**Практическая работа №3**

**Тема:** «Хэш-таблицы».

**Цель:** изучить реализацию хэш-таблицы с открытой адресацией я высокоуровневом языке программирование Python.

Ход работы :

Хеш-таблица (hash table) — это специальная структура данных для хранения пар ключей и их значений. По сути это ассоциативный массив, в котором ключ представлен в виде хеш-функции.

Создадим хэш-таблицу с открытой адресацией для простейшего телефона справочника. Для этого определим структуру контакта, которая представлена на рисунке 1

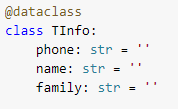


Рис. 1 Структура контакта.

Для одной ячейки таблицы определим следующую структуру, представленную на рисунке 2,

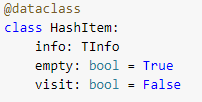


Рис. 2 Структура ячейки таблицы

где empty - флаг, указывающий, что ячейка свободна, а visit - флаг, указывающий, что ячейка просматривалась.

Для вычисления значения хэша будем использовать следующую функцию, представленную на рисунке 3

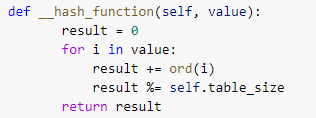


Рис. 3 Хэш-функция

Диаграмма деятельности для \_hash\_function представлена на рисунке 4

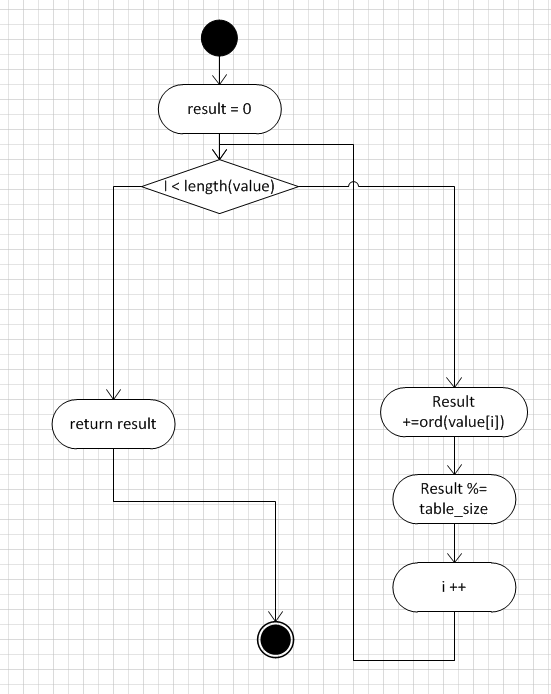


Рис. 4 Диаграмма деятельности для \_hash\_function.

Функция добавления элемента представлена на рисунке 5.

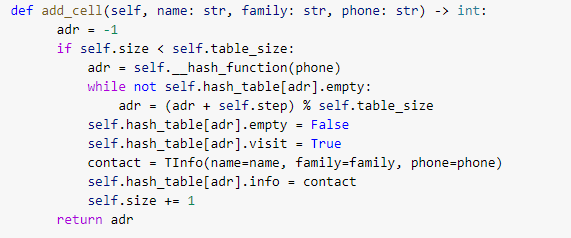


Рис. 5 Добавление элемента в хэш-таблицу.

Диаграмма деятельности для добавления элемента, в таблицу методом открытой адресации представлена на рисунке 6

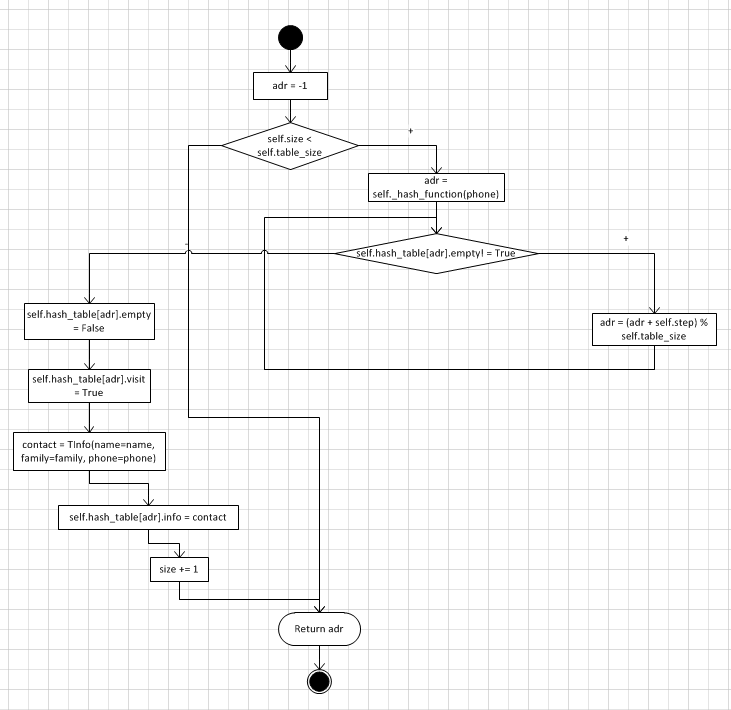


Рис. 6. Диаграмма деятельности для добавления элемента, в таблицу методом открытой адресации

Для поиска элемента, надо убедиться, что флаги visit каждой ячейки сброшены к дефолтным значениям. Для этого мы используем функцию, код которой представлен на рисунке 7.

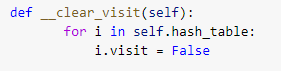


Рис. 7 Сброс значений к дефолтным

Диаграмма деятельности сброса флагов visit к дефолтным значениям представлена на рисунке 8

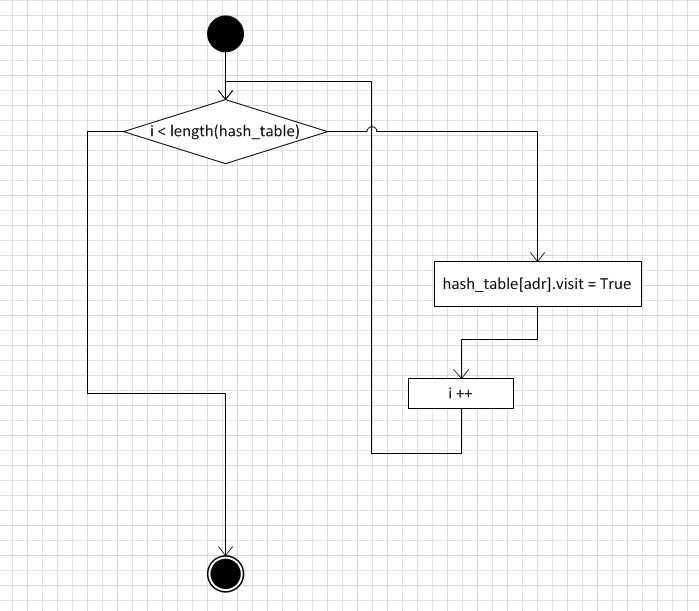


Рис. 8 Диаграмма деятельности сброса флагов visit к дефолтным значениям

Функция поиска значения в таблице представлена на рисунке 9.

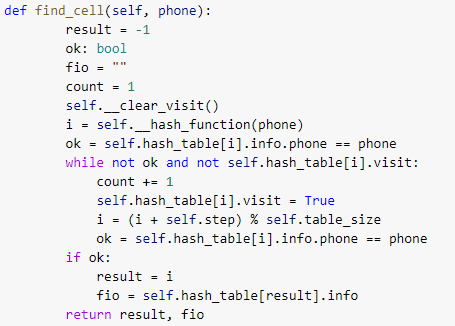


Рис. 9 Поиск элемента в таблице

Диаграмма деятельности для поиска элемента представлена на рисунке 10.

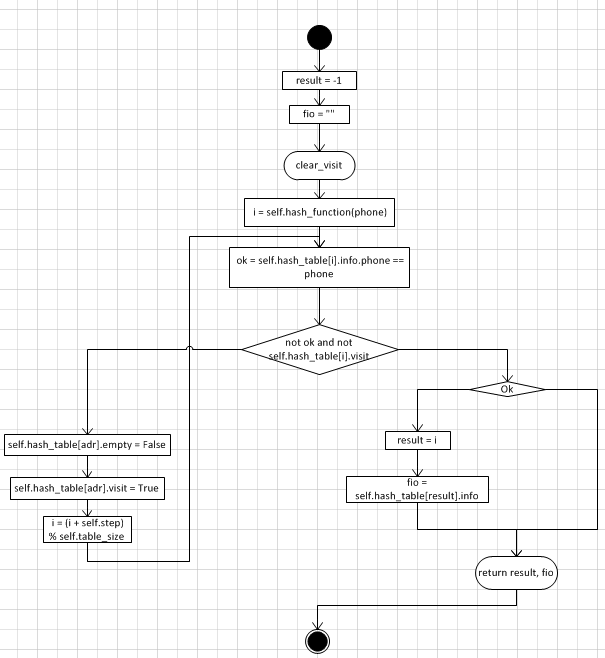


Рис. 10 Поиск элемента в хэш-таблице с открытой адресацией.

Для удаления элемента реализован метод, код которого представлен на рисунке 11.

Действие кода сводится к нахождению нужного элемента и выставление флага empty в позицию True.

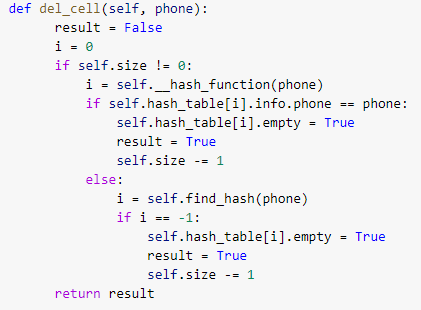


Рис. 11 Удаление элемента

Диаграмма деятельности для удаления элемента представлена на рисунке 12.

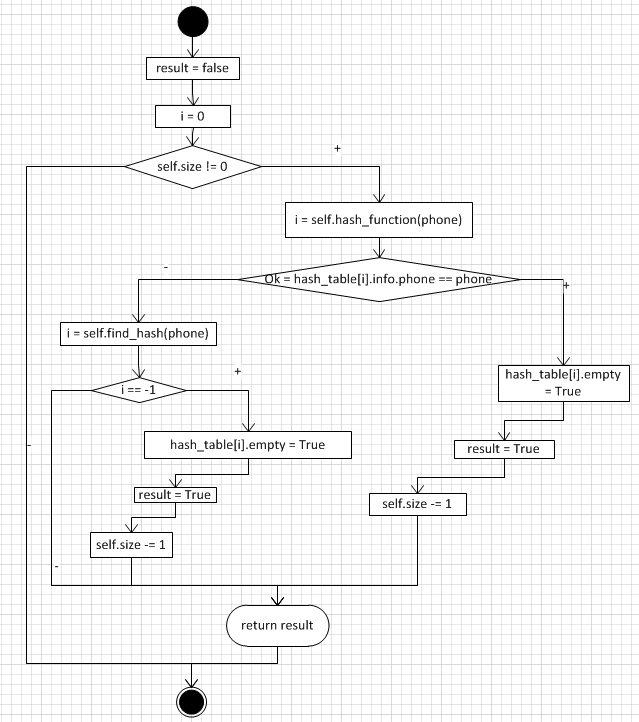


Рис. 12 Удаление элемента из хэш-таблицы.

Полностью исходный код для класса хеш-таблиц представлен ниже.

from dataclasses import dataclass

from typing import List

@dataclass

class TInfo:

    phone: str = ''

    family: str = ''

    name: str = ''

@dataclass

class HashItem:

    info: TInfo

    empty: bool = True

    visit: bool = False

class Hash:

    info: TInfo

    def \_\_init\_\_(self, table\_size=10):

        self.table\_size = table\_size

        self.info = TInfo()

        self.hash\_table = [HashItem(info=self.info) for \_ in range(self.table\_size)]

        self.size = 0

        self.step = 21

    def \_\_hash\_function(self, value):

        result = 0

        for i in value:

            result += ord(i)

            result %= self.table\_size

        return result

    def add\_cell(self, name: str, family: str, phone: str) -> int:

        adr = -1

        if self.size < self.table\_size:

            adr = self.\_\_hash\_function(phone)

            while not self.hash\_table[adr].empty:

                adr = (adr + self.step) % self.table\_size

            self.hash\_table[adr].empty = False

            self.hash\_table[adr].visit = True

            contact = TInfo(name=name, family=family, phone=phone)

            self.hash\_table[adr].info = contact

            self.size += 1

        return adr

    def \_\_clear\_visit(self):

        for i in self.hash\_table:

            i.visit = False

    def find\_cell(self, phone):

        result = -1

        ok: bool

        fio = ""

        count = 1

        self.\_\_clear\_visit()

        i = self.\_\_hash\_function(phone)

        ok = self.hash\_table[i].info.phone == phone

        while not ok and not self.hash\_table[i].visit:

            count += 1

            self.hash\_table[i].visit = True

            i = (i + self.step) % self.table\_size

            ok = self.hash\_table[i].info.phone == phone

        if ok:

            result = i

            fio = self.hash\_table[result].info

        return result, fio

    def del\_cell(self, phone):

        result = False

        i = 0

        if self.size != 0:

            i = self.\_\_hash\_function(phone)

            if self.hash\_table[i].info.phone == phone:

                self.hash\_table[i].empty = True

                result = True

                self.size -= 1

            else:

                i = self.find\_hash(phone)

                if i == -1:

                    self.hash\_table[i].empty = True

                    result = True

                    self.size -= 1

        return result

    def out(self):

        out = "{:<6}{:<20}{:<20}{:<20}".format("N", "Name", "Family", "Phone") + '\n'

        for i in range(self.table\_size):

            name: str = self.hash\_table[i].info.name

            family = self.hash\_table[i].info.family

            phone = self.hash\_table[i].info.phone

            out += "{:<6}{:<20}{:<20}{:<20}".format(i + 1, name, family, phone) + '\n'

        return out

Вывод: в данной практической работе была изучена и реализована хеш-таблица с открытой адресацией на высокоуровневом языке программирования Python.