МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРІЇНИ
НАВЧАЛЬНО НАУКОВИЙ КОМЛЕКС
“ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ”
НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

КУРСОВА РОБОТА

з дисципліни

“Алгоритми та аналіз складності”

На тему:

“Машина Тьюрінга”

Виконав:
Студент ІІ-го курсу
гр. КА – 41
Барзій Ілля
Ігорович

Прийняла:
_____
Тимощук Оксана
Леонідівна
“ ” ____ _____ 2016

КИЇВ 2016

,

Анотація

Курсова робота представляє собою реалізацію машини Тюрінга. Програма дозволяє користувачу писати код, зберігати його у файл, зчитувати з файлу та виконувати його. Код може виконуватись за різних швидкостей із можливістю зупинки та продовження виконання, покроковим виконанням, виводом необхідної інформації у інтерфейс.

Для написання програми використовувалась мова програмування python з графічним інтерфейсом Tkinter.

Аннотация

Курсовая робота представляет собой реализацию машины Тьюринга. Программа позволяет пользователю писать код, сохранять его в файл, считывать из файла и выполнять его. Код может выполняться при разных скоростях с возможностью остановки и продолжения выполнения, пошаговым выполнением, выводом необходимой информации в интерфейс.

Для написания программы использовался язык программирования python с графическим интерфейсом Tkinter.

Annotation

This coursework is Turing machine simulator. The program allows user to write code, save it in a file, read from file and execute it. Code may be executed in different speeds, execution may be stopped and continued, done step-by-step with information output on the interface.

The program was written on python using Tkinter user interface.

,

Зміст

,

1.Постановка задачі

Постановка задачі полягає у написанні програми, що реалізує машину Тюрінга. Користуючись засобами створення графічного інтерфейсу Tkinter:

1) реалізувати введення з клавіатури коду програми для машини Тюрінга, що буде виконуватися, початкового слова, початкового та кінцевих станів, швидкості виконання програми;

2)забезпечити перевірку коректності введення коду програми користувачем та видавати відповідні повідомлення у разі помилок;

3)реалізувати збереження програми користувача з можливістю її подальшого відкриття;

4) реалізувати простий та ефективний в роботі інтерфейс;

5) забезпечити можливість різних способів виконання програми - просте виконання з заданою швидкістю та можливістю зупинки машини, покрокове виконання програми.

,

2. Теоретична частина

У машині Тюрінга пам'ять представляє собою необмежену з обох сторін стрічку, розбиту на комірки. Оскільки у мові програмування python до масиві можна додавати елементи під час виконання програми, а максимальна допустима кількість елементів у масиві більша 5,3*10^8, то можна вважати, що стрічка нескінченна.

Машина має кінцеву кількість знаків (символів) що утворюють зовнішній алфавіт, в якому кодуються данні, що передаються у машину а також ті, що обробляються у ній. У написаній програмі зовнішній алфавіт складається з символів ASCII. Символ "#" є порожнім знаком або знаком порожньої комірки.

На будь-якій стадії роботи машини в кожній комірці може зберігатися не більше одного символу. До початку роботи на стрічку подається початкове слово. Робота машини складається з виконуваних один за одним тактів, по ходу котрих виконується перетворення початкової інформації на стрічці у проміжну.

В якості початкової інформації на стрічку подається будь-яка вінцева система знаків зовнішнього алфавіту, розставлене довільним чином по комірках. В залежності від початкової інформації U можливі два випадки:

1. Після кінцевого числа тактів машина зупиниться та видасть сигнал про зупинку (чи зупинка у кінцевому стані чи виникла помилка). При цьому на стрічці лишається деяка інформація B.

2. Зупинка та сигнал про зупинку ніколи не виведуться. У написаній програмі такий результат роботи машини допустимий.

В машині Тюрінга в кожному окремому такті бере участь лише одна комірка пам'яті - та, на яку вказує показник. При переході машини з одного такту до іншого положення показнику може змінюватися не більше ніж на 1 у кожному з напрямків.

В машині опрацювання інформації виконується в логічному блоці, котрий може перебувати в одному з кінцевого числа станів. У даній програмі стан може позначатись будь - яким кінцевим набором символів ASCII, не розділених пробілом.

,

3.Опис програмного продукту

Реалізуємо саму машину Тюрінга за методологією ООП.

Стрічку машини Тюрінга реалізуємо у формі класу, що має функції створення, ходу, зчитування, рухів стрічки: вправо, вліво, знаходження на місці та виводу. При створенні стрічки їй передаються початкове слово(за замовчанням порожнє), позиція(за замовчанням 0) та символ порожньої комірки. Символи стрічки зберігаються у масиві..

При виконанні функції ходу передається символ перевірки, символ заміни та напрямку руху. Якщо символ перевірки не співпадає з символом, на якому встановлений вказівник на стрічці, видається помилка. Інакше, символ замінюється символом заміни. Далі, в залежності від напрямку руху викликається функція руху стрічки у відповідному напрямку.

Функція зчитування стрічки повертає символ, на якому знаходиться вказівник стрічки.

Функції руху змінюють позицію показника на стрічці та, за потреби, додають символ порожньої комірки в початок чи кінець стрічки.

Функція виводу виводить у масив комірок (взятий масив розміром 25) слово, що міститься на стрічці так, щоб комірка, на якій стоїть показник була 13ю (посередині) а комірки, що не мають символу зі стрічки мали символ порожньої комірки.

Саму машину Тюрінга теж реалізуємо у формі класу, що має функції створення, додавання команди, виконання кроку, виконання всієї програми, збереження стану машини та запуску зі стану.

Функція створення приймає початкову стрічку, початковий стан, множину кінцевих станів, та символ порожньої комірки.

Функція додавання команди приймає поточний стан, поточну літеру, наступний стан, літеру заміни та напрям руху. Якщо програма ще не має команди для даного стану та даного символу, то команда додається.

Функція виконання кроку складається з восьми етапів, а саме:

1. Перевірка чи довжина стрічки нульова та чи машина у кінцевому стані.

2. Перевірка чи поточний стан є кінцевим. Якщо так, то вивести інформацію про успішне виконання програми.

3. Перевірка чи поточний стан є в програмі. Якщо ні, то видати повідомлення про помилку.

,

4. Зчитування символу, на якому стоїть вказівник.
5. Перевірка чи є у поточному стані команда для зчитаного символу. Якщо ні, вивести відповідну помилку.
6. Отримати зі словнику (списку команд) наступний стан, символ виводу та напрям руху.
7. Замінити поточний стан наступним.
8. Записати зміни у стрічку.

Функція виконання всієї програми виводить стрічку на екран, викликає функцію виконання кроку, збільшує лічильник виконаних команд та викликає саму себе через час, що заданий користувачем. Така структура дозволяє змінювати швидкість та зупиняти програму під час її виконання.

Функція збереження стану машини зупиняє виконання програми користувача та зберігає поточний стан машини (данні з класу) у масив.

Функція запуску зі стану знову запускає машину та передає їй як параметри данні з масиву з попередньої функції.

Далі йде опис графічного інтерфейсу та його функцій.

Функція NewFile очищує вікно вводу коду користувача.

Функція OpenFile відкриває текстовий файл та переписує його зміст у вікно вводу коду користувача.

Функція About виводить вікно, що відображає коротку інформацію про автора програми.

Функція run зчитує початкове слово, що вводиться у машину Тюрінга, початковий стан та кінцеві стани. Також функція перевіряє введені користувачем на коректність та, якщо все вірно, передає данні на виконання машині. Інакше - видає помилку.

Функція stop викликає збереження стану машини.

Функція continu викликає запуск машини зі стану.

Функція step викликає виконання кроку машини.

Функція file_save зберігає данні з вікна для вводу коду користувача у текстовий файл.

Функція quitme знищує вікно та виходить з програми.

,

4.Алгоритм роботи програми

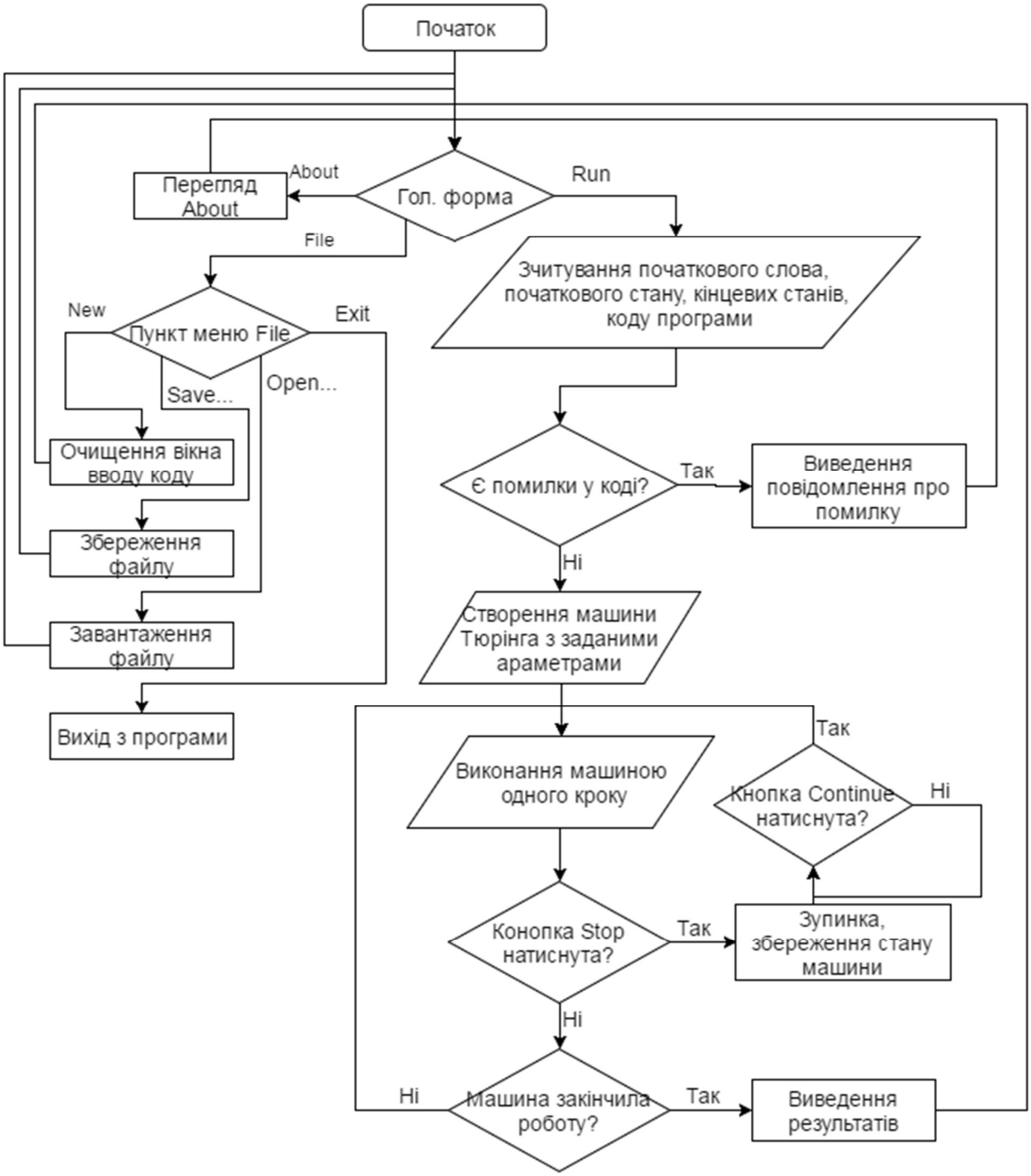


Рис.1. Структурна блок-схема програми.

,

5.Результати роботи програми

При запуску програма видає головне вікно.

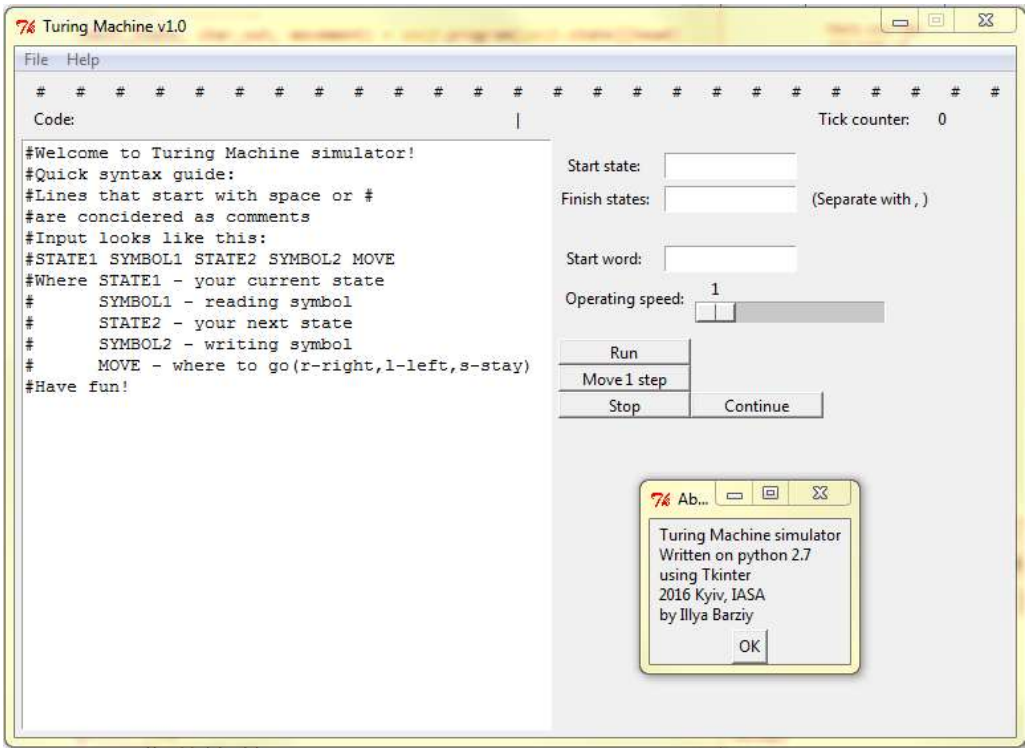
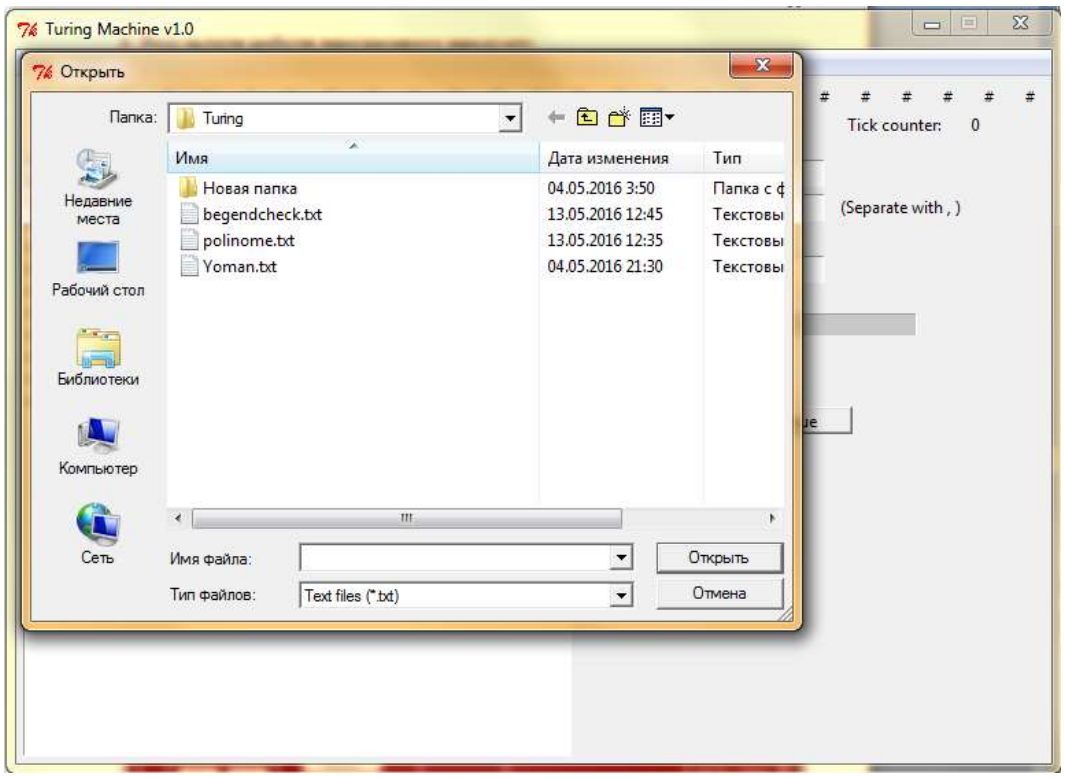


Рис.2. Головне вікно з вікном інформації про автора програми(About)

Завантажимо програму для машини з текстового файлу та введемо посаткові данні.



,

Рис. 3. Вікно вибору текстового файлу для завантаження програми.

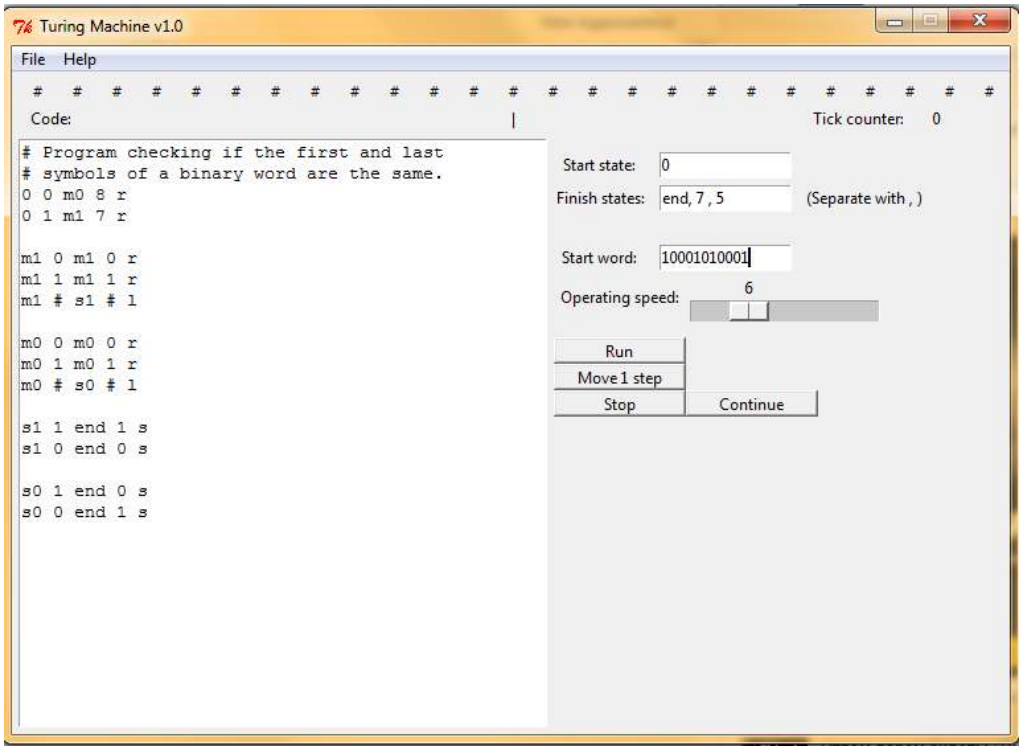


Рис. 4. Головне вікно з введеною з файлу програмою, початковими данними та швидкістю виконання.

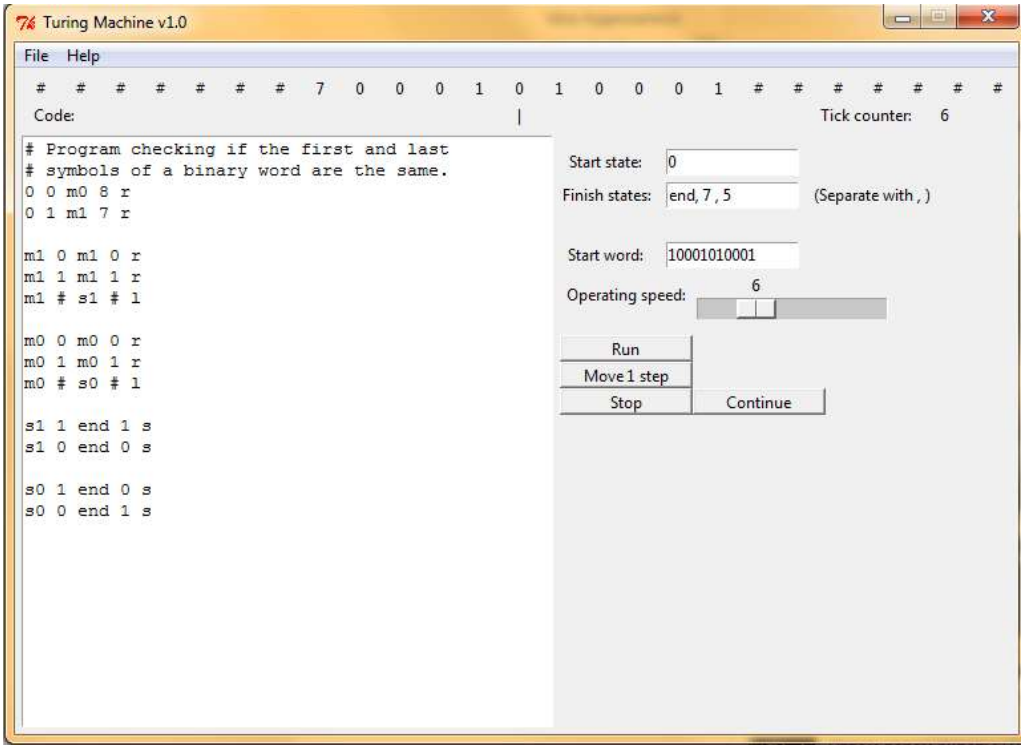


Рис.5. Інтерфейс під час виконання програми.

,

6.Висновки

В результаті виконання курсової роботи була створена програма, що моделює машину Тюрінга. Програма складається з симулятору машини , написаного у вигляді ООП та графічного інтерфейсу. Вона продемонструвала гарну працездатність, має зрозумілий інтерфейс і не повинна викликати труднощів у роботі кінцевого користувача.

У ході виконання курсової роботи були вивчені основи програмування мовою python. Написання програми сприяло закріпленню теоретичного матеріалу на практиці.

Написана програма є логічно завершеною. Перевагами є висока працездатність, легкість у використанні, зрозумілий інтерфейс та можливість розширення функціоналу через використання методології ООП.

,

7.Список використаної літератури

1. https://en.wikipedia.org/wiki/Turing_machine

2. https://docs.python.org/2/library/tkinter.html

3. https://stackoverflow.com/

,

Додаток. Лістинг програми

App.py

```
"""Here is the Turing Machine Core"""
import sys
pr = sys.stdout.write

class MachineTapeException(Exception):
        """ Turing Exception Exception """
        def __init__(self, value):
                Exception.__init__(self)
                self.value = value
        def __str__(self):
                return self.value

class TuringErrorException(Exception):
        """ Turing Exception Exception """
        def __str__(self):
                return "Crash"

class TuringAcceptException(Exception):
        """ Turing Accept Exception """
        def __str__(self):
                return "Accept"

class MachineTape:
        def __init__(self, initialString=[], initialPos=0, blank="#"):
                """ The Tape uses a simple list.  It could easily be changed into a
string if
                   need be """
                self.tape = []
                self.pos = initialPos
                self.blank = blank
                self.initialString = initialString
                if len(initialString) > 0:
                   for ch in initialString:
                           self.tape.append(ch)
                else:
                   self.tape.append(blank)

        def reinit(self):
```


,

```
            self.__init__(self.initialString)

        def move(self, check_char, changeto_char, direction):
            """ Only R, L, S directions are supported """
            # check to see if the character under the head is what we need
            if check_char != self.tape[self.pos]:
                raise MachineTapeException ("Tape head doesn't match head
character")

            # at this point the head is over the same character we are looking for
            #  change the head character to the new character
            self.tape[self.pos] = changeto_char

            if direction == "L":
                self.move_left()
            elif direction == "R":
                self.move_right()
            elif direction == "S":
               self.move_stay()
              else: raise MachineTapeException ("Direction is invalid")

        def read(self):
            """ return the character over the head """
            return self.tape[self.pos]

        def move_left(self):
            if self.pos <= 0:
                self.tape.insert(-1, self.blank)
                self.pos = 0
            else:
                self.pos -= 1

        def move_right(self):
            self.pos += 1
            if self.pos >= len(self.tape): self.tape.append(self.blank)

        def move_stay(self):
        self.pos=self.pos

        def show(self):
            """ print the tape """
            for cell in range(25):
                if cell<(12-self.pos):
                 cells[cell].set('#')
```


,

```
                    if (cell>=(12-self.pos))and(cell<(len(self.tape)+12-self.pos)):
                     cells[cell].set(self.tape[cell-12+self.pos])
                    if cell>=(len(self.tape)+12-self.pos):
                     cells[cell].set('#')
                root.update_idletasks()

"""
The program structure for the TM is created with a dictionary.
   To step algorithm:
        1. Check to see if the length of the string is zero and if we
           are in a final state
        2. If the currentstate is in the final states then raise an Accept
        3. If the currentstate is not in the program then raise an Error
        4. Check the head character
        5. If the head character is not in the program and in the current state then
           raise an Error
        6. Retrieve from the dictionary the dest_state, char_out, and movement
        7. set the current state to the new state
        8. write the tape, and move the head

Program Layout:
   [state][char_in] --> [(dest_state, char_out, movement)]
"""

class TuringMachine:
        def __init__(self, initialString, initialSt, finalStates=[], blank="#"):
                self.blank = blank
                self.tape = MachineTape(initialString)
                self.fstates = finalStates
                self.program = {}
                self.initState = initialSt
                self.state = self.initState
                self.lenStr = len(initialString)

        def reinit(self):
                self.state = self.initState
                self.tape.reinit()

        def addTransition(self, state, char_in, dest_state, char_out, movement):
                if not self.program.has_key(state):
                        self.program[state] = {}

                tup = (dest_state, char_out, movement)
                self.program[state][char_in] = tup
```


,

```
        def step(self):
                """ Steps 1 - 3 """
                if self.lenStr == 0 and self.state in self.fstates: raise
TuringAcceptException
                if self.state in self.fstates: raise TuringAcceptException
                if self.state not in self.program.keys(): raise TuringErrorException

                """ Steps 4 and 5 """
                head = self.tape.read()
                if head not in self.program[self.state].keys(): raise
TuringErrorException

                """ Steps 6 and 7 """
                # execute transition
                (dest_state, char_out, movement) = self.program[self.state][head]
                self.state = dest_state
                try:
                        """ Step 8 """
                        self.tape.move(head, char_out, movement)
                except MachineTapeException, s:
                        print s

        def execute(self):
                """ The TM will keep stepping forever until the TM accepts or rejects.
                   This does allow for looping TM's """
                try:
                        global stop
                        if not stop:
                         self.tape.show()
                         self.step()
                         t = int(1000/float(scale.get()))
                         global num
                         num.set(int(num.get())+1)
                         #root.update()
                         root.after(t,self.execute)
                except (TuringErrorException, TuringAcceptException), s:
                        print s
        def exec1(self):
                try:
                         self.tape.show()
                         self.step()
                         global num
                         num.set(int(num.get())+1)
```


,

```
                except (TuringErrorException, TuringAcceptException), s:
                        print s

        def gett(self):
                global g
                g=[]
                global stop
                g.append(self.blank)
                g.append(self.tape)
                g.append(self.fstates)
                g.append(self.program)
                g.append(self.initState)
                g.append(self.state)
                g.append(self.lenStr)
                stop=1
        def startfrom(self):
                global stop
                global g
                stop=0
                self.blank=g[0]
                self.tape=g[1]
                self.fstates=g[2]
                self.program=g[3]
                self.initState=g[4]
                self.state=g[5]
                self.lenStr=g[6]
                try:

                        if not stop:
                          self.tape.show()
                          self.step()
                          t = int(1000/float(scale.get()))
                          global num
                          num.set(int(num.get())+1)
                          #root.update()
                          root.after(t,self.execute)
                except (TuringErrorException, TuringAcceptException), s:
                        print s

"""Here is the Gui core"""

from Tkinter import
Tk,Menu,mainloop,Label,Button,DoubleVar,Scale,HORIZONTAL,\
                Text,FALSE,Entry,END,StringVar,Toplevel,Message,INSERT
```


,

```
import os
import tkFileDialog
import tkMessageBox
def NewFile():
    text.delete('1.0',END)

def OpenFile():
        ftypes = [('Text files', '*.txt'), ('All files', '*')]
        dlg = tkFileDialog.Open(filetypes = ftypes)
        fl = dlg.show()

        if fl != '':
            f = open(fl, "r")
            tex = f.read()
            text.delete('1.0',END)
            text.insert(INSERT, tex)

def About():
    top = Toplevel()
    top.title("About this application...")
    msg = Message(top, text="Turing Machine simulator\nWritten on python 2.7 \
                        using Tkinter\n2016 Kyiv, IASA\nby Illya Barziy")
    msg.pack()
    button = Button(top, text="OK", command=top.destroy)
    button.pack()

def run():
    global m
    global stop
    stop = 0
    global num
    num.set(0)
    skip=0
    stword = word.get()
    ststate = startst.get()
    finstate = finishst.get()
    finstate = "".join(finstate.split())
    finst = finstate.split(",")

    tx=text.get("1.0",END)
    tx=tx.encode('ascii','ignore')
    tx = os.linesep.join([s for s in tx.splitlines() if s])
    tx = os.linesep.join([n for n in tx.splitlines() if not n.startswith('#')])
    tx = os.linesep.join([n for n in tx.splitlines() if not n.startswith(' ')])
```


,

```
    ar=tx.split()

    g=0
    for elements in ar:
        g+=1
    if (g % 5 !=0):
            tkMessageBox.showwarning(
            "Error",
            "Error in input: Number of arguments in input command don't match")
    com=g/5
    commands = [[ar[y*5+x] for x in range(5)] for y in range(com)]

    for i in range(com):
        if (len(commands[i][1])!=1) or (len(commands[i][3])!=1):
            skip = 1
            tkMessageBox.showwarning(
            "Error",
            "Error in input: Some of symbols entered are morethen one-digit")
        nd = commands[i][4].upper()
        commands[i][4]=commands[i][4].upper()
        if not ((nd=='L') or (nd=='R') or (nd=="S")):
            tkMessageBox.showwarning(
            "Error",
            "Error in input: Incorrect move side (L R or S allowed)")
            skip = 1

    if not skip:
        print stword,ststate,finst
        m = TuringMachine(stword,ststate,finst)
        for i in range(com):
            m.addTransition(commands[(i-1)][0], commands[(i-1)][1],
                            commands[(i-1)][2],commands[(i-1)][3],
                            commands[(i-1)][4]
                        )
        m.execute()
        #m.gett()

def stop():
    global m
    m.gett()

def continu():
    global m
    m.startfrom()
```


,

```
def step():
    global m
    global stop
    stop = 1
    m.exec1()

def file_save():
    f = tkFileDialog.asksaveasfile(mode='w', defaultextension=".txt")
    if f is None: # asksaveasfile return `None` if dialog closed with "cancel".
        return
    text2save = str(text.get(1.0, END)) # starts from `1.0`, not `0.0`
    f.write(text2save)
    f.close() # `()` was missing
def quitme():
    root.destroy()

#Window set
root = Tk()
root.wm_title("Turing Machine v1.0")
root.geometry("755x500")
root.resizable(width=FALSE, height=FALSE)
#Menu set
menu = Menu(root)
root.config(menu=menu)
filemenu = Menu(menu)
menu.add_cascade(label="File", menu=filemenu)
filemenu.add_command(label="New", command=NewFile)
filemenu.add_command(label="Open...", command=OpenFile)
filemenu.add_command(label="Save...", command=file_save)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=quitme)
helpmenu = Menu(menu)
menu.add_cascade(label="Help", menu=helpmenu)
helpmenu.add_command(label="About...", command=About)
#Scale set
var = DoubleVar()
scale = Scale( root, variable = var, orient=HORIZONTAL, from_=1, to=20)
scale.place(height=40,width=150,x=510, y=150)
#Text set
text = Text(root)
text.place(height=445,width=400,x=5, y=50)
text.insert(END, "#Welcome to Turing Machine simulator!\n#Quick syntax
guide:\n#Lines that start with space or #\n#are concidered as comments\n")
```


,

```
text.insert(END, "#Input looks like this:\n#STATE1 SYMBOL1 STATE2
SYMBOL2 MOVE\n#Where STATE1 - your current state\n#      SYMBOL1 -
reading symbol\n#      STATE2 - your next state\n")
text.insert(END, "#      SYMBOL2 - writing symbol\n#      MOVE - where to go(r-
right,l-left,s-stay)\n#Have fun!\n")
#Entry set
word = Entry(root)
word.place(height=20,width=100,x=490,y=130)

startst = Entry(root)
startst.place(height=20,width=100,x=490,y=60)

finishst = Entry(root)
finishst.place(height=20,width=100,x=490,y=85)

labels = []
cells=[]
global num
num = StringVar()
num.set(0)
#Setting cells
for i in range(25):
    cells.append(StringVar())
    cells[i].set('#')
#Displaying cells
for data in range(25):
    labels.append(Label(root, textvariable=cells[data]))
    labels[data].place(height=20,width=30,x=5+data*30,y=5)
#Setting additional labels
Label(root,text="|").place(height=20,width=30,x=5+12*30,y=25)
Label(root,text="Tick counter:").place(height=20,width=70,x=605,y=25)
Label(root,textvariable=str(num)).place(height=20,width=40,x=680,y=25)
Label(root,text="Start word:").place(height=20,width=70,x=410,y=130)
Label(root,text="Start state:").place(height=20,width=70,x=410,y=60)
Label(root,text="Finish states:").place(height=20,width=70,x=410,y=85)
Label(root,text="(Separate with , )").place(height=20,width=90,x=600,y=85)
Label(root,text="Operating speed:").place(height=20,width=100,x=410,y=160)
Label(root,text="Code:").place(height=20,width=50,x=5,y=25)
#Setting operating buttons:
startbutton = Button(root)
startbutton.configure(text="Run",command=run)
startbutton.place(height=20,width=100,x=410,y=200)

stopbutton = Button(root)
```


,

```
stopbutton.configure(text="Stop",command=stop)
stopbutton.place(height=20,width=100,x=410,y=240)

playbutton = Button(root)
playbutton.configure(text="Continue",command=continu)
playbutton.place(height=20,width=100,x=510,y=240)

stepbutton = Button(root)
stepbutton.configure(text="Move 1 step",command=step)
stepbutton.place(height=20,width=100,x=410,y=220)

mainloop()
```


,