**Практическая работа №4.**

**Тема: «**Структуры данных «линейные списки».

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

Многочлен с целыми коэффициентами можно представить в виде списка, причем если , то соответствующее звено не включать в список. Необходимо вывести на экран значения элементов многочлена при случайных значениях .

Для реализации линейного списка сначала определим структуру данных для узла этого списка. Класс Node представлен на листинге 1.

Листинг 1. Класс Node.

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

def get\_data(self):

return self.data

def get\_next(self):

return self.next

def set\_next(self, next):

self.next = next

Функция для добавления элемента в начало списка представлена на листинге 2.

Листинг 2. Функция для добавления элемента в начало списка.

def push(self, value):

temp = Node(value)

temp.set\_next(self.head)

self.head = temp

Функция для вставки элемента в конец списка представлена на листинге 3. Диаграмма деятельностей для этой функции представлена на рисунке 1.

Листинг 3. Функция для добавления элемента в конец списка.

def append(self, new\_data):

new\_node = Node(new\_data)

if self.head is None:

self.head = new\_node

return

last = self.head

while last.next:

last = last.get\_next()

last.set\_next(new\_node)

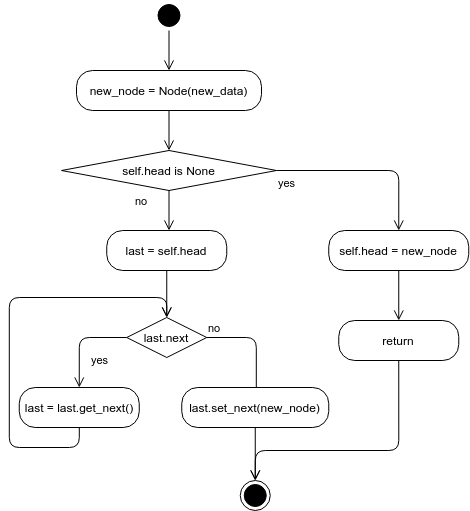


Рисунок 1. Диаграмма деятельностей для функции добавления элемента в конец списка.

Функция для добавления элемента в произвольное место списка представлена на листинге 4. Диаграмма деятельностей для этой функции представлена на рисунке 2.

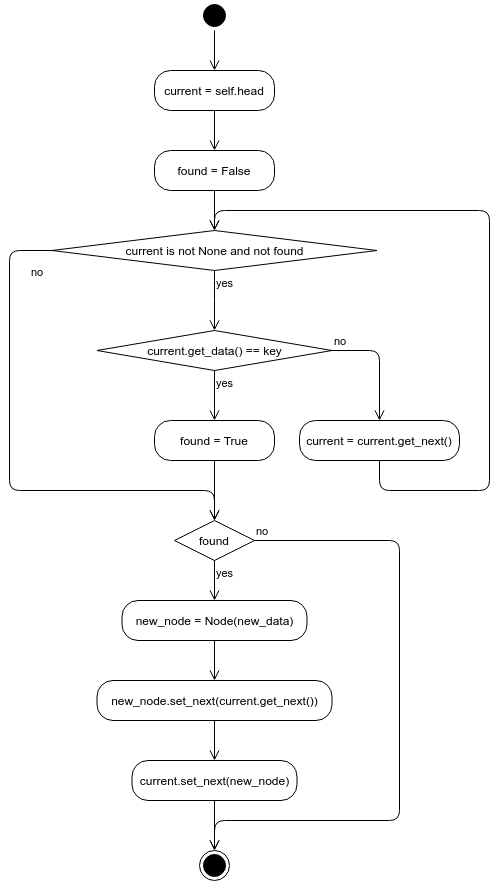


Рисунок 2. Диаграмма деятельностей для функции добавления элемента в произвольное место списка.

Листинг 4. Функция добавления элемента в произвольное место списка.

def insert\_after(self, key, new\_data):

current = self.head

found = False

while current is not None and not found:

if current.get\_data() == key:

found = True

else:

current = current.get\_next()

if found:

new\_node = Node(new\_data)

new\_node.set\_next(current.get\_next())

current.set\_next(new\_node)

Функция для получения длины списка представлена на листинге 5.

Листинг 5. Функция для получения длины списка.

def length(self):

current = self.head

count = 0

while current is not None:

count += 1

current = current.get\_next()

return count

Функция для поиска элемента в списке представлена на листинге 6. Диаграмма деятельностей для этой функции представлена на рисунке 3.

Листинг 6. Функция поиска элемента в списке.

def search(self, key):

current = self.head

found = False

while current is not None and not found:

if current.get\_data() == key:

found = True

else :

current = current.get\_next()

return found

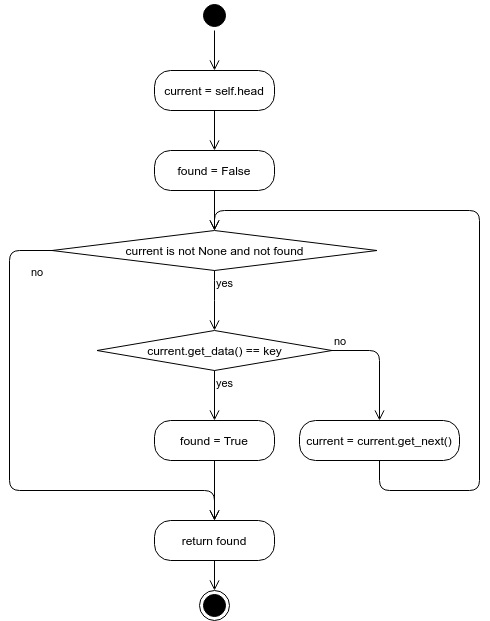


Рисунок 3. Диаграмма деятельностей для функции поиска элемента в списке.

Функция для удаления элемента списка представлена на листинге 7. Диаграмма деятельностей для этой функции представлена на рисунке 4.

Листинг 7. Функция для удаления элемента списка.

def delete\_node(self, key):

current = self.head

previous = None

found = False

while not found:

if current.get\_data() == key:

found = True

else:

previous = current

current = current.get\_next()

if previous is None:

self.head = current.get\_next()

else:

previous.set\_next(current.get\_next())

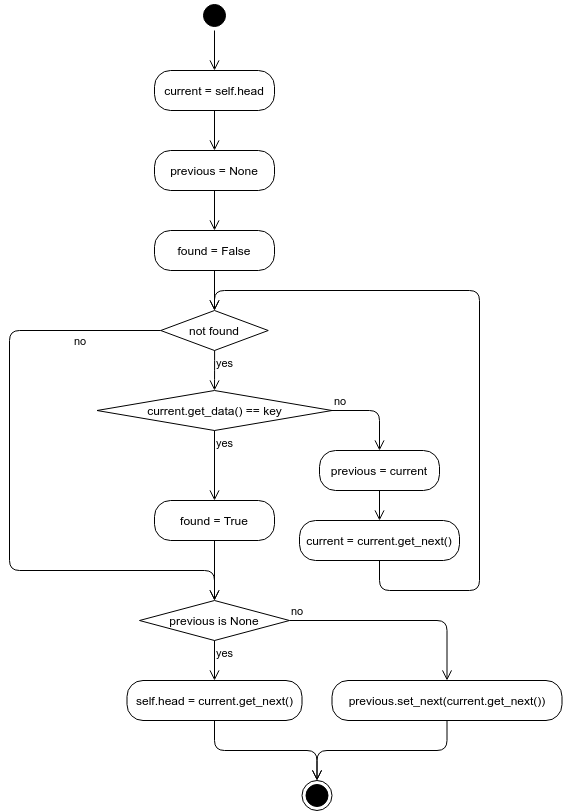


Рисунок 4. Диаграмма деятельностей для функции удаления элемента списка.

Полностью исходный код для класса MyLinkedList представлены на листинге 8.

Листинг 8. Полный исходный код класса MyLinkedList.

class MyLinkedList:

def \_\_init\_\_(self):

self.head = None

def \_\_str\_\_(self):

if self.head is not None:

current = self.head

out = "[" + str(current.get\_data())

while current.get\_next() is not None:

current = current.get\_next()

out += "," + " " + str(current.get\_data())

return out + "]"

def push(self, value):

temp = Node(value)

temp.set\_next(self.head)

self.head = temp

def append(self, new\_data):

new\_node = Node(new\_data)

if self.head is None:

self.head = new\_node

return

last = self.head

while last.next:

last = last.get\_next()

last.set\_next(new\_node)

def insert\_after(self, key, new\_data):

current = self.head

found = False

while current is not None and not found:

if current.get\_data() == key:

found = True

else:

current = current.get\_next()

if found:

new\_node = Node(new\_data)

new\_node.set\_next(current.get\_next())

current.set\_next(new\_node)

def length(self):

current = self.head

count = 0

while current is not None:

count += 1

current = current.get\_next()

return count

def search(self, key):

current = self.head

found = False

while current is not None and not found:

if current.get\_data() == key:

found = True

else :

current = current.get\_next()

return found

def delete\_node(self, key):

current = self.head

previous = None

found = False

while not found:

if current.get\_data() == key:

found = True

else:

previous = current

current = current.get\_next()

if previous is None:

self.head = current.get\_next()

else:

previous.set\_next(current.get\_next())

Исходный код программы для вывода значений элементов многочлена представлен на листинге 9.

Листинг 9. Исходный код программы для выполнения задания.

from Linked\_list.linked\_list import MyLinkedList

import random as rd

polynomial = MyLinkedList()

def polynom(x, n):

for i in range(n, -1, -1):

p = rd.randint(0, 10) \* x \*\* i

if p != 0:

polynomial.append(p)

return polynomial

print(polynom(2, 10))

**Вывод:** в ходе выполнения данной практической работы была реализована структура данных линейный список, и выполнено индивидуальное задание.