Turing Machine

Daniel Antonio Rodriguez Perez 130561

Juan Carlos Gonzales Ibarra, Teoría Computacional, UPSLP

Design a Turing Machine that, given a word w of the alphabet Σ = {0, 1}, provides its reverse, wR in **Python**.

Daniel Antonio Rodriguez Perez 130561

ABSTRACT  
The main problem of this project is to design in Python a turing machine that given a word of 0's and 1's can invert this. The solution is much more complex than that, principally because we must choose the best structure to represent an MT, the second problem is to simulate a tape of infinite length.

In the end it is not only solved the problem, the scope was much higher obtaining a deterministic turing machine of a single tape. That I could solve any problem that can be solved with this. In other words, the source code could solve any Turing Machine problem with adding the transitions, defining a final state and an initial state.

# Introduction

In this project, I planned to design in Python a turing machine that, given a word of 0's and 1's, could invert this. The solution is much more complex than that, in the end it is not only solved the problem, the scope was much higher obtaining a deterministic turing machine of a single tape. That I could solve any problem that can be solved with this.

The Turing machine, presented by Alan Turing in 1936 in On computable numbers, with an application to the Entscheidungsproblems, is the mathematical model of a device that behaves like a finite automata and that has a tape of infinite length in which they can read, write or erase symbols. There are other versions with several tapes, deterministic or not, etc., but all are equivalent (with respect to the languages ​​they accept).  
  
One of the most important theorems about Turing machines is that they can simulate the behavior of a computer (storage and control unit). Therefore, if a problem can not be solved by one of these machines, then it can not be solved by a computer (undecidable problem, NP).  
  
The notation of the Turing machines is simple and exact, so it is more convenient to work with them when studying which problems are decidable (P) and which are undecidable (NP).  
  
At the moment, the inclusion relationship between P and NP is yet to be proven, although we do know that  
  
P⊆NP  
  
In addition, we will say that the languages ​​accepted by the Finite Automata (Deterministic or not, with or without transitions-ε, with or without a battery ...) can also be accepted by some Turing machine.  
  
On this report we will find some solution for resolve the problem of reverse a word the way 01 for obtain 10 with Python.

# Methodology

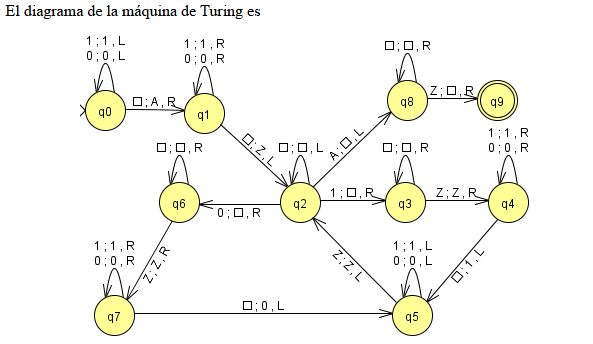
Definition of the Turing Machine  
  
We call Turing Machine (or MT) to  
  
**M = (Q, Σ, T, δ, q0, B, F)**  
  
where

* Q is the finite set of states that we will denote by  
      q0, q1, q2, ...
* Σ is the alphabet: the finite set of input symbols.
* Τ is the set of ribbon symbols. The alphabet is a subset of Τ.
* q0 is the initial state: the state in which the MT is initially located.
* B is an element of Σ: the symbol in white. It is found in all the boxes on the tape that do not have an entry symbol.
* F is the set of final states.
* δ is the function of transitions.

The expression  
  
**δ (q, X) = (p, Y, D)**  
Indicates that in the state q, if the head of the MT points to the ribbon symbol X, then the MT writes the ribbon symbol Y in the current box (changes X to Y) and moves the head one square towards D (D can be right, R; or left, L) and go to the state p.  
  
The tape of the MT is formed by infinite squares.  
  
Initially, the input word (a concatenation of symbols of the alphabet) is written in consecutive boxes of the tape and the head points to the first symbol of the word. All other boxes (left and right) contain the blank symbol.

# Solution

We assumes that the solution for the MT was provided by the teacher. Thus this is the solution.



# Source Code

Function \_\_init\_\_

Firstly, I created a class called MT and defined initial variables, which I'll list and explain later.  
self .\_\_ tabla = is a dictionary and will have every MT transition  
self .\_\_ estado\_inicial = my initial state that by definition is q0  
self .\_\_ estado\_final = my final state which in this case is q9

Self.cadena = represents the infinite tape

Then, I call the delta function which represents the function of transitions, but that will be explained later

**class** mt:

*# cadena = []*

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

self.\_\_tabla = {}

self.\_\_estado\_inicial = ""

self.\_\_estado\_final = ""

self.cadena = []

self.delta()

Function agrega\_transicion

In this function, to add the transitions that are built in the delta function and added in the table, in the image you can see how the table ends after adding transitions. Each object in the table represents a transition, which was already explained in the introduction.

Table

(q0, '0') -> ('q0', '0', 'L')

(q0, '1') -> ('q0', '1', 'L')

(q0, 'B') -> ('q1', 'A', 'R')

(q1, '0') -> ('q1', '0', 'R')

(q1, '1') -> ('q1', '1', 'R')

(q1, 'B') -> ('q2', 'Z', 'L')

(q2, 'B') -> ('q2', 'B', 'L')

(q2, 'A') -> ('q8', 'B', 'L')

(q2, '1') -> ('q3', 'B', 'R')

(q2, '0') -> ('q6', 'B', 'R')

(q8, 'B') -> ('q8', 'B', 'R')

(q8, 'Z') -> ('q9', 'B', 'R')

(q3, 'B') -> ('q3', 'B', 'R')

(q3, 'Z') -> ('q4', 'Z', 'R')

(q4, 'B') -> ('q5', '1', 'L')

(q4, '0') -> ('q4', '0', 'R')

(q4, '1') -> ('q4', '1', 'R')

(q5, '0') -> ('q5', '0', 'L')

(q5, '1') -> ('q5', '1', 'L')

(q5, 'Z') -> ('q2', 'Z', 'L')

(q6, 'B') -> ('q6', 'B', 'R')

(q6, 'Z') -> ('q7', 'Z', 'R')

(q7, 'B') -> ('q5', '0', 'L')

(q7, '0') -> ('q7', '0', 'R')

(q7, '1') -> ('q7', '1', 'R')

Initial States:

q0

Final States:

q9

**def** agrega\_transicion(self, estado, simbolo, nuevo\_estado,nuevo\_simbolo, accion):

try:

self.\_\_tabla[estado]

except KeyError:

self.\_\_tabla[estado] = {}

try:

self.\_\_tabla[estado][simbolo]

except KeyError:

self.\_\_tabla[estado][simbolo] = {}

if type(nuevo\_estado) == str:

self.\_\_tabla[estado][simbolo] = (nuevo\_estado,nuevo\_simbolo,accion)

else:

self.\_\_tabla[estado][simbolo] = zip(nuevo\_estado,nuevo\_simbolo,accion)

Function evaluate

This is the most difficult method, here the word is evaluated. To understand a bit, by definition it starts at q0 and the tape starts at the first character. Then we have the state and the symbol by which to move but if you access an index that does not exist throws an exception and ends the cycle, otherwise continue moving through tape and replacing the characters according to each transition.

**def** evaluate(self, string):

self.cadena = [i for i in string]

estado\_actual=self.\_\_estado\_inicial

apuntador=0

while True:

try:

print ("(%s, '%s')" % (estado\_actual,self.cadena[apuntador]))

estado\_actual,nuevo\_simbolo,accion=self.\_\_tabla[estado\_actual][self.cadena[apuntador]]

self.cadena[apuntador]=nuevo\_simbolo

apuntador+=self.mueve\_cinta(accion,apuntador)

print ("(%s, '%s',%s)" % (estado\_actual, nuevo\_simbolo,accion))

*# print(apuntador)*

except IndexError:

break

except KeyError:

break

pass

Function mueve\_cinta

The biggest problem is simulating an infinite tape. An array is finite and to simulate this case I have added a function called mueve\_cinta, in case it moves to the left and the index is 0 it adds a "B" element to the list header and if it moves to the right and the index is larger than the list, add an element B at the end of the list. In this way I avoid problems accessing an index that does not exist in the list, therefore the tape grows dynamically inside the Machine. The index grows or decreases according to this function.

**def** mueve\_cinta(self,accion,apuntador):

if accion == "L":

if apuntador < 1:

self.cadena.insert(0,"B")

return 0

return -1

elif accion == "R":

if apuntador > (len(self.cadena)-2):

self.cadena.append("B")

return 1

elif accion == "S":

return 0

# Results

At the end, a word is requested to evaluate by console and redirect the output to an "out.txt" file.

cadena = input("Esrcibe una cadena: ")

*#code for redirect stdout to a file*

orig\_stdout = sys.stdout

f = open('out.txt', 'w')

sys.stdout = f

*#code for redirect stdout to a file*

maquina\_turing = mt()

maquina\_turing.print\_mt()

print (maquina\_turing.evaluate(cadena))

Try the word 101ab

Escribe una cadena: 101ab

Output

['A', '1', '0', '1', 'a', 'b']

False

Try the word 01

Escribe una cadena: 01

Output

['B', 'B', 'B', 'B', 'B', '1', '0']

True

Conclusion

The personal conclusion outside the context of the turing machines was the way to simulate the tape seems to me a very dirty way to solve the problem, I would have liked to use pointers, but for me it is a relatively new language and for time I decided to use a solution easy.

(2018, 08). Turing machine. *Wikipedia*. Obtenido 11, 2018, de <https://en.wikipedia.org/wiki/Turing_machine>

(2018, 01). What is a Turing machine?. *Computer Laboratory*. Obtenido 11, 2018, de <https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/turing-machine/one.html>