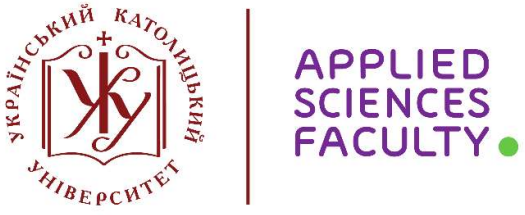
УКРАЇНСЬКИЙ КАТОЛИЦЬКИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНИХ НАУК
Комп’ютерні науки

Реалізувати машину Тюрінга

Автор: Кипибіда Діана

31 травня 2020



1. Вступ.

Результат проекту – реалізація примітивної машини Тюрінга на мові програмування python. Розроблений модуль дозволяє створити об’єкт машина Тюрінга для заданого алфавіту, створити довільну кількість станів машини і команд для проведення необхідних операцій над можливими вхідними словами, створену машину можна спробувати застосувати до довільного слова, якщо машина незастосовна до цього слова, алгоритм або зайде в безкінечний цикл, або виникне помиилка.

Знання з Дискретної математики, застосовані для реалізації проекту – матеріали 14-ої лекції.

2. Реалізація.

```
1
2   EMPTY_LETTER = None
3
4   class RIGHT:
5       '''Move to the right'''
6       pass
7   class LEFT:
8       '''Move to the lefft'''
9       pass
10  class NOWHERE:
11      '''Don't move anywhere'''
12      pass
13
```

Λ - порожня буква EMPTY_LETTER

RIGHT, LEFT, NOWHERE

(Праворуч, Ліворуч, Нікуди)

python класи що репрезентують рухи Головки читання-запису

```
13
14  class State:
15      __number = 0
16      def __init__(self):
17          self.__number = State.__number
18          State.__number += 1
19
20      def __str__(self):
21          return f"q{self.__number}"
22
```

Клас для репрезентації станів машини. Кожний створений стан отримує унікальний ідентифікатор (цифру) (Перший створений стан – q0, другий – q1, q2, …).

Літери алфавіту – python стрічки (str).

```
class Command:
    '''Команда машини Тюрінга'''
    def __init__(self, start_state, input_letter, output_letter, direction, end_sta
te):
        '''Створити команду
        Параметри:
            start_state: State – початковий стан
```

```
            input_letter: str or EMPTY_LETTER – вхідна літера
            output_letter: str or EMPTY_LETTER – вихідна літера
            direction: RIGHT, LEFT, NOWHERE – напрямок руху
            end_state: State – кінцевий стан
        Команда:
            start_state input_letter -> output_letter direction end_state
            Приклад: q1 A -> B R q2
        '''
        self.start_state = start_state
        self.input_letter = input_letter
        self.output_letter = output_letter
        self.direction = direction
        self.end_state = end_state

    def output(self) -> tuple:
        '''Повернути праву частину команди'''
        return (self.output_letter, self.direction, self.end_state)

    def __str__(self):
        '''Рядок – репрезентація лівої команди'''
        return f'{self.input_string()} -> {self.output_string()}'

    def output_string(self):
        '''Рядок – репрезентація правої частини команди'''
        return f'{self.output_letter} {d_to_l(self.direction)} {self.end_state}'

    def input_string(self):
        '''Рядок – репрезентація лівої частини команди'''
        return f'{self.start_state} {self.input_letter}'
```

```
class Cell:
    '''комірки “магнітної” стрічки'''
    __number = 0
    def __init__(self, item=None, left=None, right=None):
        self.letter = item  # Вміст Комірки
        self.left = left  # Посилання на комірку ліворуч
        self.right = right  # Посилання на комірку праворуч
        self.__number = Cell.__number
        Cell.__number += 1

    @staticmethod
    def connect(Cell1, Cell2):
        '''З’єднати послідовні комірки'''
        Cell2.left = Cell1
        Cell1.right = Cell2
```

```
class Tape:
    '''Пам’ять машини (Двозв’язний список)'''
    def __init__(self):
        self.clear()
```

```
    def write(self, word):
        for i in word:
            self.append_end(i)

    def clear(self):
        '''Очистити пам’ять'''
        self.start = Cell()
        self.end = Cell()
        Cell.connect(self.start, self.end)

    def __iter__(self):
        '''По пам’яті можна ітерувати'''
        cell = self.start
        while cell.right is not self.end:
            cell = cell.right
            yield cell

    def append_start(self, thing=EMPTY_LETTER):
        '''Додати порожню комірку на початку стрічки'''
        cell = Cell(thing)
        Cell.connect(cell, self.start.right)
        Cell.connect(self.start, cell)
        return cell

    def append_end(self, thing=EMPTY_LETTER):
        '''Додати порожню комірку вкінці стрічки'''
        cell = Cell(thing)
        Cell.connect(self.end.left, cell)
        Cell.connect(cell, self.end)
        return cell

    def right(self, head):
        '''Повернути клітинку, праворуч від даної
        Назва head використовується, щоб наголосити на тому,
        Що функція використовуватиметься для руху головки читання-запису'''
        if head.right is self.end:
            return self.append_end()
        return head.right

    def left(self, head):
        '''Повернути клітинку, ліворуч від даної
        Назва head використовується, щоб наголосити на тому,
        Що функція використовуватиметься для руху головки читання-запису'''
        if head.left is self.start:
            return self.append_start()
        return head.left

    def string(self, N=10):
       '''Рядок – репрезентація стрічки'''
        return '|'.join([str(i.letter).center(N) for i in list(self)])
```

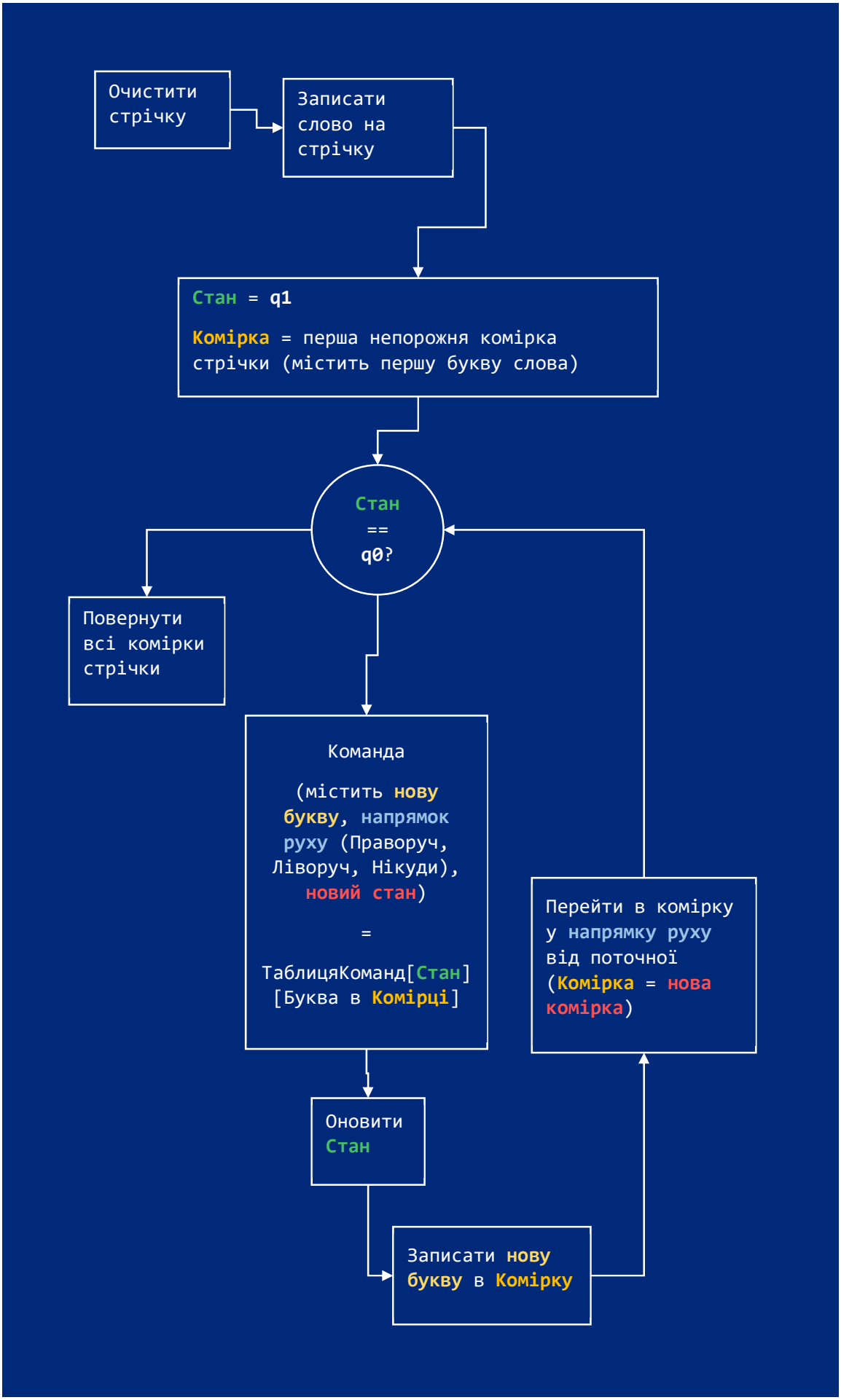
```
class CommandTable:
    '''Таблиця команд машини Тюрінга
```

```
    Фактично є словником словників, індексується за станом і літерою'''

    def __init__(self, alphabet):
        self._table = {}
        self.alphabet = list(alphabet)

    def __getitem__(self, state):
        if state not in self._table:
            self._table[state] = {}
        return self._table[state]

    def add(self, command: Command):
        '''Додати команду в таблицю
        Якщо команда з такою ж лівою частиною вже є в таблиці, перезаписує її'''
        if command.start_state in self and command.input_letter in self[command.sta
rt_state]:
            print('Warning: You overwrote an existing command!')
        self[command.start_state][command.input_letter] = command

    def remove(self, command: Command):
        '''Видалити команду з таблиці'''
        if not command.start_state in self:
            return
        if command.input_letter in self[command.start_state]:
            self[command.start_state].pop(command.input_letter)
```

```
class Machine:
    '''машина Тюрінга'''
    def __init__(self, letters: list):
        ''' Створити нову машину
        Параметри:
            letters: list of strings – список літер алфавіту
            алфавіт машини (self.alphabet) – даний алфавіт +
            {a’ (a + ‘ – буквально конкатенація стрічок) для всіх a є letters} +
{порожня літера}
        '''

        if not all(isinstance(i, str) for i in letters):
            raise ValueError('Alphabet can contain strings only')

        self.alphabet = letters+list(map(lambda x: x+"'", letters))+[EMPTY_LETTER]
        self.__command_table = CommandTable(self.alphabet)
        self.__start_state = State()
        self.__end_state = State()
        self.tape = Tape()

    def start_end_states(self):
        '''Повернути початковий і кінцевий стани машини (q1 і q0)'''
        return self.__start_state, self.__end_state
```

```
    def run(self, word):
        '''Process given word'''

        # Begin work
        current_state = self.__start_state
        self.tape.clear()  # clear tape
        self.tape.write(word)  # write word onto the tape
        tape = self.tape
        current_cell = tape.start.right  # start at the beginning of the tape

        while current_state is not self.__end_state:
            self.print_machine(current_state, current_cell)

            command = self.__command_table[current_state][current_cell.letter]
            current_cell.letter, shift, current_state = command.output()

            if shift == RIGHT:
                current_cell = tape.right(current_cell)  # move to the right
            elif shift == LEFT:
                current_cell = tape.left(current_cell)  # move to the left

        self.print_machine(current_state, current_cell)
        return tape

    def print_machine(self, stateЖ, head_cell):
        '''Вивести машину на екран'''
        …

    def add_command(self, command):
        '''Додати команду в таблицю команд'''
        if not command.input_letter in self.alphabet or \
               not command.output_letter in self.alphabet:
           raise ValueError(…)
        self.__command_table.add(command)

    def remove_command(self, command):
        '''Видалити команду з таблиці команд'''
        self.__command_table.remove(command)

    def table(self):
        return str(self.__command_table)
```

Нижче діаграми роботи методу Machine.run(word)


Очистити стрічку
Записати слово на стрічку
Стан = q1
Комірка = перша непорожня комірка стрічки (містить першу букву слова)
Стан == q0?
Повернути всі комірки стрічки
Команда
(містить нову букву, напрямок руху (Праворуч, Ліворуч, Нікуди), новий стан)
=
ТаблицяКоманд[Стан] [Буква в Комірці]
Перейти в комірку у напрямку руху від поточної (Комірка = нова комірка)
Оновити Стан
Записати нову букву в Комірку

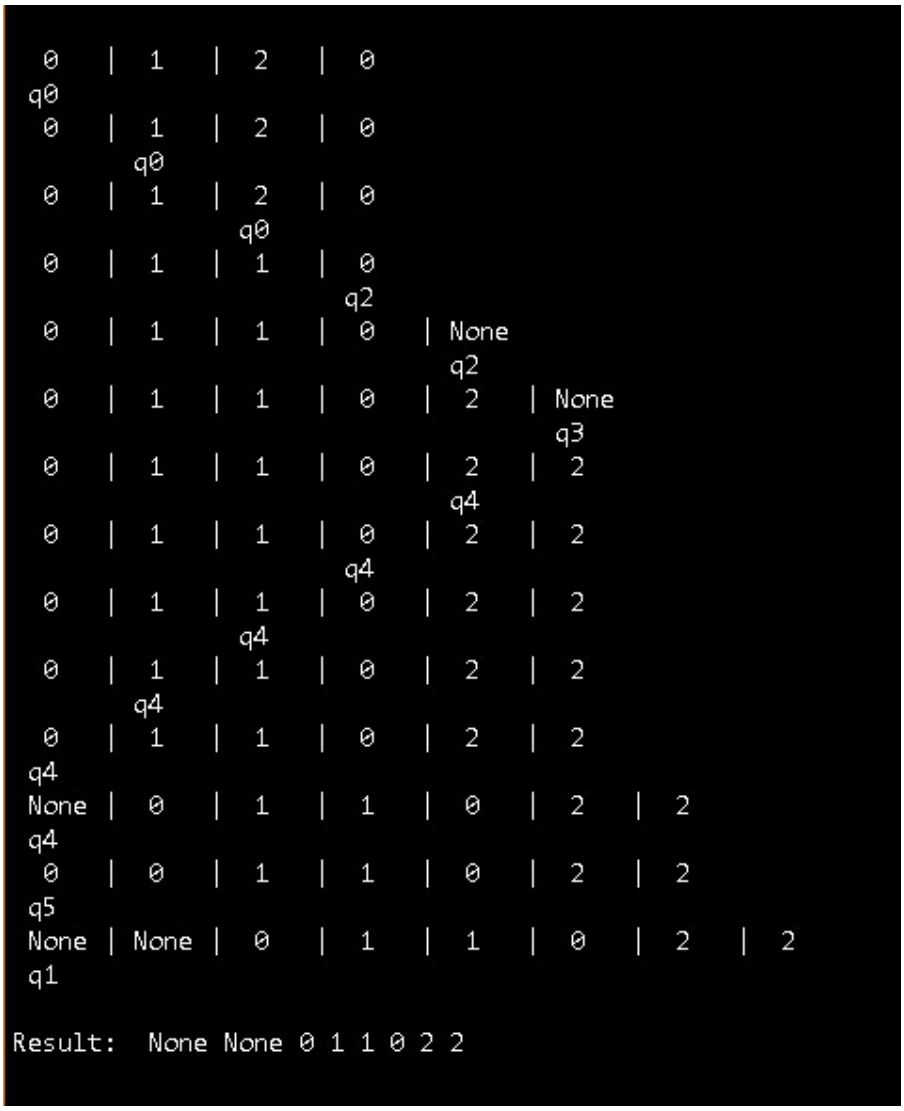
Приклад використання модуля:

```
    word = '0120'
    # Створити машину
    m = Machine(['0', '1', '2'])
    # Зберегти автоматично створені машиною q1 I q0 для подальших маніпуляцій
    q1, q0 = m.start_end_states()
    # Створити необхідну кількість станів
    q2, q3, q4, q5 = State(), State(), State(), State()

    print('Input word:', word)

    # Додати команди до машини
    m.add_command(Command(q1, '0', '0', RIGHT, q1))
    m.add_command(Command(q1, '1', '1', RIGHT, q1))
    m.add_command(Command(q1, '2', '1', RIGHT, q2))
    m.add_command(Command(q2, '0', '0', RIGHT, q2))
    m.add_command(Command(q2, EMPTY_LETTER, '2', RIGHT, q3))
    m.add_command(Command(q3, EMPTY_LETTER, '2', LEFT, q4))
    m.add_command(Command(q4, '2', '2', LEFT, q4))
    m.add_command(Command(q4, '1', '1', LEFT, q4))
    m.add_command(Command(q4, '0', '0', LEFT, q4))
    m.add_command(Command(q4, EMPTY_LETTER, '0', NOWHERE, q5))
    m.add_command(Command(q5, '0', EMPTY_LETTER, LEFT, q0))

    # Вивести таблицю команд
    print('\n', m.table(), '\n')

    print('\nResult: ', m.run(word), '\n\n')
```

Виведена таблиця команд:

```
           |     0      |     1      |     2      |     0'     |     1'     |     2'     |     None
  |
-------------------------------------------------------------------------------------------------------
------
     q0    |   0 R q0   |   1 R q0   |   1 R q2   |     -      |     -      |     -      |     -
  |
     q2    |   0 R q2   |     -      |     -      |     -      |     -      |     -      |   2 R q3
  |
     q3    |     -      |     -      |     -      |     -      |     -      |     -      |   2 L q4
  |
     q4    |   0 L q4   |   1 L q4   |   2 L q4   |     -      |     -      |     -      |   0 N q5
  |
     q5    | None L q1  |     -      |     -      |     -      |     -      |     -      |     -
  |
-------------------------------------------------------------------------------------------------------
------
```

Робота машни покроково:



У цієї реалізації машини є деякі особливості. По перше, машина повертає всі створені комірки стрічки, включно з порожніми (None). По-друге, немає ніяких обмежень на створення команд, крім того, щоб всі використані літери були в алфавіті машини (для того, щоб позначити букву нее потрібно змінювати стан і т.д.).