

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 15

Выполнил:
Скворцов Д.А.
К3140

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №2. Телефонная книга	3
Задача №4. Прошитый ассоциативный массив	7
Дополнительные задачи	12
Задача №6. Фибоначчи возвращается	12
Задача №8. Почти интерактивная хеш-таблица	15
Вывод	19

Задачи по варианту

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.

Листинг кода.

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

class HashTable:
    def __init__(self, max_size):
        self.size = 0
        self.table = [None] * max_size
        self.max_size = max_size

    def _hash(self, k):
        return hash(k) % self.max_size

    def insert(self, k, v):
        index = self._hash(k)

        if self.table[index] is None:
            self.table[index] = Node(k, v)
            self.size += 1
        else:
            cur_node = self.table[index]
            while cur_node is not None:
                if cur_node.key == k:
                    cur_node.value = v
                    return
                cur_node = cur_node.next

            new_node = Node(k, v)
            new_node.next = self.table[index]
            self.table[index] = new_node
            self.size += 1

    def search(self, k):
        index = self._hash(k)
        cur_node = self.table[index]
        while cur_node is not None:
            if cur_node.key == k:
                return cur_node.value
            cur_node = cur_node.next
        return None # raise KeyError(key)
```

```

def remove(self, k):
    index = self._hash(k)
    prev = None
    cur_node = self.table[index]

    while cur_node is not None:
        if cur_node.key == k:
            if prev is None:
                self.table[index] = cur_node.next
            else:
                prev.next = cur_node.next
            self.size -= 1
            return
        prev = cur_node
        cur_node = cur_node.next

    # raise KeyError(key)

def __len__(self):
    return self.size

def solution(commands):
    dct = HashTable(len(commands))
    ans = []
    for command in commands:
        cmd = command.split()
        if cmd[0] == 'add':
            dct.insert(int(cmd[1]), cmd[2])
        elif cmd[0] == 'del':
            dct.remove(int(cmd[1]))
        else:
            ret = dct.search(int(cmd[1]))
            ans.append("not found" if (ret is None) else ret)
    return ans

```

Обработчик запросов реализован при помощи самописного красса Хеш-таблицы. Для каждого номера генерируется хэш, по которому происходит гораздо более быстрый поиск в словаре. Аналогично происходит удаление и добавление.

Lab6. Task 2

Input	Output
8	not found
find 3839442	granny
add 123456 me	me
add 0 granny	not found
find 0	
find 123456	
del 0	
del 0	
find 0	

Тест 100 элементов:
 Время работы: 0.000103 секунд
 Затрачено памяти: 7.2 Килобайт

Тест 10e3 элементов:
 Время работы: 0.00188 секунд
 Затрачено памяти: 67.1 Килобайт

Тест 10e5 элементов:
 Время работы: 0.14511 секунд
 Затрачено памяти: 6.5 Мегабайт

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000103 секунд	7.2 Килобайт
Медиана диапазона значений входных данных из текста задачи	0.00188 секунд	67.1 Килобайт
Верхняя граница диапазона значений входных данных из текста задачи	0.14511 секунд	6.5 Мегабайт

Вывод по задаче:

Алгоритм работает за линейное время, каждая операция по отдельности занимает константное время. В зависимости от качества алгоритма хеширования время может отличаться.

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 .
- **prev x** – вывести значение, соответствующее ключу, находящемуся в ассоциативном массиве, который был вставлен позже всех, но до x, или , b 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 , 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 ;
 - 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 находится 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.

Листинг кода.

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

        self.next_arr = None
        self.prev_arr = None

class HashTable:
    def __init__(self, max_size):
        self.size = 0
        self.table = [None] * max_size
        self.max_size = max_size

    def _hash(self, k):
        return hash(k) % self.max_size

    def insert(self, k, v) -> Node:
        index = self._hash(k)

        if self.table[index] is None:
            self.table[index] = Node(k, v)
            self.size += 1
            return self.table[index]

        cur_node = self.table[index]

        while cur_node is not None:
            if cur_node.key == k:
                cur_node.value = v
                return cur_node
            cur_node = cur_node.next
```



```

        new_node = Node(k, v)
        new_node.next = self.table[index]
        self.table[index] = new_node
        self.size += 1

    return new_node

def search(self, k, mode=None) -> Node:
    index = self._hash(k)
    cur_node = self.table[index]
    while cur_node is not None:
        if cur_node.key == k:
            return (cur_node.value if mode is None else cur_node)
        cur_node = cur_node.next
    return None

def remove(self, k) -> Node:
    index = self._hash(k)
    prev = None
    cur_node = self.table[index]

    while cur_node is not None:
        if cur_node.key == k:
            if prev is None:
                self.table[index] = cur_node.next
            else:
                prev.next = cur_node.next
            self.size -= 1
            return cur_node
        prev = cur_node
        cur_node = cur_node.next

    return None

def __len__(self):
    return self.size

def __contains__(self, k):
    return self.search(k) is None

class AssociationArray(HashTable):
    def __init__(self, max_size):
        super().__init__(max_size)

```

```

        self.head = None

    def insert(self, k, v):
        node = super().insert(k, v)
        if self.head is not None:
            connect(self.head, node)
        self.head = node

    def remove(self, k):
        node = super().remove(k)
        if node is not None:
            reconnect(node)

    def prev(self, k):
        node = super().search(k, mode="Node")
        if node is not None and node.prev_arr is not None:
            return node.prev_arr.value
        return None

    def next(self, k):
        node = super().search(k, mode="Node")
        if node is not None and node.next_arr is not None:
            return node.next_arr.value
        return None

def connect(node1, node2):
    if node1 is not None:
        node1.next_arr = node2
    if node2 is not None:
        node2.prev_arr = node1

def reconnect(node):
    connect(node.prev_arr, node.next_arr)

def solution(commands):
    dct = AssociationArray(len(commands))
    ans = []
    for command in commands:
        cmd = command.split()
        if cmd[0] == 'put':
            dct.insert(cmd[1], cmd[2])
        elif cmd[0] == 'delete':
            dct.remove(cmd[1])

```

```

elif cmd[0] in ('get', 'prev', 'next'):
    if cmd[0] in 'get':
        ret = dct.search(cmd[1])
    elif cmd[0] == 'prev':
        ret = dct.prev(cmd[1])
    else:
        ret = dct.next(cmd[1])
    ans.append("<none>" if (ret is None) else ret)
return ans

```

Для каждого элемента при добавлении также запоминаем следующий и предыдущий, чтобы при помощи связанного списка производить быстрое удаление и поиск следующего/предыдущего.

Lab6. Task 4

Input	Output
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	

Lab6. Task 4

Тест 100 элементов:
 Время работы: 0.000121 секунд
 Затрачено памяти: 8.9 Килобайт

Тест 10e3 элементов:
 Время работы: 0.001698 секунд
 Затрачено памяти: 83.7 Килобайт

Тест 5*10e5 элементов:
 Время работы: 0.518511 секунд
 Затрачено памяти: 41.2 Мегабайт

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000121 секунд	8.9 Килобайт
Медиана диапазона значений входных данных из текста задачи	0.001698 секунд	83.7 Килобайт
Верхняя граница диапазона значений входных данных из текста задачи	0.518511 секунд	41.2 Мегабайт

Вывод по задаче:

Алгоритм работает.

Дополнительные задачи

Задача №6. Фибоначчи возвращается

Вам дается последовательность чисел. Для каждого числа определите, является ли оно числом Фибоначчи. Напомним, что числа Фибоначчи определяются, например, так: $F_0 = F_1 = 1$ (1) $F_i = F_{i-1} + F_{i-2}$ для $i \geq 2$.

- **Формат ввода / входного файла (input.txt).** Первая строка содержит одно число N ($1 \leq N \leq 10^6$) - количество запросов. Следующие N строк содержат по одному целому числу. При этом соблюдаются следующие ограничения при проверке: 1. Размер каждого числа не превосходит 5000 цифр в десятичном представлении. 2. Размер входа не превышает 1 Мб.
- **Формат вывода / выходного файла (output.txt).** Для каждого числа, данного во входном файле, выведите «Yes», если оно является числом Фибоначчи, и «No» в противном случае.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 128 мб.

Листинг кода.

```
def fill_fib(x, last_n):
    global fibs
    global fibs_by_num
    i = last_n
    while True:
        new_fib = fibs_by_num[i-1] + fibs_by_num[i]
        fibs.add(new_fib)
        fibs_by_num.append(new_fib)
        i += 1
        if new_fib > x:
            return False, i
        elif new_fib == x:
            break
    return True, i

def solution(lst):
    global fibs
    global fibs_by_num
    fibs = set([1])
    fibs_by_num = [1, 1]
    res = []
    last_n = 1
    for i in lst:
        x = int(i)
        if x not in fibs:
            if x > fibs_by_num[last_n]:
                verdict, last_n = fill_fib(x, last_n)
                res.append("Yes" if verdict else "No")
            else:
                res.append("No")
        else:
            res.append("Yes")

    return res
```

Для решения функция `fill_fib` находит все новые числа фибоначчи, начиная с последнего известного, и возвращает результат, как только доходит до проверяемого числа или перескакивает его.

Lab6. Task 6

Input	Output
8	Yes
1	Yes
2	Yes
3	No
4	Yes
5	No
6	No
7	Yes
8	

Тест 100 элементов:
 Время работы: 5.4e-05 секунд
 Затрачено памяти: 3.8 Килобайт

Тест 10e6 элементов:
 Время работы: 0.252006 секунд
 Затрачено памяти: 8.1 Мегабайт

Тест 5000 разрядов:
 Время работы: 0.052138 секунд
 Затрачено памяти: 28.1 Мегабайт

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000054 секунд	3.8 Килобайт
Медиана диапазона значений входных данных из текста задачи	0.252006 секунд	8.1 Мегабайт
Верхняя граница диапазона значений входных данных из текста задачи	0.052138 секунд	28.1 Мегабайт

Вывод по задаче:

Время работы алгоритма - $O(n)$, но он расходует память на сохранение всех найденных чисел Фибоначчи. Каждое последующее число считается

итеративно и при помощи последних известных, что позволяет не экономить память.

Задача №8. Почти интерактивная хеш-таблица

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:

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

Требуется N раз выполнить следующую последовательность операций:

- Если X содержится в таблице, то установить $A \leftarrow (A + AC) \bmod 103$, $B \leftarrow (B + BC) \bmod 10^{15}$.
- Если X не содержится в таблице, то добавить X в таблицу и установить $A \leftarrow (A + AD) \bmod 10^3$, $B \leftarrow (B + BD) \bmod 10^{15}$.
- Установить $X \leftarrow (X \cdot A + B) \bmod 10^{15}$. Начальные значения X, A и B , а также N, AC, BC, AD и BD 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 AC, BC, AD и BD такие что $0 \leq AC, AD < 103, 0 \leq BC, BD < 10^{15}$.
- **Формат выходного файла (output.txt).** Выведите значения X, A и B после окончания работы. 11
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 256 мб.

Листинг кода.

```
class HashSet:
    def __init__(self, max_size):
        self.size = 0
        self.table = array('q', [EMPTY] * max_size)
```

```

        self.aux_list = []
        self.max_size = max_size

    def _hash(self, k):
        return k % self.max_size

    def add(self, k):
        index = self._hash(k)

        if self.table[index] == EMPTY:
            self.table[index] = k
            self.size += 1
            return

        cur_value = self.table[index]
        if cur_value == k: return

        if cur_value >= 0:
            idx = len(self.aux_list)
            lst = [cur_value, k]
            self.aux_list.append(lst)
            self.table[index] = cur_value = -idx
        else:
            lst = self.aux_list[-cur_value]
            for key in lst:
                if key == k: return
            lst.append(k)
        self.size += 1

    def __contains__(self, k):
        index = self._hash(k)
        cur_node = self.table[index]
        if cur_node == EMPTY:
            return False
        elif cur_node >= 0:
            return cur_node == k

        return k in self.aux_list[-cur_node]

    def __len__(self):
        return self.size

```



```
def solution(lst):
    n,x,a,b, ac,bc,ad,bd = lst
    size = n * 2 if n < 10**6 else 2017963
    hash_set = HashSet(size) # prime for distribution
    t = 10 ** 15
    for _ in range(n):
        if x in hash_set:
            a = (a + ac) % 1000
            b = (b + bc) % t
        else:
            a = (a + ad) % 1000
            b = (b + bd) % t
            hash_set.add(x)
        x = (x * a + b) % t
    return x, a, b
```

Чтобы сделать самописный hash-set более эффективным для задания размера его памяти используется простое число, так как распределение хешей в нем более равномерное. Также для оптимизации по памяти используется агрегат, позволяющий хранить либо число, либо ссылку на несколько чисел в выносном массиве.

Lab6. Task 8

Input	Output
4 0 0 0	3 1 1
1 1 0 0	

Тест 100 элементов:
 Время работы: 0.000153 секунд
 Затрачено памяти: 6.7 Килобайт

Тест 10e5 элементов:
 Время работы: 0.170469 секунд
 Затрачено памяти: 7.2 Мегабайт

Тест 10e6 элементов:
 Время работы: 1.392178 секунд
 Затрачено памяти: 42.3 Мегабайт

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.000153 секунд	6.7 Килобайт

входных данных из текста задачи		
Медиана диапазона значений входных данных из текста задачи	0.170469 секунд	7.2 Мегабайт
Верхняя граница диапазона значений входных данных из текста задачи	1.392178 секунд	42.3 Мегабайт

Вывод по задаче:

Алгоритм работает быстро, за линейное время и при этом затрачивает минимум памяти. поскольку не создает связного списка, если элемент с определённым хешем единственен.

Вывод

Структуры данных на основе хэширования (Hash-table, Hash-set) позволяют значительно ускорять работу с данными, требующую частого поиска или добавления/удаления объектов по их значению. Они существенно уменьшают время выполнения программы. Взамен делается упор на качество хеш-функции и выделение памяти, которые надо компенсировать при постоянном увеличении размеров.