Практическая работа №3

Тема: «Хэш-таблицы».

Цель работы: изучить реализацию хэш-таблиц.

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 реализацией метода открытой адрессации для простейшего телефона справочника. Для этого определим структуру контакта, которая представлена на рис. 1.

```
5   @dataclass
6   class TInfo:
7       phone: str = ''
8       family: str = ''
9       name: str = ''
10
```

Рис. 1. Структура контакта.

Для одной ячейки таблицы определим следующую структуру, представленную на рис. 2.

```
12  @dataclass
13  class HashItem:
14      info: TInfo
15      empty: bool = True
16      visit: bool = False
```

Рис. 2. Структура ячейки таблицы.

Где empty - флаг, указывающий, что ячейка свободна, в независимости от содержащихся там данных.

visit - флаг, указывающий, что ячейка просматривалась.

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

```
29 ▼    def __hash_function(self, value):
30          result = 0
31 ▼        for i in value:
32              result += ord(i)
33              result %= self.table_size
34          return result
```

Рис. 3. Хэш-функция.

Диаграмма деятельности для этой функции представлена на рис. 4.

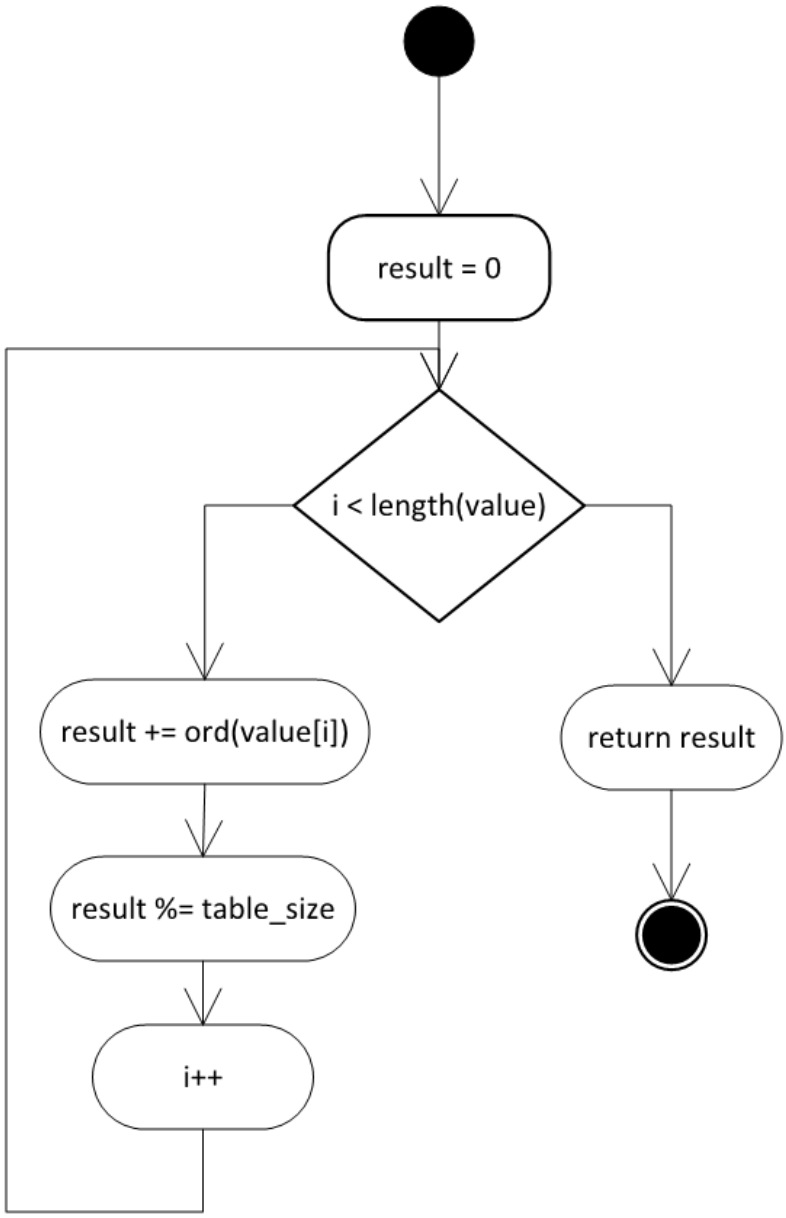

Рис. 4. Диаграмма деятельности для _hash_function.

Функция добавления элемента представлена на рис. 5.

```
36      def add_cell(self, name: str, family: str, phone: str) -> int:
37          adr = -1
38          if self.size < self.table_size:
39              adr = self.__hash_function(phone)
40              while not self.hash_table[adr].empty:
41                  adr = (adr + self.step) % self.table_size
42              self.hash_table[adr].empty = False
43              self.hash_table[adr].visit = True
44              contact = TInfo(name=name, family=family, phone=phone)
45              self.hash_table[adr].info = contact
46              self.size += 1
47          return adr
```

Рис. 5. Добавление элемента в хэш-таблицу.

Диаграмма деятельности для добавления элемента представлена на рис. 6.

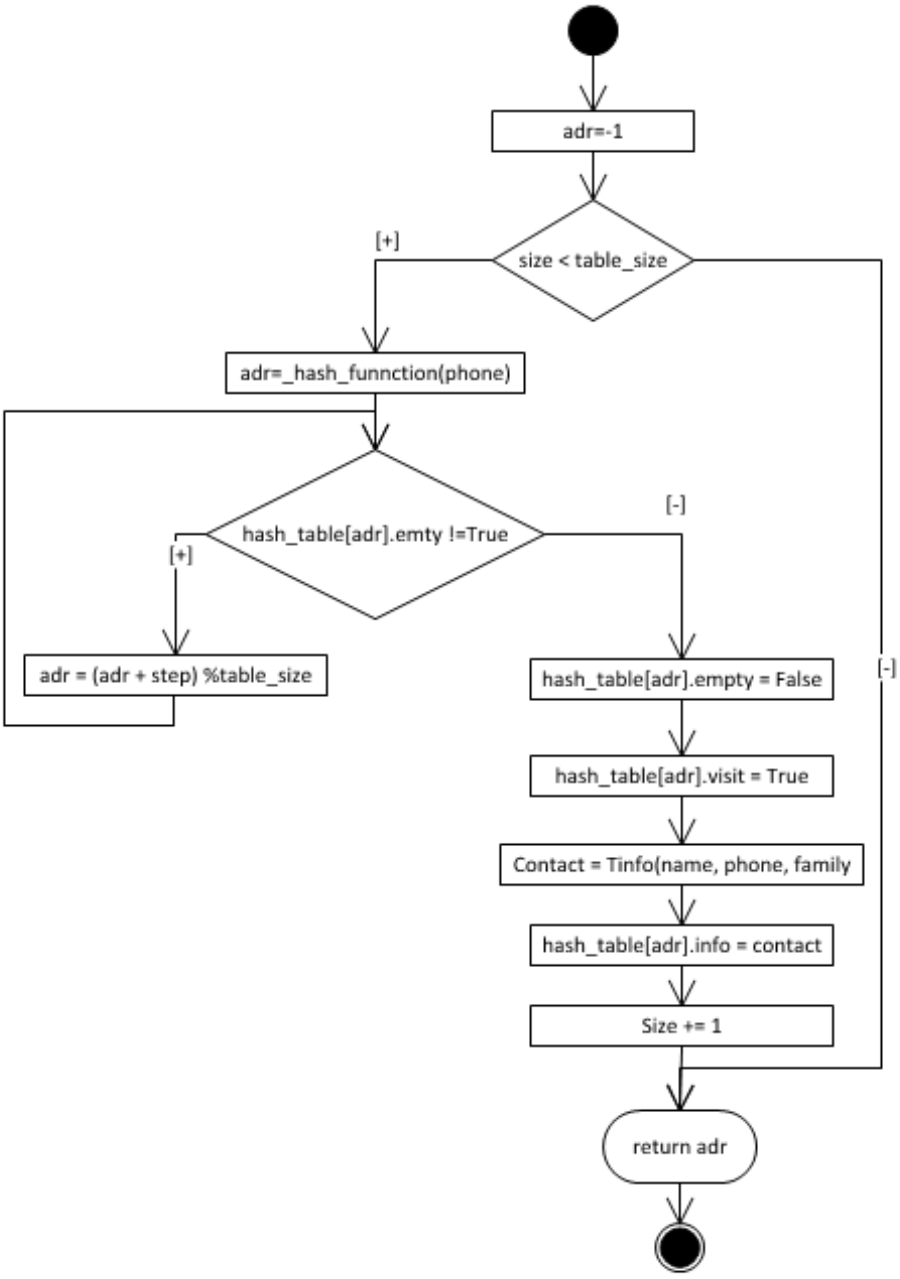

Рис. 6. Диаграма деятельности для добавления элемента, в таблицу методом открытой адрессации.

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

```
49 ▼    def __clear_visit(self):
50          for i in self.hash_table:
51              i.visit = False
```

Рис. 7. Сброс значений к дефолтным.

Диаграмма деятельности представлена для нее на рис. 8.

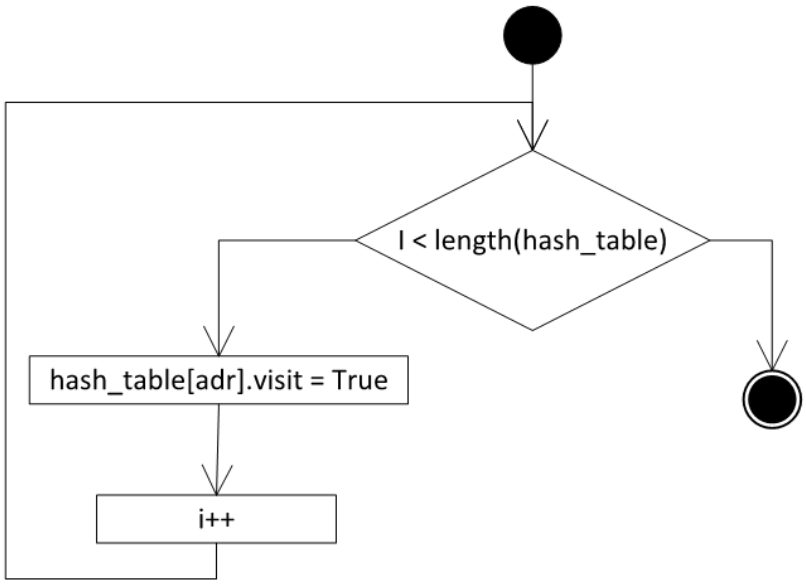

Рис. 8. Сброс флагов visit к дефолтным значениям.

Функция поиска значения в таблице представлена на рис. 9.

```
53      def find_cell(self, phone):
54          result = -1
55          ok: bool
56          fio = ""
57          count = 1
58          self.__clear_visit()
59          i = self.__hash_function(phone)
60          ok = self.hash_table[i].info.phone == phone
61          while not ok and not self.hash_table[i].visit:
62              count += 1
63              self.hash_table[i].visit = True
64              i = (i + self.step) % self.table_size
65              ok = self.hash_table[i].info.phone == phone
66          if ok:
67              result = i
68              fio = self.hash_table[result].info
69          return result, fio
```

Рис. 9. Поиск элемента в таблице.

Диаграма деятельности для поиска элемента представлена на рисунке 10.

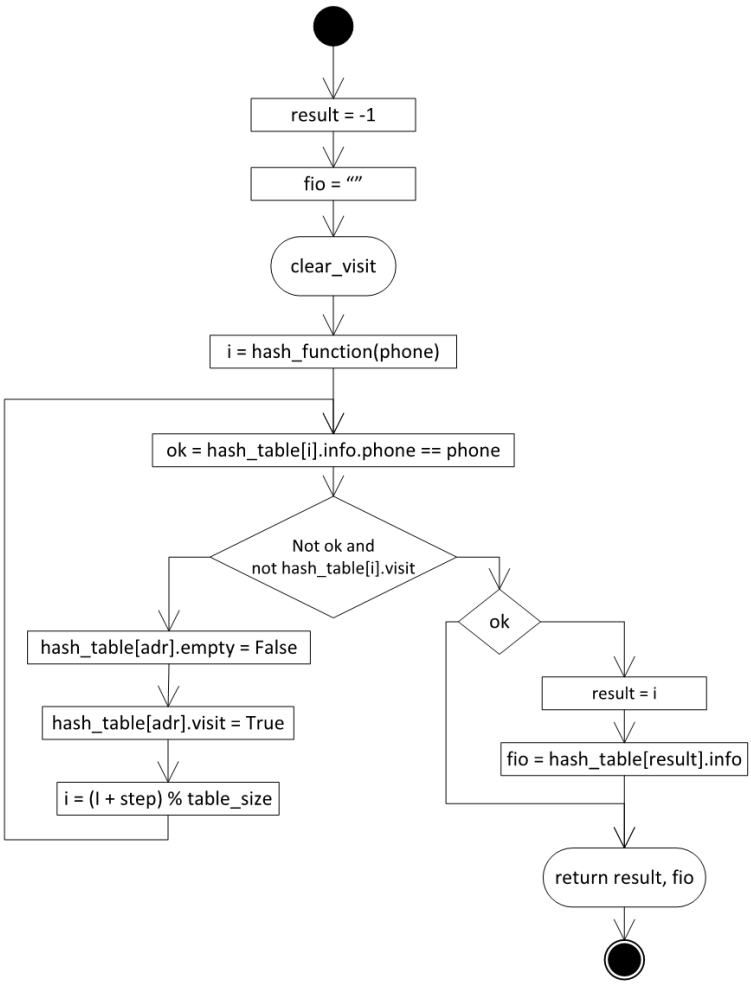

Рис. 10. Поиск элемента в хэш-таблице с открытой адресацией.

Для удаления элемента реализован метод, код которого представлен на рис. 11. Действие кода сводится к нахождению нужного элемента и выставление флага empty в позицию True.

```
71      def del_cell(self, phone):
72          result = False
73          i = 0
74          if self.size != 0:
75              i = self.__hash_function(phone)
76              if self.hash_table[i].info.phone == phone:
77                  self.hash_table[i].empty = True
78                  result = True
79                  self.size -= 1
80              else:
81                  i = self.find_hash(phone)
82                  if i == -1:
83                      self.hash_table[i].empty = True
84                      result = True
85                      self.size -= 1
86          return result
```

Рис. 11. Удаление элемента.

Диаграмма деятельности для удаления элемента представлена на рис. 12.

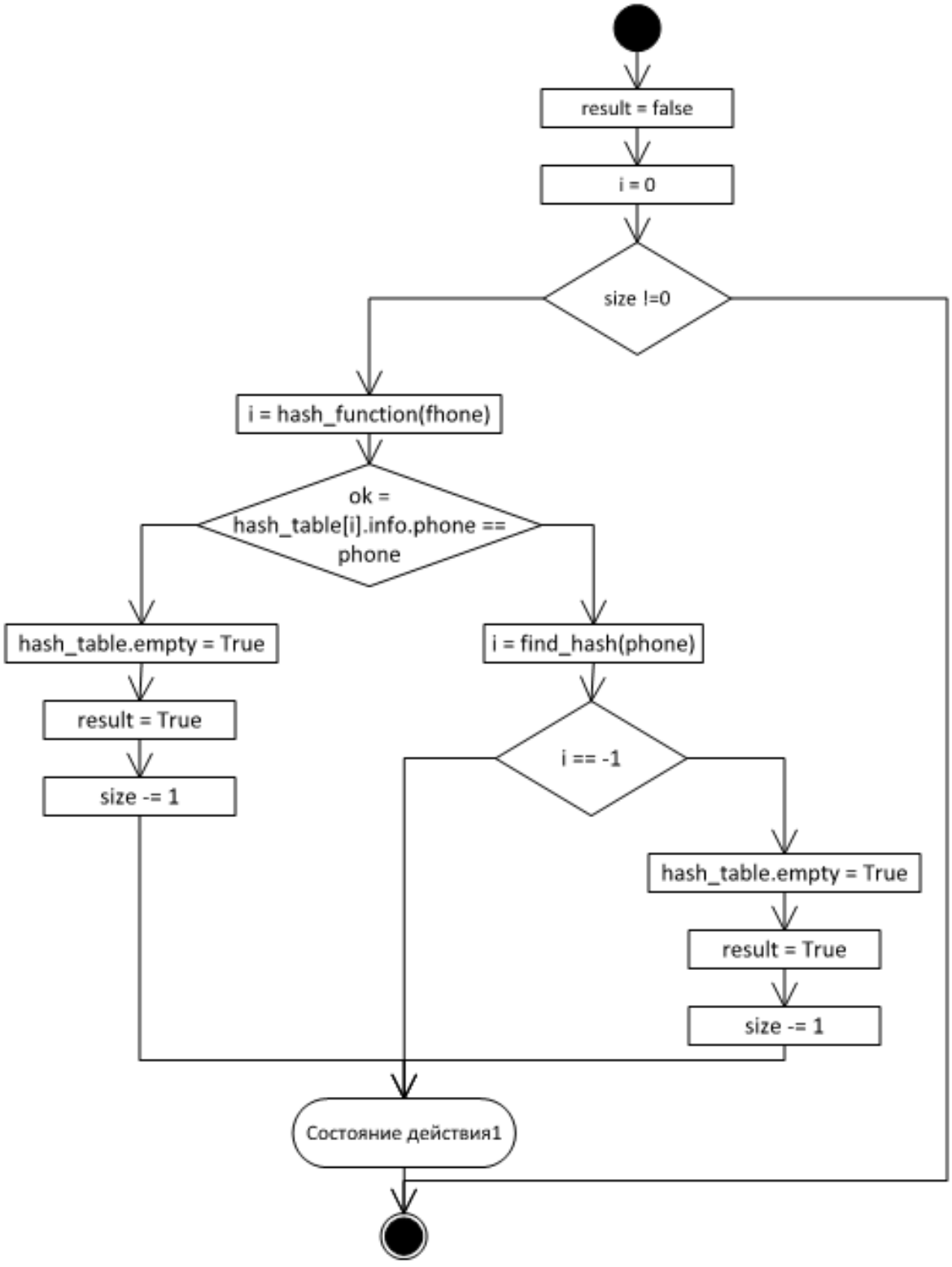

Рис. 12. Удаление элемента из хэш-таблицы.

Так же реализуем хэш-таблиц по методу цепочек. Для этого определим классы данных, как на рис. 13.

```python
@dataclass
class TInfo:
    name: str = ''
    family: str = ''
    phone: str = ''


@dataclass
class SubCell:
    info: TInfo = TInfo(name='', family='', phone='')
```

Рис. 13. Классы данных для метода цепочек.

Реализацию функции для хэширования оставим без измененений.

Изменим функцию добавления нового значения (рис. 14) и ее диаграмма деятельности представлена на рис. 15.

```python
    def add_cell(self, info:TInfo):
        adr = self.__hash_func(info.phone)
        i = len(self.hash_table[adr]) - 1
        self.hash_table[adr][i] = SubCell(info=info)
        self.hash_table[adr].append(SubCell(info=TInfo()))
```

Рис. 14. Функция добавления новой записи в таблицу.

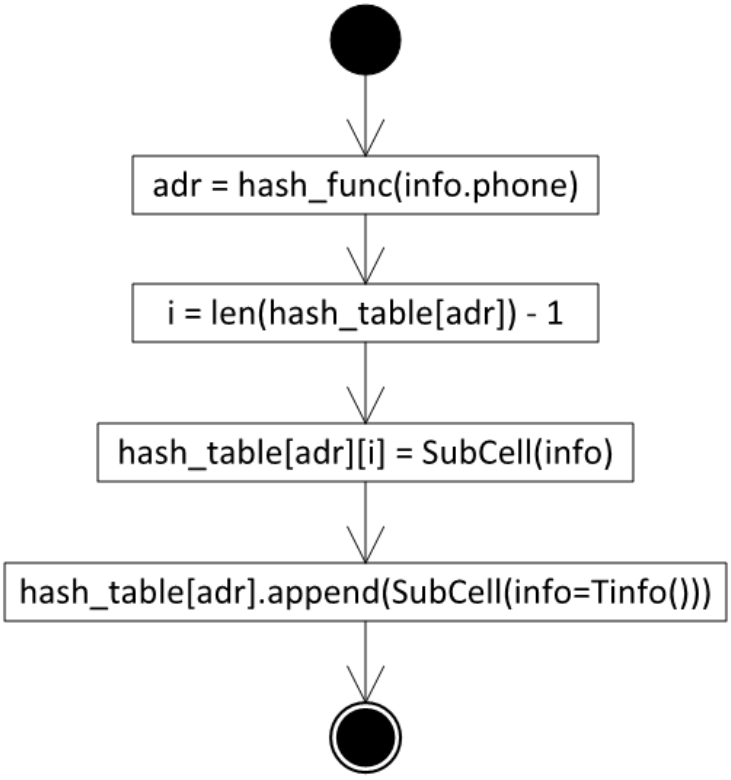

Рис. 15. Добавление нового элемента в таблицу.

Функция удаления элемента представлена на рис. 16. Диаграмма деятельности для нее представлена на рис. 17.

```
37      def del_cell(self, info):
38          adr = self.__hash_func(info.phone)
39          i = 0
40          while self.hash_table[adr][i].info != info:
41              i+=1
42          del self.hash_table[adr][i]
```

Рис. 16. Удаление элемента.

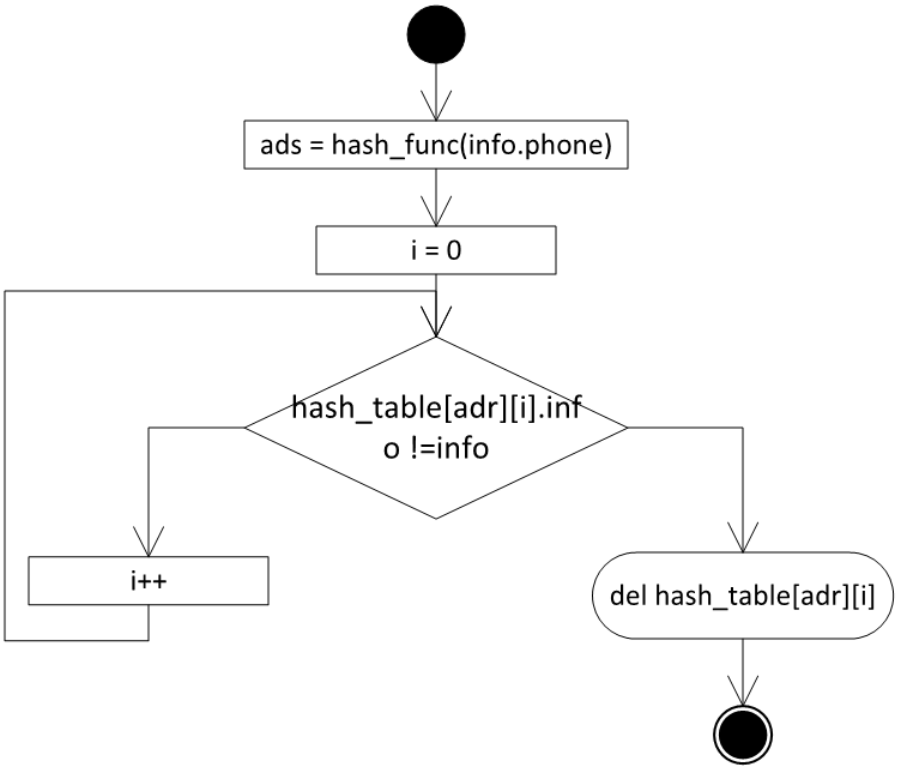

Рис. 17. Удаление элемента.

Функция поиска элемента представлена на рис. 18. Диаграмма деятельности на рис. 19.

```
44      def find_cell(self, info):
45          adr = self.__hash_func(info.phone)
46          i = 0
47          while self.hash_table[adr][i].info != info:
48              i += 1
49          return adr, i
```

Рис. 18. Функция поиска элемента.

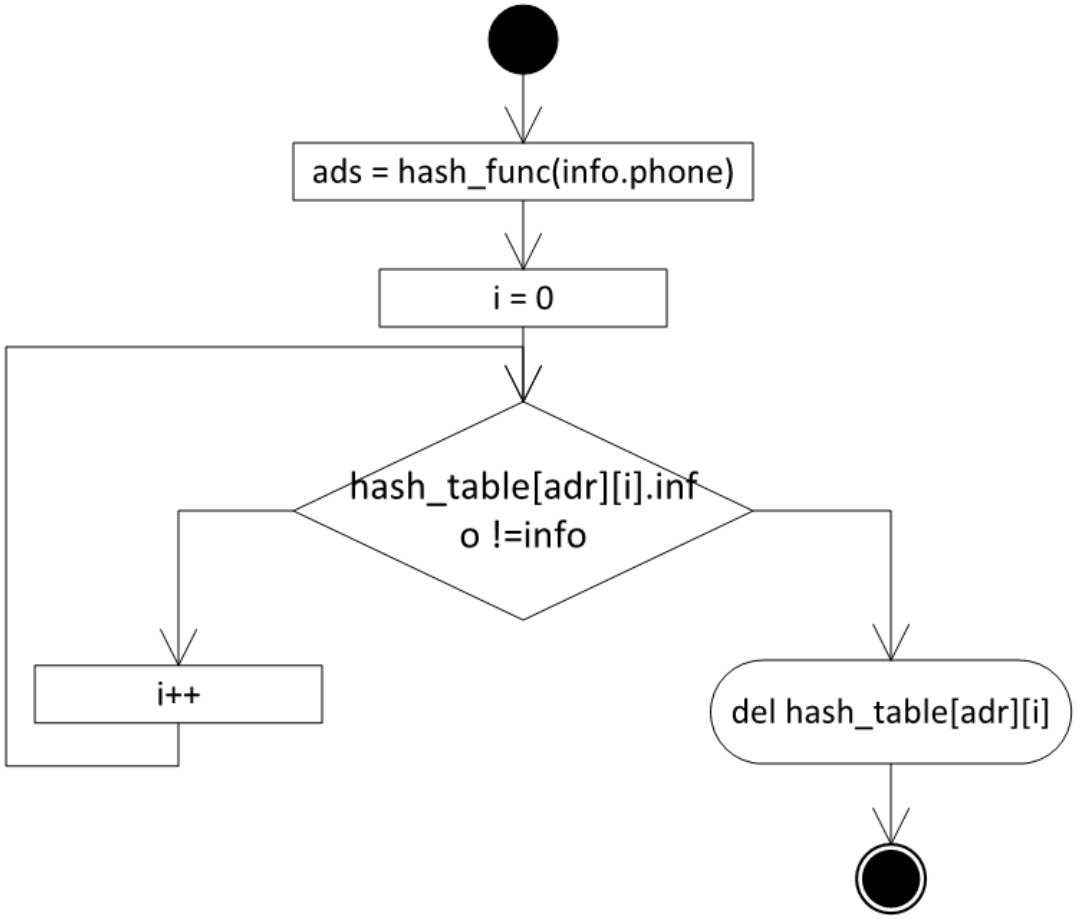

Рис. 19. Поиск элемента в хэш-таблице.

Вывод: в ходе выполнения практической работы были изучены хэш- таблицы и методы их реализации на языке Python.