

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 14

Выполнила:
Рудникова Виктория Олеговна
К3125

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Множество	3
Задача №2. Телефонная книга	7
Задача №4. Прошитый ассоциативный массив	13
Вывод:	18

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.

```
from time import process_time
from tracemalloc import start, get_traced_memory
```

```
class HashSet:
    def __init__(self):
        self.set = {}
        self.length = len(self.set)
```

```
    def add(self, element):
        self.set[element] = self.length
```

```
    def delete(self, element):
        try:
            self.set.pop(element)
        except KeyError:
            pass
```

```
    def check(self, element):
        if self.set.get(element) is None:
            return 'N'
```

```

        return 'Y'

    def repr(self):
        return repr(self.set)

def file_output(array):
    with open('output.txt', 'w+') as f:
        f.write('\n'.join([str(i) for i in array]))

if __name__ == '__main__':
    start()
    hashset = HashSet()
    stack = []

    with open('input.txt') as f:
        n = int(f.readline())
        for _ in range(n):
            command, number = f.readline().split()
            match command:
                case 'A':
                    hashset.add(number)
                case 'D':
                    hashset.delete(number)
                case '?':
                    stack.append(hashset.check(number))
                    print(hashset.check(number))

    file_output(stack)
    print('Time:', str(process_time()), 'sec')
    print('Memory usage:', str(get_traced_memory()[1] / 1024), 'KB')

```

Реализован класс HashSet, представляющий собой множество с операциями добавления, удаления и проверки элемента на существование.

Результат работы кода на примерах из текста задачи:

```
input.txt ×
1      8
2      A 2
3      A 5
4      A 3
5      ? 2
6      ? 4
7      A 2
8      D 2
9      ? 2

output.txt ×
1      Y
2      N
3      N
```

Результат работы кода на максимальных и минимальных значениях:

```

input.txt x
1 500000
2 A 384239482
3 A 23245234
4 A 543256765
5 A 654324543
6 A 2343234
7 A 3434543
8 D 23245234
9 ? 23245234
10 ? 25555555

output.txt x
1 N
2 N
3 Y
4 N

```

```

input.txt x
1 2
2 A 5
3 ? 5

output.txt x
1 Y

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.078375 sec	17.244140625 KB
Пример из задачи	0.078375 sec	17.244140625 KB
Верхняя граница	0.078375 sec	17.244140625 KB

диапазона значений входных данных из текста задачи		
--	--	--

Вывод по задаче: в этой задаче я реализовала множество.

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.

```
from time import process_time
from tracemalloc import start, get_traced_memory

class PhoneBook:
    def __init__(self):
        self.arr = [[] for _ in range(0, 101)]
        self.output_file = open('output.txt', 'w')

    def add(self, number, name):
        idx = self.hash(number)
        if type(self.arr[idx]) != int:
            for element in self.arr[idx]:
                if element[0] == number:
                    element[1] = name
            return
        self.arr[idx].append([number, name])
    else:
        self.arr[idx] = [number, name]

    def remove(self, number):
        idx = self.hash(int(number))
        self.arr[idx] = []

    def find(self, number):
        idx = self.hash(int(number))
        for i in self.arr[idx]:
            if i[0] == number:
                self.output_file.write(i[1] + '\n')
            return
        self.output_file.write('not found\n')

    @staticmethod
    def hash(value):
        return value % 101

    def __repr__(self):
        return repr(self.arr)

if __name__ == '__main__':
    start()
    p = PhoneBook()

    with open('input.txt', 'r') as f:
        lines = f.readlines()
        for i in range(1, len(lines)):
            temp = lines[i].split()
            command = temp[0]
            number = int(temp[1])
```



```
        match command:
            case 'add':
                p.add(number, temp[2])
            case 'del':
                p.remove(number)
            case 'find':
                p.find(number)
        print(p.arr)

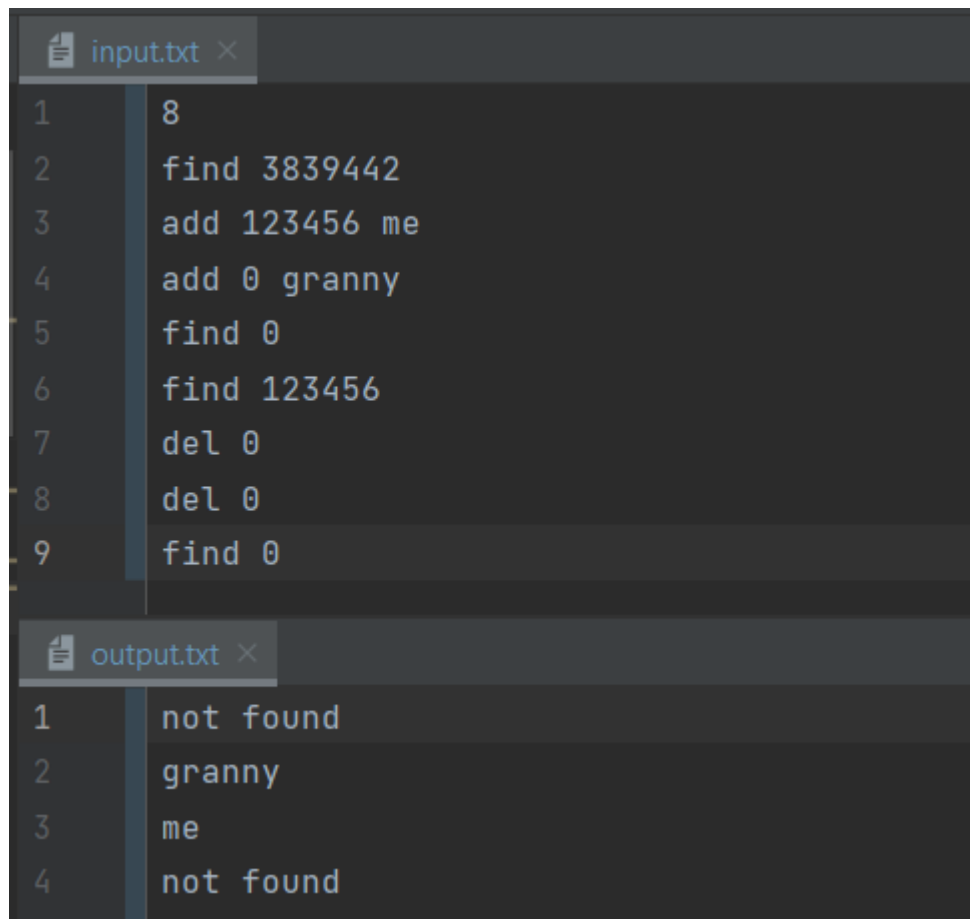
    print('Time:', str(process_time()), 'sec')
    print('Memory usage:', str(get_traced_memory()[1] / 1024), 'KB')
```

С помощью хеш-таблицы реализована телефонная книга с операциями добавления контакта, удаления и поиска контакта по номеру телефона.

Результат работы кода на примерах из текста задачи:

```
input.txt ×
1 12
2 add 911 police
3 add 76213 Mom
4 add 17239 Bob
5 find 76213
6 find 910
7 find 911
8 del 910
9 del 911
10 find 911
11 find 76213
12 add 76213 daddy
13 find 76213

output.txt ×
1 Mom
2 not found
3 police
4 not found
5 Mom
6 daddy
```



The image shows a code editor with two tabs: 'input.txt' and 'output.txt'. The 'input.txt' tab is active and contains 9 lines of code. The 'output.txt' tab is also visible and contains 4 lines of output. The code in 'input.txt' includes a value '8', a 'find' operation with '3839442', an 'add' operation with '123456' and 'me', an 'add' operation with '0' and 'granny', and several 'find' and 'del' operations with '0'.

input.txt	output.txt
1 8	1 not found
2 find 3839442	2 granny
3 add 123456 me	3 me
4 add 0 granny	4 not found
5 find 0	
6 find 123456	
7 del 0	
8 del 0	
9 find 0	

Результат работы кода на максимальных и минимальных значениях:

```
input.txt x
1 69
2 add 56 lkkkkksld
3 find 56
4 del 56
5 add 56 lkkkkksld
6 find 56
7 del 56
8 add 56 lkkkkksld
9 find 56
10 del 56

output.txt x
1 lkkkkksld
2 lkkkkksld
3 lkkkkksld
4 lkkkkksld
5 lkkkkksld
6 lkkkkksld
7 lkkkkksld
8 lkkkkksld
```

```
input.txt x
1 2
2 add 67 gfd
3 find 67

output.txt x
1 gfd
2
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.109375 sec	17.244140625 KB

Пример из задачи	0.109375 sec	17.244140625 KB
Пример из задачи	0.109375 sec	17.244140625 KB
Верхняя граница диапазона значений входных данных из текста задачи	0.109375 sec	19.87890625 KB

Вывод по задаче: в задаче я использовала хеш-таблицу для реализации телефонной книги.

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 `<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.
- Формат входного файла (`input.txt`). В первой строке входного файла на-

ходится 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.

```
from time import process_time
from tracemalloc import start, get_traced_memory
```

```
class Array:
    def __init__(self):
        self.array = {}

    def get(self, x):
        if self.array.get(x) is None:
            return '<none>'
        else:
            return self.array.get(x)

    def prev(self, x):
        lst = list(self.array)
        if x in lst:
            index = lst.index(x) - 1
            if lst.index(x) != 0:
                next_key = lst[index]
                return self.get(next_key)
            else:
                return '<none>'
        else:
            return '<none>'

    def next(self, x):
        lst = list(self.array)
        if x in lst:
            index = lst.index(x) + 1
            if index != len(lst):
                next_key = lst[index]
                return self.get(next_key)
            else:
                return '<none>'
        else:
            return '<none>'

    def put(self, x, y):
        self.array[x] = y
```

```

def delete(self, element):
    try:
        self.array.pop(element)
    except KeyError:
        pass

def __repr__(self):
    return repr(self.array)

def file_output(array):
    with open('output.txt', 'w+') as f:
        f.write('\n'.join([str(i) for i in array]))

if __name__ == '__main__':
    start()
    arr = Array()
    output = []

    with open('input.txt') as f:
        n = int(f.readline())

        for _ in range(n):
            string = list(f.readline().strip('\n').split())
            try:
                command, x, y = string
            except ValueError:
                command, x = string
            match command:
                case 'get':
                    output.append(arr.get(x))
                case 'prev':
                    output.append(arr.prev(x))
                case 'next':
                    output.append(arr.next(x))
                case 'put':
                    arr.put(x, y)
                case 'delete':
                    arr.delete(x)

    file_output(output)
    print('Time:', str(process_time()), 'sec')
    print('Memory usage:', str(get_traced_memory()[1] / 1024), 'KB')

```

Реализован прошитый ассоциативный массив, поддерживающий операции добавления элемента, его удаления, поиска значения по ключу, вывода предыдущего и следующего добавленного ключа.

Результат работы кода на примерах из текста задачи:

```
input.txt x
1      14
2      put zero a
3      put one b
4      put two c
5      put three d
6      put four e
7      get two
8      prev two
9      next two
10     delete one
11     delete three
12     get two
13     prev two
14     next two
15     next four

output.txt x
1      c
2      b
3      d
4      c
5      a
6      e
7      <none>
```

Результат работы кода на максимальных и минимальных значениях:


```
input.txt x
1 500000
2 put 1 a
3 put 2 b
4 put 3 c
5 put 4 d
6 put 5 e
7 put 6 f
8 put 7 g
9 put 8 y
10 put 9 t
11 put 10 c
12 put 11 c
13 put 12 c
...

output.txt x
1 b
2 c
3 g
4 f
```

```
input.txt x
1 2
2 put 5 ttttt
3 get 6

output.txt x
1 <none>
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.09375 sec	17.244140625 KB

входных данных из текста задачи		
Пример из задачи	0.09375 sec	17.244140625 KB
Верхняя граница диапазона значений входных данных из текста задачи	0.09375 sec	17.244140625 KB

Вывод по задаче: в задаче я реализовала прошитый ассоциативный массив.

Вывод:

В работе я вспомнила хеширование и хеш-таблицы, научилась их реализовывать и применять их для решения практических задач.