
## Centro de Ciências Exatas e Tecnológicas

**Teoria da computação 2021.1 - Ramon Pereira Lopes**

**Éverton Gomes dos Santos**

# Máquina de Turing não determinística

## Introdução

### O que é?

A Máquina de Turing é um dispositivo teórico conhecido como máquina universal, que foi concebido pelo matemático britânico Alan Turing em seu artigo publicado em 1936. É um modelo abstrato de um computador, que se restringe apenas aos aspectos lógicos do seu funcionamento (memória, estados e transições), e não a sua implementação física. (wikipedia)
Informalmente uma máquina de Turing consiste de:

- Uma **fita** ilimitada dividida em células que contem símbolos de um alfabeto.
- Um **cabeçote** que pode ler e escrever símbolos na fita e se move para ambos os lados.
- Uma **unidade de controle** que contem um registrador de estados e uma tabela de transições pré determinadas que dizem a máquina o estado atual, se deve ler ou escrever símbolos na fita e o estado de destino.

### Qual a importância?

De acordo com *Botellho* (p. 1, apud BITTENCOURT, 2001; LOVE, 4004; SETZER 2006) A máquina de Turing teve importância fundamental no desenvolvimento das áreas de computabilidade,

teoria dos autômatos formais e análise de algoritmos. A distinção entre hardware e software por meio do conceito de máquina universal é considerada como um dos triunfos do século XX.

# Projeto e Implementação

Foi solicitado um simulador de uma Máquina de Turing padrão não determinística onde a entrada é dada pelos estados, alfabeto de entrada, alfabeto da fita, símbolo delimitador e símbolo branco, quantidade de transições e transições a serem fetias, estado inicial, estados finais e as palavras a serem computadas. As transições são especificadas como uma quíntupla <a, b, c, d, e> onde 'a' é o estado de origem, 'b' é o caractere a ser lido, 'c' é o estado de destino, 'd' é o símbolo a ser escrito e, por fim, 'e' é a direção, imóvel (I), esquerda (E) e direita (D).
O protótipo recebe os dados como descrito no paragrafo anterior como strings, com excessão da quantidade de transições que são inteiros. A principal informação a ser mencionada são as **transições** que são armazenadas utilizando a estrutura de dados *dicionários* no qual as chaves são formadas pelo estado de origem e caractere lido, e os valores são o estado de destino, simbolo a ser escrito e a direção.

A implementação do algoritmo foi realizada utilizando a linguagem python por ser mais palpável para solução do problema e apresentar simplicidade e robustez. A ferramenta de desenvolvimento utilizada foi o replit por ser uma IDE online que permite a implementação em qualquer lugar e ainda possui armazenamento e controle de versão próprio. A captação das entradas e a inicialização do dicionáro é mostrada a seguir:

In [ ]:
```
estados = input().split()
alfabetoEntrada=input().split()
alfabetoFita = input().split()
delimitador = input()
simboloBranco = input()
quantTransicoes=int(input())
transicoes=dict()
for i in range(quantTransicoes):
    quintupla = input().split()
    chave = (quintupla[0], quintupla[1])
    if chave not in transicoes:
        transicoes[chave]=([[quintupla[2], quintupla[3], quintupla[4]]])
    else:
        transicoes[chave].append([quintupla[2], quintupla[3], quintupla[4]])
estadoInicial = input()
estadosFinais = input().split()
palavras = input().split()
```

O núcleo do algoritmo é verificar se a palavra é aceita ou não pela máquina descrita. Para isso ao receber a palavra ela é partida em símbolos e dado o estado incial verifica se o conjunto estado símbolo corresponde a uma das chaves contidas no dicionário de transições. A fita é representada por uma estrutura de *lista* composta pelo símbolo delimitador, a palavra recebida e o símbolo branco para cada uma das palavras recebidas. Foi utilizada ainda uma pilha auxiliar para que dada a palavra se a chave da computação atual estiver no dicionário é atribuida a esta pilha a configuração atual com posição do cabeçote e fita. Assim temos o não determinismo onde para cada computação a busca de estado atual e símbolo é realizada.
Este procedimento se repete até que a palavra chegue ao fim e a máquina esteja em um estado

final.

O procedimento descrito é codificado a seguir:

In [ ]:

```
aux = 1
for palavra in palavras:
    fita=list((delimitador+palavra+simboloBranco))
    pilha=[(estadoInicial, aux, fita)]
    aceita=False
    while len(pilha)>0:
        p=pilha.pop()
        estadoAtual=p[0]
        indice_p=p[1]
        fita_p=p[2][:]
        chave_p=(estadoAtual,fita_p[indice_p])
        if chave_p not in transicoes and estadoAtual in estadosFinais:
            aceita=True
            break
        if chave_p in transicoes:
            for a in transicoes[(chave_p)]:
                estadoAtual = a[0]
                fita_p[indice_p] = a[1]
                direcao = a[2]
                if direcao == 'E':
                    pilha.append((estadoAtual, indice_p-1, fita_p))
                elif direcao == 'D':
                    if (indice_p+1)==len(fita):
                        fita_p.append(simboloBranco)
                    pilha.append((estadoAtual, indice_p+1, fita_p))
                elif direcao == 'I':
                    pilha.append((estadoAtual, indice_p, fita_p))
    if aceita:
        print("S")
    else:
        print("N")
```

# Resultados

## Estudo empírico de tempo de execução

Para o estudo de tempo de execução foi adicionado ao protótipo o calculo de tempo, utilizando o tempo inicial e final de cada execução para cada palavra, pode-se ter uma relação entre tempo de execução e tamanho da palavra que será usado para estudos. Para isso foi utilizada a MT apresentada na descrição do problema prático variando o tamanho das palavras de entrada **apenas com palavras aceitas** pela máquina.

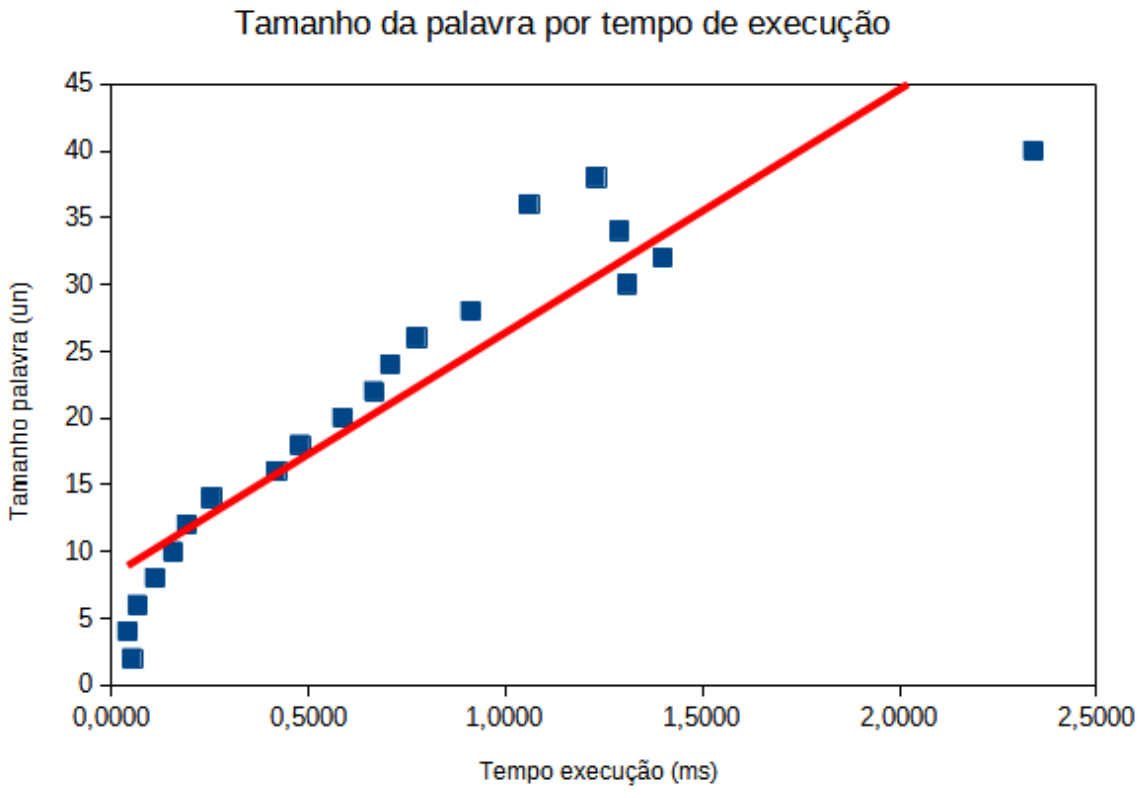
O software utilizado para realizar a regressão linear simples foi o *LibreOffice calc*. Uma observação é que esse procedimento poderia ser realizado utilizando o próprio python, porém devido a falta de experiência e o tempo extinto, optou-se pela utilização do software de planilhas.

Foram utilizadas 20 entradas aumentando-se o tamanho em 2 símbolos em cada palavra. A tabela a seguir apresenta a amostra:

| Tamanho palavra(un) | Tempo execução(ms) |
|---|---|
| 2 | 0,0536 |
| 4 | 0,0412 |

| Tamanho palavra(un) | Tempo execução(ms) |
|---|---|
| 6 | 0,0672 |
| 8 | 0,1110 |
| 10 | 0,1570 |
| 12 | 0,1920 |
| 14 | 0,2550 |
| 16 | 0,4190 |
| 18 | 0,4800 |
| 20 | 0,5870 |
| 22 | 0,6670 |
| 24 | 0,7080 |
| 26 | 0,7750 |
| 28 | 0,9120 |
| 30 | 1,3100 |
| 32 | 1,4000 |
| 34 | 1,2900 |
| 36 | 1,0600 |
| 38 | 1,2300 |
| 40 | 2,3400 |

Com esta amostra foi gerado o gráfico a seguir:



A reta que melhor se ajusta aos pontos foi gerada a partir da aplicação da função: $ f(x) = m*x + n $, onde $ f(x) $ é a variável dependente igual a *tamanho da palavra* em unidades. $ x $ é a variável independente igual ao *tempo de execução* em milisegundos. e $ m $ e $ n $ são parâmetros descobertos justamente aplicando a regressão linear simples.

A equação da reta é: $ f(x) = 18,206*x + 8,206 $

Outros dados da regressão como fator $ R^2 = 85,38\% $ que apresenta o quão próximos os dados estão da linha de regrssão, erro padrão, análise de vacância dentre outros, podem ser encontrados aqui.

## Conclusão

Pode-se concluir que a medida que o tamanho da palavra aumenta o tempo de execução também aumenta. Porém destaca-se que a quantidade de transições para uma determinada palavra também influencia no tempo de execução da mesma pois haverá palavras em que a máquina percorrerá mais passos e mais estados para poder solucioná-la, assim palavras de mesmo tamanho podem ser determinadas pela máquina por passos diferentes e assim tempos diferentes.