

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

|  |
| --- |
| **Институт информационных технологий (ИТ)** |
| **Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)** |

**ОТЧЁТ ПО ПРАКТИЧЕСКОМУ ЗАДАНИЮ №2**

**по дисциплине «Структуры и алгоритмы обработки данных»**

Тема: **«**Хеширование и организация быстрого поиска данных»

|  |  |  |  |
| --- | --- | --- | --- |
| Отчет представлен к рассмотрению:  Студент группы ИНБО-01-20 | «28» сентября 2021 г. |  | Салов В.Д. |
| (подпись) | | | |
| Преподаватель | «28» сентября 2021 г. |  | Сорокин А.В. |
|  | (подпись) | | |

Москва, 2021 г.

CОДЕРЖАНИЕ

[Цель работы 3](#_Toc84265575)

[Постановка задачи 3](#_Toc84265576)

[Имеющиеся данные: 3](#_Toc84265577)

[Результат: 3](#_Toc84265578)

[Управление файлом: 3](#_Toc84265579)

[Подход к решению. 3](#_Toc84265580)

[Алгоритмы операций на псевдокоде. 5](#_Toc84265581)

[Код приложения. 5](#_Toc84265582)

[Результаты выполнения операций с хеш-таблицами. 9](#_Toc84265583)

[Скриншоты содержания файла и хеш-таблицы. 11](#_Toc84265584)

[Время поиска элемента с заданным ключом. 12](#_Toc84265585)

[Вывод 14](#_Toc84265586)

[Список информационных источников 15](#_Toc84265587)

# Цель работы

Получение навыков по разработке хеш-таблиц и их применению.

# Постановка задачи

Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к записям файла, структура записи которого приведена в варианте.

**Вариант 9.**

|  |  |
| --- | --- |
| Тип хеш-таблицы  (способ реализации коллизий) | Структура записи файла (ключ – подчеркнутое поле) |
| Цепное хеширование | Страховой полис: номер, компания, фамилия владельца |

## Имеющиеся данные:

* хеш-таблица для реализации коллизий по методу *Цепное хеширование*;
* двоичный файл с записями фиксированной длины;
* структура записи файла: номер, компания, фамилия владельца.

## Результат:

* приложение, выполняющее операции;
* управление хеш-таблицей: вставка ключа, удаление ключа, нахождение ключа в хеш-таблице, рехеширование.

## Управление файлом:

* посредством хеш-таблицы: считывание записи из файла при поиске записи;
* добавление записи в файл, удаление записи из файла, чтение записи файла по заданному номеру записи.

## Подход к решению.

1. Разработан класс хеш-таблицы, содержащий размер хеш-таблицы и методы добавления, удаления, получения элемента, рехеширования и вывода в консоль.
2. Создан класс узла однонаправленного списка, содержащий информационную часть и ссылку на следующий узел.
3. Добавлены классы для работы с файлом и для управления однонаправленным списком.

Класс для работы с файлом включает в себя методы получения и изменения строки, переключения между файлами (при рехешировании), а также генератор прохода по всем строкам, содержащим информационную часть узлов.

Класс однонаправленного списка содержит конструктор, генерирующий список по строке, методы добавления и удаления элементов, а также метод получения строкового представления.

Таким образом данные классы инкапсулируют работу с файлом и работу с однонаправленным списком в простой интерфейс.

1. Разработаны методы обработки хеш-таблицы:
   * Вставка: вычисление хеша нового элемента и его расположение в соответствующем списке.
   * Удаление: нахождение соответствующего списка и удаление из него элемента по ключу.
   * Поиск: нахождение по хеш-функции соответствующего списка и поиск нужного элемента в нём.
   * Вывод: вывод содержимого файла в консоль.
   * Рехеширование: увеличение размера таблицы и распределение по ней всех элементов.
2. Для хранения данных используется текстовый файл со следующей структурой:

<номер>,<компания>,<фамилия >;<номер>,<компания>,<фамилия>

<номер>,<компания>,<фамилия>

Каждый список записывается в строку, соответствующей его номеру; каждый узел отделяется от другого с помощью символа «;»; каждое поле узла отделяется от другого с помощью символа «,».

Добавлены методы редактирования файла:

* Для добавления записи в файл строка, соответствующая данному элементу, заменяется на новое строковое отображение списка.
* Для удаления записи из файла строка, соответствующая данному элементу, заменяется на новое строковое отображение списка.

## Алгоритмы операций на псевдокоде.

Вставка в таблицу:

func вставка(узел\_списка):

h := хеш-функция(узел\_списка.ключ);

ul := однонаправленный\_список(строка\_в\_файле\_на\_позиции (h));

ul.добавить(узел\_списка);

записать\_в\_файл\_строку\_на\_позицию(ul.строка, h);

Удаление из таблицы:

func удаление(ключ):

h := хеш-функция(узел\_списка.ключ);

ul := однонаправленный\_список(строка\_в\_файле\_на\_позиции (h));

ul.удалить(ключ);

записать\_в\_файл\_строку\_на\_позицию(ul.строка, h);

Поиск по ключу:

func поиск(ключ):

h := хеш-функция(узел\_списка.ключ);

ul := однонаправленный\_список(строка\_в\_файле\_на\_позиции (h));

res := ul.поиск(ключ);

возврат res;

## Код приложения.

Класс узла списка:

class Node:  
 def \_\_init\_\_(self, number, company, name):  
 self.number = number  
 self.company = company  
 self.name = name  
 self.next = None  
  
 @classmethod  
 def from\_string(cls, line: str):  
 lines = line.split(',')  
 return Node(int(lines[0]), lines[1], lines[2])  
  
 def \_\_str\_\_(self):  
 return f'{self.number},{self.company},{self.name}'

Класс однонаправленного спиcка:

from .Node import Node  
  
class UnidirectionalList:  
 def \_\_init\_\_(self, line: str):  
 self.len = 0  
 self.start = None  
 if line != '' and line != '\n':  
 lines = line.split(';')  
 for node\_line in lines:  
 self.add(Node.from\_string(node\_line))  
  
 def \_\_str\_\_(self):  
 current\_node = self.start  
 res = []  
 while current\_node is not None:  
 res.append(str(current\_node))  
 current\_node = current\_node.next  
 res = ';'.join(res[::-1]).replace('\n', '')  
 return res  
  
 def add(self, node: Node):  
 old\_start = self.start  
 self.start = node  
 node.next = old\_start  
 self.len += 1  
 return self.len  
  
 def remove(self, key: int):  
 if self.start is not None:  
 if self.start.number == key:  
 self.start = self.start.next  
 else:  
 current\_node = self.start  
 while current\_node.next is not None:  
 if current\_node.next.number == key:  
 current\_node.next = current\_node.next.next  
 else:  
 current\_node = current\_node.next  
  
 def get(self, key: int):  
 current\_node = self.start  
 while current\_node is not None:  
 if current\_node.number == key:  
 return current\_node  
 current\_node = current\_node.next

Класс для чтения файла:

class FileWorking:  
 def \_\_init\_\_(self):  
 self.work\_file = 'file1.txt'  
 self.temp\_file = 'file2.txt'  
  
 def get\_line(self, n: int):  
 file = open(self.work\_file, 'r')  
 res = file.readlines()[n]  
 file.close()  
 res.replace('\n', '')  
 return res  
  
 def set\_line(self, n: int, line: str):  
 file = open(self.work\_file, 'r')  
 lines = file.readlines()  
 lines = list(map(lambda x: x.replace('\n', ''), lines))  
 file.close()  
 lines[n] = line  
 file = open(self.work\_file, 'w')  
 file.writelines(map(lambda x: x + '\n', lines))  
 file.close()  
  
 def set\_lines\_count(self, n):  
 file = open(self.work\_file, 'w')  
 file.writelines(['\n'] \* n)  
 file.close()  
  
 def iterate\_throw\_old\_nodes(self):  
 file = open(self.temp\_file, 'r')  
 lines = file.readlines()  
 file.close()  
 for line in lines:  
 for node\_line in line.split(';'):  
 if node\_line != '\n':  
 yield node\_line  
  
 def swap\_files(self):  
 self.work\_file, self.temp\_file = self.temp\_file, self.work\_file  
  
 def get\_all\_lines(self):  
 file = open(self.work\_file, 'r')  
 lines = file.readlines()  
 file.close()  
 return lines  
  
  
fw = FileWorking()

Класс хеш-таблицы:

from .UnidirectionalList import UnidirectionalList, Node  
from .FileWorking import fw  
  
class HashTable:  
 def \_\_init\_\_(self):  
 self.size = 2  
 fw.set\_lines\_count(2)  
  
 def hash(self, x: int):  
 return x % self.size  
  
 def add(self, number: int, company: str, name: str):  
 node = Node(number, company, name)  
 h = self.hash(node.number)  
 ul = UnidirectionalList(fw.get\_line(h))  
 count = ul.add(node)  
 fw.set\_line(h, str(ul))  
 if count > self.size \* 0.75:  
 self.rehash()  
  
 def remove(self, number):  
 h = self.hash(number)  
 ul = UnidirectionalList(fw.get\_line(h))  
 ul.remove(number)  
 fw.set\_line(h, str(ul))  
  
 def get(self, number):  
 h = self.hash(number)  
 ul = UnidirectionalList(fw.get\_line(h))  
 return str(ul.get(number)).replace('\n', '')  
  
 def rehash(self):  
 fw.swap\_files()  
 self.size \*= 2  
 fw.set\_lines\_count(self.size)  
 for node\_line in fw.iterate\_throw\_old\_nodes():  
 self.\_\_add\_node(Node.from\_string(node\_line))  
  
 def fill\_from\_file(self, file\_path):  
 file = open(file\_path, 'r')  
 for line in file:  
 line = line.replace('\n', '')  
 if line != '':  
 self.\_\_add\_node(Node.from\_string(line))  
  
 def print(self):  
 print('-------')  
 print(''.join(fw.get\_all\_lines()), end='')  
 print('-------')  
  
 def \_\_add\_node(self, node: Node):  
 self.add(node.number, node.company, node.name)

## Результаты выполнения операций с хеш-таблицами.

Для проведения тестирования было добавлено несколько элементов.

Код теста:

from Lab2N import \*  
  
hs = HashTable()  
hs.add(1, 'ab', 'gh')  
hs.print()  
hs.add(2, 'cd', 'ij')  
hs.print()  
hs.add(5, 'ef', 'kl')  
hs.print()

Результат:

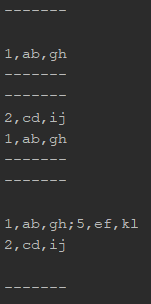
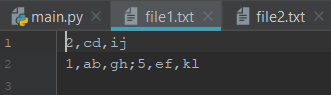
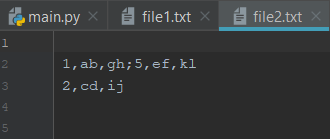


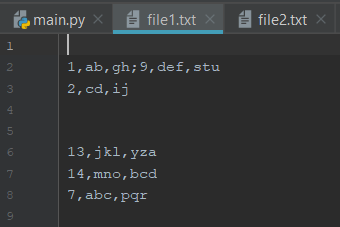
Рисунок 1 – ввод элементов.

Таблица вывелась три раза с новыми элементами. После добавления третьего элемента произошло **рехеширование**. Это можно увидеть по размеру таблицы. Также произошла **коллизия**, и элемент с ключом «5» встал в список за элементом с ключом «1», так как значение их хеш-функций совпало.

Добавим ещё несколько элементов, а затем удалим часть элементов.

Код теста:

hs.add(7, 'abc', 'pqr')  
hs.add(9, 'def', 'stu')  
hs.add(12, 'ghi', 'vwx')  
hs.add(13, 'jkl', 'yza')  
hs.add(14, 'mno', 'bcd')  
hs.print()  
  
hs.remove(12)  
hs.remove(5)  
hs.print()

Результат:

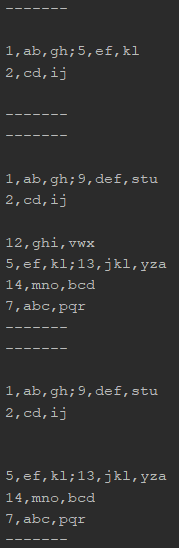
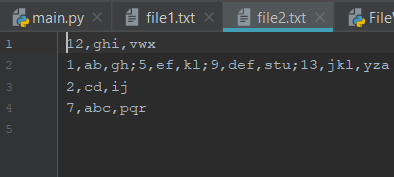


Рисунок 2 – Результаты тестирования.

Как можно заметить, указанные нам записи добавились, а после удалились. Далее протестируем возможность поиска элементов по ключу.

Код теста:

print(hs.get(9))  
print(hs.get(2))  
print(hs.get(5))

Результат:



Рисунок 3 – Результат тестирования.

Программа работает верно, поскольку существующие элементы были найдены и выведены на экран, а не существующие – нет, что можно понять из надписи «None».

## Скриншоты содержания файла и хеш-таблицы.

Полный вывод программы при тестировании:

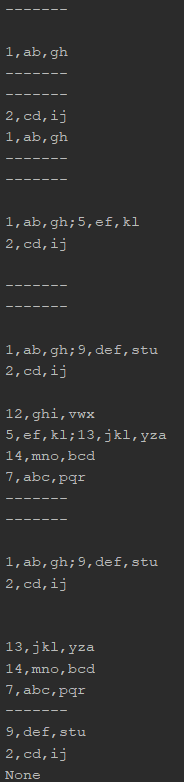


Рисунок 4 – Полный вывод программы при тестировании

Содержимое файла:

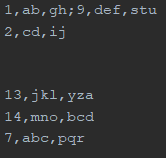


Рисунок 5 – Содержимое файла.

## Время поиска элемента с заданным ключом.

Заполним хеш-таблицу большим количеством элементов. Для примера заполним её случайным образом, добавляя в неё элементы со значением ключа от 0 до 9999. Также добавим в хеш-таблицу элементы с ключом 0, 5120 и 9999 соответственно. После чего произведём поиск данных элементов (первого элемента, элемента «где-то посередине» и последнего элемента соответственно) и посчитаем затрачиваемое время для каждого поиска.



Рисунок 6 – Код для проверки на время работы метода получения элемента.

В результате было выяснено, что время доступа для всех записей одинаково.

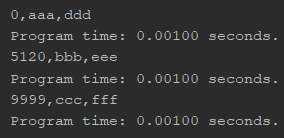


Рисунок 7 – Время получения разных элементов.

# Вывод

В ходе работы были приобретены практические навыки по использованию хеш-таблиц и их применению.

# Список информационных источников

1. Лекции по дисциплине «Структуры и алгоритмы обработки данных» /

Л. А. Скворцова, МИРЭА – Российский технологический университет, 2021.