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 1

Выполнил:
Бен Шамех Абделазиз
Группа: K3239

Проверила:
Ромакина Оксана Михайловна

Санкт-Петербург
2025 г.

Содержание

1 Задача 1. Множество

1.1 Условие

Реализуйте множество с операциями «добавление ключа» (A), «удаление ключа» (D), «проверка существования ключа» (?).

1.2 Листинг кода

```python
class HashSet:
    def __init__(self, size=6*10**5):
        self.size = size
        self.table = [[] for i in range(size)]

    def add(self, key):
        ind = self.get_hash(key)
        if key not in self.table[ind]:
            self.table[ind].append(key)

    def pop(self, key):
        ind = self.get_hash(key)
        if key in self.table[ind]:
            self.table[ind].remove(key)

    def check(self, key):
        return "Y" if key in self.table[self.get_hash(key)] else "N"

    def get_hash(self, key):
        return hash(key) % self.size
```

1.3 Текстовое объяснение

Для реализации множества используется хеш-таблица с открытой адресацией методом цепочек. Каждая ячейка массива содержит список элементов, имеющих одинаковый хеш-код. Операция поиска и удаления в среднем выполняется за $O(1)$, в худшем случае (при большом количестве коллизий) — за $O(N)$.

1.4 Результаты выполнения

Ввод: 8 операций (A 2, A 5, A 3, ? 2, ? 4, A 2, D 2, ? 2).
Вывод: `Y, N, N`.

Сценарий	Время выполнения	Затраты памяти
Нижняя граница (n=3)	0.09136 sec	14.56 Mb
Пример из задачи	0.11091 sec	16.71 Mb
Верхняя граница (n=10^5)	0.87454 sec	36.76 Mb

2 Задача 2. Телефонная книга

2.1 Листинг кода

```
class PhoneBook:
    def __init__(self):
        self.contacts = {}

    def add(self, number, name):
        self.contacts[number] = name

    def delete(self, number):
        if number in self.contacts:
            del self.contacts[number]

    def find(self, number):
        return self.contacts.get(number, "not found")
```

2.2 Текстовое объяснение

Телефонная книга реализована с использованием ассоциативного массива (словаря). Номер телефона выступает в роли уникального ключа. Если при добавлении номер уже существует, имя перезаписывается. Поиск и удаление осуществляются по ключу за константное время.

2.3 Результаты выполнения

Ввод: `add 911 police, find 911, del 911, find 911`.
Вывод: `police, not found`.

Сценарий	Время выполнения	Затраты памяти
Пример из задачи	0.00267 sec	14.79 Mb
Верхняя граница (n=10^5)	0.43443 sec	35.12 Mb

3 Задача 4. Прошитый ассоциативный массив

3.1 Листинг кода

```
class StitchedHashMap:
    def __init__(self):
        self.prev_links = {}
        self.next_links = {}
        self.data = {}
        self.last_inserted = None

    def put(self, key, value):
        if key not in self.data:
            if self.last_inserted is not None:
                self.next_links[self.last_inserted] = key
                self.prev_links[key] = self.last_inserted
            self.last_inserted = key
        self.data[key] = value

    def get(self, key):
        return self.data.get(key, "<none>")

    def prev(self, key):
        p_key = self.prev_links.get(key)
        return self.data.get(p_key, "<none>") if p_key in self.data
   else "<none>"

    def next(self, key):
        n_key = self.next_links.get(key)
        return self.data.get(n_key, "<none>") if n_key in self.data
   else "<none>"
```

3.2 Текстовое объяснение

Прошитый массив сочетает в себе свойства хеш-таблицы и двусвязного списка. При каждой вставке нового элемента мы обновляем "ссылки"(`prev` и `next`) в дополнительных словарях. Это позволяет не только быстро получать значение по ключу, но и мгновенно находить элементы, вставленные непосредственно до или после данного.

3.3 Результаты выполнения

Ввод: `put zero a, put one b, prev one.`
Вывод: `a.`

4 Задача 5. Выборы в США

4.1 Листинг кода

```python
import sys

def count_votes():
    votes = {}
    for line in sys.stdin:
        if not line.strip(): continue
        name, count = line.split()
        votes[name] = votes.get(name, 0) + int(count)

    for name in sorted(votes.keys()):
        print(f"{name} {votes[name]}")
```

4.2 Текстовое объяснение

Задача решается путем аккумулирования значений в словаре. Фамилия кандидата используется как ключ, а количество голосов прибавляется к текущему значению в словаре. После обработки всех данных ключи сортируются в алфавитном порядке для итогового вывода результатов.

4.3 Результаты выполнения

Ввод: `McCain 10, McCain 5, Obama 9.`
Вывод: `McCain 15, Obama 9.`

5 Задача 7. Драгоценные камни

5.1 Листинг кода

```python
def count_gem_pairs(n, k, S, pairs):
    index_dict = {c: [] for c in "abcdefghijklmnopqrstuvwxyz"}
    for i, char in enumerate(S):
        index_dict[char].append(i)

    total_pairs = 0
    for a, b in pairs:
        list_a, list_b = index_dict[a], index_dict[b]
        i = j = 0
        while i < len(list_a) and j < len(list_b):
            if list_b[j] > list_a[i]:
                total_pairs += (len(list_b) - j)
                i += 1
            else:
                j += 1
    return total_pairs
```

5.2 Текстовое объяснение

Алгоритм сначала индексирует строку, сохраняя позиции каждой буквы в отсортированных списках. Для каждой пары "красивых"камней (a, b) мы ищем количество таких индексов $i \in list\_a$ и $j \in list\_b$, что $i < j$. Благодаря тому, что списки индексов отсортированы, задача решается за линейное время методом двух указателей.

5.3 Результаты выполнения

Ввод: `7 1, abacaba, aa`.
Вывод: `6`.

6 Вывод

1. В данной лабораторной работе были изучены и реализованы основные структуры данных на базе хеширования: множества, словари и прошитые ассоциативные массивы.

2. Была доказана эффективность хеш-таблиц для задач накопления данных (подсчет голосов) и быстрого поиска в больших массивах.

3. Применение дополнительных структур (связок) поверх хеш-таблицы позволяет сохранять порядок поступления данных без потери скорости доступа $O(1)$.