**TRƯỜNG ĐẠI HỌC CẦN THƠ**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG**

**KHOA KHOA HỌC MÁY TÍNH**

****

**BÁO CÁO HP TIN HỌC LÝ THUYẾT (CT121)**

**ĐỀ TÀI:**

**Xây dựng NFAε và kiểm tra một chuỗi có**

**thuộc NFAε đã cho không**

**GVHD: Cô Phạm Xuân Hiền**

**Sinh viên: Nguyễn Phước Khải (STT: 26)**

**Mã số: B2207531**

**Khóa: K48**

**HK2 2024-2025**

**Cần Thơ, 11/2024**

**NHẬN XÉT CỦA GIẢNG VIÊN**

-----------------------------------------------------------------

-----------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------

*Cần Thơ, ngày tháng năm*

(Ký và ghi rõ họ tên)

**Mục Lục**

[PHẦN 1: TỔNG HỢP CÁC BÀI THỰC HÀNH 4](#_Toc183896568)

[I. Lab 1 4](#_Toc183896569)

[II. Lab 2 7](#_Toc183896570)

[III. Lab 3 11](#_Toc183896571)

[PHẦN 2: PHẦN BÁO CÁO 20](#_Toc183896572)

[I. Cơ sở lý thuyết 20](#_Toc183896573)

[1. Automata hữu hạn (FA: Finite Automata) 20](#_Toc183896574)

[2. Automata hữu hạn không đơn định – NFA (Non-deterministic Finite Automaton) 20](#_Toc183896575)

[3. NFA với ε-dịch chuyển (NFAε) 21](#_Toc183896576)

[II. Mục tiêu bài toán 23](#_Toc183896577)

[1. Xây dựng và mô phỏng một NFAε 23](#_Toc183896578)

[2. Kiểm tra chuỗi với NFAε 23](#_Toc183896579)

[3. Tích hợp với file 23](#_Toc183896580)

[4. Chuyển đổi biểu thức chính quy sang NFAε 23](#_Toc183896581)

[III. Phương pháp thực hiện 24](#_Toc183896582)

[1. Phân tích bài toán và thiết kế cấu trúc dữ liệu: 24](#_Toc183896583)

[2. Cài đặt thuật toán kiểm tra chuỗi với NFAε 24](#_Toc183896584)

[3. Chức năng đọc file dữ liệu NFAε cho trước 26](#_Toc183896585)

[4. Chức năng chuyển regex sang NFAε 26](#_Toc183896586)

[5. Giao diện thực hiện chức năng chương trình 28](#_Toc183896587)

[IV. Thiết kế và cài đặt 29](#_Toc183896588)

[1. Cài đặt các thư viện 29](#_Toc183896589)

[2. Cấu trúc lưu trữ NFAε 30](#_Toc183896590)

[3. Lớp State 31](#_Toc183896591)

[4. Lớp NFAe 31](#_Toc183896592)

[5. Hàm hỗ trợ 34](#_Toc183896593)

[6. Lớp NFAeMenu 39](#_Toc183896594)

[V. Kết quả đạt được 43](#_Toc183896595)

[1. Xây dựng giao diện người dùng hoàn chỉnh: 43](#_Toc183896596)

[2. Xử lý file dữ liệu 44](#_Toc183896597)

[3. Hiển thị sơ đồ trạng thái NFAε 45](#_Toc183896598)

[4. Chức năng kiểm tra chuỗi đầu vào 46](#_Toc183896599)

[VI. Hướng phát triển 48](#_Toc183896600)

[VII. Tài liệu tham khảo 49](#_Toc183896601)

# PHẦN 1: TỔNG HỢP CÁC BÀI THỰC HÀNH

# Lab 1

# 1. Hoán đổi 2 ký tự đầu tiên của 2 chuỗi và nối chúng lại

def cau1(str1, str2):

    lst1 = list(str1)

    lst2 = list(str2)

    temp = lst1[0:2]

    lst1[0:2] = lst2[0:2]

    lst2[0:2] = temp

    return ''.join(lst1)+ ' ' + ''.join(lst2)

print(cau1('abc', 'efg'))

Kết quả: efc abg

# 2. Loại bỏ ký tự tại vị trí chẵn trong chuỗi

def cau2(string):

    return string[1::2]

print(cau2('python'))

Kết quả: yhn

# 3. Đếm số lần xuất hiện của các từ trong câu

from collections import Counter

def cau3(string):

    lst = string.split(' ')

    counter = Counter(lst)

    return dict(counter)

print(cau3('no pain no gain'))

Kết quả: {'no': 2, 'pain': 1, 'gain': 1}

# 4. Mã hóa chuỗi bằng cách dịch trái các ký tự 3 bước

def cau4(ciphertext, shift = 3):

    deciphered\_text = ""

    for char in ciphertext:

        # Check if the character is an uppercase letter

        if char.isupper():

            deciphered\_text += chr((ord(char) - shift - 65) % 26 + 65)

        # Check if the character is a lowercase letter

        elif char.islower():

            deciphered\_text += chr((ord(char) - shift - 97) % 26 + 97)

        else:

            # If it's not a letter, don't change it

            deciphered\_text += char

    return deciphered\_text

print(cau4('def'))

Kết quả: abc

# 5. Kiểm tra chuỗi có hợp lệ không (có được sinh từ bộ ký tự cho trước)

def cau5(string, inputS):

    for c in string:

        if c not in inputS:

            return False

    return True

print(cau5('0011', {'0', '1', '2'}))

print(cau5('123', {'0', '1', '2'}))

Kết quả:

True

False

# 6. Chuyển chuỗi thành danh sách các từ

def cau6(string):

    return string.split(' ')

print(cau6('This is a list'))

Kết quả: ['This', 'is', 'a', 'list']

# 7. In ra ký tự không lặp lại đầu tiên trong chuỗi

def cau7(string):

    counter = Counter(string)

    for k, v in counter.items():

        if v == 1:

            return k

    return None

print(cau7('abcdef'))

print(cau7('abcabcdef'))

print(cau7('aabbbcc'))

Kết quả:

a

d

None

# 8. Loại bỏ khoảng trắng trong chuỗi

def cau8(string):

    return string.replace(' ', '')

print(cau8('a b c'))

Kết quả: abc

# 9. In ra từ được lặp lại đầu tiên trong chuỗi

def cau9(string):

    words = string.split()

    seen\_words = set()

    for word in words:

        if word in seen\_words:

            return word

        seen\_words.add(word)

    return None

print(cau9('ab ca bc ca ab bc'))

print(cau9('ab ca bc ab'))

Kết quả:

ca

ab

# 10. Tìm độ dài tối đa của chuỗi con gồm các số 0 liên tiếp

def cau10(binary\_string):

    # Tách chuỗi nhị phân dựa trên '1' để lấy các chuỗi con toàn là '0'

    zero\_groups = binary\_string.split('1')

    # Tìm chuỗi '0' có độ dài lớn nhất

    max\_zeros = max(len(group) for group in zero\_groups)

    return max\_zeros

print(cau10('1110001110000'))

Kết quả: 4

#Bài tập nâng cao 1: Kiểm tra 2 danh sách khác nhau có thể sinh ra cùng một chuỗi hay không?

def cau1NangCao(lstA, lstB):

    permA = permutations(lstA)

    permB = permutations(lstB)

    setA = {''.join(x) for x in permA}

    setB = {''.join(x) for x in permB}

    if setA & setB:

        return True

    else:

        return False

print(cau1NangCao(['110', '0011', '0110'], ['110110', '00', '110']))

print(cau1NangCao(['0011', '11', '1101'], ['101', '011', '110']))

print(cau1NangCao(['100', '0', '1'], ['1', '100', '0']))

Kết quả:

True

False

True

#Bài tập nâng cao 2

def cau2NangCao(genString):

    for c in genString:

        if c not in {'A', 'C', 'G', 'T'}:

            return False

    if len(genString) % 3 != 0:

        return False

    else:

        start\_codon = 'ATG'

        end\_codon = {'TAA', 'TAG', 'TGA'}

        condonList = split\_into\_chunks(genString, 3)

        if condonList[0] == start\_codon and condonList[-1] in end\_codon:

            for condon in condonList[1:-1]:

                    if len(set(condon)) == 1:

                        return False

            return True

        else:

            return False

print(cau2NangCao('ATGCCCTAG'))

print(cau2NangCao('ATGCGTTGA'))

Kết quả:

False

True

# Lab 2

#Câu 1: Chỉnh sửa chương trình cho phép nhận DFA từ file hoặc do người dùng nhập vào, và

#kiểm tra một chuỗi nhập từ bàn phím có thuộc DFA đã cho hay không?

# Xây dựng DFA

class DFA(object):

    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

        self.current\_state = start\_state

    def transition\_to\_state\_with\_input(self, input\_value):

        if ((self.current\_state, input\_value) not in self.transition\_function.keys()):

            self.current\_state = None

            return

        self.current\_state = self.transition\_function[(self.current\_state, input\_value)]

        return

    def in\_accept\_state(self):

        return self.current\_state in self.accept\_states

    def go\_to\_initial\_state(self):

        self.current\_state = self.start\_state

        return

    def run\_with\_input\_list(self, input\_list):

        self.go\_to\_initial\_state()

        for inp in input\_list:

            self.transition\_to\_state\_with\_input(inp)

        return self.in\_accept\_state()

def dfa\_file(filename):

    with open(filename, 'r') as f:

        L = f.readlines()

        states = set(map(int, L[0].split(',')))

        alphabet = set(L[1].strip().split(','))

        start\_state = int(L[2])

        accept\_states = set(map(int, L[3].split(',')))

        tf = {}

        for line in L[4:]:

            part = line.strip().split(',')

            state\_from = int(part[0])

            char = part[1]

            state\_to = int(part[2])

            tf[(state\_from, char)] = state\_to

        return DFA(states, alphabet, tf, start\_state, accept\_states)

L = list('1011101')

dfa1 = dfa\_file(r"./DFA\_sample.txt")

dfa1.go\_to\_initial\_state()

print(dfa1.run\_with\_input\_list(L))

file DFA\_sample.txt

0,1,2

0,1

0

0

0,0,1

0,1,1

1,0,2

1,1,0

2,0,1

2,1,2

Kết quả: True

#Câu 2:Dựa trên DFA đã xây dựng, mở rộng xây dựng lớp NFA và kiểm tra 1 chuỗi có thuộc ngôn ngữ sinh bởi NFA đã cho

# Xây dựng NFA

class NFA():

    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

        self.current\_state:list = self.start\_state

    def transition\_to\_state\_with\_input(self, input\_value):

        print("state: ", self.current\_state)

        temp\_state = []

        for c\_state in self.current\_state:

            if (c\_state, input\_value) in self.transition\_function.keys():

                state = self.transition\_function[(c\_state, input\_value)]

                if isinstance(state, (int)):

                    if state not in temp\_state:

                        temp\_state.append(state)

                else:

                    temp\_state += state

        if len(temp\_state) == 0:

            self.current\_state = []

        else:

            self.current\_state = temp\_state.copy()

        return

    def in\_accept\_state(self):

        for c\_state in self.current\_state:

            if c\_state in self.accept\_states:

                return True

        return False

    def go\_to\_initial\_state(self):

        self.current\_state = self.start\_state

        return

    def run\_with\_input\_list(self, input\_list):

        self.go\_to\_initial\_state()

        for inp in input\_list:

            self.transition\_to\_state\_with\_input(inp)

        return self.in\_accept\_state()

states = {0, 1, 2, 3 ,4}

alphabet = {'0', '1'}

start\_state = [0]

accept\_states = {2, 4}

tf = dict()

tf[(0, '0')] = [0, 3]

tf[(0, '1')] = [0, 1]

tf[(1, '1')] = 2

tf[(2, '0')] = 2

tf[(2, '1')] = 2

tf[(3, '0')] = 4

tf[(4, '0')] = 4

tf[(4, '1')] = 4

L = list('10111011')

nfa1 = NFA(states, alphabet, tf, start\_state, accept\_states)

print(nfa1.run\_with\_input\_list(L))

# Lab 3

import re

#file test.txt

this is a test with re

This line has exactly twenty char

this should meet the required minimum length?!

Do you agree?!

Some random words a, r, s, m, l appear here.

No commas or periods here just words

A mouse is in the house

aaaaabbbb test

ambition arises amidst amazing abundance

myemail@gmail.com

hello@domain.com, user@yahoo.com, support@outlook.com

#Cau A

#Cau A1: Các dòng bắt đầu bằng ‘t’ hoặc ‘h’, và có chứa ‘re’ (sử dụng phương thức re.match())

with (open('test.txt') as file):

    for line in file.readlines():

        line = line.strip()

        # 1)

        result = re.match(r'^[th].\*re', line)

        if result:

            print(line)

Kết quả:

this is a test with re

this should meet the required minimum length?!

#Cau A2: Các dòng có chiều dài tối thiểu 20 kí tự

with (open('test.txt') as file):

    for line in file.readlines():

        line = line.strip()

        # 2)

        result = re.match(r'.{20,}', line)

        if result:

            print(line)

Kết quả

this is a test with re

This line has exactly twenty char

this should meet the required minimum length?!

Some random words a, r, s, m, l appear here.

No commas or periods here just words

A mouse is in the house

ambition arises amidst amazing abundance

hello@domain.com, user@yahoo.com, support@outlook.com

#Cau A3: Các dòng kết thúc bởi cặp dấu ‘?!’

with (open('test.txt') as file):

    for line in file.readlines():

        line = line.strip()

        # 3)

        result = re.search(r'\?!$', line)

        if result:

            print(line)

Kết quả

this should meet the required minimum length?!

Do you agree?!

#Cau A4: Các dòng chứa những ký tự: a, r, s, m, l (không cần liên tục)

with (open('test.txt') as file):

    for line in file.readlines():

        line = line.strip()

        # 4)

        result = re.search(r'^(?=.\*a)(?=.\*r)(?=.\*s)(?=.\*m)(?=.\*l).\*', line)

        if result:

            print(line)

Kết quả

Some random words a, r, s, m, l appear here.

hello@domain.com, user@yahoo.com, support@outlook.com

#Cau A5: Nội dung file ko chứa các dấu ‘,’ và ‘.’

with (open('test.txt') as file):

    for line in file.readlines():

        line = line.strip()

        # 5)

        result = re.search(r'[,.]', line)

        if not result:

            print(line)

Kết quả:

this is a test with re

This line has exactly twenty char

this should meet the required minimum length?!

Do you agree?!

No commas or periods here just words

A mouse is in the house

aaaaabbbb test

ambition arises amidst amazing abundance

#Cau A6: Các dòng có chứa chữ “mouse”

with (open('test.txt') as file):

    for line in file.readlines():

        line = line.strip()

        # 6)

        result = re.search(r'mouse', line)

        if result:

            print(line)

Kết quả:

A mouse is in the house

#Cau A7: Các từ có số chữ ‘a’ bất kỳ và theo sau bởi ‘b’

with (open('test.txt') as file):

    for line in file.readlines():

        line = line.strip()

        # 7)

        result = re.search(r'a+b', line)

        if result:

            print(line)  
Kết quả:

aaaaabbbb test

ambition arises amidst amazing abundance

#Cau A8: Domain địa chỉ mail (ví dụ: abc@gmail.com, in gmail.com )

with (open('test.txt') as file):

    for line in file.readlines():

        line = line.strip()

        # 8)

        result = re.search(r'@([\w.-]+\.[a-zA-Z]{2,})\b', line)

        if result:

            print(line)

Kết quả

myemail@gmail.com

hello@domain.com, user@yahoo.com, support@outlook.com

#Cau A9: Nội dung giữa cặp tag <head> </head>

# 9)

test\_str = "<head>This is a sample text within head tags.</head>"

result = re.search(r'<head>(.\*?)</head>', test\_str)

print(result.group(1))

#Cau B

import re

def is\_valid\_sentence(sentence):

    # 1) Câu được bắt đầu bằng ký tự in hoa, theo sau bởi ký tự thường

    if not re.search(r"^[A-Z][a-z]", sentence):

        return False

    # 2) Kết thúc bằng dấu chấm hoặc sau một ký tự in hoa

    if not re.search(r"[A-Za-z]\.$|[A-Z]$", sentence):

        return False

    # 3) Các từ cách nhau bằng một khoảng trắng, không chấp nhận nhiều hơn 1 khoảng trắng liên tiếp

    if re.search(r"\s{2,}", sentence):

        return False

    # 4) Không tồn tại hai ký tự liên tiếp viết hoa

    if re.search(r"[A-Z]{2,}", sentence):

        return False

    return True

# Ví dụ sử dụng:

sentences = [

    "Hello world.",

    "bat dau bang ky tu in thuong.",

    "This is a Test sentence.",

    "Ket thuc la mot ky tu in hoA",

    "Incorrect  sentence due to  extra spaces.",

    "AnotherIncorrectSentence",

    "Ends with period.",

    "NO TWO uppercase letters"

]

for sentence in sentences:

    print(f"\"{sentence}\": {is\_valid\_sentence(sentence)}")

Kết quả:

"Hello world.": True

"bat dau bang ky tu in thuong.": False

"This is a Test sentence.": True

"Ket thuc la mot ky tu in hoA": True

"Incorrect sentence due to extra spaces.": False

"AnotherIncorrectSentence": False

"Ends with period.": True

"NO TWO uppercase letters": False

#Cau C

import re

def process\_text(text):

    words = text.split()

    # Các từ thỏa mãn từng điều kiện

    results = {

        'contains\_a\_and\_digit': [],

        'contains\_a\_followed\_by\_b': [],

        'starts\_with\_a\_and\_ends\_with\_b': [],

        'contains\_only\_a\_to\_z\_and\_underscore': [],

        'length\_is\_5': [],

        'contains\_h': [],

        'starts\_with\_digit': [],

        'contains\_underscore': [],

        'date\_format\_conversion': []

    }

    for word in words:

        # 1. Các từ có chứa các ký tự thường ‘a-z’ và số từ ‘0-9’

        if re.search(r'[a-z].\*\d|[0-9].\*[a-z]', word):

            results['contains\_a\_and\_digit'].append(word)

        # 2. Các từ có chứa ký tự ‘a’ theo sau bởi b (b xuất hiện ít nhất 0 lần)

        if re.search(r'a\*b', word):

            results['contains\_a\_followed\_by\_b'].append(word)

        # 3. Các từ bắt đầu bằng ‘a’, theo sau là ký tự bất kỳ và kết thúc bằng ‘b’

        if re.match(r'^a.\*b$', word):

            results['starts\_with\_a\_and\_ends\_with\_b'].append(word)

        # 4. Các từ chỉ chứa ký tự thường ‘a-z’ và ‘\_’

        if re.match(r'^[a-z\_]+$', word):

            results['contains\_only\_a\_to\_z\_and\_underscore'].append(word)

        # 5. Các từ có chiều dài là 5

        if re.match(r'\b\w{5}\b', word):

            results['length\_is\_5'].append(word)

        # 6. Các từ có chứa ký tự ‘h’

        if 'h' in word:

            results['contains\_h'].append(word)

        # 7. Các từ bắt đầu là số từ ‘0-9’

        if re.match(r'^\d', word):

            results['starts\_with\_digit'].append(word)

        # 8. Các từ có chứa dấu ‘\_’ và thay bằng khoảng trắng

        if '\_' in word:

            results['contains\_underscore'].append(word.replace('\_', ' '))

        # 9. Có chứa định dạng mm-dd-yy, và chuyển thành định dạng dd-mm-yy

        if re.match(r'\d{2}-\d{2}-\d{2}', word):

            new\_date = re.sub(r'(\d{2})-(\d{2})-(\d{2})', r'\2-\1-\3', word)

            results['date\_format\_conversion'].append(new\_date)

    return results

# Ví dụ sử dụng:

text = "abc123 abc bbb abb a\_z 12345 hello 11-06-24 a\_ b\_ h h\_ hello\_123"

result = process\_text(text)

# In kết quả

for key, value in result.items():

    print(f"{key}: {value}")

Kết quả:

contains\_a\_and\_digit: ['abc123', 'hello\_123']

contains\_a\_followed\_by\_b: ['abc123', 'abc', 'bbb', 'abb', 'b\_']

starts\_with\_a\_and\_ends\_with\_b: ['abb']

contains\_only\_a\_to\_z\_and\_underscore: ['abc', 'bbb', 'abb', 'a\_z', 'hello', 'a\_', 'b\_', 'h', 'h\_']

length\_is\_5: ['12345', 'hello']

contains\_h: ['hello', 'h', 'h\_', 'hello\_123']

starts\_with\_digit: ['12345', '11-06-24']

contains\_underscore: ['a z', 'a ', 'b ', 'h ', 'hello 123']

date\_format\_conversion: ['06-11-24']

#Cau D

#Cau D1: Hoán đổi 2 ký tự đầu tiên của 2 chuỗi và nối chúng lại

def hoandoi(str1, str2):

# Sử dụng re.sub để hoán đổi hai ký tự đầu tiên

return re.sub(r'^(..)', lambda m: str2[:2], str1) + " " + re.sub(r'^(..)', lambda m: str1[:2], str2)

print(hoandoi('abc', 'efg'))

#Cau D2: Loại bỏ ký tự tại vị trí chẵn trong chuỗi

def loaibokytuchan(s):

return re.sub(r'(.)', lambda m: m.group(1) if m.start() % 2 != 0 else '', s)

print(loaibokytuchan('python'))

#Cau D3: Đếm số lần xuất hiện của các từ trong câu

def demsotu(cau):

tu = re.findall(r'\b\w+\b', cau)

tu\_dem = {}

for word in tu:

tu\_dem[word] = tu\_dem.get(word, 0) + 1

return tu\_dem

print(demsotu('no pain no gain'))

#Cau D4: Mã hóa chuỗi bằng cách dịch trái các ký tự 3 bước

def dichchuyenCeasar(s):

return re.sub(r'[a-z]', lambda m: chr((ord(m.group(0)) - 97 - 3) % 26 + 97), s)

print(dichchuyenCeasar('def'))

#Cau D5: Kiểm tra chuỗi có hợp lệ không (có được sinh từ bộ ký tự cho trước)

def chuoihople(s, valid\_set):

valid\_characters = ''.join(valid\_set)

return bool(re.fullmatch(f'[{valid\_characters}]+', s))

print(chuoihople('0011', {'0', '1', '2'}))

print(chuoihople('123', {'0', '1', '2'}))

#Cau D6: Chuyển chuỗi thành danh sách các từ

def stringtolist(s):

return re.findall(r'\b\w+\b', s)

print(stringtolist('This is a list'))

#Cau D7: In ra ký tự không lặp lại đầu tiên trong chuỗi

from collections import Counter

def inkhonglap(s):

# Sử dụng biểu thức chính quy để tìm tất cả các ký tự

chars = re.findall(r'.', s)

# Đếm số lần xuất hiện của mỗi ký tự

char\_count = Counter(chars)

# Duyệt qua các ký tự và tìm ký tự không lặp lại đầu tiên

for char in chars:

if char\_count[char] == 1:

return char

return None

print(inkhonglap('abcdef'))

print(inkhonglap('abcabcdef'))

print(inkhonglap('aabbcc'))

#Cau D8: Loại bỏ khoảng trắng trong chuỗi

def loaibospace(s):

return s.replace(' ', '')

print(loaibospace('a b c'))

#Cau D9:

def inlap(s):

# Tìm tất cả các từ trong chuỗi

words = re.findall(r'\b\w+\b', s)

seen = set()

# Duyệt qua các từ và tìm từ lặp lại đầu tiên

for word in words:

if word in seen:

return word

seen.add(word)

return None

print(inlap('ab ca bc ca ab bc')) # Kết quả: 'ca'

print(inlap('ab ca bc ab')) # Kết quả: 'ab'

#Cau D10: Tìm độ dài tối đa của chuỗi con gồm các số 0 liên tiếp

def chuoiconso0(bin\_str):

zeros = re.findall(r'0+', bin\_str)

if zeros:

# Tính độ dài của chuỗi con dài nhất

return max(len(zeros\_str) for zeros\_str in zeros)

return 0

print(chuoiconso0('1110001110000'))

# PHẦN 2: PHẦN BÁO CÁO

## Cơ sở lý thuyết

### Automata hữu hạn (FA: Finite Automata)

Automata hữu hạn – FA là một mô hình tỉnh toán của hệ thống với sự mô tả bởi các input và output. Tại mỗi thời điểm, hệ thống có thể được xác định ở một trong số hữu hạn các cấu hình nội bộ gọi là các trạng thái (states). Mỗi trạng thái của hệ thống thể hiện sự tóm tắt các thông tin liên quan đến những input đã chuyển qua và xác định các phép chuyển kế tiếp trên dãy input tiếp theo.

### Automata hữu hạn không đơn định – NFA (Non-deterministic Finite Automaton)

Automata hữu hạn không đơn định – NFA là một máy Automata hữu hạn trong đó chấp nhận không, một hoặc nhiều hơn một phép chuyển từ cụng một trạng thái trên cùng một ký hiệu nhập.

Một NFA được định nghĩa bởi bộ 5 thành phần:

Trong đó:

* **:** Tập hợp hữu hạn các trạng thái.
* **:** Tập hợp hữu hạn các ký tự (bảng chữ cái đầu vào).
* **:** Hàm chuyển trạng thái, với:
* **:** Trạng thái bắt đầu .
* **:** Tập các trạng thái kết thúc .

### NFA với ε-dịch chuyển (NFAε)

Trong lý thuyết Automata, **NFAε** (Non-deterministic Finite Automaton with epsilon transitions) là một loại máy tự động hữu hạn không đơn định, được mở rộng với khả năng chuyển trạng thái thông qua **epsilon transitions** (ký hiệu là ε). Epsilon transition cho phép chuyển từ một trạng thái này sang trạng thái khác mà không cần đọc bất kỳ ký tự nào từ chuỗi đầu vào.

#### 3.1. Định nghĩa và Cấu trúc của NFAε

Một NFAε được định nghĩa bởi bộ 5 thành phần:

Trong đó:

* : Tập hợp hữu hạn các trạng thái.
* : Tập hợp hữu hạn các ký tự (bảng chữ cái đầu vào).
* : Hàm chuyển trạng thái, với:
* : Trạng thái bắt đầu .
* : Tập các trạng thái kết thúc .

Hàm chuyển trạng thái trong NFAε khác với NFA ở chỗ nó có thể nhận một ký tự từ tập **Σ** hoặc **ε** (epsilon). Một chuyển đổi epsilon có nghĩa là từ một trạng thái, máy có thể chuyển sang trạng thái khác mà không cần phải đọc bất kỳ ký tự nào từ chuỗi đầu vào.

#### 3.2. Hàm chuyển trạng thái (Transition Function)

Trong NFAε, hàm chuyển trạng thái có dạng:

Điều này có nghĩa là hàm chuyển trạng thái nhận vào một cặp (trạng thái, ký tự đầu vào) và trả về một tập hợp các trạng thái có thể đạt được.

Ví dụ:

* Nếu máy đang ở trạng thái **q₁** và đọc ký tự **a**, nó có thể chuyển đến một hoặc nhiều trạng thái, chẳng hạn như **q₂** và **q₃**.
* Nếu máy đang ở trạng thái **q₁** và đọc ký tự **ε**, nó có thể chuyển đến trạng thái **q₂** mà không cần đọc thêm ký tự nào.

#### 3.3. Cách thức hoạt động của NFAε

Khi kiểm tra một chuỗi đầu vào, NFAε không chỉ theo dõi một chuỗi các trạng thái mà nó có thể chuyển qua, mà còn theo dõi tất cả các khả năng chuyển trạng thái mà nó có thể đạt được, bao gồm cả qua các chuyển đổi epsilon.

* Bắt đầu tại trạng thái ban đầu: NFAε bắt đầu từ trạng thái ban đầu q₀.
* Duyệt qua chuỗi đầu vào:
  + Với mỗi ký tự trong chuỗi đầu vào, NFAε sẽ tính toán các trạng thái mà nó có thể chuyển đến thông qua các chuyển đổi epsilon (nếu có) và các chuyển đổi thông thường.
  + Nếu trong một bước, có nhiều khả năng chuyển trạng thái, ta sẽ "thử" tất cả các khả năng này cùng lúc.
* Kiểm tra trạng thái kết thúc: Sau khi đọc hết chuỗi đầu vào, nếu NFAε kết thúc tại một trạng thái thuộc F (tập trạng thái kết thúc), thì chuỗi đầu vào được coi là hợp lệ. Nếu không, chuỗi bị từ chối.

#### 3.4. ε-closure

Một khái niệm quan trọng trong NFAε là ε-closure.

ε-closure của một trạng thái q là tập hợp tất cả các trạng thái mà từ q có thể đạt được qua các chuyển đổi epsilon, bao gồm cả chính nó.

ε-closure(q) = Tập hợp các trạng thái có thể đạt được từ trạng thái q thông qua các chuyển đổi epsilon.

**Ví dụ**:

Nếu từ trạng thái q₁ có chuyển đổi epsilon đến q₂, và từ q₂ lại có chuyển đổi epsilon đến q₃, thì ε-closure của q₁ sẽ là {q₁, q₂, q₃}.

## Mục tiêu bài toán

Mục tiêu của bài toán này là xây dựng một **demo/giao diện** cho việc xây dựng và kiểm tra chuỗi đầu vào của một NFAε (NFA với ε-dịch chuyển), giúp người dùng có thể tạo và chỉnh sửa một NFAε, hỗ trợ phát thảo sơ đồ NFAε, sau đó kiểm tra một chuỗi đầu vào xem có thuộc ngôn ngữ mà NFAε nhận diện hay không.

Cụ thể, bài toán này có các mục tiêu chính như sau:

### 1. Xây dựng và mô phỏng một NFAε

* Tạo ra một công cụ để người dùng có thể xây dựng NFAε từ các thành phần cơ bản: các trạng thái, ký tự đầu vào, hàm chuyển trạng thái (bao gồm các chuyển đổi epsilon).
* Giao diện cho phép người dùng sử dụng một NFAε do mình tự tạo một cách dễ dàng thông qua việc đọc file.
* Hỗ trợ người dung vẽ sơ đồ NFAε của mình thông qua các thư viện có sẵn.

### 2. Kiểm tra chuỗi với NFAε

* Cung cấp một chức năng để người dùng kiểm tra một chuỗi có thuộc ngôn ngữ của NFAε được người dung cho trước hay không.
* Xác định xem máy có thể chuyển từ trạng thái ban đầu đến trạng thái chấp nhận khi đọc chuỗi đầu vào (bao gồm các ký tự đầu vào và các bước epsilon).

### 3. Tích hợp với file

* Hỗ trợ đọc và ghi dữ liệu từ file. Người dùng có thể lưu sẵn một NFAε vào một file và sau đó tải lên giao diện để vẽ sơ đồ NFAε, đồng thời kiểm tra chuỗi đầu vào tương ứng.
* Cung cấp một cách dễ dàng để nhập dữ liệu NFAε và chuỗi từ file và thực hiện kiểm tra mà không cần nhập liệu thủ công qua giao diện.

### 4. Chuyển đổi biểu thức chính quy sang NFAε

* Cho phép người dùng nhập biểu thức chính quy và kiểm tra chuỗi có thuộc biểu thức chính quy hay không bằng cách qua NFAε.

## Phương pháp thực hiện

Để thực hiện bài toán "Xây dựng NFAε và kiểm tra chuỗi có thuộc NFAε hay không", chúng ta sẽ thực hiện theo các bước chính sau:

### 1. Phân tích bài toán và thiết kế cấu trúc dữ liệu:

Trước tiên, chúng ta sẽ tiến hành phân tích và thiết kế các cấu trúc dữ liệu cần thiết để mô phỏng NFAε. Cấu trúc dữ liệu bao gồm các thành phần chính sau:

* **Trạng thái (States)**: Đây là tập hợp các trạng thái của NFAε. Một trạng thái có thể là trạng thái bắt đầu (start state) hoặc trạng thái chấp nhận (accept state), và có thể bao gồm nhiều trạng thái chấp nhận trong NFAε. Mỗi trạng thái sẽ được đại diện bằng một chuỗi ký tự hoặc số, giúp dễ dàng tham chiếu và thao tác.
* **Ký tự đầu vào (Alphabet)**: Đây là tập hợp các ký tự mà NFAε có thể nhận diện. Ví dụ: Tập hợp các ký tự có thể là {a, b, c} trong ngôn ngữ chuỗi. Ký tự đầu vào giúp NFAε nhận diện chuỗi đầu vào. Trong phần này e sẽ được dung làm đại diện cho ε nhằm đồng bộ.
* **Trạng thái bắt đầu (Start State)**: Đây là trạng thái mà bắt đầu khi thực hiện kiểm tra chuỗi đầu vào. Trong NFAε, trạng thái bắt đầu là điểm xuất phát và có thể chuyển qua nhiều trạng thái khác thông qua các chuyển đổi epsilon hoặc các ký tự đầu vào.
* **Tập rạng thái kết thúc (Accept States)**: Đây là các trạng thái mà NFAε có thể kết thúc khi chuỗi đầu vào được xử lý thành công. Nếu sau khi đọc toàn bộ chuỗi mà kết thúc ở một trong các trạng thái này, thì chuỗi được xem là thuộc ngôn ngữ của NFAε.
* **Hàm chuyển trạng thái (Transitions)**: Đây là một bảng hoặc đồ thị mô tả các chuyển đổi giữa các trạng thái trong NFAε. Điều đặc biệt ở NFAε là mỗi chuyển đổi có thể là epsilon (ε), tức là không cần đọc ký tự đầu vào mà vẫn có thể chuyển từ trạng thái này sang trạng thái khác. Do đó, hàm chuyển trạng thái sẽ phải xử lý cả chuyển đổi ký tự đầu vào và epsilon (ε).

### 2. Cài đặt thuật toán kiểm tra chuỗi với NFAε

Sau khi thiết kế cấu trúc dữ liệu, bước tiếp theo là xây dựng thuật toán kiểm tra chuỗi đầu vào xem có thuộc ngôn ngữ mà NFAε nhận diện hay không. Việc này bao gồm:

* **Xử lý ε-closure:** Trong NFAε, ε-closure là dùng để xác định tất cả các trạng thái có thể chuyển đến từ một trạng thái bất kỳ mà không cần đọc ký tự đầu vào. Điều này đặc biệt quan trọng vì NFAε có thể chuyển từ trạng thái này sang trạng thái khác chỉ qua các chuyển đổi epsilon(ε). Thuật toán ε-closure sẽ trả về tất cả các trạng thái có thể đạt được từ trạng thái hiện tại thông qua các epsilon transition.
  + **Khởi tạo ngăn xếp và tập closure:** Đặt trạng thái ban đầu vào ngăn xếp và thêm nó vào tập closure. Đây là bước khởi đầu, nơi ta thiết lập ngăn xếp và khởi tạo một tập hợp chứa trạng thái đầu tiên.
  + **Duyệt qua các trạng thái:** Lặp qua từng trạng thái trong ngăn xếp. Mỗi lần lấy ra một trạng thái từ ngăn xếp, ta kiểm tra nó và kiểm tra các chuyển đổi epsilon có thể xảy ra.
  + **Kiểm tra các chuyển đổi epsilon:** Duyệt qua tất cả các chuyển đổi epsilon từ trạng thái hiện tại, thêm các trạng thái đích vào ngăn xếp nếu chưa được thêm vào tập closure.
  + **Trả về kết quả:** Khi ngăn xếp trống, trả về tập closure đã xây dựng, đó là tập hợp các trạng thái có thể đạt được từ trạng thái ban đầu thông qua chuyển đổi epsilon.
* **Các bước di chuyển:** các trạng thái có thể chuyển đến từ một hoặc nhiều trạng thái hiện tại khi đọc một ký tự đầu vào hoặc epsilon. Đây là một phần quan trọng trong việc kiểm tra chuỗi đầu vào trong NFAε, vì nó giúp xác định các lựa chọn mà máy có thể thực hiện ở mỗi bước trong quá trình xử lý chuỗi.
  + **Khởi tạo tập trạng thái tiếp theo**: Tạo một tập hợp rỗng **next\_states** để chứa các trạng thái mà tự động có thể chuyển đến khi đọc ký tự đầu vào từ một tập các trạng thái đã cho.
  + **Duyệt qua các trạng thái trong tập đã cho**: Lặp qua từng trạng thái trong tập trạng thái đã cho. Với mỗi trạng thái, ta kiểm tra xem có tồn tại chuyển đổi với ký tự đầu vào **char** hay không.
  + **Kiểm tra các chuyển đổi và cập nhật trạng thái tiếp theo**: Nếu có chuyển đổi từ trạng thái hiện tại tới các trạng thái mới khi đọc ký tự **char**, ta thêm tất cả các trạng thái mới này vào tập **next\_states.**
  + **Trả về kết quả**: Sau khi duyệt qua tất cả các trạng thái trong tập đầu vào, hàm sẽ trả về tập **next\_states**, đó là tập hợp các trạng thái mà tự động có thể chuyển đến sau khi đọc ký tự đầu vào từ các trạng thái ban đầu.
* **Kiểm tra chấp nhận chuỗi:** thực hiện việc kiểm tra xem một chuỗi đầu vào có được NFAε chấp nhận hay không. Quá trình này bao gồm việc bắt đầu từ trạng thái ban đầu và tính toán tất cả các trạng thái có thể đạt được qua các chuyển đổi epsilon. Kiểm tra xem có ít nhất một trạng thái chấp nhận nào được đạt đến sau khi xử lý xong chuỗi, từ đó đưa ra kết luận chuỗi có được chấp nhận hay không.
  + **Khởi tạo trạng thái ban đầu**: Tính toán ε-closure của trạng thái bắt đầu và lưu vào **current\_states**.
  + **Duyệt qua từng ký tự trong chuỗi**: Lặp qua từng ký tự trong chuỗi, hiển thị thông tin về ký tự đang xử lý.
  + **Tính toán các trạng thái tiếp theo**: Với mỗi trạng thái trong **current\_states**, tính toán các trạng thái tiếp theo từ chuyển đổi với ký tự hiện tại, sau đó cập nhật **next\_states**.
  + **Cập nhật trạng thái hiện tại**: Tính toán ε-closure của các trạng thái trong **next\_states** và cập nhật **current\_states**.
  + **Kiểm tra kết quả**: Kiểm tra xem **current\_states** có chứa bất kỳ trạng thái chấp nhận nào. Nếu có, chuỗi được chấp nhận, ngược lại bị từ chối.

### 3. Chức năng đọc file dữ liệu NFAε cho trước

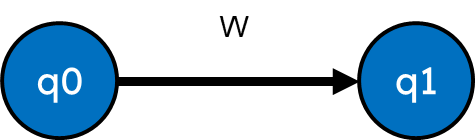
Chức năng này hướng đến việc xử lý file đầu vào NFAε làm cơ sở dữ liệu để xây dựng chương trình, chúng chịu trách nhiệm đọc cấu trúc và dữ liệu từ file. Từ các dữ liệu này, chương trình sẽ có thể khởi tạo được đối tượng với đầy đủ các thông tin phù hợp với cấu trúc dữ liệu được thiết lập.

* **Mở và đọc file đầu vào:** file được mở ở chế độ đọc và tất cả các dòng được lưu vào danh sách **lines**.
* **Tuân thủ quy tắc cấu trúc dữ liệu cho trước:** file được chia thành các dòng bao gồm:
* **States:** dòng thứ nhất, tức lines[0], chứa các trạng thái.
* **Alphabet:** dòng thứ hai, tức lines[1], chứa các ký tự đầu vào và epsilon.
* **Start:** dòng thứ ba, tức lines[2], chứa trạng thái bắt đầu.
* **Accept:** dòng thứ tư, tức lines[3], chứa trạng thái kết thúc.
* **Transition:** các dòng từ dòng số năm trở đi, tức lines[5:], chứa các chuyển đổi.

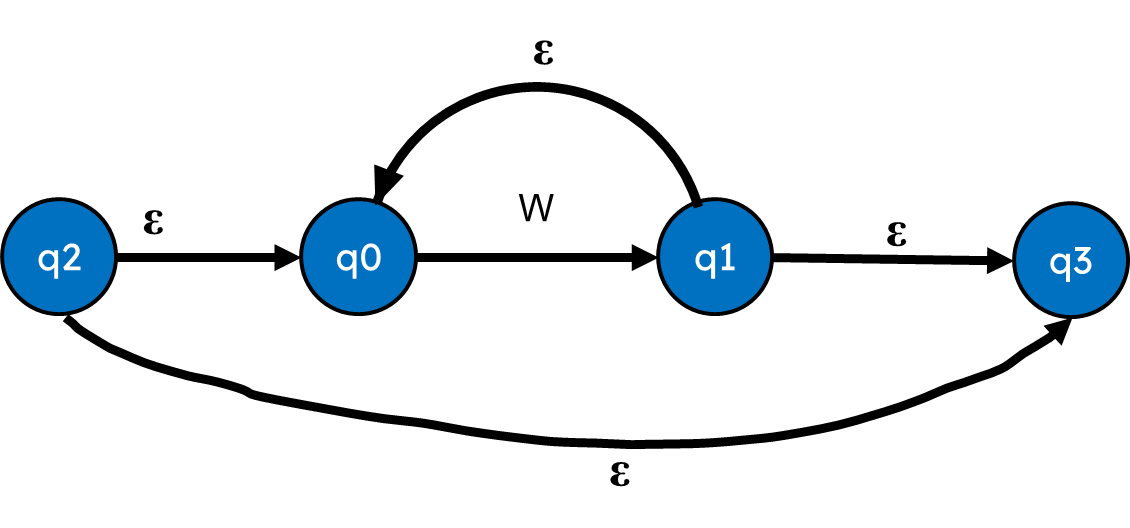
### 4. Chức năng chuyển regex sang NFAε

Sau khi người dùng nhập regex và xác nhận, chương trình sẽ khởi tạo NFAε và cho phép thực hiện kiểm tra chuỗi.

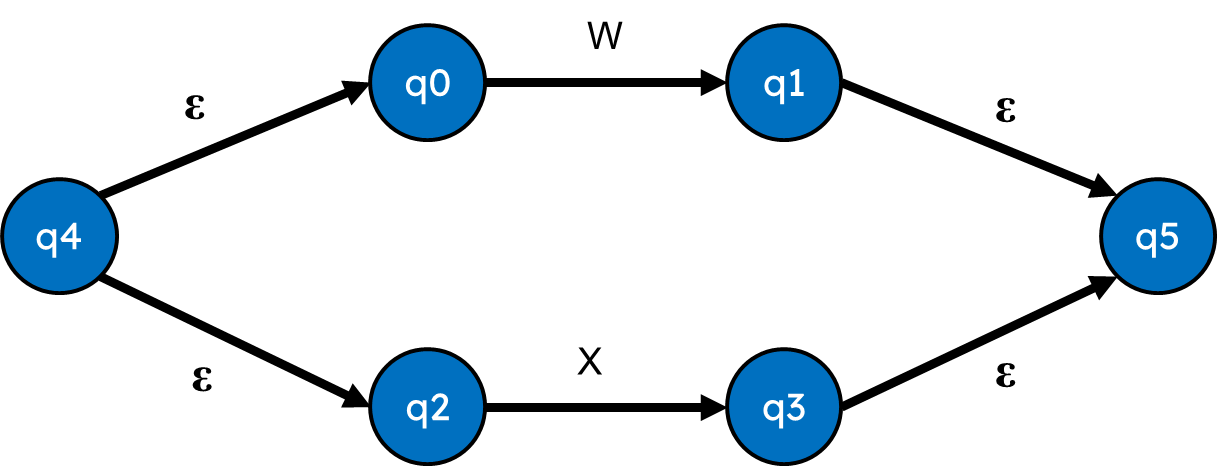
* **Thêm ký tự nối kết vào regex**: Duyệt qua từng ký tự trong regex và thêm ký tự nối kết ‘.’.
  + Ví dụ:
    - abc → a.b.c
    - 01\*|1 → 0.1\*|1
* **Chuyển đổi regex từ infix sang postfix:**
  + Ta có độ ưu tiên tăng dần của các toán tử là: ‘|’ , ‘.’, ‘\*’
  + Khởi tạo ta có một mảng output và một stack rỗng – stack có nhiệm vụ lưu các toán tử và cặp dấu ‘(’, ‘)’
  + Duyệt qua từng ký tự trong regex:
    - Nếu là ký tự nhập, ta push vào output.
    - Nếu gặp toán tử ‘|’ , ‘.’, ‘\*’:
      * Thực viện vòng lặp kiểm tra, nếu ở đỉnh stack là toán tử, mà nó có độ ưu tiên **lớn hơn hoặc bằng** toán tử hiện tại thì ta lấy toán tử đó ra (pop) khỏistack và push vào output.
      * Push toán tử hiện tại vào stack.
    - Nếu gặp dấu ‘(’, ta push vào stack.
    - Nếu gặp dấu  ‘)’, ta thực hiện vòng lặp lấy các toán tử trong stack và push vào output cho đến khi gặp dấu ‘(’ (ta pop luôn cả dấu ‘(’ ra khỏi stack).
  + Sau khi kết thúc vòng lặp, nếu trong stack vẫn còn toán tử, ta sẽ push hết vào mảng output.
  + Ví dụ:
    - 0|1 → 01|
    - 0.1\*|1 → 01\*.1|
* **Chuyển đổi regex ở dạng postfix sang NFAε:** Duyệt qua từng ký tự trong regex:
  + Nếu là ký tự nhập:



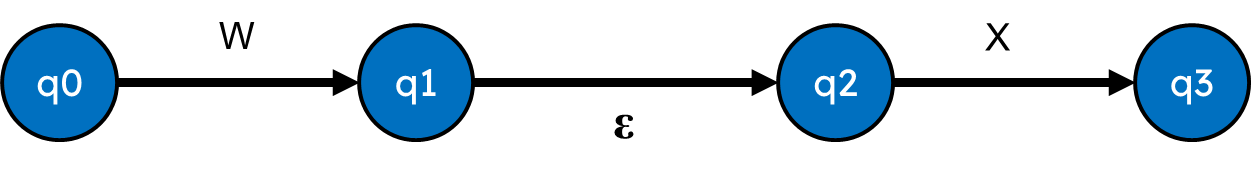
* + Nếu là ký tự ‘\*’ (phép bao đóng):



* + Nếu là ký tự ‘|’ (phép hợp):



* + Nếu là ký tự ‘.’ (phép nối kết):



### 5. Giao diện thực hiện chức năng chương trình

Chương trình sẽthực hiện việc kiểm tra chuỗi đầu vào có được NFAε chấp nhận hay không và hiển thị kết quả rõ ràng qua giao diện. Giao diện này được thực hiện với sự hỗ trợ của thư viện Tkinter. Giao diện có thể thực hiện các chức năng:

* **Thực hiện tải file cơ sở dữ liệu NFAε:** file sẽ được gọi và dữ liệu trên sẽ là cơ sở để kiểm tra chuỗi có thuộc ngôn ngữ đã cho hay không. Nếu chưa tải file mà thực hiện kiểm tra chuỗi, chương trình sẽ thông báo lỗi và buộc người dùng phải tải file mới có thể thực hiện việc kiểm tra.
* **Chức năng nhập regex và chuyển thành NFAε:** sau khi người dùng nhập regex và xác nhận, chương trình sẽ khởi tạo NFAε và cho phép thực hiện kiểm tra chuỗi.
* **Chức năng xem đồ thị NFAε:** sau khi tải file, chương trình sẽ tự động sơ đồ bằng thư viện graphviz. Thư viện này sẽ vẽ sơ đồ và lưu chúng vào một file định dạng png. Chúng ta sau khi load file thành công có thể xem trực tiếp sơ đồ thông qua giao diện hoặc xem lại sơ đồ nếu cần qua việc hình ảnh sẽ được lưu lại trên máy.
* **Nhập chuỗi đầu vào và kiểm tra chuỗi:** với điều kiện đã load file thành công, chúng ta có thể nhập vào một chuỗi để kiểm tra chúng có thuộc ngôn ngữ đã cho hay không. Giao diện cũng sẽ hiển thị từng bước di chuyển của các trạng thái đồng thời cho ra kết quả chấp nhận hay không chấp nhận chuỗi. Điều này giúp ta dễ dàng theo dõi và hiểu cách mà NFAε hoạt động.

## Thiết kế và cài đặt

Chương trình được thiết kế dựa trên ngôn ngữ Python, bao gồm các thành phần chính như đọc dữ liệu NFAε từ file, xử lý thuật toán liên quan (ε-closure, di chuyển trạng thái, kiểm tra chấp nhận chuỗi), và giao diện đồ họa người dùng. Việc cài đặt tập trung vào tính đơn giản và dễ sử dụng, đảm bảo người dùng có thể nạp dữ liệu, kiểm tra chuỗi và xem sơ đồ NFAε một cách trực quan.

### Cài đặt các thư viện

Các thư viện được cài đặt nhằm hỗ trợ xây dựng chương trình dễ dàng và trực quan hơn.

import os

import traceback

import tkinter as tk

from tkinter import filedialog, messagebox

from graphviz import Digraph

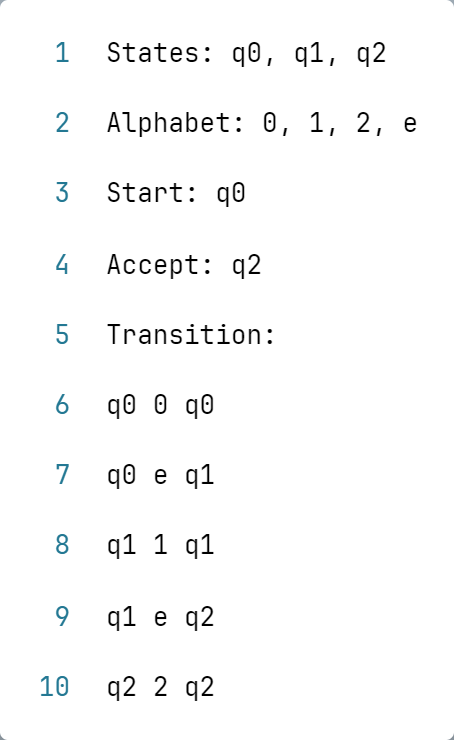
from PIL import Image, ImageTk

from NFAe import NFAe

* **tkinter:** Xây dựng giao diện người dùng và quản lý sự kiện tương tác. Thư viện này giúp tạo cửa sổ chính, các nút bấm, khung nhập liệu, và hiển thị kết quả để người dùng dễ dàng thao tác với chương trình.
* **os:** Làm việc với hệ thống file, giúp kiểm tra sự tồn tại của file hoặc đường dẫn, đảm bảo tính chính xác khi xử lý các file liên quan như sơ đồ trạng thái NFAε.
* **traceback:** hiển thị chi tiết lỗi khi có lỗi phát sinh.
* **filedialog:** Hiển thị hộp thoại chọn file để người dùng dễ dàng tải lên file chứa thông tin NFAε. Điều này đơn giản hóa quá trình nạp dữ liệu vào chương trình.
* **messagebox:** Hiển thị thông báo, lỗi hoặc kết quả dưới dạng hộp thoại. Giúp người dùng nhận phản hồi nhanh về trạng thái hoạt động của chương trình, như thông báo lỗi khi tải file hoặc kết quả kiểm tra chuỗi.
* **graphviz:** Vẽ sơ đồ trạng thái NFAε, hỗ trợ trực quan hóa các trạng thái và chuyển đổi của NFAε. Kết quả được lưu thành ảnh để người dùng dễ dàng xem qua giao diện.
* **PIL:** Xử lý và hiển thị hình ảnh sơ đồ trạng thái NFAε trong giao diện người dùng. Điều này giúp người dùng trực tiếp xem và kiểm tra sơ đồ trạng thái mà không cần mở bằng phần mềm ngoài.

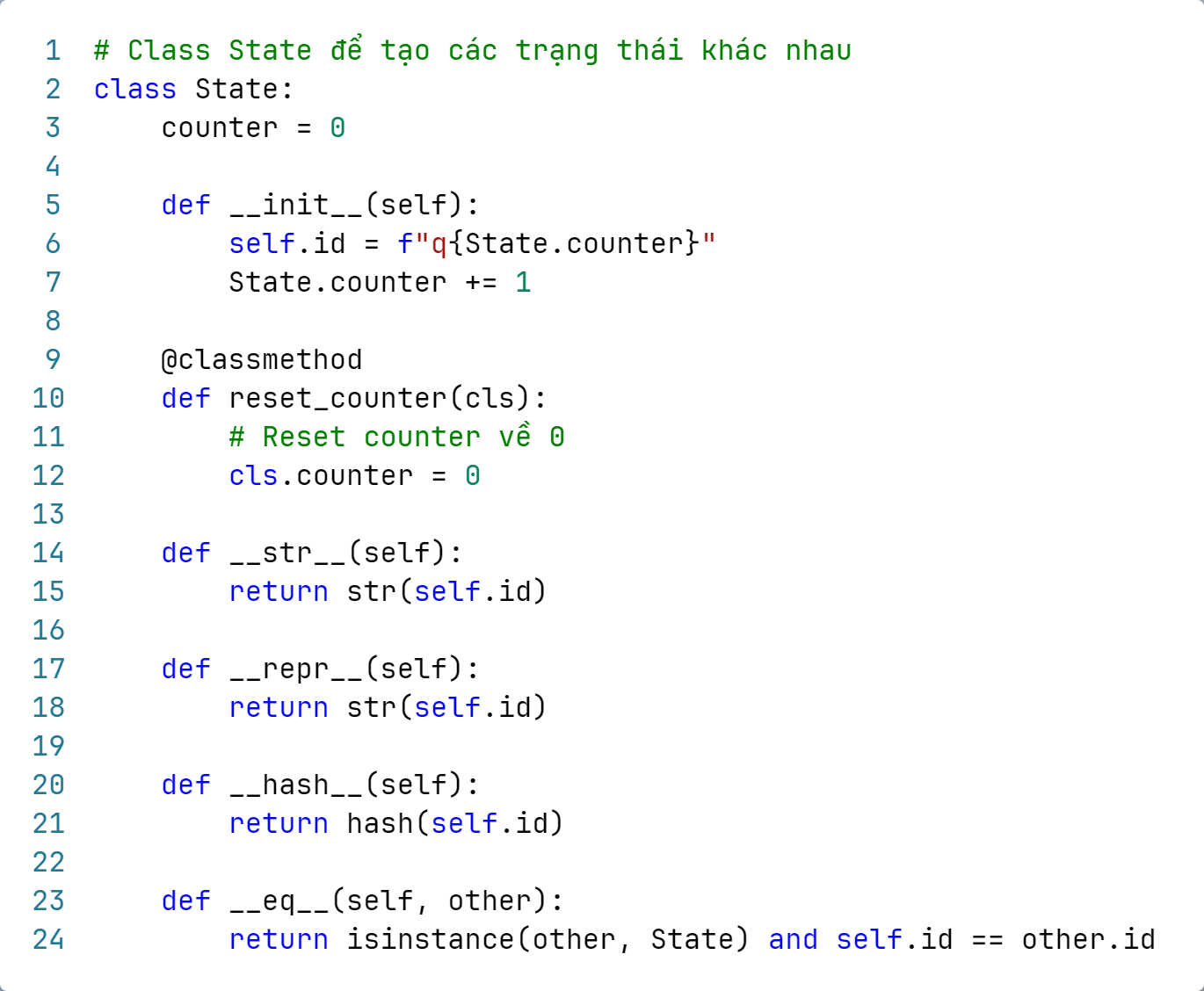
### Cấu trúc lưu trữ NFAε

Cấu trúc lưu trữ của NFAε dùng file txt: gồm các trạng thái, ký hiệu đầu vào (bao gồm ε), trạng thái bắt đầu, các trạng thái kết thúc và các hàm chuyển trạng thái.



### Lớp State

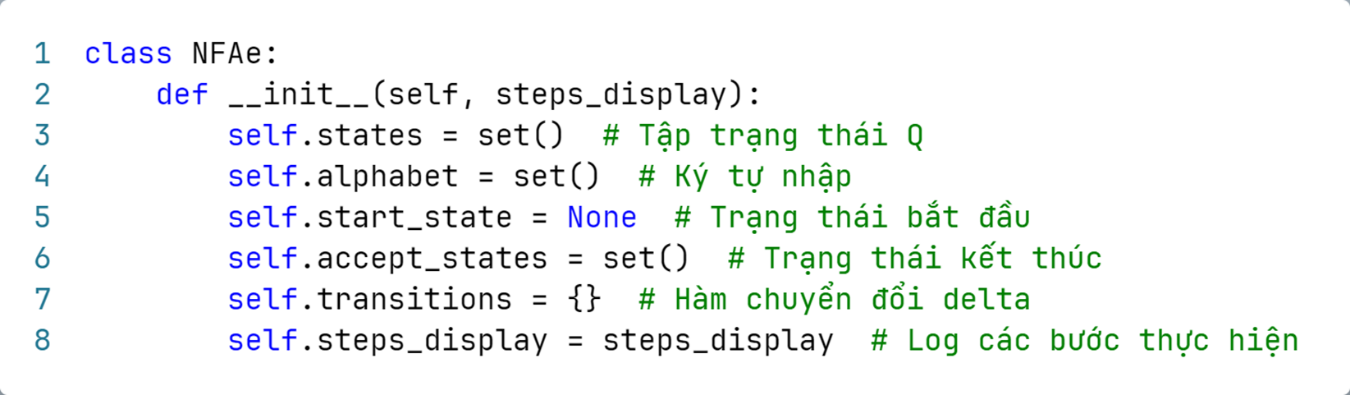
Lớp này lưu trữ cấu trúc state (trạng thái) của NFAε. Mục đích là giúp thuận tiện đặt tên cho các trạng thái trong quá trình xây dựng NFAε.



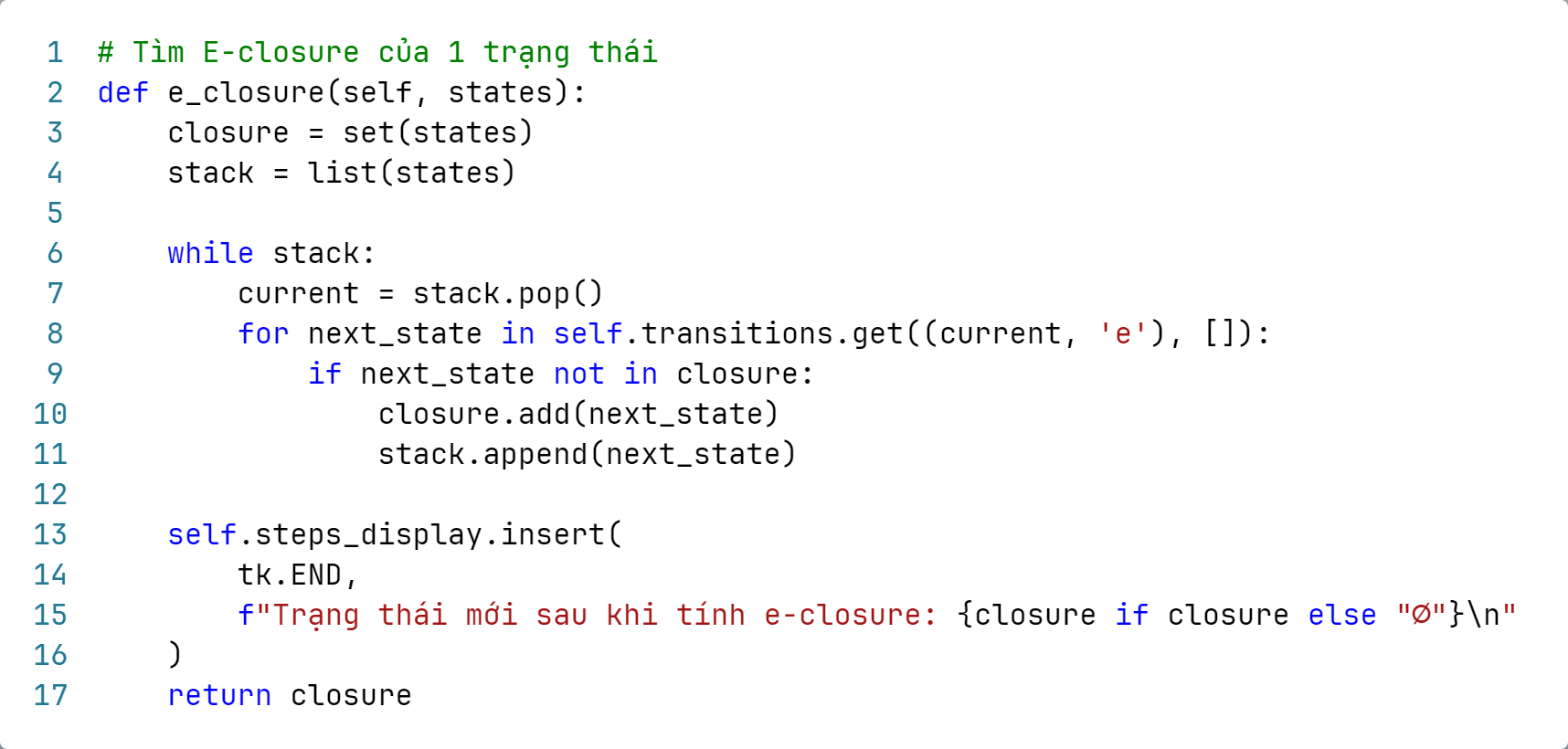
### Lớp NFAe

Lớp NFAe là thành phần cốt lõi của chương trình, chịu trách nhiệm xử lý liên quan đến NFAε. Nó cung cấp các phương thức để thao tác với các trạng thái và chuyển đổi, từ đó xác định xem một chuỗi có được chấp nhận bởi NFAε hay không.

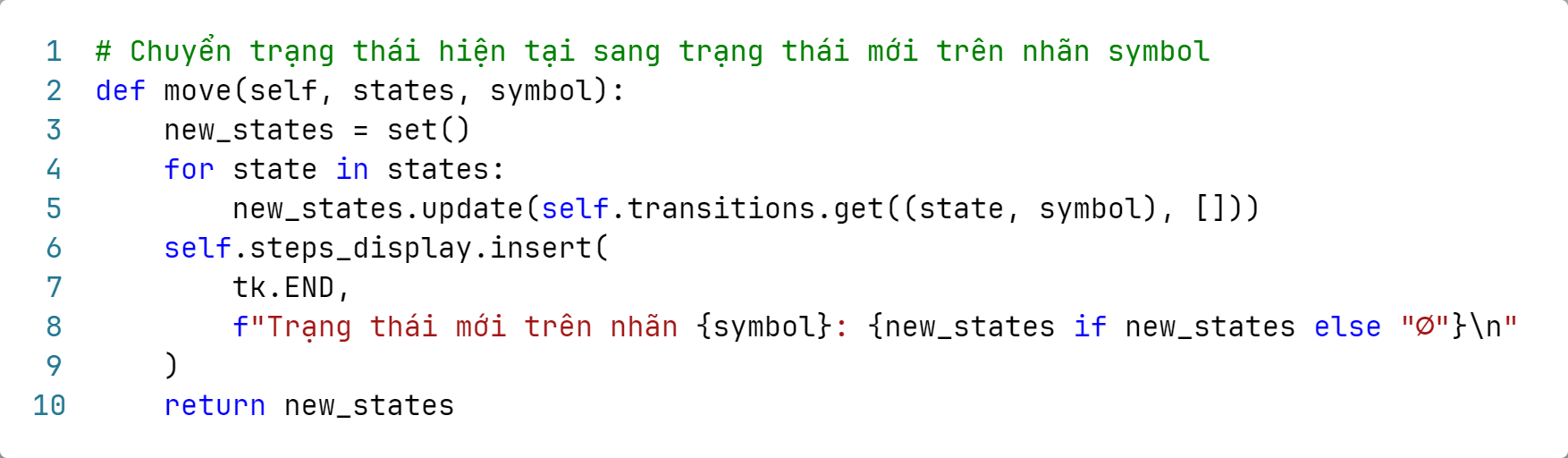
* **Các thuộc tính:**
  + **states:** Tập hợp tất cả các trạng thái của NFAε.
  + **alphabet:** Tập hợp các ký tự đầu vào mà NFAε có thể xử lý (bao gồm cả epsilon).
  + **start\_state:** Trạng thái bắt đầu, nơi quá trình kiểm tra chuỗi được bắt đầu.
  + **accept\_states:** Tập trạng thái kết thúc, dùng để quyết định xem một chuỗi có được chấp nhận hay không.
  + **transitions:** Hàm chuyển đổi , lưu trữ các quy tắc chuyển trạng thái dựa trên ký tự đầu vào hoặc epsilon.
  + **steps\_display:** Khung giao diện hiển thị các bước xử lý (kết nối với giao diện Tkinter).

****

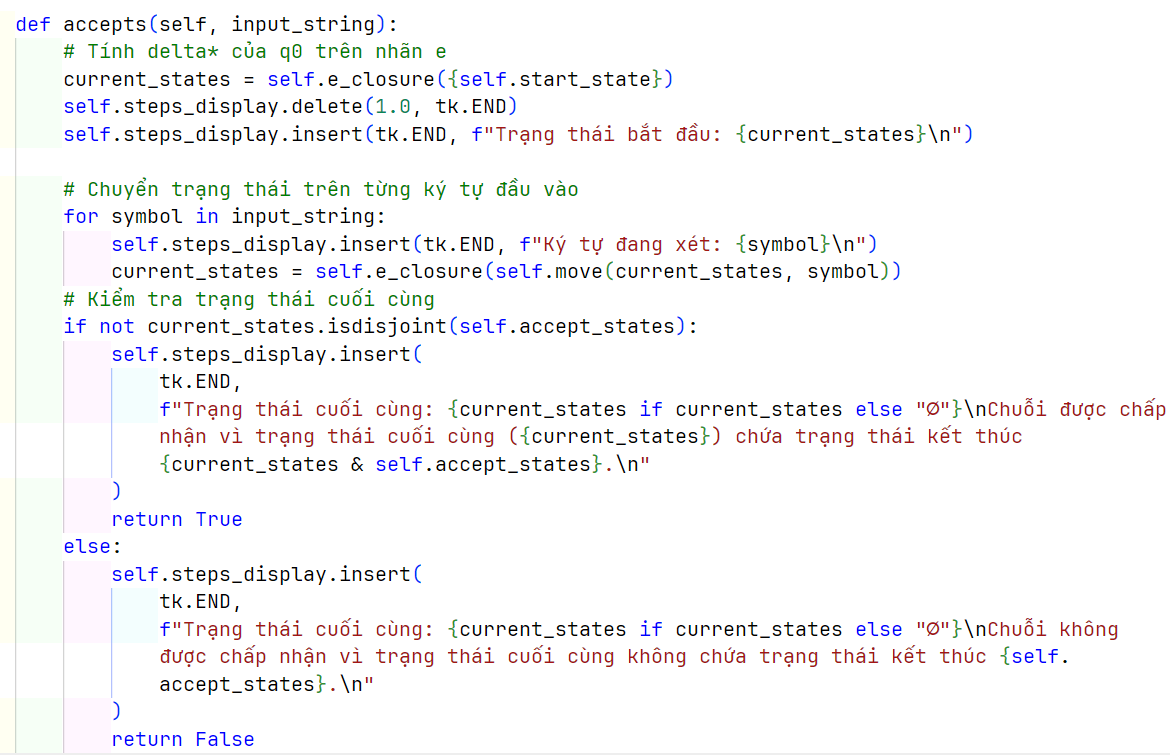
* **Các phương thức:**
  + **epsilon\_closure(self, state):** Tính tập ε-closure, bao gồm tất cả các trạng thái có thể đạt tới từ trạng thái đầu vào qua các bước epsilon.

****

* + **move(self, states, char):** Tính toán các trạng thái có thể chuyển đến từ một tập trạng thái với ký tự đầu vào.

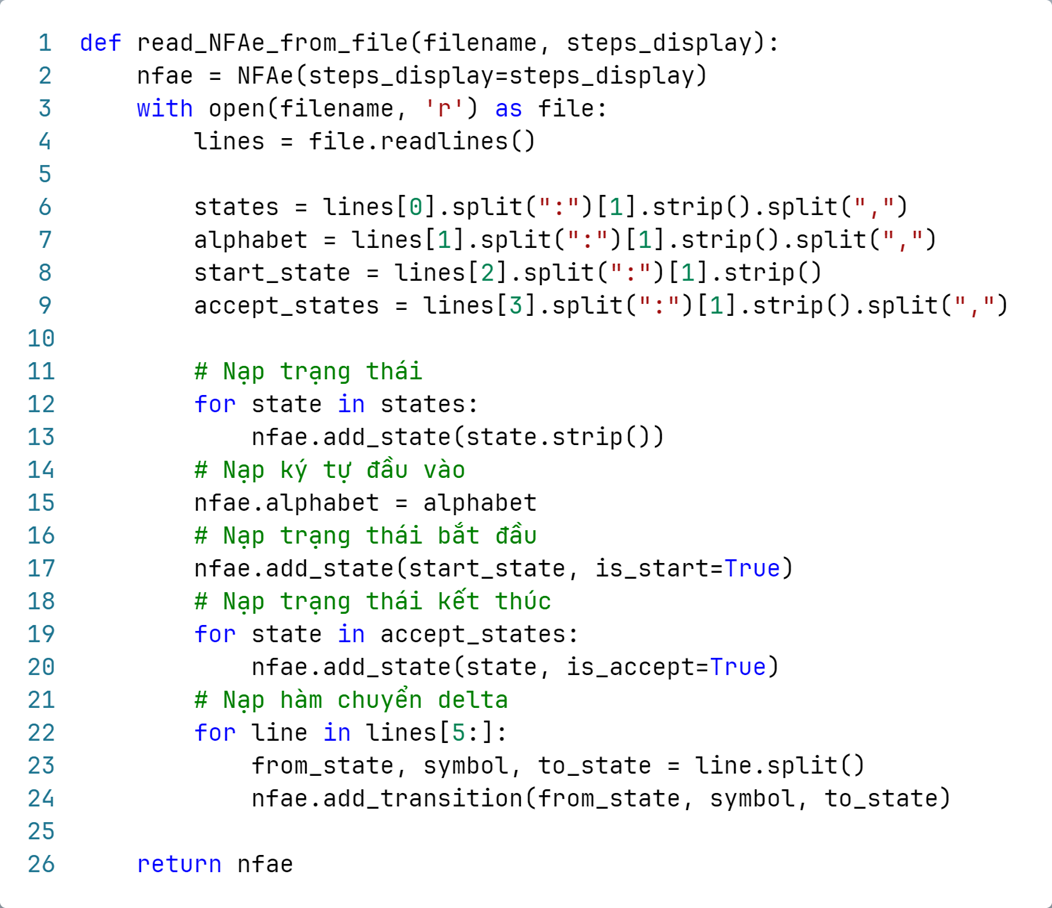
****

* + **accepts(self, string):** Kiểm tra xem chuỗi đầu vào có được chấp nhận bởi NFAε không. Phương thức này sử dụng epsilon\_closure và move để duyệt qua các trạng thái.

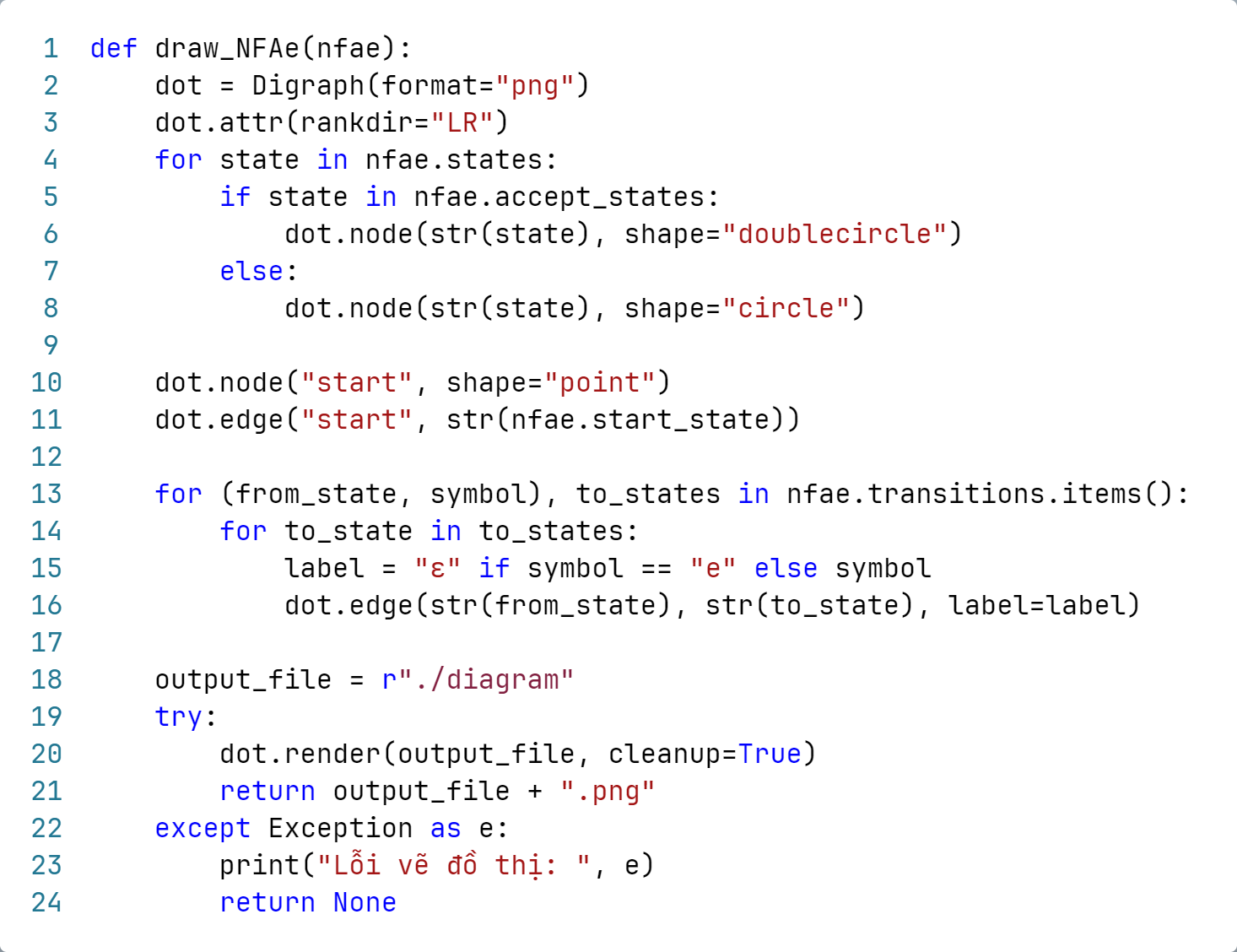
****

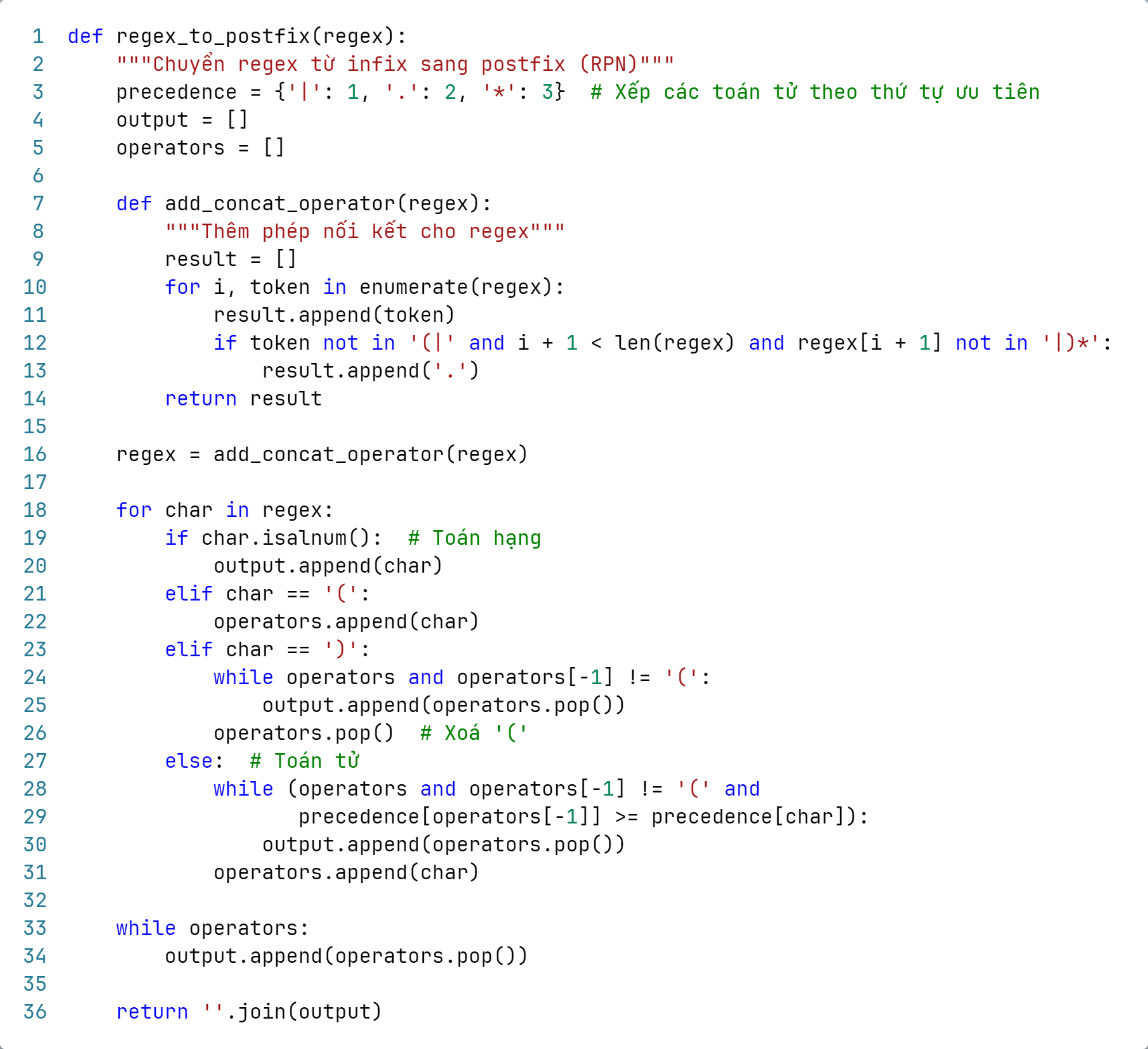
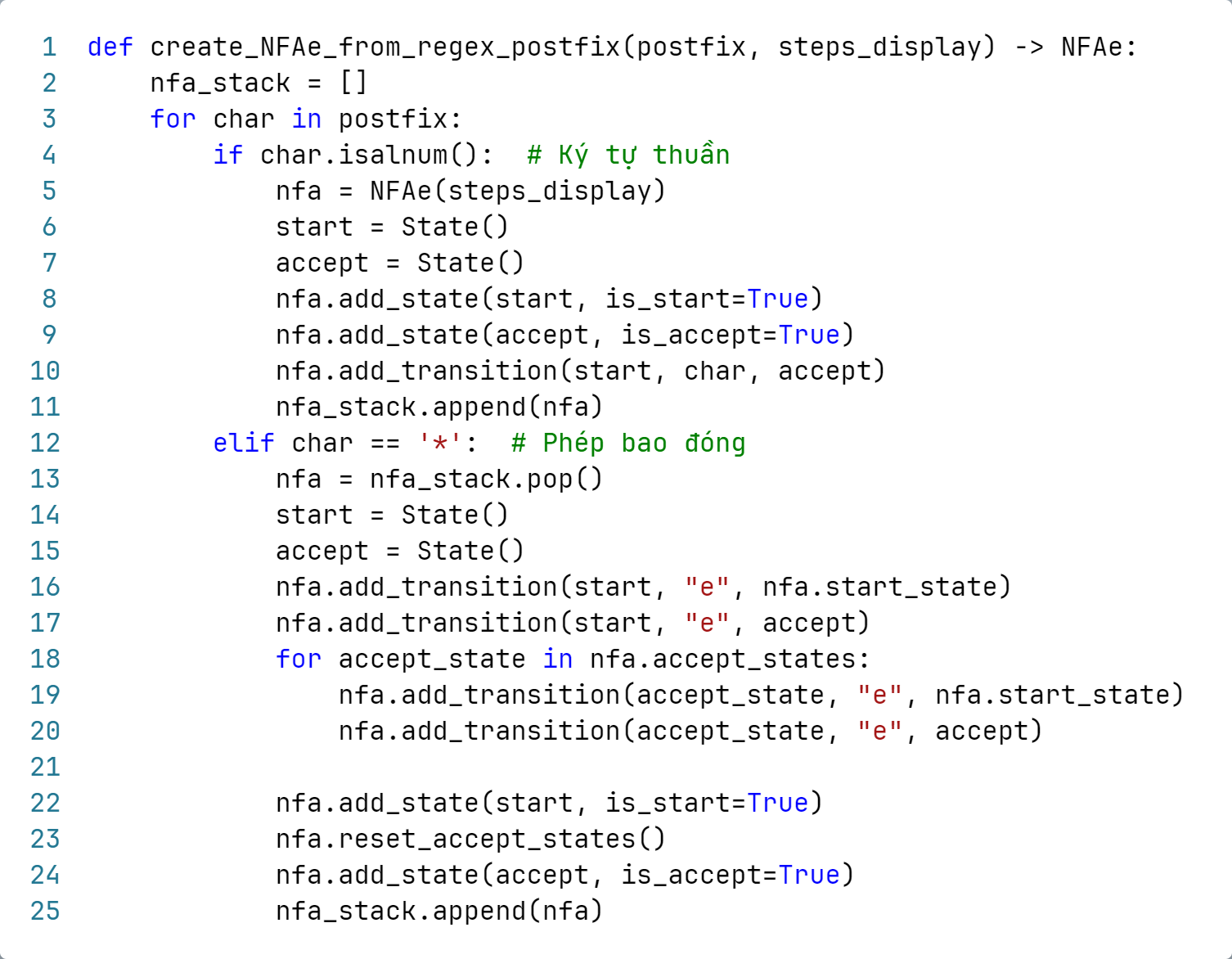
### Hàm hỗ trợ

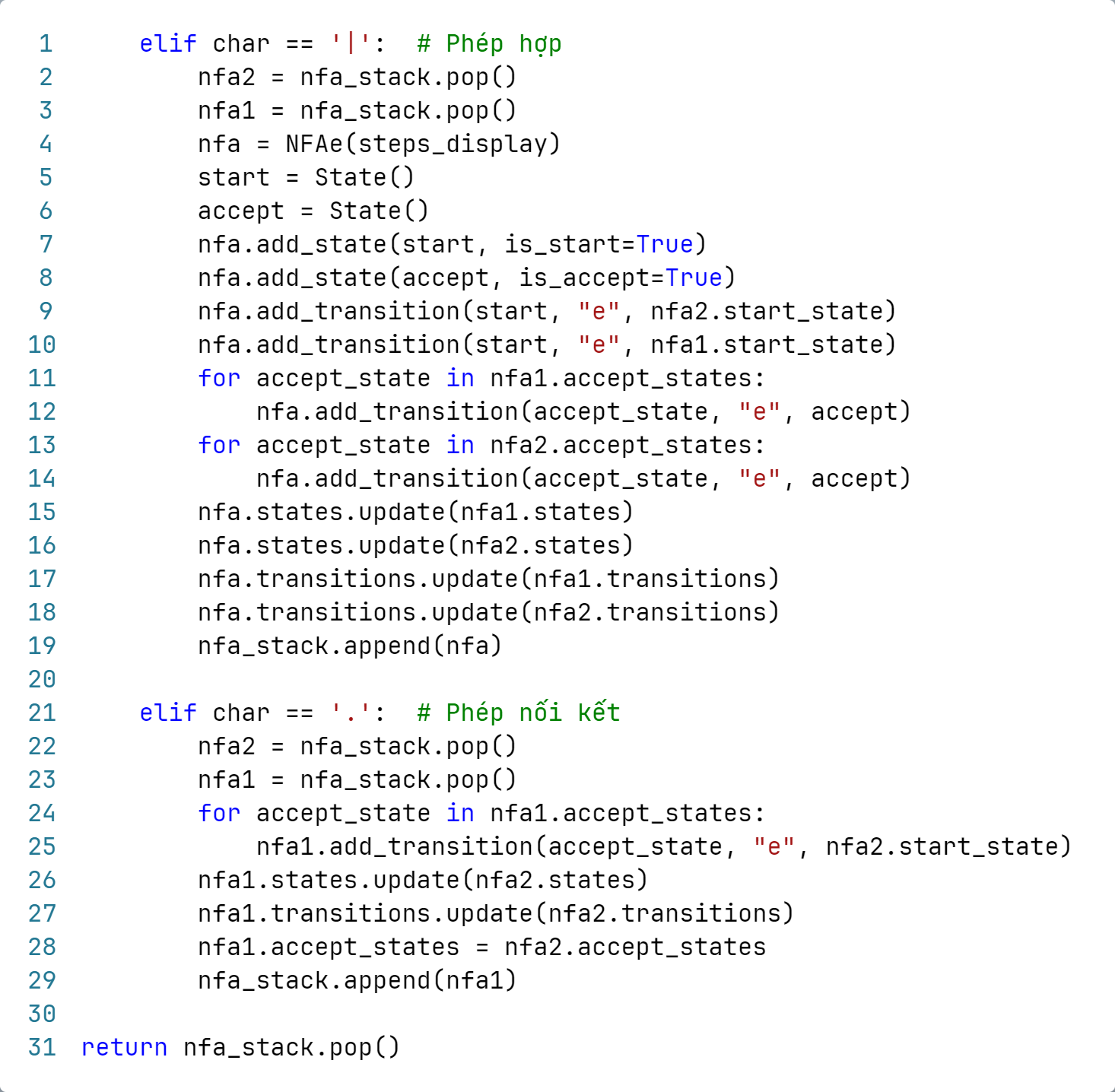
* **Hàm read\_NFAe\_from\_file:** Đọc dữ liệu từ file và khởi tạo đối tượng NFAε.



* **Hàm draw\_nfae:** Vẽ sơ đồ trạng thái NFAε và lưu thành file ảnh bằng thư viện graphviz.

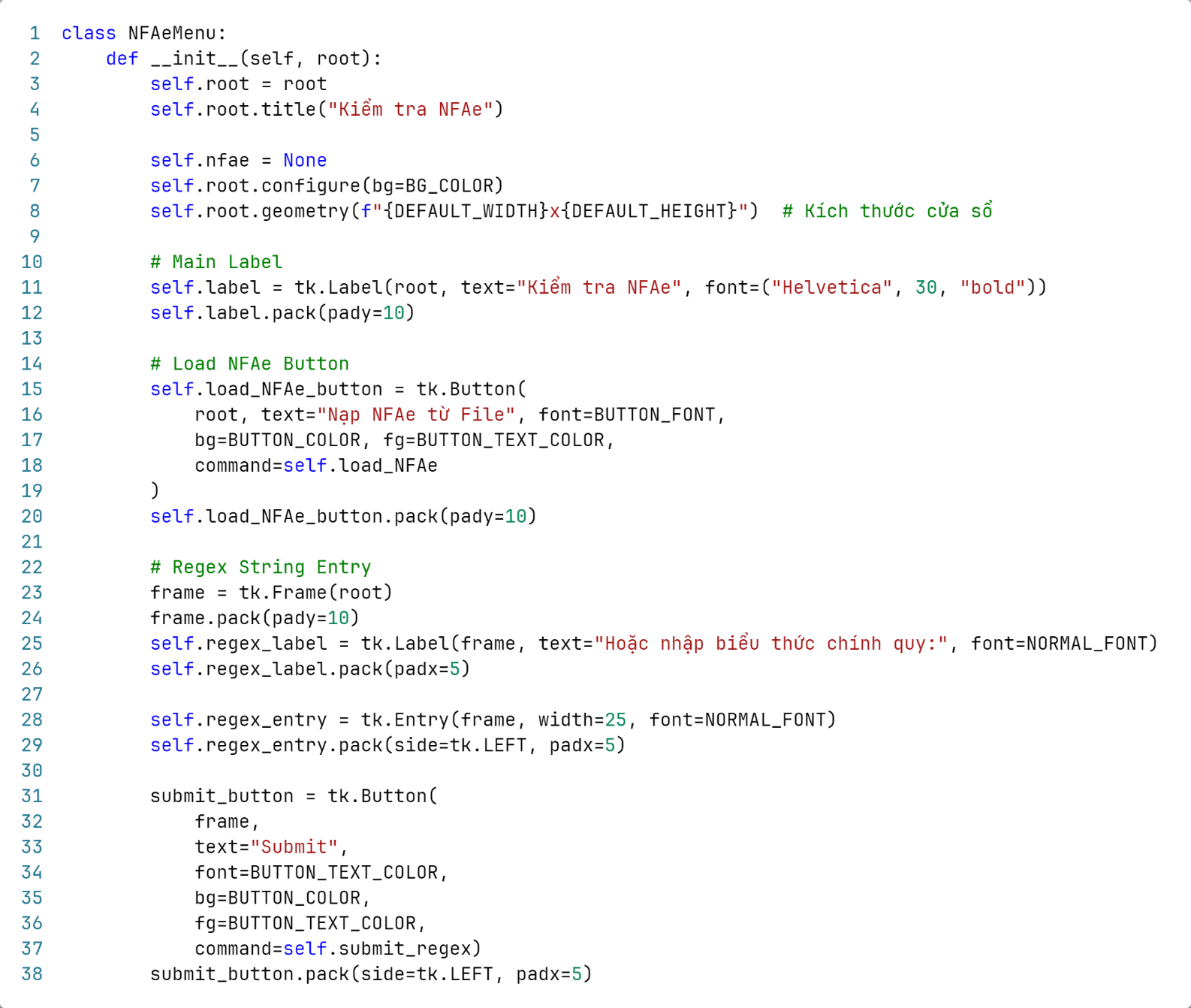


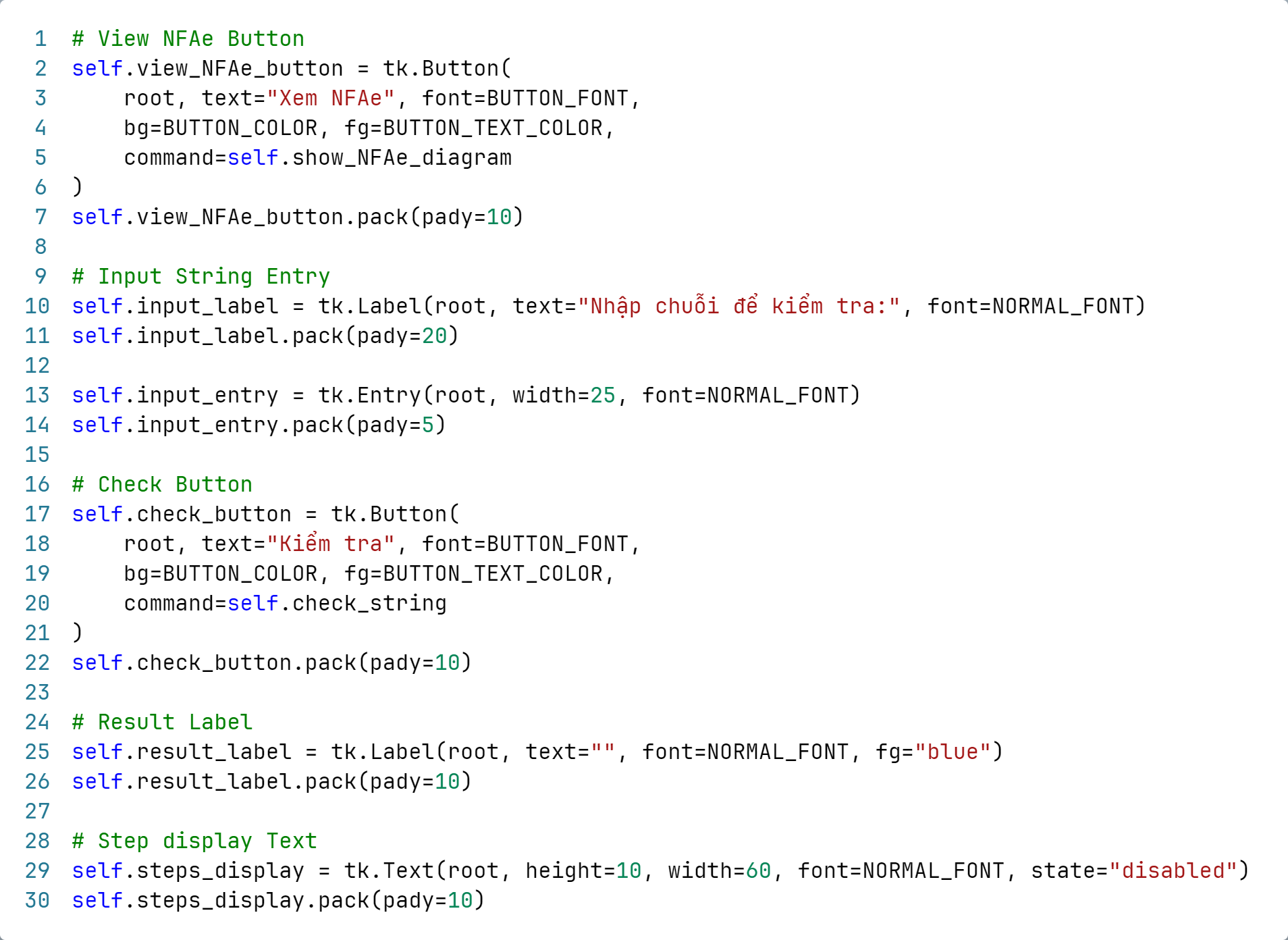
* **Hàm regex\_to\_postfix:** Chuyển regex từ infix sang postfix.
* **Hàm create\_NFAe\_from\_regex\_postfix:** Chuyển regex ở dạng postfix sang NFAε. 



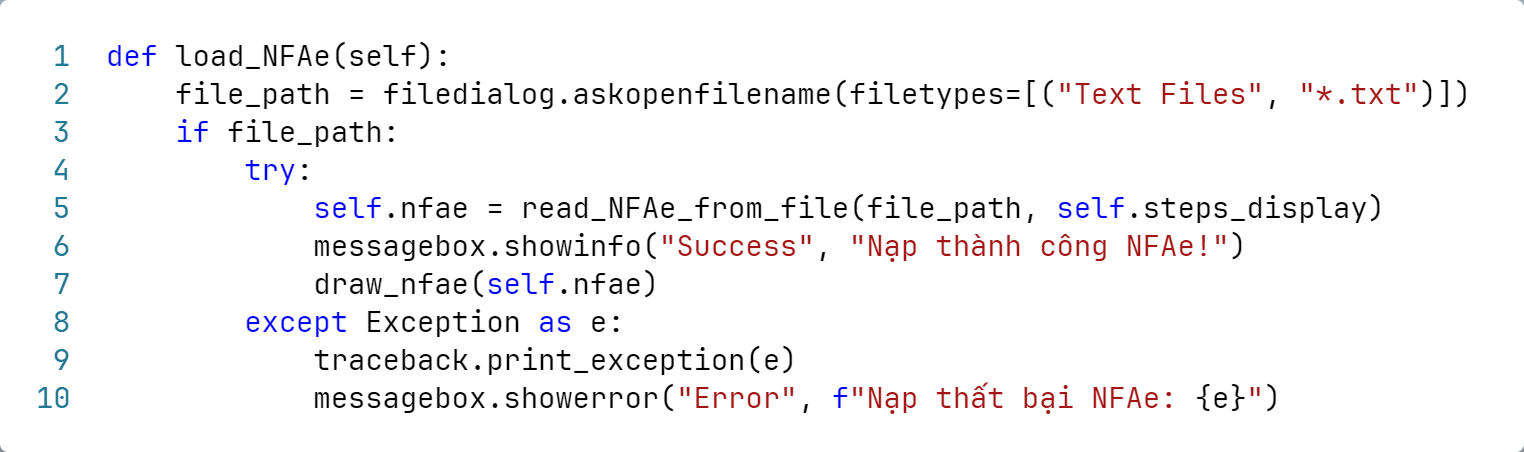
### Lớp NFAeMenu

Lớp xử lý các tác vụ liên quan đến giao diện người dùng, mục đích chính là giúp người dùng thuận tiện trong quá trình sử dụng ứng dụng.

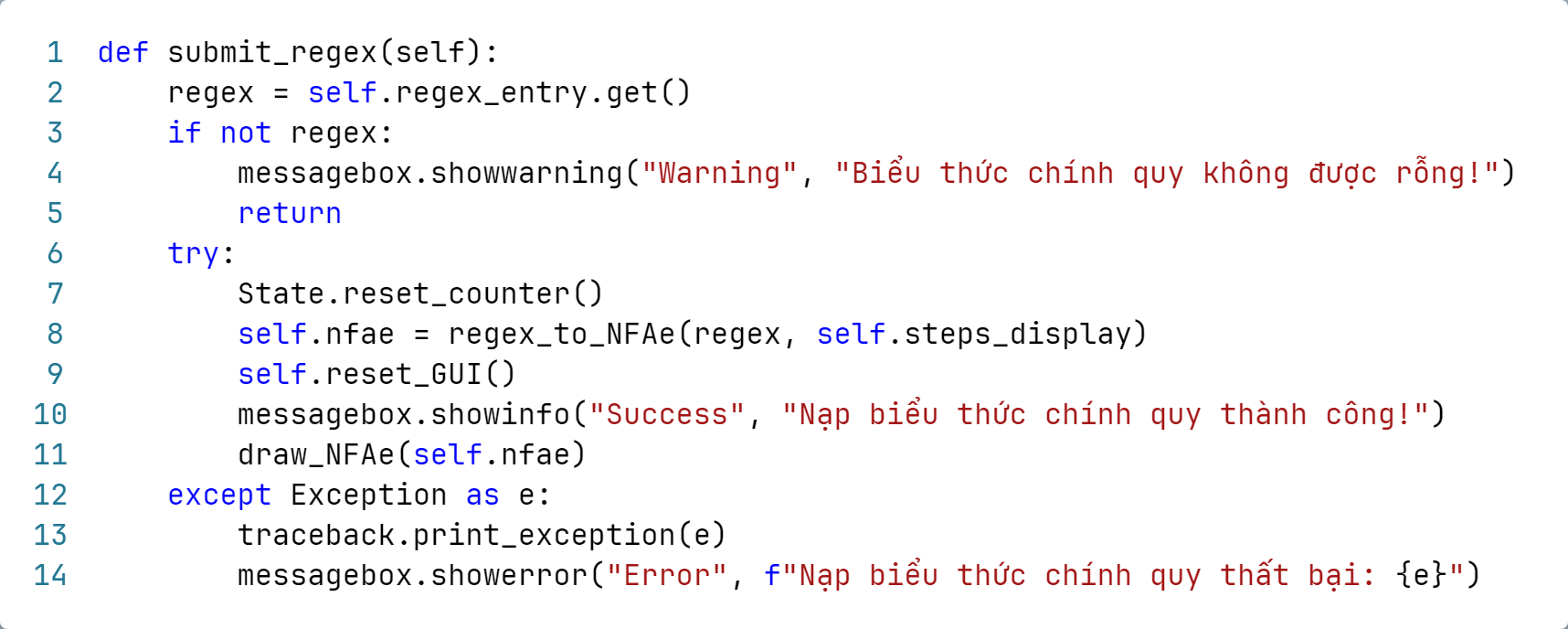


****

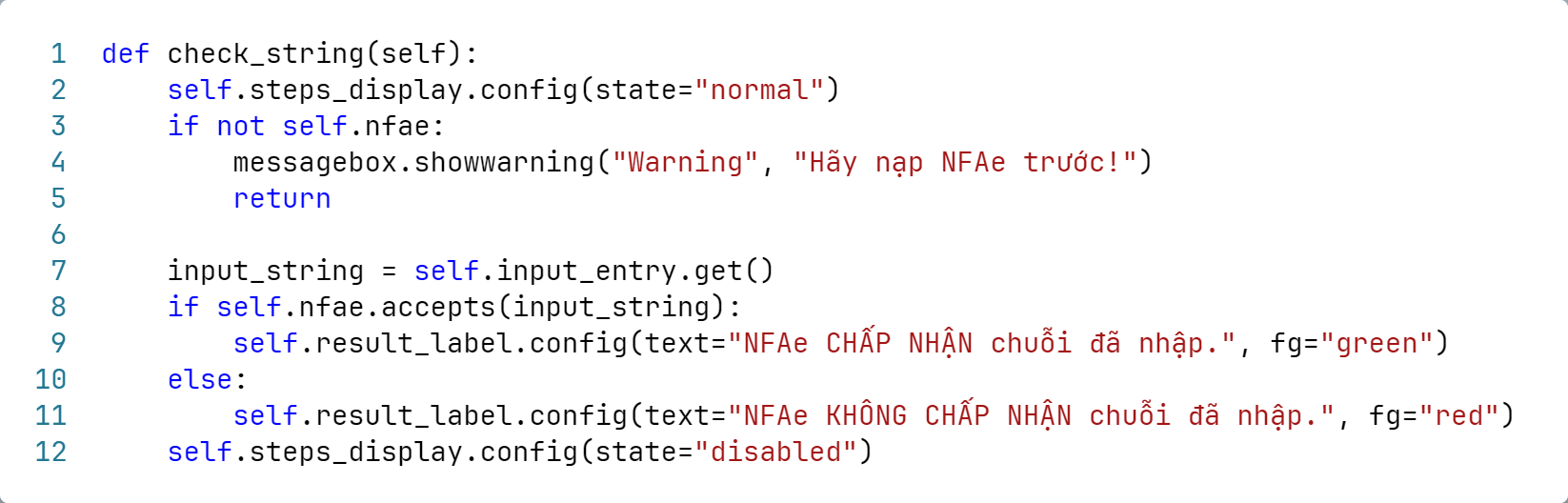
* **Hàm load\_NFAe:** Chức năng này cho phép người dùng tải một file NFAε vào chương trình, vẽ sơ đồ NFAε và thông báo kết quả nạp file thành công hay thất bại thông qua giao diện.



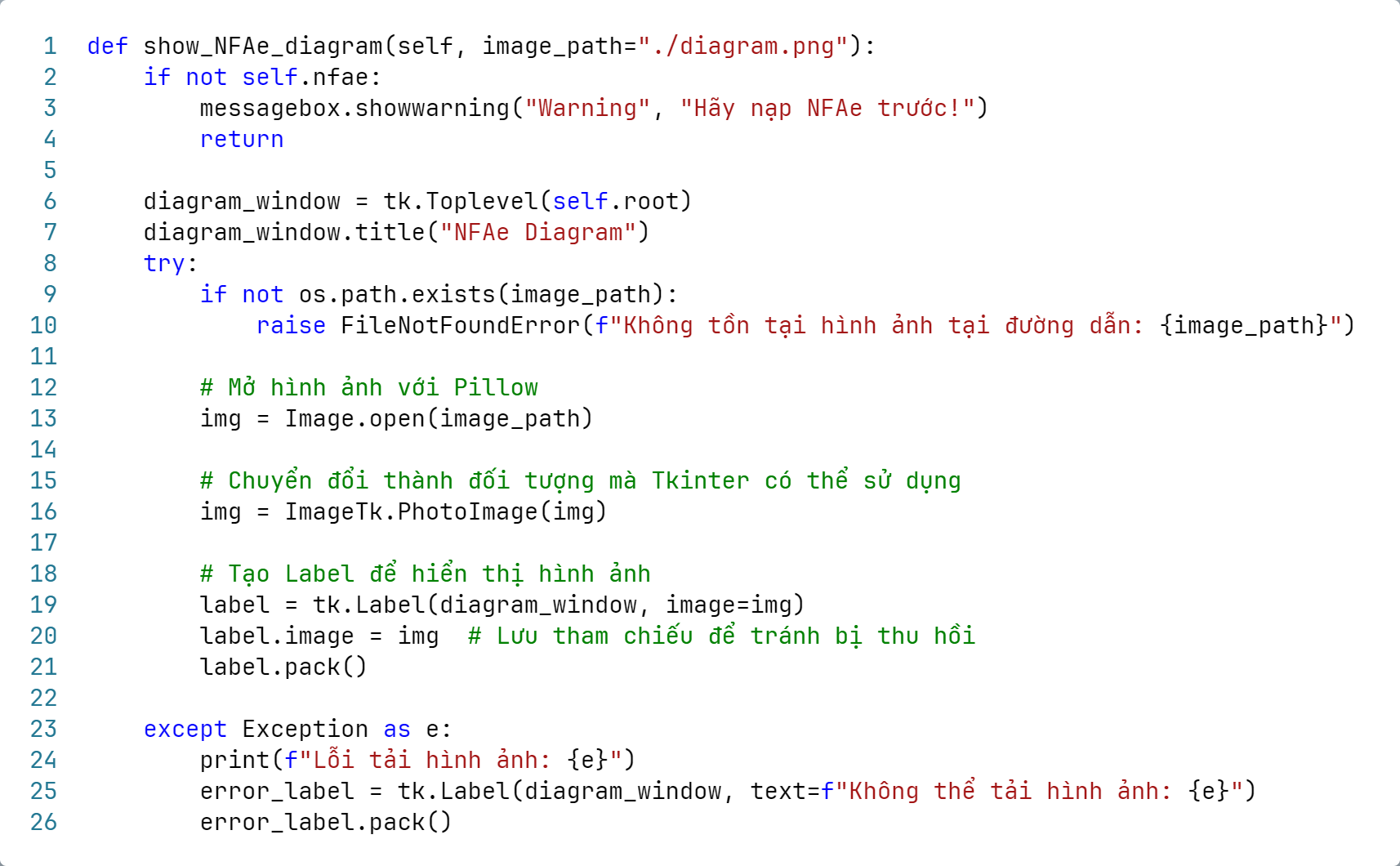
* **Hàm submit\_regex:** Chức năng này cho phép người dùng nhập một chuỗi regex, chuyển sang NFAε và thông báo kết quả nạp file thành công hay thất bại thông qua giao diện.



* **Hàm check\_string:** Hàm này kiểm tra xem một chuỗi nhập vào có được NFAε chấp nhận hay không và hiển thị kết quả cho người dùng thông qua giao diện.



* **Hàm show\_NFAe\_diagram:** có nhiệm vụ hiển thị sơ đồ trạng thái của NFAε dưới dạng hình ảnh cho người dùng xem trên giao diện.

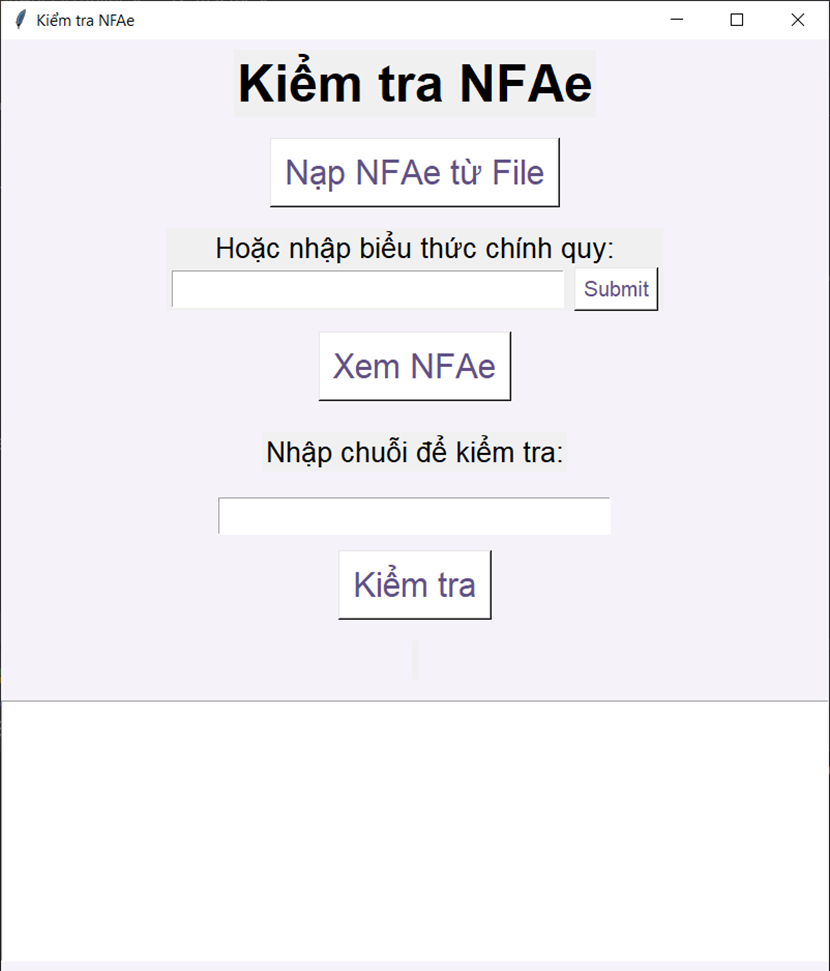


## Kết quả đạt được

### 1. Xây dựng giao diện người dùng hoàn chỉnh:

Giao diện sử dụng thư viện Tkinter, cho phép người dùng dễ dàng tương tác với chương trình để tải file NFAε, kiểm tra chuỗi đầu vào, và xem sơ đồ trạng thái của NFAε.

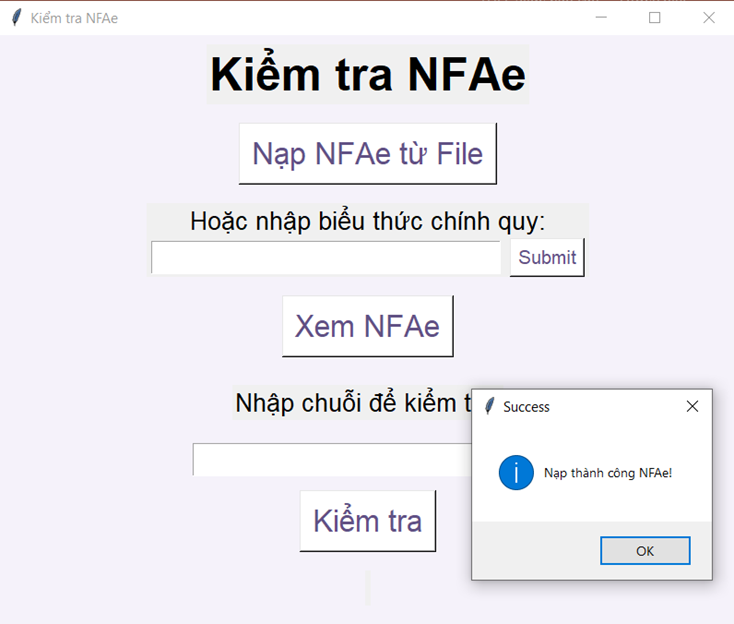
Các chức năng chính như tải file NFAε, kiểm tra chuỗi, hiển thị bước kiểm tra chuỗi, và vẽ sơ đồ trạng thái đều được tích hợp và hoạt động mượt mà trên giao diện**.**

****

### 2. Xử lý file dữ liệu

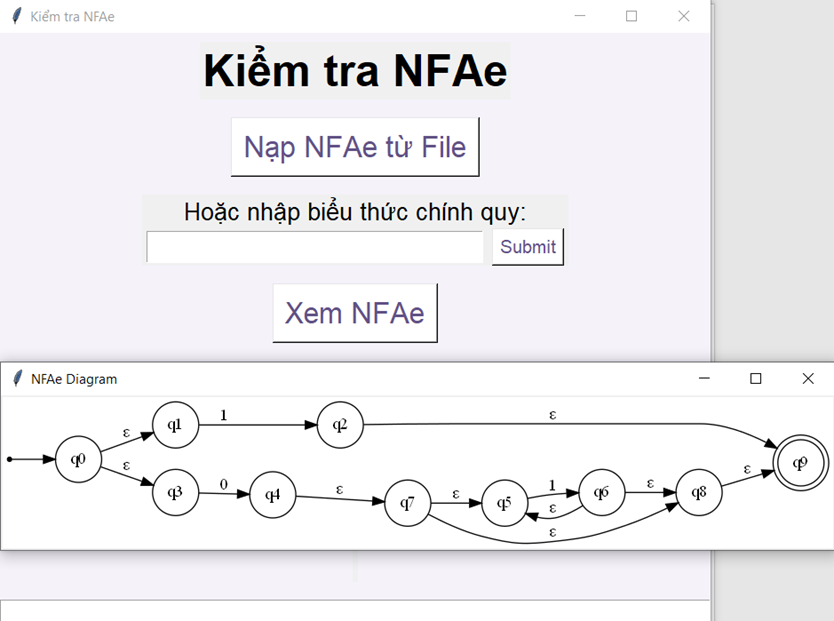
Người dùng có thể tải một file dữ liệu chứa mô tả NFAε (theo định dạng txt), chương trình sẽ đọc và phân tích file này để xây dựng đối tượng NFAε.

Phần đọc file xử lý được nhiều loại thông tin như tập trạng thái, bảng chuyển tiếp, trạng thái bắt đầu và trạng thái kết thúc.



### 3. Hiển thị sơ đồ trạng thái NFAε

Chương trình có thể vẽ sơ đồ trạng thái của NFAε bằng thư viện Graphviz và hiển thị hình ảnh này trong cửa sổ mới. Người dùng có thể dễ dàng kiểm tra trực quan cấu trúc của NFAε.



### 4. Chức năng kiểm tra chuỗi đầu vào

Chương trình có thể nhận chuỗi đầu vào từ người dùng, kiểm tra chuỗi có được chấp nhận bởi NFAε đã cho hay không.

Quá trình kiểm tra chuỗi sẽ hiển thị từng bước chuyển trạng thái, giúp người dùng hiểu rõ cách mà NFAε xử lý chuỗi đầu vào.

* **Chuỗi được chấp nhận:**



* **Chuỗi bị từ chối:**



## Hướng phát triển

Chúng ta có thể có các cải tiến và mở rộng có thể áp dụng vào chương trình trong tương lai để nâng cao hiệu quả và tính năng. Dưới đây là một số hướng phát triển tiềm năng mà ta có thể xem xét:

* **Hỗ trợ nhiều định dạng tệp NFAε**: Hiện tại, hệ thống chỉ hỗ trợ đọc dữ liệu NFAε từ tệp văn bản txt. Trong tương lai, có thể mở rộng để hỗ trợ nhiều định dạng tệp khác nhau như JSON, XML hoặc thậm chí các định dạng đồ họa như GraphML để người dùng có thể nhập dữ liệu dễ dàng hơn.
* **Tối ưu hóa giao diện người dùng**: Giao diện người dùng hiện tại có thể được cải tiến để trở nên trực quan hơn, bao gồm việc thêm các tính năng như kéo thả các trạng thái hoặc chuyển tiếp, tự động vẽ sơ đồ NFAε khi người dùng tạo ra nó.
* **Tính toán và hiển thị các bước chi tiết hơn**: Mặc dù hệ thống hiện tại có thể hiển thị các bước kiểm tra chuỗi, nhưng có thể mở rộng để cung cấp thông tin chi tiết hơn, chẳng hạn như phân tích sâu về mỗi bước của quá trình chuyển trạng thái và ε-closure.
* **Mở rộng hỗ trợ cho các loại máy tự động khác**: Hệ thống có thể được phát triển thêm để hỗ trợ các loại tmáy tự động khác ngoài NFAε, như DFA (Deterministic Finite Automata), hoặc hỗ trợ việc chuyển đổi giữa các loại tự động hóa này.

## Tài liệu tham khảo

**[1]. Giáo trình, tài liệu bộ môn tin học lý thuyết**

**[2]. Tài liệu về Graphviz**

(<https://graphviz.gitlab.io/documentation/>)

**[3]. Tài liệu về PIL (Python Imaging Library)** (<https://pillow.readthedocs.io/en/stable/>)

**[4]. Tài liệu về Tkinter**

(<https://docs.python.org/3/library/tkinter.html>)

**[5]. Cách chuyển từ infix sang postfix**

([Reserve Polish Notation](https://viblo.asia/p/reserve-polish-notation-924lJrG8lPM))