**Documentação: Minimização de Autômatos Finitos Determinísticos (DFA)**

**Introdução**

Autômatos Finitos Determinísticos (DFAs) são máquinas de estado finito utilizadas na teoria da computação e na ciência da computação para reconhecer padrões dentro de cadeias de caracteres. A minimização de DFAs é o processo de transformar um DFA em outro DFA equivalente que tenha o menor número possível de estados.

Esta documentação descreve o processo de minimização de DFAs, apresentando o código utilizado, os resultados obtidos (principalmente os diagramas dos autômatos antes e depois da minimização), e um manual do usuário para orientar a utilização do código.

## Código Utilizado

O código a seguir define uma classe DFA que permite a criação, visualização e minimização de autômatos finitos determinísticos.

import networkx as nx

import matplotlib.pyplot as plt

# Define a classe DFA (Deterministic Finite Automaton)

class DFA:

def \_\_init\_\_(self, states, alphabet, transition\_function, start\_state, accept\_states):

# Inicializa os estados do DFA

self.states = states

# Inicializa o alfabeto do DFA

self.alphabet = alphabet

# Inicializa a função de transição do DFA

self.transition\_function = transition\_function

# Inicializa o estado inicial do DFA

self.start\_state = start\_state

# Inicializa os estados de aceitação do DFA

self.accept\_states = accept\_states

# Método para remover estados inacessíveis

def remove\_inaccessible\_states(self):

# Conjunto para armazenar estados acessíveis

accessible\_states = set()

# Fila de estados a serem verificados, começando pelo estado inicial

queue = [self.start\_state]

while queue:

# Remove o primeiro estado da fila

state = queue.pop(0)

if state not in accessible\_states:

# Adiciona estado ao conjunto de estados acessíveis

accessible\_states.add(state)

# Para cada símbolo no alfabeto

for symbol in self.alphabet:

# Obtém o próximo estado a partir da função de transição

next\_state = self.transition\_function.get((state, symbol))

if next\_state and next\_state not in accessible\_states:

# Adiciona o próximo estado à fila se ainda não foi visitado

queue.append(next\_state)

# Atualiza os estados, função de transição e estados de aceitação com apenas os estados acessíveis

self.states = accessible\_states

self.transition\_function = {(state, symbol): next\_state for (state, symbol), next\_state in self.transition\_function.items() if state in accessible\_states and next\_state in accessible\_states}

self.accept\_states = {state for state in self.accept\_states if state in accessible\_states}

# Método para minimizar o DFA

def minimize(self):

# Primeiro, remove estados inacessíveis

self.remove\_inaccessible\_states()

# Inicialmente, dois conjuntos: estados de aceitação e não aceitação

P = [set(self.accept\_states), set(self.states) - set(self.accept\_states)]

# Fila de conjuntos para verificar distinção

W = [set(self.accept\_states), set(self.states) - set(self.accept\_states)]

while W:

# Remove um conjunto da fila

A = W.pop()

for symbol in self.alphabet:

# X é o conjunto de estados que transitam para um estado em A com o símbolo atual

X = {state for state in self.states if self.transition\_function.get((state, symbol)) in A}

for Y in P[:]:

# Interseção de X e Y

intersection = X & Y

# Diferença entre Y e X

difference = Y - X

if intersection and difference:

# Remove Y de P

P.remove(Y)

# Adiciona interseção a P

P.append(intersection)

# Adiciona diferença a P

P.append(difference)

if Y in W:

# Remove Y de W

W.remove(Y)

# Adiciona interseção a W

W.append(intersection)

# Adiciona diferença a W

W.append(difference)

else:

if len(intersection) <= len(difference):

# Adiciona menor conjunto a W

W.append(intersection)

else:

# Adiciona maior conjunto a W

W.append(difference)

# Criar novos estados

new\_states = {frozenset(s): i for i, s in enumerate(P)}

# Nova função de transição

new\_transition\_function = {}

# Novo conjunto de estados de aceitação

new\_accept\_states = set()

for group in P:

# Pega um representante do grupo

representative = next(iter(group))

# Estado novo correspondente ao grupo

new\_state = new\_states[frozenset(group)]

if representative in self.accept\_states:

# Se o representante é estado de aceitação, adiciona o novo estado aos estados de aceitação

new\_accept\_states.add(new\_state)

for symbol in self.alphabet:

# Obtém o próximo estado a partir do representante

next\_state = self.transition\_function.get((representative, symbol))

if next\_state:

# Encontra o grupo ao qual o próximo estado pertence

next\_group = next(s for s in P if next\_state in s)

# Define a nova transição

new\_transition\_function[(new\_state, symbol)] = new\_states[frozenset(next\_group)]

# Atualiza os estados com os novos estados

self.states = set(new\_states.values())

# Atualiza a função de transição

self.transition\_function = new\_transition\_function

# Atualiza os estados de aceitação

self.accept\_states = new\_accept\_states

# Atualiza o estado inicial

self.start\_state = new\_states[frozenset(next(s for s in P if self.start\_state in s))]

# Método para representar o DFA como string

def \_\_str\_\_(self):

return f"States: {self.states}\nAlphabet: {self.alphabet}\nStart state: {self.start\_state}\nAccept states: {self.accept\_states}\nTransitions: {self.transition\_function}"

# Método para desenhar o grafo do DFA

def draw(self):

# Cria um grafo direcionado

G = nx.DiGraph()

# Adiciona nós ao grafo

G.add\_nodes\_from(self.states)

for (state, symbol), next\_state in self.transition\_function.items():

# Adiciona arestas ao grafo

G.add\_edge(state, next\_state, label=symbol)

# Layout do grafo

pos = nx.spring\_layout(G)

# Desenha o grafo

nx.draw(G, pos, with\_labels=True, node\_size=2000, node\_color="lightblue", font\_size=12, font\_weight="bold", arrows=True)

# Define os rótulos das arestas

edge\_labels = {(u, v): d['label'] for u, v, d in G.edges(data=True)}

# Desenha os rótulos das arestas

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, font\_color='red')

# Exibe o grafo

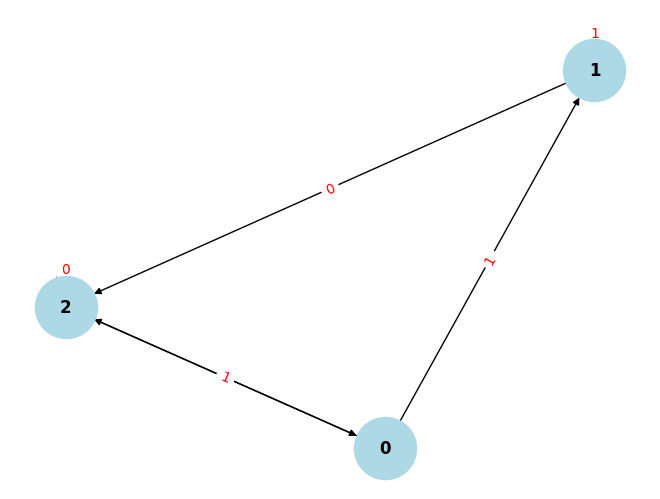
plt.show()

## Plots: Exemplo 1:

Original

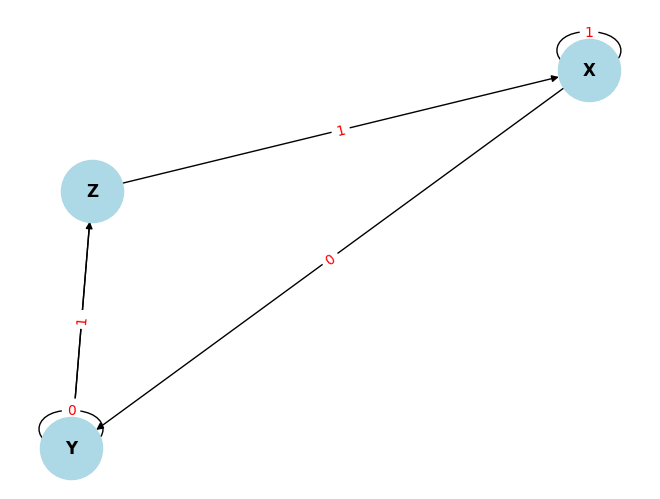
## 

Minimizado

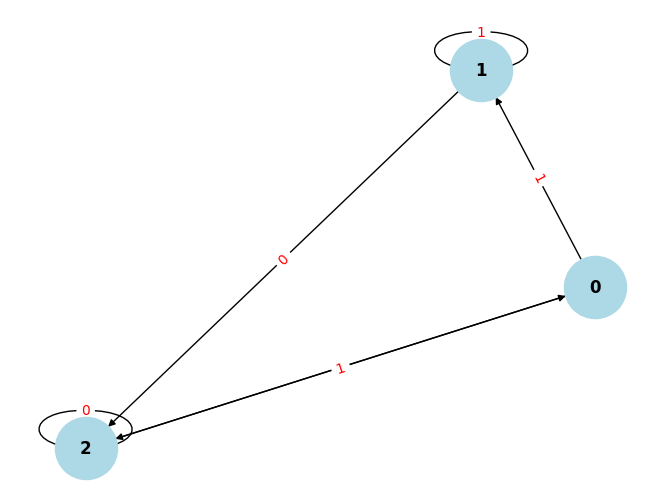


Exemplo 2:

Original

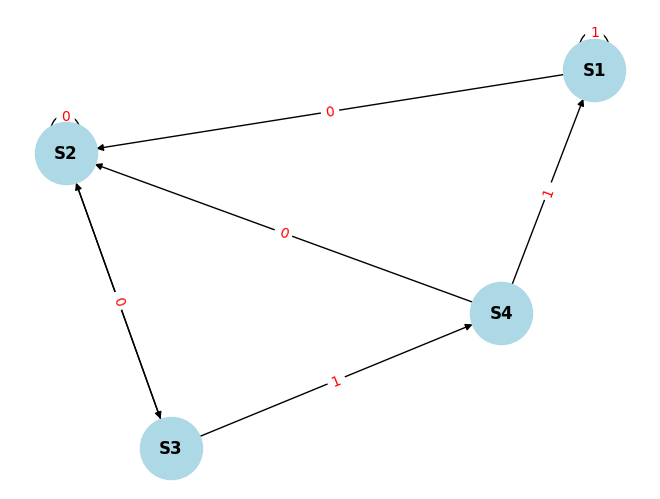


Minimizado

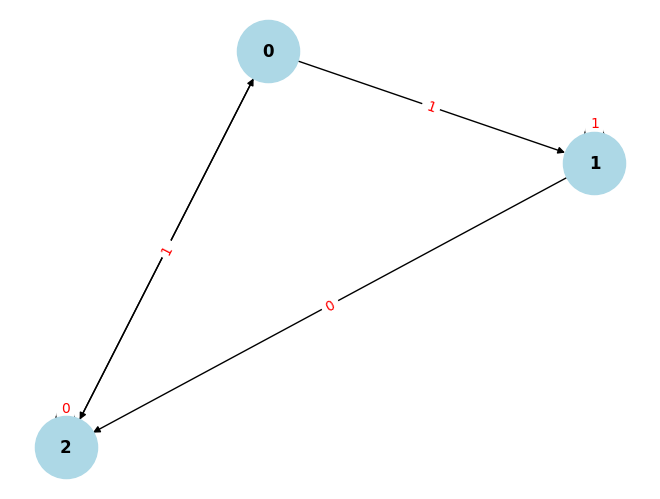


Exemplo 3

Original



Minimizado



## Exemplos de Uso

### Exemplo 1: Cadeias que terminam com "01"

### DFA Original

states = {'A', 'B', 'C', 'D'}

alphabet = {'0', '1'}

transition\_function = {

('A', '0'): 'B',

('A', '1'): 'A',

('B', '0'): 'B',

('B', '1'): 'C',

('C', '0'): 'B',

('C', '1'): 'D',

('D', '0'): 'B',

('D', '1'): 'A'

}

start\_state = 'A'

accept\_states = {'C'}

dfa1 = DFA(states, alphabet, transition\_function, start\_state, accept\_states)

print("Original DFA 1:")

print(dfa1)

dfa1.draw()

#### DFA Minimizado

dfa1.minimize()

print("\nMinimized DFA 1:")

print(dfa1)

dfa1.draw()

### Exemplo 2: Cadeias que contêm "01" no final

#### DFA Original

states = {'X', 'Y', 'Z'}

alphabet = {'0', '1'}

transition\_function = {

('X', '0'): 'Y',

('X', '1'): 'X',

('Y', '0'): 'Y',

('Y', '1'): 'Z',

('Z', '0'): 'Y',

('Z', '1'): 'X'

}

start\_state = 'X'

accept\_states = {'Z'}

dfa2 = DFA(states, alphabet, transition\_function, start\_state, accept\_states)

print("Original DFA 2:")

print(dfa2)

dfa2.draw()

#### DFA Minimizado

dfa2.minimize()

print("\nMinimized DFA 2:")

print(dfa2)

dfa2.draw()

### Exemplo 3: Cadeias que terminam com "01" (estrutura diferente)

#### DFA Original

states = {'S1', 'S2', 'S3', 'S4'}

alphabet = {'0', '1'}

transition\_function = {

('S1', '0'): 'S2',

('S1', '1'): 'S1',

('S2', '0'): 'S2',

('S2', '1'): 'S3',

('S3', '0'): 'S2',

('S3', '1'): 'S4',

('S4', '0'): 'S2',

('S4', '1'): 'S1'

}

start\_state = 'S1'

accept\_states = {'S3'}

dfa3 = DFA(states, alphabet, transition\_function, start\_state, accept\_states)

print("Original DFA 3:")

print(dfa3)

dfa3.draw()

#### DFA Minimizado

dfa3.minimize()

print("\nMinimized DFA 3:")

print(dfa3)

dfa3.draw()

## Manual do Usuário

1. **Instalação das Dependências:** Certifique-se de ter o networkx e o matplotlib instalados. Utilize os comandos abaixo para instalar:  
     
   pip install networkx matplotlib
2. **Criação de um DFA:** Para criar um DFA, instancie a classe DFA com os estados, alfabeto, função de transição, estado inicial e estados de aceitação.  
     
   dfa = DFA(states, alphabet, transition\_function, start\_state, accept\_states)
3. **Visualização do DFA:** Use o método draw() para visualizar o DFA.  
     
   dfa.draw()
4. **Minimização do DFA:** Utilize o método minimize() para minimizar o DFA.  
     
   dfa.minimize()
5. **Visualização do DFA Minimizado:** Após a minimização, use novamente o método draw() para visualizar o DFA minimizado.  
     
   dfa.draw()