class AF:

def \_init\_(self, states, alphabet, transitions, initial\_state, final\_states):

self.states = states

self.alphabet = alphabet

self.transitions = transitions

self.initial\_state = initial\_state

self.final\_states = final\_states

@staticmethod

def from\_file(filename):

with open(filename, 'r') as file:

content = file.read()

return AF.parse(content)

@staticmethod

def from\_input():

content = ""

print("Introduceți datele automatului (respectând structura specificată):")

while True:

line = input()

if line == "": # Stop on empty line

break

content += line + "\n"

return AF.parse(content)

@staticmethod

def parse(content):

# Extrage elementele conform EBNF utilizând regex

states = re.findall(r'stari:\s\*(.\*)', content)[0].split(', ')

alphabet = re.findall(r'alfabet:\s\*(.\*)', content)[0].split(', ')

transitions = re.findall(r'tranzitii:\s\*(.\*)', content)[0].split('; ')

initial\_state = re.findall(r'stare\_initiala:\s\*(.\*)', content)[0]

final\_states = re.findall(r'stari\_finale:\s\*(.\*)', content)[0].split(', ')

transition\_dict = {}

for t in transitions:

src, rest = t.split(', ')

symbol, dest = rest.split('->')

if (src, symbol) not in transition\_dict:

transition\_dict[(src, symbol)] = dest

return AF(states, alphabet, transition\_dict, initial\_state, final\_states)

def display\_elements(self):

print("1. Mulțimea stărilor:", self.states)

print("2. Alfabetul:", self.alphabet)

print("3. Tranzițiile:")

for (src, symbol), dest in self.transitions.items():

print(f" {src} --{symbol}--> {dest}")

print("4. Starea inițială:", self.initial\_state)

print("5. Mulțimea stărilor finale:", self.final\_states)

def accept\_sequence(self, sequence):

state = self.initial\_state

for symbol in sequence:

if (state, symbol) in self.transitions:

state = self.transitions[(state, symbol)]

else:

return False

return state in self.final\_states

def longest\_prefix(self, sequence):

state = self.initial\_state

prefix = ""

longest\_accept = ""

for symbol in sequence:

if (state, symbol) in self.transitions:

state = self.transitions[(state, symbol)]

prefix += symbol

if state in self.final\_states:

longest\_accept = prefix

else:

break

return longest\_accept

Descriere

AutomatFinit:

States: Set<String>

Alphabet: Set<String>

Transitions: Map<<String,String>String>

initialState: String

finalState: String