

**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 Абделаиз
Группа: К3239

Проверила:
Ромакина Оксана Михайловна

Санкт-Петербург
2025 г.

Содержание

1	Задача 1. Множество	2
1.1	Условие	2
1.2	Листинг кода	2
1.3	Объяснение решения	2
1.4	Результаты выполнения	2
2	Задача 2. Телефонная книга	3
2.1	Листинг кода	3
2.2	Объяснение решения	3
2.3	Результаты выполнения	3
3	Задача 4. Прошитый ассоциативный массив	4
3.1	Листинг кода	4
3.2	Объяснение решения	4
3.3	Результаты выполнения	4
4	Задача 5. Выборы в США	5
4.1	Листинг кода	5
4.2	Результаты выполнения	5
5	Задача 7. Драгоценные камни	6
5.1	Листинг кода	6
5.2	Объяснение решения	6
5.3	Результаты выполнения	6
6	Вывод	7

1 Задача 1. Множество

1.1 Условие

Реализуйте множество с операциями «добавление ключа», «удаление ключа», «проверка существования ключа».

1.2 Листинг кода

```
1 class HashSet:
2     def __init__(self, size=6*10**5):
3         self.size = size
4         self.table = [[] for i in range(size)]
5
6     def add(self, key):
7         ind = self.get_hash(key)
8         if key not in self.table[ind]:
9             self.table[ind].append(key)
10
11    def pop(self, key):
12        ind = self.get_hash(key)
13        if key in self.table[ind]:
14            self.table[ind].remove(key)
15
16    def check(self, key):
17        return key in self.table[self.get_hash(key)]
18
19    def get_hash(self, key):
20        return hash(key) % self.size
```

1.3 Объяснение решения

Класс с операциями добавления, удаления и проверки. Обращение происходит через хеш по модулю в массиве списков (метод цепочек для обработки коллизий).

1.4 Результаты выполнения

Метрики эффективности:

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=100000)	0.87454 sec	36.76 Mb

2 Задача 2. Телефонная книга

2.1 Листинг кода

```
1 class PhoneAdapter:
2     def __init__(self):
3         self.numb_dict = {}
4
5     def add_num(self, number, name):
6         self.numb_dict[number] = name
7
8     def pop_num(self, number):
9         if number in self.numb_dict:
10             del self.numb_dict[number]
11
12     def find_num(self, number):
13         return self.numb_dict.get(number, "not found")
```

2.2 Объяснение решения

Создаем класс, который управляет номерами и именами через встроенный словарь Python, обеспечивающий быстрый доступ по ключу.

2.3 Результаты выполнения

Сценарий	Время выполнения	Затраты памяти
Нижняя граница (n=3)	0.00133 sec	14.33 Mb
Пример из задачи	0.00267 sec	14.79 Mb
Верхняя граница (n=100000)	0.43443 sec	35.12 Mb

3 Задача 4. Прошитый ассоциативный массив

3.1 Листинг кода

```
1 class StitchedHashMap:
2     def __init__(self):
3         self.prev_dep = {}
4         self.next_dep = {}
5         self.data = {}
6         self.values_queue = []
7
8     def get(self, item):
9         return self.data.get(item, "<none>")
10
11    def prev(self, val):
12        if val in self.prev_dep and self.prev_dep[val] in self.data:
13            return self.data[self.prev_dep[val]]
14        else: return "<none>"
15
16    def next(self, item):
17        if item in self.next_dep and self.next_dep[item] in self.data:
18            return self.data[self.next_dep[item]]
19        else: return "<none>"
20
21    def put(self, key, val):
22        if key not in self.data:
23            if self.values_queue:
24                last_key = self.values_queue[-1]
25                self.next_dep[last_key] = key
26                self.prev_dep[key] = last_key
27                self.values_queue.append(key)
28            self.data[key] = val
```

3.2 Объяснение решения

Создаем класс со словарем данных, массивом порядка и словарями связей (предыдущий-/следующий), чтобы поддерживать историю вставок.

3.3 Результаты выполнения

Сценарий	Время выполнения	Затраты памяти
Нижняя граница (n=3)	0.00598 sec	14.78 Mb
Пример из задачи	0.00641 sec	15.07 Mb
Верхняя граница (n=100000)	0.51925 sec	36.19 Mb

4 Задача 5. Выборы в США

4.1 Листинг кода

```
1 class ElectionsAdapter:
2     def __init__(self):
3         self.elect_dict = {}
4
5     def add_num(self, name, cnt):
6         self.elect_dict[name] = self.elect_dict.get(name, 0) + cnt
7
8     def get_data(self):
9         names_sorted = sorted(self.elect_dict.keys())
10        return [f"{name} {self.elect_dict[name]}" for name in
names_sorted]
```

4.2 Результаты выполнения

Сценарий	Время выполнения	Затраты памяти
Нижняя граница (n=5)	0.00234 sec	14.77 Mb
Пример из задачи	0.00254 sec	14.83 Mb
Верхняя граница (n=100000)	0.47529 sec	37.17 Mb

5 Задача 7. Драгоценные камни

5.1 Листинг кода

```
1 def get_b_pairs(st_rocks, arr_pairs):
2     ans = 0
3     index_dict = {i:[] for i in 'abcdefghijklmnopqrstuvwxyz'}
4     for i in range(len(st_rocks)):
5         index_dict[st_rocks[i]].append(i)
6
7     for x, y in arr_pairs:
8         arr1, arr2 = index_dict[x], index_dict[y]
9         i = j = 0
10        while i < len(arr1) and j < len(arr2):
11            if arr2[j] > arr1[i]:
12                ans += (len(arr2) - j)
13                i += 1
14            else:
15                j += 1
16    return ans
```

5.2 Объяснение решения

Подсчитываем индексы вхождений каждого типа камня в словарь. Затем для каждой "красивой" пары через два указателя вычисляем количество комбинаций за линейное время.

5.3 Результаты выполнения

Сценарий	Время выполнения	Затраты памяти
Нижняя граница (n=5)	0.01938 sec	14.77 Mb
Пример из задачи	0.02316 sec	16.02 Mb
Верхняя граница (n=100000)	0.63208 sec	36.35 Mb

6 Вывод

1. В этой лабораторной работе мы познакомились и научились создавать множества, хеш-таблицы и хеш-функции, а также реализовывать алгоритмы для этих структур.
2. Алгоритмы $O(n)$ и $O(n \log n)$ выполняются достаточно быстро. Затраты памяти растут линейно пропорционально количеству элементов.
3. С помощью методов `time.perf_counter()` и `psutil.Process().memory_info().rss` можно эффективно отслеживать ресурсозатратность алгоритмов.