МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Информатика»
Тема “Моделирование работы Машины Тьюринга”

Студентка гр. 1384 _____ Логинова А. Ю.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2021

Цель работы.

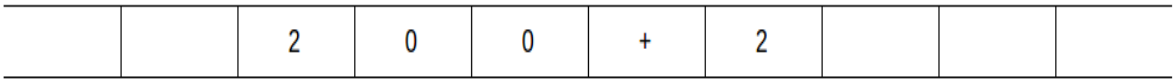
Смоделировать поведение машины Тьюринга на задачи сложения и вычитания троичного числа и троичной цифры.

Задание.

На вход программе подается строка неизвестной длины. Каждый элемент является значением в ячейке памяти ленты Машины Тьюринга.

На ленте находится троичное число, знак (плюс или минус) и троичная цифра.

Напишите программу, которая выполнит арифметическую операцию. Указатель на текущее состояние Машины Тьюринга изначально находится слева от числа (но не на первом его символе). По обе стороны от числа находятся пробелы. Результат арифметической операции запишите на месте первого числа. Для примера выше лента будет выглядеть так:



Ваша программа должна вывести полученную ленту после завершения работы.

Алфавит:

- 0
- 1
- 2
- +
- -
- " " (пробел)

Соглашения:

1. Направление движения автомата может быть одно из R (направо), L (налево), N (неподвижно).
2. Число обязательно начинается с единицы или двойки.
3. Числа и знак операции между ними идут непрерывно.
4. Гарантируется, что в результате операции вычитания не может получиться отрицательного числа.

В отчет включите таблицу состояний. Отдельно кратко опишите каждое состояние, например:

q1 - начальное состояние, которое необходимо, чтобы найти первую цифру первого числа.

Выполнение работы.

На вход программе подается строка, которая делится посимвольно. Список (далее *tape*) полученных символов с таблицей (далее *table*) состояний, зависимых от значений, класса *ActionTable* передаются в качестве аргумента в класс, моделирующий машину Тьюринга *TuringMachine.* Таблица состояний приводится в таблице 2. *Table* реализован с помощью вложенных словарей, где ключ первого уровня – состояние, а его значение – вложенный словарь, в котором ключ второго уровня – символ в ячейке на ленте, а его значение – объект класса *Action*, в который были переданы: значение, необходимое записать в ячейку, на которой остановился указатель, ход и обновленное состояние.

Далее вызывается функция класса *TuringMachine process(),* которая запускает процесс работы машины Тьюринга, реализованный с помощью цикла *while* и вызова функции *__make_step()* и выполняется до тех пор, пока не встретит состояние *Qt* (Qterminate). В конце функция *__delete_zeros()* избавляется от нулей, образовавшихся перед первым слагаемым.

Исходя из настоящего состояния машины Тьюринга, функция *__make_step()* запрашивает объект *action* класса *Action* из *table* с помощью функции *get_action()* класса *ActionTable*, а затем обновляет свое состояние.

Далее вызывается функция класса *TuringMachine print_tape(),* которая выводит обработанную ленту в виде строки с помощью метода *.join().*

Для удобства состояния в классе *TuringMachine* были реализованы с помощью класса *States*, который наследует класс *Enum,* реализующий перечисление. Таким образом, состояния от *Q0* до *Qt* будут определены как значения от 0 до 8. Таким же образом были реализованы ходы в классе *Action L, R, N,* которые были определены как -1, 1, 0 соответственно. Для того чтобы было возможно прибавлять к *pointer* ход (в функции *__make_step())* вместо *Enum* был использован *IntEnum*.

Разработанный программный код приводится в приложении А. Результаты тестирования приводятся в таблице 1. Таблица состояний приводится в таблице 2.

Тестирование.

Таблица 1 – результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарии
1	0-0	0-0	Программа работает корректно
2	1000000000-2	222222221-2	Программа работает корректно

Справочная информация.

Таблица 2 - таблица состояний.

	0	**1**	**2**	**+**	**-**	**‘ ’**
Q0	<0;R;Q0>	<1;R;Q0>	<2;R;Q0>	<+;R;Q1>	<-;R;Q2>	<’ ’;R;Q0>
Q1	<0;N;Qt>	<1;L;Q3>	<2;L;Q4>			<’ ’;R;Q1>
Q2	<0;N;Qt>	<1;L;Q5>	<2;L;Q6>			<’ ’;R;Q2>
Q3	<1;N;Qt>	<2;N;Qt>	<0;L;Q7>	<+;L;Q3>	<-;L;Q3>	<’ ’;L;Q3>
Q4	<2;N;Qt>	<0;L;Q7>	<1;L;Q7>	<+;L;Q4>	<-;L;Q4>	<’ ’;L;Q4>
Q5	<2;L;Q5>	<0;N;Qt>	<1;N;Qt>	<+;L;Q5>	<-;L;Q5>	<’ ‘;L;Q5>
Q6	<1;L;Q5>	<2;L;Q5>	<0;N;Qt>	<+;L;Q6>	<-;L;Q6>	<’ ‘;L;Q6>
Q7	<1;N;Qt>	<2;N;Qt>	<0;L;Q7>			<1;N;Qt>

Пропущенные в таблице ячейки означают, что такой связи состояние – значение не существует.

Q0 – начальное состояние, которое необходимо, чтобы найти + или -.

Q1 – состояние, в котором ищется слагаемое для суммы (первая троичная цифра).

Q2 – состояние, в котором ищется вычитаемое для разности (первая троичная цифра).

Q3 – состояние, в котором ищется последняя цифра первого слагаемого, если второе слагаемое – 1, что аналогично прибавлению единице.

Q4 – состояние, в котором ищется последняя цифра первого слагаемого, если второе слагаемое – 2, что аналогично прибавлению двойки.

Q5 – состояние, в котором ищется последняя цифра уменьшаемого, если вычитаемое – 1, что аналогично вычитанию единицы.

Q6 – состояние, в котором ищется последняя цифра уменьшаемого, если вычитаемое – 2, что аналогично вычитанию двойки.

Q7 – состояние, необходимое для обработки переполнений.

Выводы.

Была написана программа, моделирующая работу машины Тьюринга на задачи сложения и вычитания троичного числа и троичной цифры.

ПРИЛОЖЕНИЕ А.

Исходный код программы.

Название файла: **main.py**

```
from TuringMachine import TuringMachine, States
from ActionTable import ActionTable
from Action import Action, Move

turing_machine = TuringMachine(list(input()),
                               ActionTable(
                                   {
                                       States.Q0:
                                           {
                                               '0': Action('0', Move.R,
States.Q0),
                                               '1': Action('1', Move.R,
States.Q0),
                                               '2': Action('2', Move.R,
States.Q0),
                                               '+': Action('+', Move.R,
States.Q1),
                                               '-': Action('-', Move.R,
States.Q2),
                                               ' ': Action(' ', Move.R,
States.Q0)
                                           },
                                       States.Q1:
                                           {
                                               '0': Action('0', Move.N,
States.Qt),
                                               '1': Action('1', Move.L,
States.Q3),
                                               '2': Action('2', Move.L,
States.Q4),
                                               ' ': Action(' ', Move.R,
States.Q1)
                                           },
                                       States.Q2:
                                           {
                                               '0': Action('0', Move.N,
States.Qt),
                                               '1': Action('1', Move.L,
States.Q5),
                                               '2': Action('2', Move.L,
States.Q6),
                                               ' ': Action(' ', Move.R,
States.Q2)
                                           },
                                       States.Q3:
                                           {
                                               '0': Action('1', Move.N,
States.Qt),
                                               '1': Action('2', Move.N,
States.Qt),
                                               '2': Action('0', Move.L,
```

```
States.Q7),
                    '+': Action('+', Move.L,
States.Q3),
                    '-': Action('-', Move.L,
States.Q3),
                    ' ': Action(' ', Move.L,
States.Q3)
                },
            States.Q4:
                {
                    '0': Action('2', Move.N,
States.Qt),
                    '1': Action('0', Move.L,
States.Q7),
                    '2': Action('1', Move.L,
States.Q7),
                    '+': Action('+', Move.L,
States.Q4),
                    '-': Action('-', Move.L,
States.Q4),
                    ' ': Action(' ', Move.L,
States.Q4)
                },
            States.Q5:
                {
                    '0': Action('2', Move.L,
States.Q5),
                    '1': Action('0', Move.N,
States.Qt),
                    '2': Action('1', Move.N,
States.Qt),
                    '+': Action('+', Move.L,
States.Q5),
                    '-': Action('-', Move.L,
States.Q5),
                    ' ': Action(' ', Move.L,
States.Q5)
                },
            States.Q6:
                {
                    '0': Action('1', Move.L,
States.Q5),
                    '1': Action('2', Move.L,
States.Q5),
                    '2': Action('0', Move.N,
States.Qt),
                    '+': Action('+', Move.L,
States.Q6),
                    '-': Action('-', Move.L,
States.Q6),
                    ' ': Action(' ', Move.L,
States.Q6)
                },
            States.Q7:
                {
                    '0': Action('1', Move.N,
States.Qt),
```

```
                                                    '1': Action('2', Move.N,
States.Qt),
                                                    '2': Action('0', Move.L,
States.Q7),
                                                    ' ': Action('1', Move.N,
States.Qt)
                                                }
                                        }))

turing_machine.process()
turing_machine.print_tape()
```

Название файла: **TuringMachine.py**

```
from enum import Enum



class States(Enum):
    Q0 = 0,
    Q1 = 1,
    Q2 = 2,
    Q3 = 3,
    Q4 = 4,
    Q5 = 5,
    Q6 = 6,
    Q7 = 7,
    Qt = 8



class TuringMachine:

    def __init__(self, tape, table):
        self.tape = tape
        self.pointer = 0
        self.state = States.Q0
        self.table = table

    def __make_step(self):
        action = self.table.get_action(self.state,
self.tape[self.pointer])
        self.tape[self.pointer] = action.new_val
        self.pointer += action.move
        self.state = action.new_state
```

```
def __delete_zeros(self):
    for index, value in enumerate(self.tape):
        if value == '0' and self.tape[index+1] in ('0', '1', '2'):
            self.tape[index] = ' '
        elif value in ('+', '-', '1', '2'):
            return

def process(self):
    while self.state != States.Qt:
        self.__make_step()
    self.__delete_zeros()

def print_tape(self):
    print(''.join(self.tape))
```

Название файла: **ActionTable.py**

```
class ActionTable:

def __init__(self, table):
    self.table = table

def get_action(self, state, value):
    return self.table[state][value]
```

Название файла: **Action.py**

```
from enum import IntEnum


class Move(IntEnum):
    L = -1,
    R = 1,
    N = 0


class Action:
```

```
def __init__(self, new_val, move, new_state):
    self.new_val = new_val
    self.move = move
    self.new_state = new_state
```