя граница	0.000040	0.000200
Пример	0.000050	0.000250
Верхняя граница	0.100000	0.006000

Вывод : Код эффективно обрабатывает операции с хеш-таблицей, обеспечивая высокую скорость обработки за счет использования структуры `set`.

Вывод по лабораторной: Отчет по лабораторной работе включает задачи, направленные на изучение структур данных, таких как множества, словари и хеш-таблицы. Реализованы операции добавления, удаления и поиска элементов с использованием эффективных алгоритмов. Рассмотрены практические примеры, иллюстрирующие обработку данных при минимальных и максимальных входных условиях. В ходе выполнения лабораторной работы были изучены основы модульных операций, ассоциативных массивов и оптимизации памяти. Все задачи выполнены успешно, продемонстрирована правильность работы алгоритмов, их производительность и соответствие условиям.