Практическая работа № 3

Тема: «Хэш-таблицы»

Цель работы: изучить реализацию хэш-таблиц. Хеш-табли́ца — 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 в листинге 1.

Листинг 1. Структура контакта.

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

Для одной ячейки таблицы определим следующую структуру, представленную в листинге 2.

Листинг 2.Структура ячейки таблицы.

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

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

visit – флаг, указывающий, что ячейка просматривалась.

					АиСД.09.03.02.260000		
Изм	Лист	№ докум.	Подпись	Дата			
Разраб.		Якубов Р.О.			**Практическая работа № 3**	Лит	Лист	Листов
Провер.		Береза А.Н.			**Тема:** <<Хеш-таблицы>>			
						ИСОиП(ф)ДГТУ ИСТ-Tb21		
Н.контр.								
Утв.								

Для вычисления значения хэша будем использовать следующую функцию, представленную в листинге 3.

Листинг 3. Хэш-функция.

```
def __hash_function(self, value):
    result = 0
    for i in value:
        result += ord(i)
        result %= self.table_size
    return result
```

Диаграмма деятельности для этой функции представлена на рисунке 1.

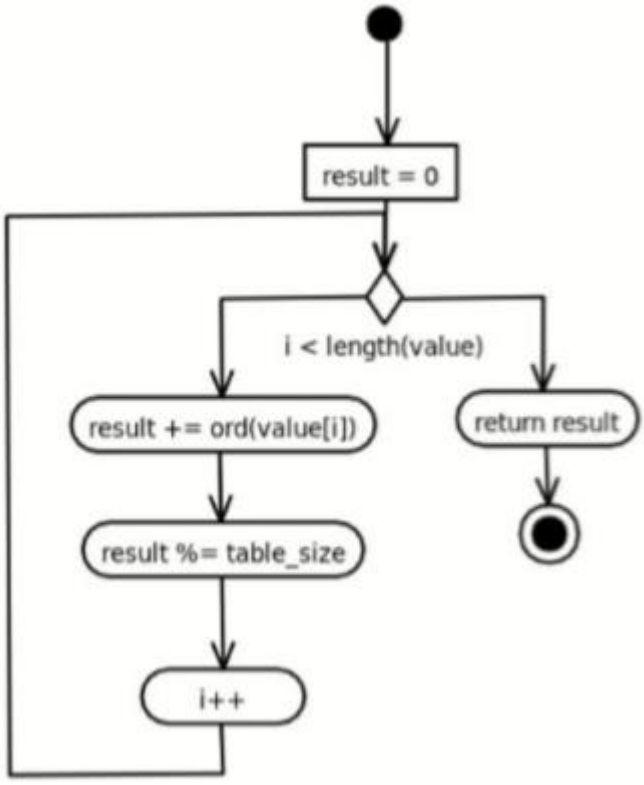

Рисунок 1 - Диаграмма деятельности для __hash_function

Листинг функции добавления элемента представлен в Листинге 4.

Листинг 4. Добавление элемента в хэш-таблицу.

```
def add_hash(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
```

Диаграмма деятельности для добавления элемента представлена на рисунке .

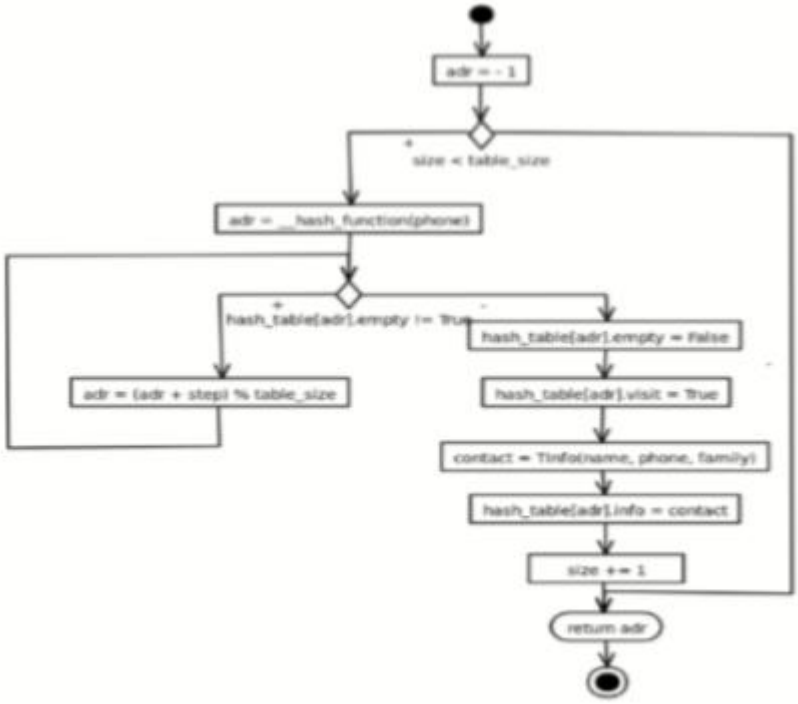

Рисунок 2 - Диаграма деятельности для добавления элемента, в таблицу методом открытой адрессации.

Для поиска элемента, надо убедиться, что флаги visit каждой ячейки сброшены к дефолтным значениям. Для этого мы используем функциию, код которой представлен в листинге 5.

Листинг 5. Сброс значений к дефолтным.

```
def __clear_visit(self):
```

```
    for i in self.hash_table:
        i.visit = False
```

Диаграмма деятельности представлена для нее на рисунке 3.

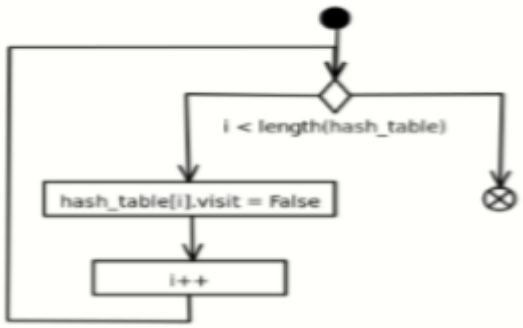

Рисунок 3 - Сброс флагов visit к дефолтным значениям

Функция поиска значения в таблице представлена в листинге 6.

Листинг 6. Поиск элемента в таблице.

```
def find_hash(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
```

Диаграма деятельности для поиска элемента представлена на рисунке .

Рисунок 4 - Поиск элемента в хэш-таблице с открытой адресацией

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

Листинг 7. Удаление элемента.

```
def del_hash(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
```

Диаграмма деятельности для этого представлена на рисунке 5.

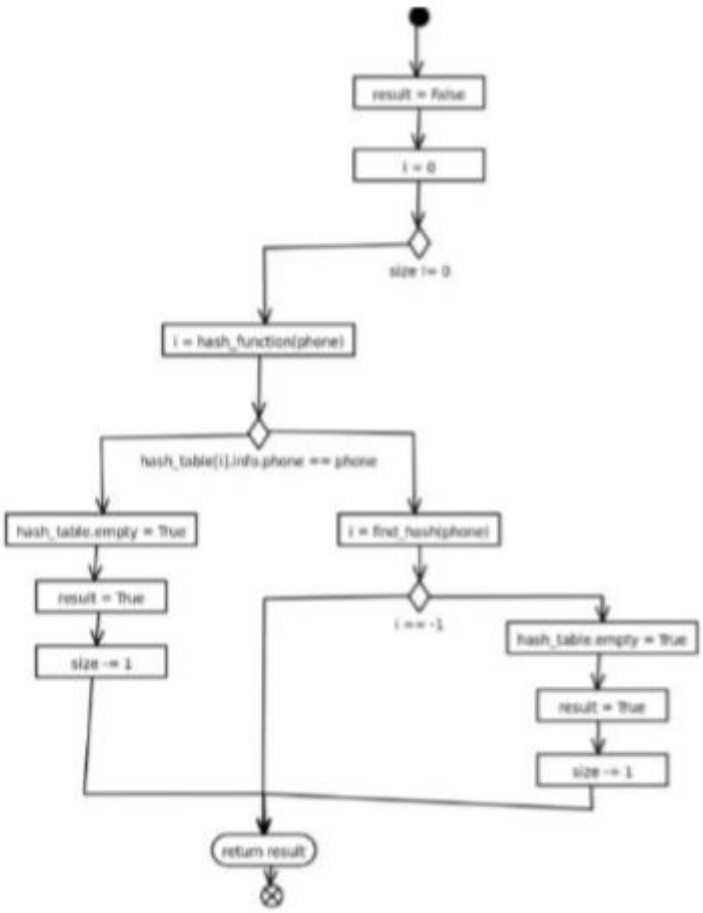

Рисунок 5 - Удаление элемента из хэш-таблицы

Так же реализуем хэш-таблиц по методу цепочек. Для этого определим классы данных, как в листинге 8.

Листинг 8. Классы данных для метода цепочек.

```
@dataclass
class TInfo:
    name: str = ''
```

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

Реализацию функции для хэширования оставим без измененений.

Изменим функцию добавления нового значения (листинг 9) и ее диаграмма деятельности представлена на рисунке 6.

Листинг 9. Функция добавления новой записи в таблицу.

```
def add_item(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=TInf
```

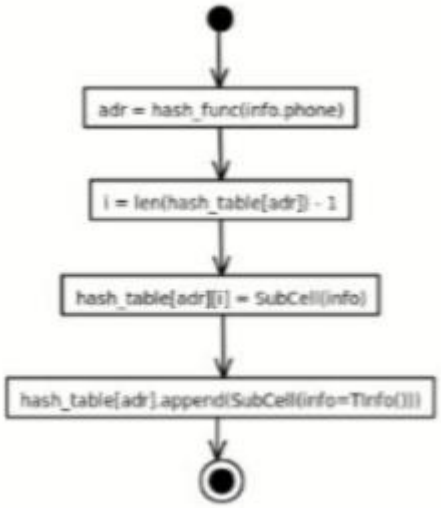

Рисунок 6 - Добавление нового элемента в таблицу

Функция удаления элемента представлена в листинге 10. Диаграмма деятельности для нее представлена на рисунке 7.

Листинг 10. Удаление элемента.

```
def del_item(self, info):
    adr = self.__hash_func(info.phone)
```

```
i = 0
while self.hash_table[adr][i].info != info:
    i+=1
del self.hash_table[adr][i]
```

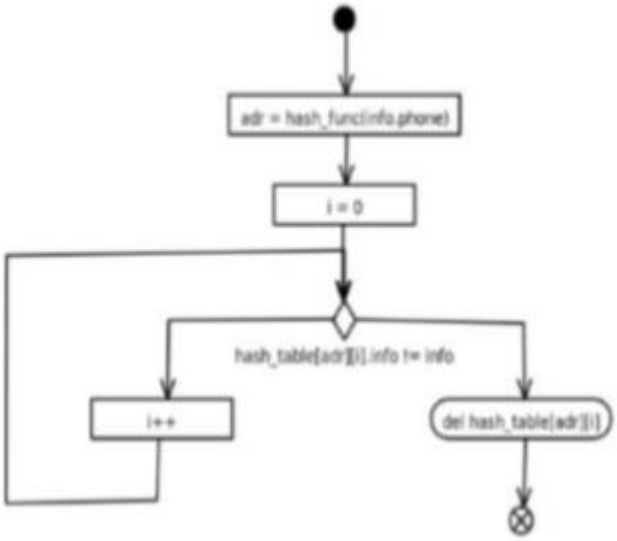

Рисунок 7 - Удаление элемента

Функция поиска элемента представлена в листинге 11. Диаграмма деятельности на рисунке 8.

Листинг 11. Функция поиска элемента. .

```
def find_item(self, info):
    adr = self.__hash_func(info.phone)
    i = 0
    while self.hash_table[adr][i].info != info:
        i += 1
    return adr, i
```

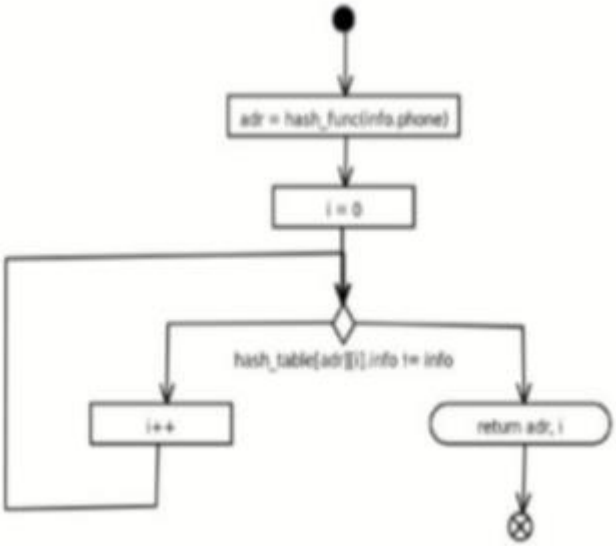

Рисунок 8 - Поиск элемента в хэш-таблице

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