Automata Programming paradigm

Construction of DFA that accepts substring 01

class DFA:

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

self.states = states

self.alphabet = alphabet

self.transition\_function = transition\_function

self.start\_state = start\_state

self.accept\_states = accept\_states

def run(self, input\_string):

current\_state = self.start\_state

for char in input\_string:

if char not in self.alphabet:

return False

current\_state = self.transition\_function[current\_state][char]

return current\_state in self.accept\_states

# Example usage:

states = {'q0', 'q1', 'q2'}

alphabet = {'0', '1'}

class DFA:

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

self.states = states

self.alphabet = alphabet

self.transition\_function = transition\_function

self.start\_state = start\_state

self.accept\_states = accept\_states

def run(self, input\_string):

current\_state = self.start\_state

for char in input\_string:

if char not in self.alphabet:

return False

current\_state = self.transition\_function[current\_state][char]

return current\_state in self.accept\_states

states = {'q0', 'q1', 'q2'}

alphabet = {'0', '1'}

transition\_function = {

'q0': {'0': 'q1', '1': 'q0'},

'q1': {'0': 'q1', '1': 'q2'},

'q2': {'0': 'q1', '1': 'q2'}

}

start\_state = 'q0'

accept\_states = {'q2'}

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

print(dfa.run('01')) # prints True

print(dfa.run('1110')) # prints False

Construction of NFA that accepts substring aba.

class NFA:

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

self.states = states

self.alphabet = alphabet

self.transitions = transitions

self.start\_state = start\_state

self.accept\_states = accept\_states

def accepts(self, string):

current\_states = [self.start\_state]

for char in string:

next\_states = []

for state in current\_states:

if (state, char) in self.transitions:

next\_states.extend(self.transitions[(state, char)])

current\_states = next\_states

return any(state in self.accept\_states for state in current\_states)

# Define the states, alphabet, transitions, start state, and accept states

states = {0, 1, 2, 3}

alphabet = {'a', 'b'}

transitions = {

(0, 'a'): {0,1},

(0, 'b'): {0},

(1, 'b'): {2},

(2, 'a'): {3},

(3, 'a'): {3},

(3, 'b'): {3},

}

start\_state = 0

accept\_states = {3}

# Create an NFA object and test whether it accepts a given string

nfa = NFA(states, alphabet, transitions, start\_state, accept\_states)

print(nfa.accepts('aaabab'))

Socket Programming Paradigm

TCP SOCKET

TCP Server Program

import socket

# Define host and port

HOST = '127.0.0.1'

PORT = 8881

# Create a socket object

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Bind the socket to a specific address and port

s.bind((HOST, PORT))

# Listen for incoming connections

s.listen(1)

print(f'Server is listening on {HOST}:{PORT}...')

while True:

# Wait for a client to connect

conn, addr = s.accept()

print(f'Connected by {addr}')

# Send a welcome message to the client

message = 'Welcome to the server!'

conn.sendall(message.encode())

# Close the connection

conn.close()

TCP client Program:

import socket

# Define host and port

HOST = '127.0.0.1'

PORT = 8881

# Create a socket object

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Connect to the server

s.connect((HOST, PORT))

# Receive the welcome message from the server

message = s.recv(1024)

print(f'Received message: {message.decode()}')

# Close the connection

s.close()

UDP SOCKET

UDP Server Program

import socket

# create a UDP socket

udp\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# bind the socket to a local address and port

local\_address = ('', 12345)

udp\_socket.bind(local\_address)

print('UDP server listening on {}:{}'.format(\*local\_address))

# receive data from clients

while True:

data, address = udp\_socket.recvfrom(1024)

print('Received data from {}:{}'.format(\*address))

print('Data:', data.decode())

# send a response back to the client

message = 'Hello, client!'

udp\_socket.sendto(message.encode(), address)

UDP Client Program

import socket

# create a UDP socket

udp\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# send a message to the server

server\_address = ('localhost', 12345)

message = 'Hello, server!'

udp\_socket.sendto(message.encode(), server\_address)

# receive a response from the server

data, address = udp\_socket.recvfrom(1024)

print('Received data from {}:{}'.format(\*address))

print('Data:', data.decode())

udp\_socket.close()