

Практическая работа №3.

Тема: «Хэш-таблицы».

Цель работы: изучить реализацию хэш-таблицы с открытой адресацией на языке Python.

Хеш-таблица — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

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

Структура контакта.

```
@dataclass
class TInfo:
    phone: str = " "
    name: str = " "
```

Для ячеек таблицы определим следующую структуру, представленную на листинге 2.

Структура ячейки таблицы.

```
@dataclass
class HashItem:
    info: TInfo
    empty: bool = True
    visit: bool = False
```

где empty указывает на то, что ячейка пуста, а visit – на то, что ячейка посещалась.

Для вычисления значения хэша будем использовать следующую хэш-функцию, представленную на листинге 3.

АИСД.09.03.02.210000

Изм	Лист	№ докум.	Подпис	Дат
Разраб.		Семикин	ь	ПР
Провер.		Бережа А.Н.		
Реценз				
Н. Контр.				
Утверд.				

Практическая
работа №3 «Хэш-
таблицы».

Лит.	Лист	Листов
	2	
ИСОиП (филиал) ДГТУ в г.Шахты ИСТ-Тб21		

Хэш-функция.

```
def __hash_function(self, s):  
    result = 0  
    for i in range(len(s)):  
        result += int(s[i]) * i  
    result //= self.size_table  
    return result
```

Диаграмма деятельности для данной хэш-функции представлена на рисунке 1.

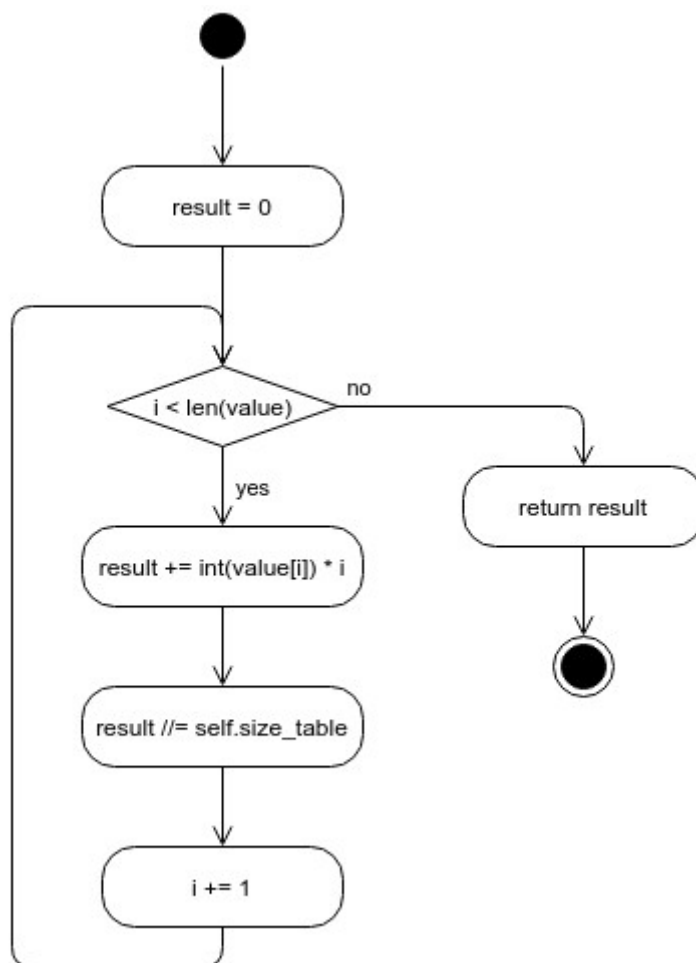


Рисунок 1. Диаграмма деятельности для хэш-функции.

Функция для добавления элемента в хэш-таблицу представлена на листинге 4. Диаграмма деятельности для функции добавления элемента представлена на рисунке 2.

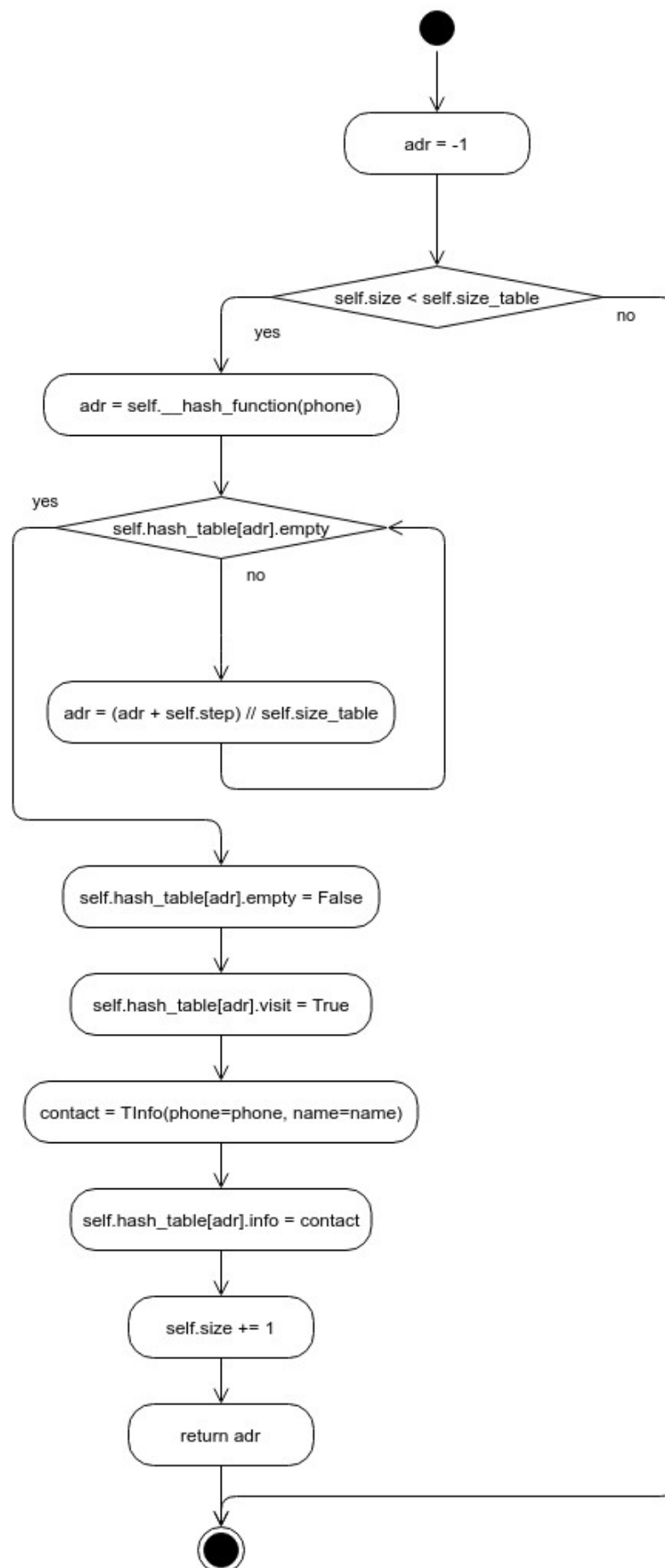


Рисунок 2. Диаграмма деятельности для добавления элемента в хэш-таблицу.

Функция добавления элемента.

```
def add_hash(self, name: str, phone: str):
    adr = -1
    if self.size < self.size_table:
        adr = self.__hash_function(phone)
        while not self.hash_table[adr].empty:
            adr = (adr + self.step) // self.size_table
        self.hash_table[adr].empty = False
        self.hash_table[adr].visit = True
        contact = TInfo(phone=phone, name=name)
        self.hash_table[adr].info = contact
        self.size += 1
    return adr
```

Для поиска элемента в хэш-таблице необходимо установить флаги visit в значение False. Для этого используется функция, представленная на листинге 5.

Функция для обновления флагов visit.

```
def __clear_visit(self):
    for i in self.hash_table:
        i.visit = False
```

Функция для поиска элемента в хэш-таблице представлена на листинге 6. Диаграмма деятельности для этой функции представлена на рисунке 3.

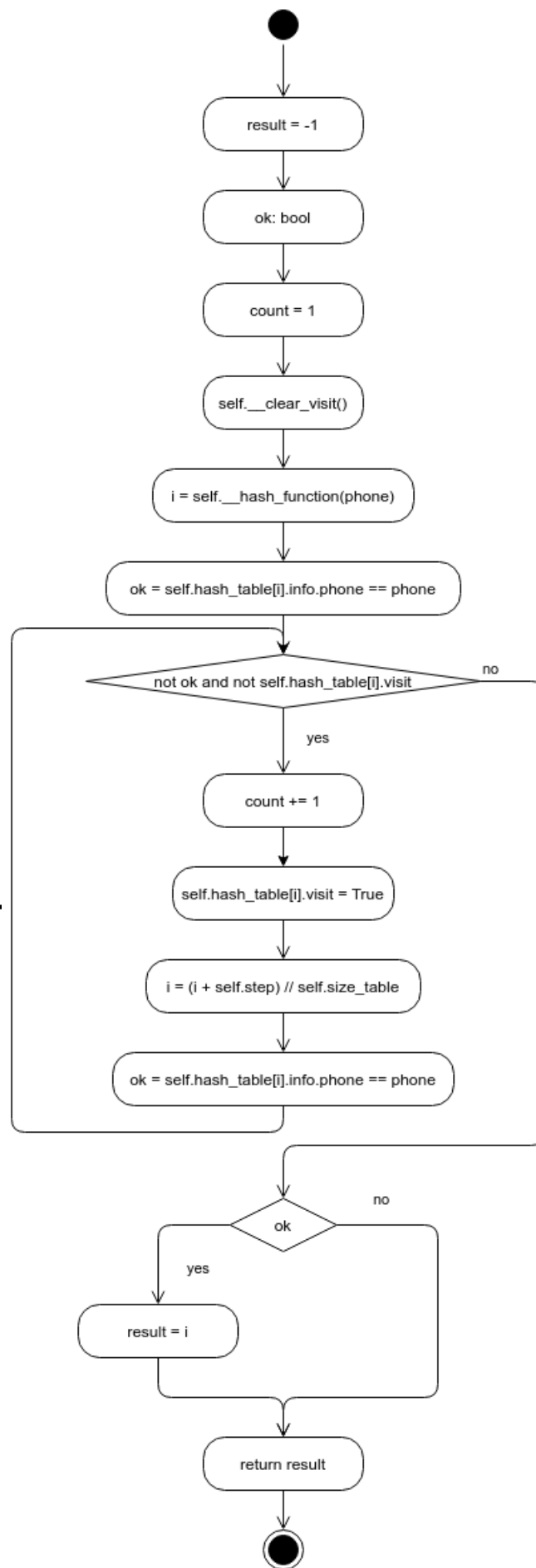


Рисунок 3. Диаграмма деятельности для функции поиска элемента.

Функция для поиска элемента.

```
def find_hash(self, phone: str):
    result = -1
    ok: bool
    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.size_table
        ok = self.hash_table[i].info.phone == phone
    if ok:
        result = i + 1
    return result
```

Для удаления элемента хэш-таблицы реализуем функцию, представленную на листинге 7. Суть данной операции состоит в вычислении хэш-функции для элемента, или его поиске, и в дальнейшем обнулении значения контакта и выставлении флага empty в значение True. Диаграмма деятельности для этой функции представлена на рисунке 4.

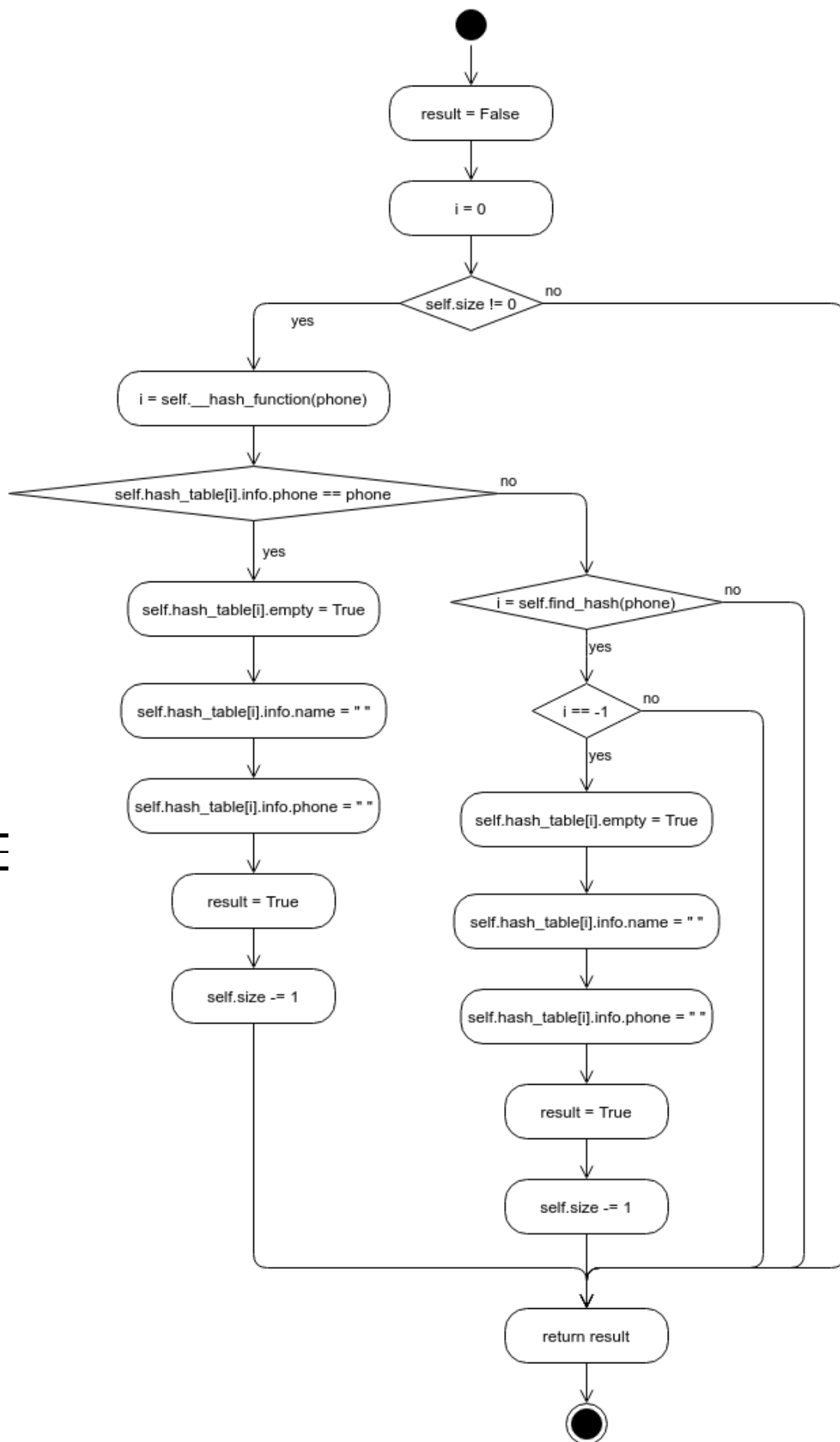


Рисунок 4. Диаграмма деятельности для функции удаления элемента.

```

def del_hash(self, phone: str):
    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
            self.hash_table[i].info.name = " "
            self.hash_table[i].info.phone = " "
            result = True
            self.size -= 1
        else:
            i = self.find_hash(phone)
            if i == -1:
                self.hash_table[i].empty = True
                self.hash_table[i].info.name = " "
                self.hash_table[i].info.phone = " "
                result = True
                self.size -= 1
    return result

```

Полностью исходный код для класса хэш-таблицы и вспомогательных классов представлен ниже.

Полный исходный код программы.

```

from dataclasses import dataclass
from typing import List

```

```

@dataclass
class TInfo:
    phone: str = " "
    name: str = " "

```

```

@dataclass
class HashItem:

```



```
info: TInfo
empty: bool = True
visit: bool = False
```

```
class MyHash:
    hash_table: List[HashItem]
    info: TInfo

    def __init__(self, size_table):
        self.size_table = size_table
        self.info = TInfo()
        self.hash_table = [HashItem(info=self.info) for _ in
range(self.size_table)]
        self.size = 0
        self.step = 21

    def __hash_function(self, s):
        result = 0
        for i in range(len(s)):
            result += int(s[i]) * i
            result //= self.size_table
        return result

    def add_hash(self, name: str, phone: str):
        adr = -1
        if self.size < self.size_table:
            adr = self.__hash_function(phone)
            while not self.hash_table[adr].empty:
                adr = (adr + self.step) // self.size_table
            self.hash_table[adr].empty = False
            self.hash_table[adr].visit = True
            contact = TInfo(phone=phone, name=name)
            self.hash_table[adr].info = contact
            self.size += 1
        return adr
```

Лис
АиСД.09.03.02.210000 ПР 11

```

def __clear_visit(self):
    for i in self.hash_table:
        i.visit = False

def find_hash(self, phone: str):
    result = -1
    ok: bool
    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.size_table
        ok = self.hash_table[i].info.phone == phone
    if ok:
        result = i + 1
    return result

```

```

def del_hash(self, phone: str):
    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
            self.hash_table[i].info.name = " "
            self.hash_table[i].info.phone = " "
            result = True
            self.size -= 1
        else:
            i = self.find_hash(phone)
            if i == -1:
                self.hash_table[i].empty = True

```

```

        self.hash_table[i].info.name = " "
        self.hash_table[i].info.phone = " "
        result = True
        self.size -= 1

    return result

def __str__(self):
    out = ""
    head = "{:<6}{:<20}{:<20}".format("N", "NAME",
"PHONE")

    out += head
    out += "\n"
    for i in range(self.size_table):
        name: str = self.hash_table[i].info.name
        phone: str = self.hash_table[i].info.phone
        string = "{:<6}{:<20}{:<20}".format(i + 1, name,
phone)

        out += string
        out += "\n"

    return out

```

Вывод: в ходе выполнения данной практической работы была реализована хэш-таблица с открытой адресацией на языке Python.