Практическая работа №4

Тема: Структуры данных «Линейные списки».

Цель работы: Изучить СД типа «линейный список», научиться их программно реализовывать и использовать.

Связный список - это рекурсивная структура, так как узел всегда содержит указатель на следующий узел. Это позволяет использовать простой рекурсивный алгоритм для таких операций, как объединение двух списков или изменение порядка элементов на обратный.

Для реализации «линейного списка» определим сначала структуру узла, код которого представлен ниже:

```
class Node:
    def __init__(self, x = None, next = None):
        self.x = x
        self.next = next
```

Для определения связного списка потребуется еще один класс – LinkedList (LinkList), в конструкторе которого будут определяться первый и последний элементы списка и его длина. Также в классе будут использоваться встроенный метод str для распечатки содержимого списка и метод clear для очистки списка.

```
class LinkList:
    def __init__(self):
        self.length = 0
        self.first = None
        self.last = None

    def __str__(self):
        if self.first != None:
            current = self.first
            out ='LinkList [\n'+str(current.value)+'\n'
            while current.next != None:
                current = current.next
                out += str(current.value) + '\n'
            return out + ']'
return 'LinkList []'

    def clear(self):
       self.__init__()
```

Код метода добавления элемента в конец списка. Диаграмма деятельности метода добавления элемента в конце списка представлена на рисунке 1.

```
def add(self, x):
        self.length += 1
        if self.first == None:
            self.last = self.first = Node(x, None)
        else:
            self.last.next = self.last = Node(x, None)
```

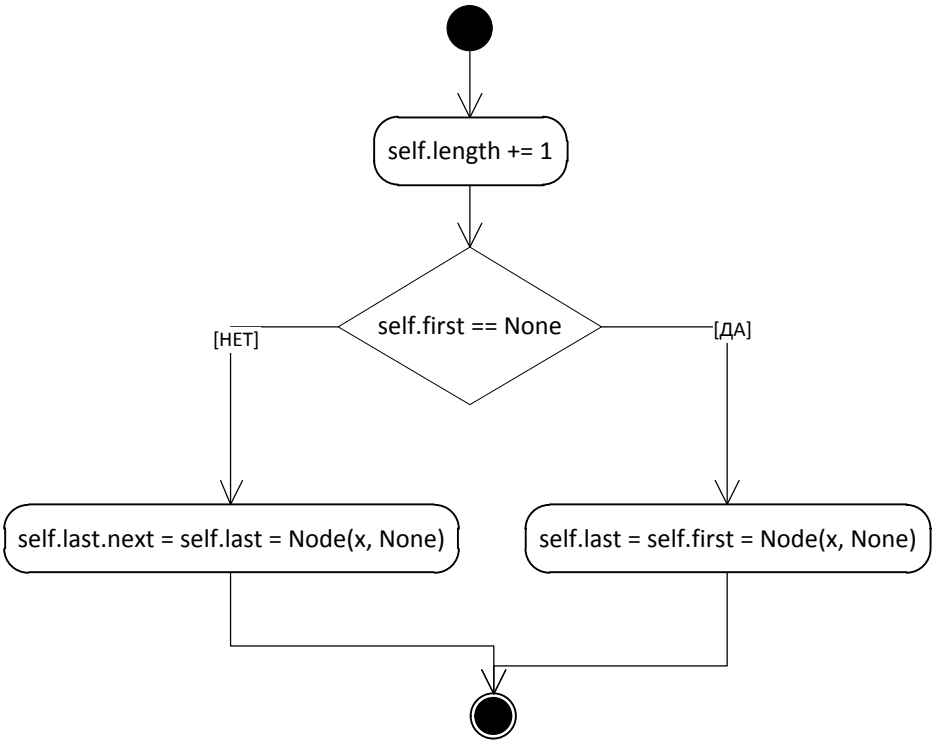

Рисунок 1 – Диаграмма деятельности метода добавления элемента в конце списка.

Код метода добавления элемента в начало списка. Диаграмма деятельности метода добавления элемента в начало списка представлена на рисунке 2.

```
def push(self, x):
        self.length += 1
        if self.first == None:
            self.last = self.first = Node(x, None)
        else:
            self.first = Node(x, self.first)
```

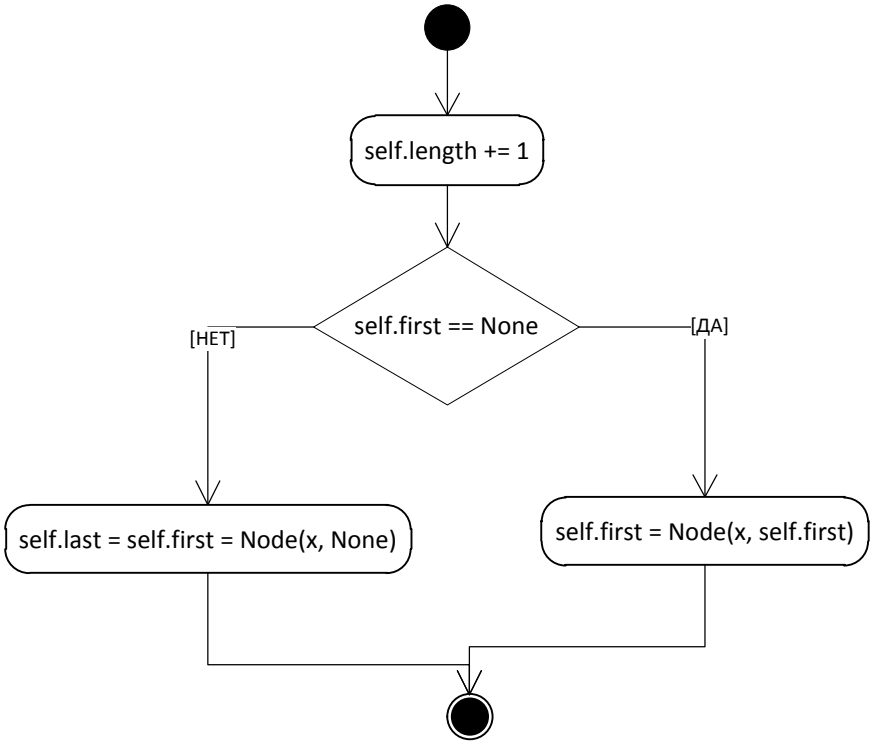

Рисунок 2 – Диаграмма деятельности метода добавления элемента в начало списка.

Код метода добавления элемента в произвольное место списка. Диаграмма деятельности метода добавления элемента в произвольное место списка представлена на рисунке 3.

```
def insert(self, i, x):
        if self.first == None:
            self.first = Node(x, self.first)
            self.last = self.first.next
            return
        if i == 0:
            self.push(x)
            return
        curr = self.first
        count = 0
        while curr != None:
            if count == i - 1:
                curr.next = Node(x, curr.next)
                if curr.next == None:
                    self.last = curr.next
                break
            curr = curr.next
            count += 1
```

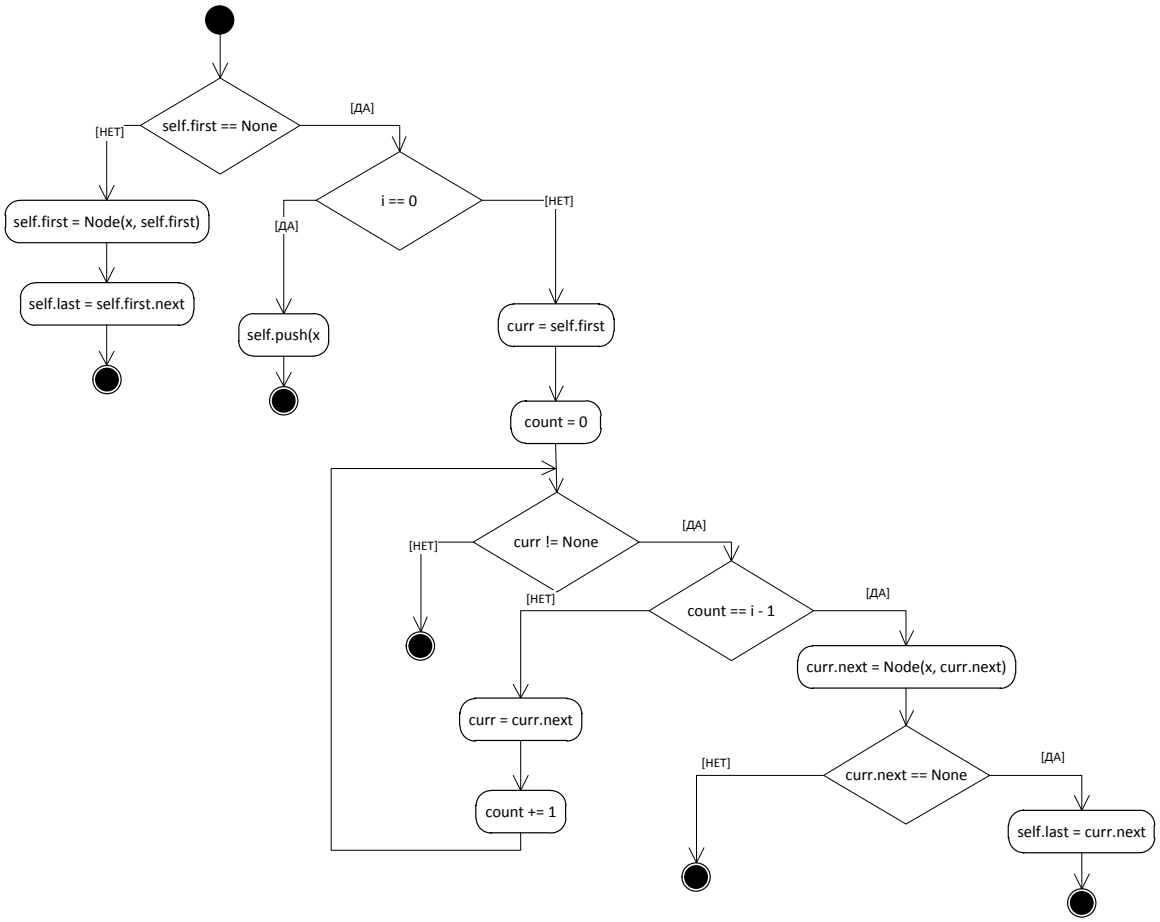

Рисунок 3 – Диаграмма деятельности метода добавления элемента в произвольное место списка.

Код метода удаления головного элемента из списка. Диаграмма деятельности метода удаления головного элемента в списке представлена на рисунке 4.

```
def delete(self, i):
        if self.first == None:
            return
        curr = self.first
        count = 0
        if i == 0:
            self.first = self.first.next
            return
        while curr != None:
            if count == i:
                if curr.next == None:
                    self.last = curr
                old.next = curr.next
                break
            old = curr
            curr = curr.next
            count += 1
```

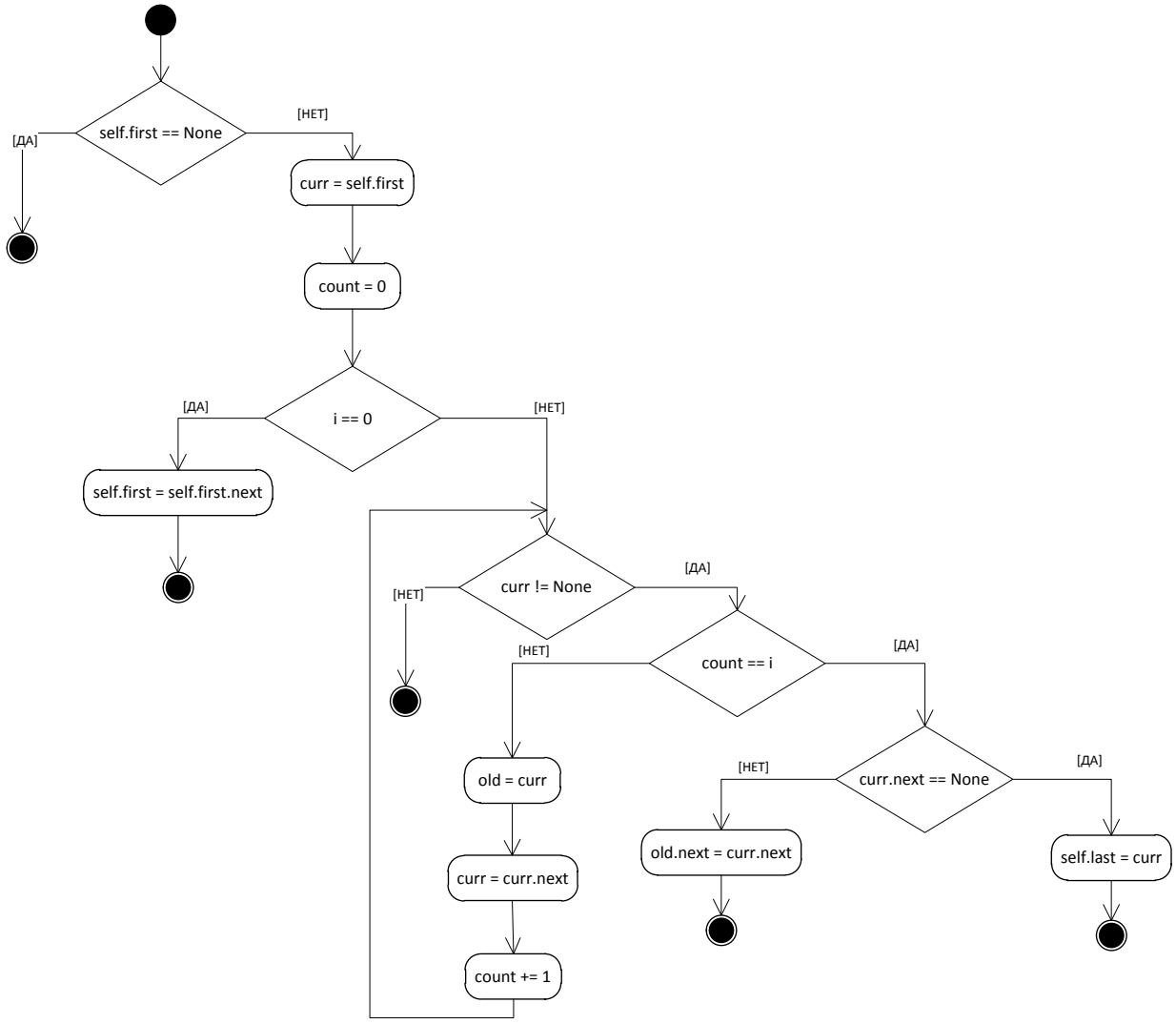

Рисунок 4 – Диаграмма деятельности метода удаления головного элемента из списка.

Код метода поиска элемента по его значению. Диаграмма деятельности метода поиска элемента по его значению представлена на рисунке 5.

```
def search(self, x):
        curr = self.first
        count = 0
        while curr != None and current.x != x:
            count += 1
            curr = curr.next
        if curr == None or curr.x != x:
            count = -1
        return count
```

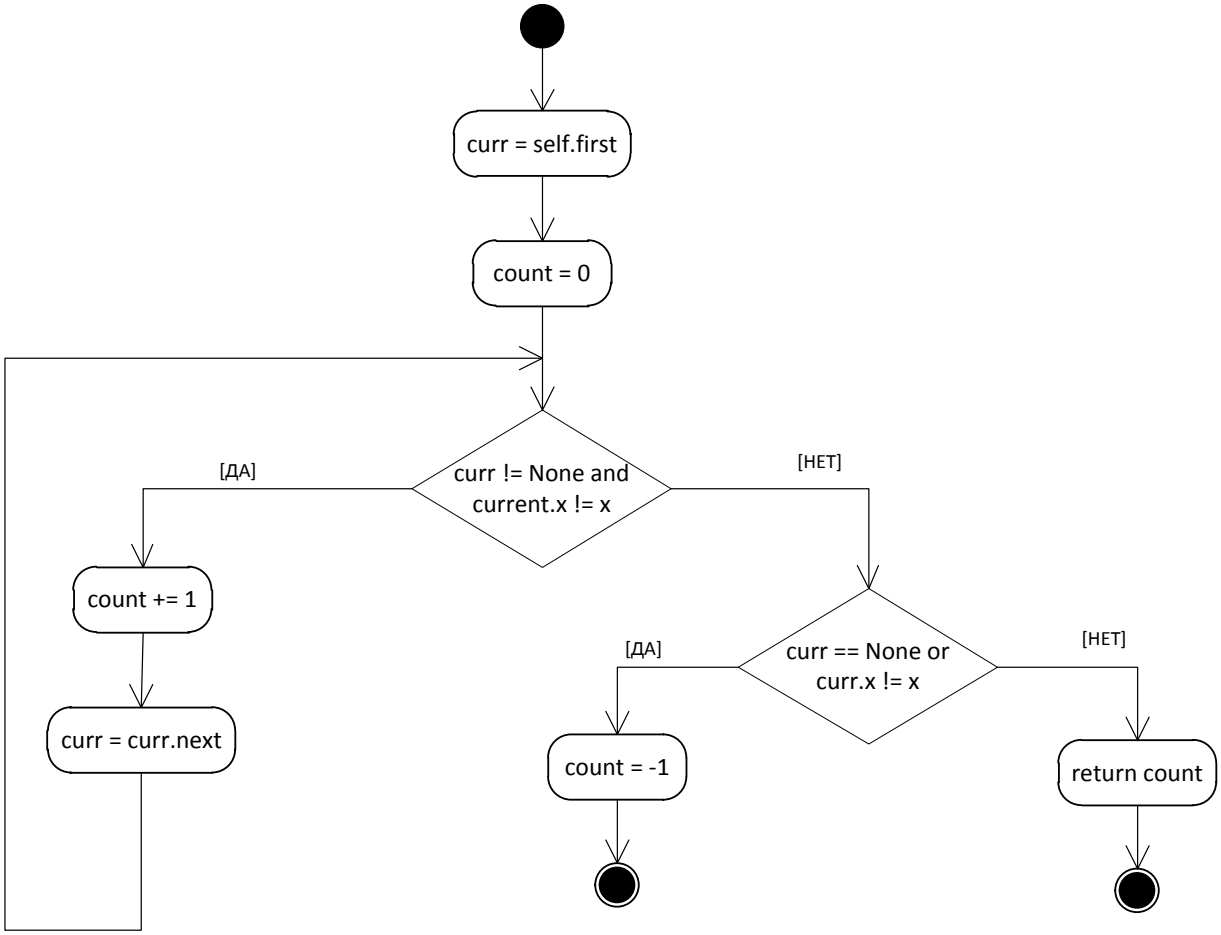

Рисунок 5 – Диаграмма деятельности метода поиска элемента по его значению.

Код для решения задачи Вариант 6:

```
# Выполнение задания Вариант 6
import random

n = int(input('Введите натуральное число элементов (от 1), n = '
))
while (n < 1):
    n = int(input("По условию n >= 1. Повторите ввод: "))
massive = []
for j in range(n):
    massive.append(random.randint (0, 9))
mLinkList = LinkList()
for j in range (n):
    c = mLinkList.search(massive[j])
    if c < 1:
        mLinkList.add(massive[j])
print ('\nЧисла:')
print (massive)

print ('\nСимволы, которые входят в нее по одному разу:')
print (mLinkList)
```

Скриншот рабочей программы представлен на рисунке 6:

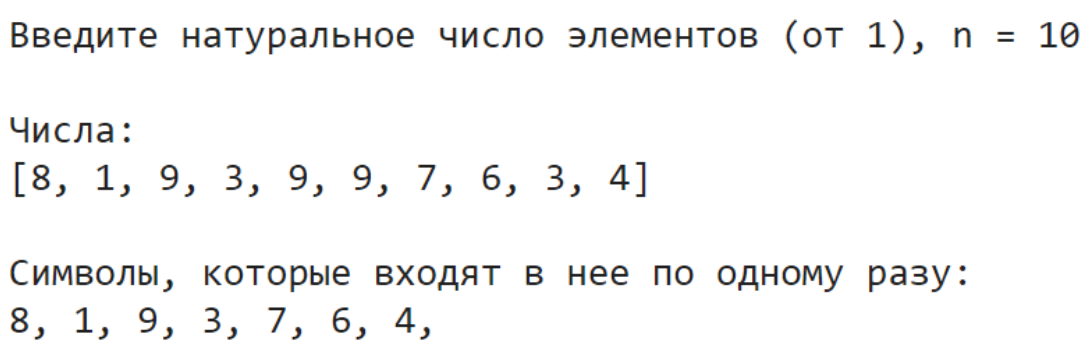

Рисунок 6 – Скриншот рабочей программы.

Вывод. Мы изучили СД типа «линейный список», научиться их программно реализовывать и использовать, добавлять элемент в начало списка, в произвольное место списка, удалять головной элемент из списка, осуществлять поиск элемента по его значению.