

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №7
по курсу «Алгоритмы и структуры данных»
Тема: Хеширование. Хеш-таблицы
Вариант 27

Выполнил:
Филиппов А.Э.
К3139

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3-19
Задача №1. Множество	3-7
Задача №2. Телефонная книга	7-13
Задача №4. Прошитый ассоциативный массив	13-19
Вывод	20

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 на ходится 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 1018.

- **Формат выходного файла (output.txt).** Выведите последовательно результаты выполнения всех операций «?». Следуйте формату выходного файла из примера.

- Ограничение по времени. 3 сек.

- Ограничение по памяти. 256 мб.

- Пример:

input.txt	output.txt
8	Y
A 2	N
A 5	N
A 3	
? 2	

? 4	
A 2	
D 2	
? 2	

- Примечание.

Эту задачу можно решить совершенно разными способами, включая использование различных средств стандартных библиотек (правда, не всех - в стандартных библиотеках некоторых языков программирования используются слишком предсказуемые методы хеширования). Именно по этой причине ее разумно использовать для проверки реализаций хеш-таблиц, которые понадобятся в следующих задачах этой работы.

Код:

```
import time
import tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

# length of the bucket array
CAPACITY = 1000000

# node is a node of linked list (I just don't wanna create LinkedList class)

class Node:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None

    def __repr__(self):
        return str(self.value)

    def __str__(self):
        return f"key: {self.key}, value: {self.value}, next: {self.next}"

class HashTable:
    def __init__(self):
```

```

self.size = 0
self.buckets = [None]*CAPACITY

def hash(self, key):
    summ = 0
    for i in range(len(key)):
        summ += (ord(key[i])) * (125 ** i)

    return summ % CAPACITY

def insert(self, key, value):
    self.size += 1
    index = self.hash(key)
    node = self.buckets[index]

    if node is None:
        self.buckets[index] = Node(key, value)
    else:
        prev = node
        while node is not None:
            prev = Node
            node = node.next
        prev.next = Node(key, value)

def get(self, key):
    index = self.hash(key)
    node = self.buckets[index]

    while node is not None and node.key != key:
        node = node.next

    return node.value if node is not None else None

def exists(self, key):
    return True if self.get(key) != None else False

def remove(self, key):
    index = self.hash(key)
    node = self.buckets[index]

    # persist previous element, so we can remove linked list node
    prev = None

    while node is not None and node.key != key:
        prev = node
        node = node.next
    if node is not None:
        self.size -= 1
        if prev is None:
            self.buckets[index] = node.next
        else:
            prev.next = node.next

```

```

table = HashTable()
n = int(lines[0])
for i in range(n):
    op, val = lines[i+1].split()
    match op:
        case "A":
            table.insert(val, True)
        case "D":
            table.remove(val)
        case "?":
            out.write(("Y" if table.exists(val) else "N") + "\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Для решения данной задачи была реализована Хеш-таблица. Хеш-таблица использует функцию хеширования по остаткам.

Результат работы кода:

```

100000
1 D 86438661941897033
2 ? 51340412550044146
3 D 845187623061312058
4 ? 412036782026898349
5 ? 609589100318869729
6 A 674218869523180292
7 D 808187345263190142
8 ? 439782592041548696
9 D 26542042281897096
10 ? 634580322450987212
11 ? 157012470431587244
12 ? 649001096457355998
13 ? 884221781468363747
RMAL input.txt

N
1 N
2 N
3 N
4 N
5 N
6 N
7 N
8 N
9 N
10 N
11 N
12 N
13 N
RMAL output.txt

```

```

1
? 1
RMAL input.txt
put.txt" 2L, 6B w
N
RMAL output.txt

```

```

8
A 2
A 5
A 3
? 2
? 4
A 2
D 2
? 2
RMAL input.txt
put.txt" 9L, 34B
Y
N
N
RMAL output.txt

```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.003404877999855671	8017903
Пример из задачи	0.010401446000287251	8021112
Верхняя граница диапазона значений входных данных из текста задачи	2.534391541000332	17315634

Вывод по задаче: Использование Хеш-таблицы может значительно оптимизировать проверку по ключу. Поскольку ассоциативные массивы в языках программирования реализованы через красно-черные деревья, а не хеш-таблицы.

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 ($1 \leq N \leq 105$) - 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 каж дого поискового запроса `find` – имя, соответствующее номеру телефона, или «not found» (без кавычек), если в телефонной книге нет человека с та ким 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.
- Примеры:

input.txt
12
add 911
police
add 76213
Mom add
17239
Bob
find 76213
find 910
find 911
del 910
del 911
find 911
find 76213

add 76213 daddy find 76213
output.txt
Mom not found police not found Mom daddy

input.txt
8 find 3839442 add 123456 me add 0 granny find 0 find 123456 del 0 del 0 find 0
output.txt
not found granny me not found

Описание примера 1. 76213 - это номер Mom, 910 - нет в телефонной книге, 911 - это номер police, но затем он был удален из телефонной книги, поэтому второй поиск 911 вернул «not found». Также обратите внимание, что когда daddy был добавлен с тем же номером телефона 76213, что и номер телефона Mom, имя контакта было переписано, и теперь поиск 76213 возвращает «daddy» вместо «Mom».

Код:

```
import time
import tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

# length of the bucket array
CAPACITY = 1000000

# node is a node of linked list (I just don't wanna create LinkedList class)

class Node:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None

    def __repr__(self):
        return str(self.value)

    def __str__(self):
        return f"key: {self.key}, value: {self.value}, next: {self.next}"

class HashTable:
    def __init__(self):
        self.size = 0
        self.buckets = [None]*CAPACITY

    def hash(self, key):
        summ = 0
```

```

    for i in range(len(key)):
        summ += (ord(key[i])) * (125 ** i)

    return summ % CAPACITY

def insert(self, key, value):
    self.size += 1
    index = self.hash(key)
    node = self.buckets[index]

    if node is None:
        self.buckets[index] = Node(key, value)
    else:
        prev = node
        while node is not None:
            if prev.key == key:
                prev.value = value
                return
            prev = Node
            node = node.next
        prev.next = Node(key, value)

def get(self, key):
    index = self.hash(key)
    node = self.buckets[index]

    while node is not None and node.key != key:
        node = node.next

    return node.value if node is not None else None

def exists(self, key):
    return True if self.get(key) != None else False

def remove(self, key):
    index = self.hash(key)
    node = self.buckets[index]

    # persist previous element, so we can remove linked list node
    prev = None

    while node is not None and node.key != key:
        prev = node
        node = node.next
    if node is not None:
        self.size -= 1
        if prev is None:
            self.buckets[index] = node.next
        else:
            prev.next = node.next

```

```
table = HashTable()
```

```

n = int(lines[0])
for i in range(n):
    inputs = lines[i+1].split()
    key = inputs[1]
    match inputs[0]:
        case "add":
            val = lines[i+1].split()[2]
            table.insert(key, val)
        case "del":
            table.remove(key)
        case "find":
            val = table.get(key)
            if val == None:
                out.write("not found\n")
            else:
                out.write(str(val) + "\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Для решения данной задачи была использована Хеш-таблица. Данная структура позволяет находить ключ за константное время $O(1)$.

Результат работы кода:

<pre> 12 1 add 911 police 2 add 76213 Mom 3 add 17239 Bob 4 find 76213 5 find 910 6 find 911 7 del 910 8 del 911 9 find 911 10 find 76213 11 add 76213 daddy 12 find 76213 </pre> <p>ORMAL input.txt input.txt" [New] 13L, 138B written</p> <pre> 1 Mom 2 not found 3 police 4 not found 5 Mom 6 daddy </pre> <p>ORMAL output.txt output.txt" 6L, 41B</p>	<pre> 8 8 7 find 3839442 6 add 123456 me 5 add 0 granny 4 find 0 3 find 123456 2 del 0 1 del 0 find 0 </pre> <p>ORMAL input.txt input.txt" 9L, 80B written</p> <pre> 1 not found 2 granny 3 me 4 not found </pre> <p>ORMAL output.txt output.txt" 4L, 31B</p>	<pre> 1 find 3839442 </pre> <p>ORMAL input.txt input.txt" 2L, 15B written</p> <pre> not found </pre> <p>ORMAL output.txt output.txt" 1L, 10B</p>	<pre> 100000 1 find 9191015 2 del 3434583 3 del 7307686 4 add 8067796 wgyqccwqx 5 del 147309 6 find 3193649 7 find 2391980 8 del 7194915 9 add 1112766 tweb 10 find 6165267 11 find 9500121 12 del 7203805 13 find 5601789 </pre> <p>ORMAL input.txt input.txt" 100001L, 1589239B written</p> <pre> not found 1 not found 2 not found 3 not found 4 not found 5 not found 6 not found 7 not found 8 not found 9 not found 10 not found 11 not found 12 not found 13 not found </pre> <p>ORMAL output.txt output.txt" 33365L, 333671B</p>
---	---	--	--

диапазона значений входных данных из текста задачи	4	
Пример из задачи	0.003505848000713740 5	8021327
Пример из задачи	0.00378958699911891	8020214
Верхняя граница диапазона значений входных данных из текста задачи	1.054035653000028	16841330

Вывод по задаче: Эффективность Хеш-таблицы зависит от функции хеширования. Неэффективный алгоритм подсчета хеша для ключа может стать бутылочным горлышком для хеш-таблицы.

Задача №4. Прошитый ассоциативный массив

Реализуйте прошитый ассоциативный массив. Ваш алгоритм должен поддер живать 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 ассо циативном 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 ассо циативном 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 оказыва ется 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 результаты выполнения всех операций get, prev, next. Следуйте формату выходного файла из примера.
- Ограничение по времени. 4 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

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

prev two next two next four	
-----------------------------------	--

- Р.с. Задача на [openedu](#), 8 неделя.

Код:

```
import time
import tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", "w")

# length of the bucket array
CAPACITY = 1000000

# node is a node of linked list (I just don't wanna create LinkedList class)

class Node:
    def __init__(self, key, value, prev=None, next_out=None):
        self.key = key
        self.value = value
        self.prev = prev
        self.next_out = next_out
        self.next = None

    def __repr__(self):
        return str(self.value)

    def __str__(self):
        return f"'key': {self.key}, 'value': {self.value}"

class HashTable:
    def __init__(self):
        self.size = 0
        self.buckets = [None]*CAPACITY
        self.top = None

    def hash(self, key):
        summ = 0
        for i in range(len(key)):
            summ += (ord(key[i])) * (125 ** i)

        return summ % CAPACITY
```

```

def insert(self, key, value):
    self.size += 1
    index = self.hash(key)
    node = self.buckets[index]

    if node is None:
        self.buckets[index] = Node(key, value)
        if self.top is not None:
            self.buckets[index].prev = self.top
            self.top.next_out = self.buckets[index]
        self.top = self.buckets[index]
    else:
        prev = node
        while node is not None:
            if prev.key == key:
                prev.value = value
                return
            prev = Node
            node = node.next
        prev.next = Node(key, value)
        if self.top is not None:
            prev.next.prev = self.top.value
        self.top.next_out = prev.next
        self.top = prev.next

def next(self, key):
    index = self.hash(key)
    node = self.buckets[index]

    while node is not None and node.key != key:
        node = node.next

    return node.next_out if node is not None else None

def prev(self, key):
    index = self.hash(key)
    node = self.buckets[index]

    while node is not None and node.key != key:
        node = node.next

    return node.prev if node is not None else None

def get(self, key):
    index = self.hash(key)
    node = self.buckets[index]

    while node is not None and node.key != key:
        node = node.next

    return node.value if node is not None else None

```



```

def exists(self, key):
    return True if self.get(key) != None else False

def remove(self, key):
    index = self.hash(key)
    node = self.buckets[index]

    # persist previous element, so we can remove linked list node
    prev = None

    while node is not None and node.key != key:
        prev = node
        node = node.next
    if node is not None:
        self.size -= 1
        if prev is None:
            if self.top == node:
                self.top = node.prev
            if node.next_out is not None:
                if node.prev is not None:
                    node.prev.next_out = node.next_out
                try:
                    node.next_out.prev = node.prev
                except Exception:
                    print(node.prev)
                    print(node.next_out)
            else:
                if node.prev is not None:
                    node.prev.next_out = None
        self.buckets[index] = node.next
    else:
        if self.top == node:
            self.top = node.prev
        if node.next_out is not None:
            if node.prev is not None:
                node.prev.next_out = node.next_out
            node.next_out.prev = node.prev
        else:
            if node.prev is not None:
                node.prev.next_out = None
        prev.next = node.next

```

```

table = HashTable()
n = int(lines[0])
for i in range(n):
    inputs = lines[i+1].split()
    key = inputs[1]
    match inputs[0]:
        case "put":
            val = inputs[2]
            table.insert(key, val)
        case "delete":

```

```

        table.remove(key)
    case "next":
        output = table.next(key)
        if output is None:
            out.write("<none>\n")
        else:
            out.write(str(output.value) + "\n")
    case "prev":
        output = table.prev(key)
        if output is None:
            out.write("<none>\n")
        else:
            out.write(str(output.value) + "\n")
    case "get":
        val = table.get(key)
        if val == None:
            out.write("<none>\n")
        else:
            out.write(str(val) + "\n")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
      str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()

```

Для решения данной задачи была реализована Хеш-таблица с последним элементом. Связный список содержит узлы, указывающие на предыдущий и следующий элемент

Результат работы кода:

<pre> 1 get two RMAL input.txt <none> RMAL output.txt </pre>	<pre> 4 1 put zero a 2 put one b 3 put two c 4 put three d 5 put four e 6 get two 7 prev two 8 next two 9 delete one 10 delete three 1 get two 2 prev two 3 next two RMAL input.txt /projects/labs/sem1/6_alg/ 6 1 b 2 d 3 c 4 a 5 e 6 <none> RMAL output.txt </pre>	<pre> 5 put whnyz r 2 delete vogpmlacb 1 put sbygiejzcuhatnr atpdrynxmxop 10 next boyibqrnlgzykpyul 9 next fzatnddywpwhcj 8 put crezo kzcqi 7 put ipgmifrkisrc qfnwppxrnmprysf 6 put ca ggq 5 delete stncnpkopgd 4 prev ukwulaexmbjjqylmg 3 delete lycrxarynavumqqvjzt 2 prev kfoafnlmeffyyqyncegx 1 delete xuvjtt get mmm AL input.txt <none> 1 <none> 2 dsaejdhdxmptepgjhy 3 <none> 4 <none> 5 <none> 6 <none> 7 unpvqdebcr 8 <none> 9 <none> 10 <none> 1 <none> 2 <none> 3 <none> AL output.txt </pre>
--	--	---

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0019445200014160946	8018346
Пример из задачи	0.002013150999118807	8022844
Верхняя граница диапазона значений входных данных из текста задачи	2.8191723149975587	32853679

Вывод по задаче: С помощью Хеш-таблицы можно реализовать прошитый ассоциативный массив. В некоторых случаях он работает быстрее красно-черного дерева

Вывод

Хеш-таблица реализует интерфейс ассоциативного массива, хоть и не везде эту структуру используют в данном ключе. Данная структура данных имеет широкое применение на практике. Без нее нельзя представить привычных нам сайтов и сервисов.