УКРАЇНСЬКИЙ КАТОЛИЦЬКИЙ УНІВЕРСИТЕТ

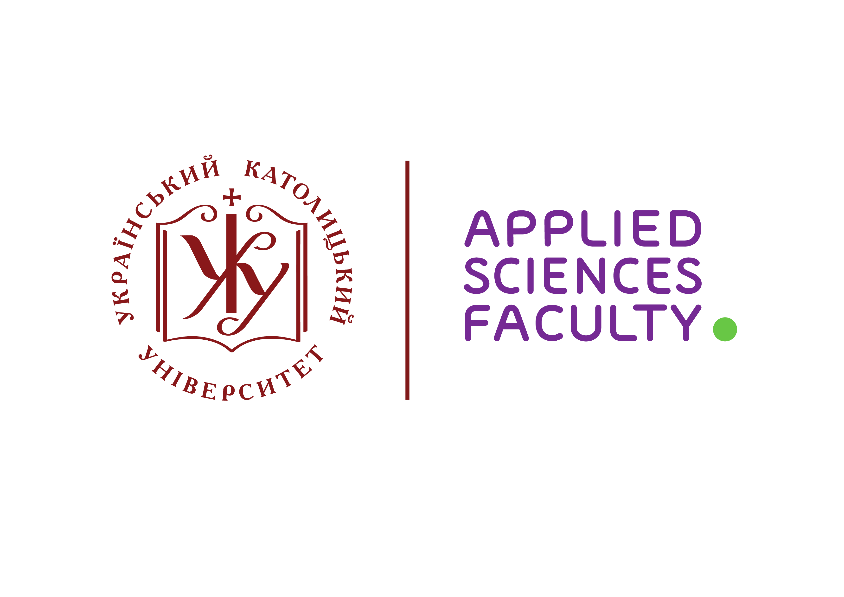
ФАКУЛЬТЕТ ПРИКЛАДНИХ НАУК

Комп’ютерні науки

**Реалізувати машину Тюрінга**

*Автор: Кипибіда Діана*

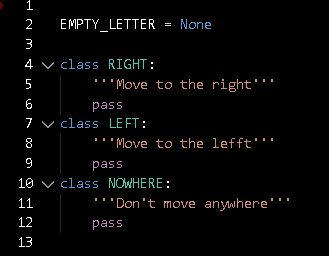
31 травня 2020



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.

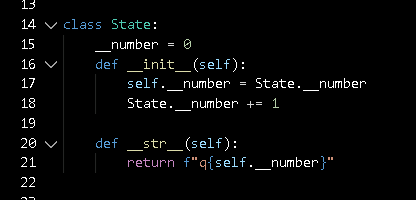
Знання з Дискретної математики, застосовані для реалізації проекту – матеріали 14-ої лекції.

1. **Реалізація.**

**ꓥ** - порожня буква EMPTY\_LETTER

**RIGHT, LEFT, NOWHERE (Праворуч, Ліворуч, Нікуди**)

python класи що репрезентують рухи Головки читання-запису



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\_state):

        '''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.start\_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’ (а + ‘ – 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

Команда

(містить **нову букву**, **напрямок руху** (Праворуч, Ліворуч, Нікуди), **новий стан**)

=

ТаблицяКоманд[**Стан**][Буква в **Комірці**]

Оновити **Стан**

Записати слово на стрічку

**Стан** = **q1**

**Комірка** = перша непорожня комірка стрічки (містить першу букву слова)

Повернути всі комірки стрічки

Записати **нову букву** в **Комірку**

Перейти в комірку у **напрямку руху** від поточної (**Комірка** = **нова комірка**)

1. **Висновки.**

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))

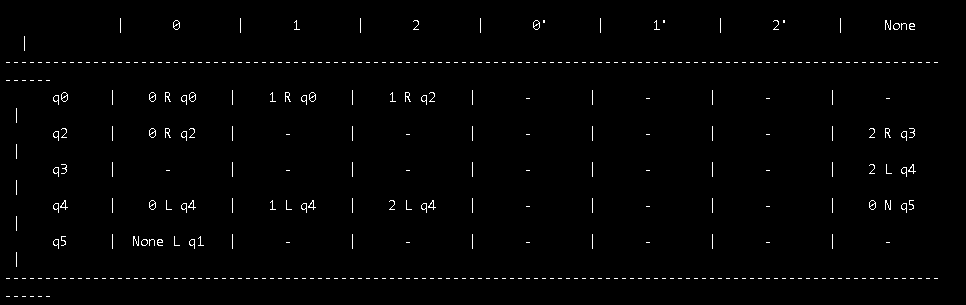
    m.add\_command(Command(q5, '0', EMPTY\_LETTER, LEFT, q0))

    # Вивести таблицю команд

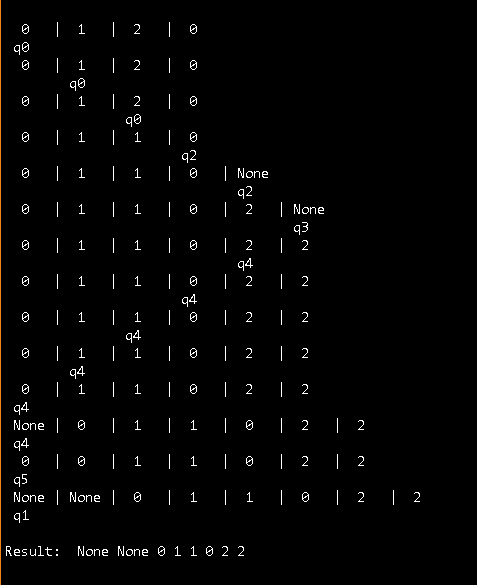
    print('\n', m.table(), '\n')

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

Виведена таблиця команд:



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.).