

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 NFA ϵ và kiểm tra một chuỗi có
thuộc NFA ϵ 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
I. Lab 1	4
II. Lab 2	7
III. Lab 3	11
PHẦN 2: PHẦN BÁO CÁO.....	20
I. Cơ sở lý thuyết	20
1. Automata hữu hạn (FA: Finite Automata)	20
2. Automata hữu hạn không đơn định – NFA (Non-deterministic Finite Automaton)	20
3. NFA với ϵ -dịch chuyển (NFA ϵ)	21
II. Mục tiêu bài toán	23
1. Xây dựng và mô phỏng một NFA ϵ	23
2. Kiểm tra chuỗi với NFA ϵ	23
3. Tích hợp với file	23
4. Chuyển đổi biểu thức chính quy sang NFA ϵ	23
III. Phương pháp thực hiện	24
1. Phân tích bài toán và thiết kế cấu trúc dữ liệu:.....	24
2. Cài đặt thuật toán kiểm tra chuỗi với NFA ϵ	24
3. Chức năng đọc file dữ liệu NFA ϵ cho trước	26
4. Chức năng chuyển regex sang NFA ϵ	26
5. Giao diện thực hiện chức năng chương trình	28
IV. Thiết kế và cài đặt.....	29
1. Cài đặt các thư viện	29
2. Cấu trúc lưu trữ NFA ϵ	30
3. Lớp State	31
4. Lớp NFA ϵ	31
5. Hàm hỗ trợ	34
6. Lớp NFA ϵ Menu.....	39
V. Kết quả đạt được	43
1. Xây dựng giao diện người dùng hoàn chỉnh:	43

2. Xử lý file dữ liệu	44
3. Hiện thị sơ đồ trạng thái NFA ϵ	45
4. Chức năng kiểm tra chuỗi đầu vào	46
VI. Hướng phát triển	48
VII. Tài liệu tham khảo	49

PHẦN 1: TỔNG HỢP CÁC BÀI THỰC HÀNH

I. 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']))
```

Đề tài: Xây dựng NFA ϵ và kiểm tra một chuỗi có thuộc NFA ϵ đã cho không

```
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
```

II. 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))
```

III. 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"\n{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:

Đề tài: Xây dựng NFA ϵ và kiểm tra một chuỗi có thuộc NFA ϵ đã cho không

```
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 nói 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

I. Cơ sở lý thuyết

1. Automata hữu hạn (FA: Finite Automata)

Automata hữu hạn – FA là một mô hình tính 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.

2. 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 cùng một trạng thái trên cùng một ký hiệu nhập.

Một NFA được định nghĩa bởi bộ 5 thành phần:

$$M = (Q, \Sigma, \delta, q_0, F)$$

Trong đó:

- Q : Tập hợp hữu hạn các trạng thái.
- Σ : Tập hợp hữu hạn các ký tự (bảng chữ cái đầu vào).
- δ : Hàm chuyển trạng thái, với:

$$\delta: Q \times \Sigma \rightarrow 2^Q$$

- q_0 : Trạng thái bắt đầu ($q_0 \in Q$).
- F : Tập các trạng thái kết thúc ($F \subseteq Q$).

3. NFA với ϵ -dịch chuyển (NFA ϵ)

Trong lý thuyết Automata, **NFA ϵ** (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 NFA ϵ

Một NFA ϵ được định nghĩa bởi bộ 5 thành phần:

$$M = (Q, \Sigma, \delta, q_0, F)$$

Trong đó:

- Q : Tập hợp hữu hạn các trạng thái.
- Σ : Tập hợp hữu hạn các ký tự (bảng chữ cái đầu vào).
- δ : Hàm chuyển trạng thái, với:

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$$

- q_0 : Trạng thái bắt đầu ($q_0 \in Q$).
- F : Tập các trạng thái kết thúc ($F \subseteq Q$).

Hàm chuyển trạng thái δ trong NFA ϵ 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 NFA ϵ , hàm chuyển trạng thái δ có dạng:

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$$

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 q_1 và đọc ký tự a , nó có thể chuyển đến một hoặc nhiều trạng thái, chẳng hạn như q_2 và q_3 .
- Nếu máy đang ở trạng thái q_1 và đọc ký tự ϵ , nó có thể chuyển đến trạng thái q_2 mà không cần đọc thêm ký tự nào.

3.3. Cách thức hoạt động của NFA ϵ

Khi kiểm tra một chuỗi đầu vào, NFA ϵ 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: NFA_ϵ bắt đầu từ trạng thái ban đầu q_0 .
- Duyệt qua chuỗi đầu vào:
 - Với mỗi ký tự trong chuỗi đầu vào, NFA_ϵ 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 NFA_ϵ 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 NFA_ϵ 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 q_1 có chuyển đổi epsilon đến q_2 , và từ q_2 lại có chuyển đổi epsilon đến q_3 , thì ϵ -closure của q_1 sẽ là $\{q_1, q_2, q_3\}$.

II. 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 NFA_ϵ (NFA với ϵ -dịch chuyển), giúp người dùng có thể tạo và chỉnh sửa một NFA_ϵ , hỗ trợ phát thảo sơ đồ NFA_ϵ , sau đó kiểm tra một chuỗi đầu vào xem có thuộc ngôn ngữ mà NFA_ϵ nhận diện hay không.

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

1. Xây dựng và mô phỏng một NFA_ϵ

- Tạo ra một công cụ để người dùng có thể xây dựng NFA_ϵ 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 NFA_ϵ do mình tự tạo một cách dễ dàng thông qua việc đọc file.
- Hỗ trợ người dùng vẽ sơ đồ NFA_ϵ của mình thông qua các thư viện có sẵn.

2. Kiểm tra chuỗi với NFA_ϵ

- Cung cấp một chức năng để người dùng kiểm tra một chuỗi có thuộc ngôn ngữ của NFA_ϵ được người dùng 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 NFA_ϵ vào một file và sau đó tải lên giao diện để vẽ sơ đồ NFA_ϵ , đồng thời kiểm tra chuỗi đầu vào tương ứng.
- Cung cấp một cách dễ dàng để nhập dữ liệu NFA_ϵ 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 NFA_ϵ

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

III. Phương pháp thực hiện

Để thực hiện bài toán "Xây dựng NFA ϵ và kiểm tra chuỗi có thuộc NFA ϵ 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 NFA ϵ . Cấu trúc dữ liệu bao gồm các thành phần chính sau:

- **Trạng thái (States):** Đây là tập hợp các trạng thái của NFA ϵ . 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 NFA ϵ . 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 NFA ϵ 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 NFA ϵ nhận diện chuỗi đầu vào. Trong phần này ϵ sẽ được dùng 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 NFA ϵ , 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 trạng thái kết thúc (Accept States):** Đây là các trạng thái mà NFA ϵ 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 NFA ϵ .
- **Hàm chuyển trạng thái (Transitions):** Đây là một bảng hoặc đồ thị mô tả các chuyển đổi giữa các trạng thái trong NFA ϵ . Điều đặc biệt ở NFA ϵ 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 NFA ϵ

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 NFA ϵ nhận diện hay không. Việc này bao gồm:

- **Xử lý ϵ -closure:** Trong NFA ϵ , 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 NFA ϵ 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 NFA_ϵ , 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 NFA_ϵ 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 NFA_ϵ cho trước

Chức năng này hướng đến việc xử lý file đầu vào NFA_ϵ 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.

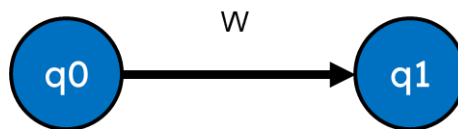
- **Mở và đọc file đầu vào:** file được mở ở chế độ đọc và tất cả các dòng được lưu vào danh sách `lines`.
- **Tuân thủ quy tắc cấu trúc dữ liệu cho trước:** file được chia thành các dòng bao gồm:
 - **States:** dòng thứ nhất, tức `lines[0]`, chứa các trạng thái.
 - **Alphabet:** dòng thứ hai, tức `lines[1]`, 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 NFA_ϵ

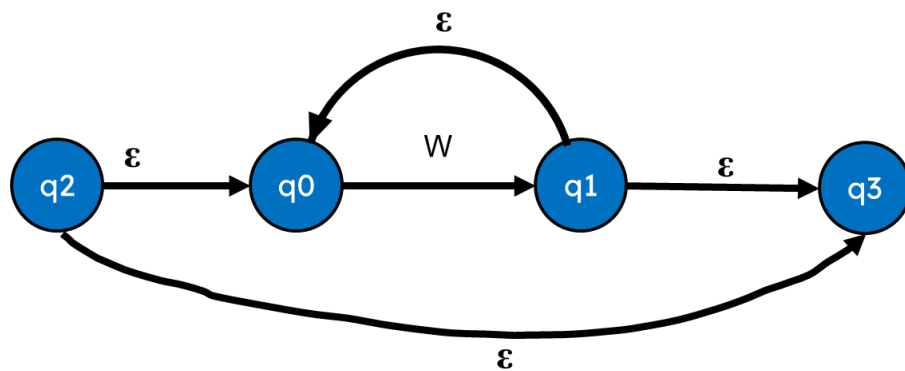
Sau khi người dùng nhập regex và xác nhận, chương trình sẽ khởi tạo NFA_ϵ và cho phép thực hiện kiểm tra chuỗi.

- **Thêm ký tự nối kết vào regex:** Duyệt qua từng ký tự trong regex và thêm ký tự nối kết `'.'`.
 - Ví dụ:
 - `abc` \rightarrow `a.b.c`
 - `01*|1` \rightarrow `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 `'|'`, `'.'`, `'*'`:

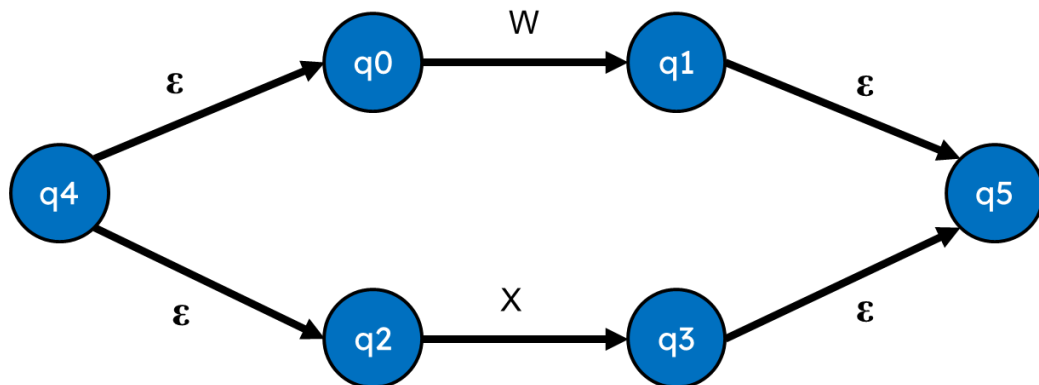
- Thực hiện vòng lặp kiểm tra, nếu ở đỉnh stack là toán tử, mà nó có độ ưu tiên **lớn hơn hoặc bằng** toán tử hiện tại thì ta lấy toán tử đó ra (pop) khỏi stack và push vào output.
- Push toán tử hiện tại vào stack.
- Nếu gặp dấu '(', ta push vào stack.
- Nếu gặp dấu ')', ta thực hiện vòng lặp lấy các 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:
 - $0|1 \rightarrow 01|$
 - $0.1*|1 \rightarrow 01*.1|$
- **Chuyển đổi regex ở dạng postfix sang NFA_ϵ :** Duyệt qua từng ký tự trong regex:
 - Nếu là ký tự nhập:



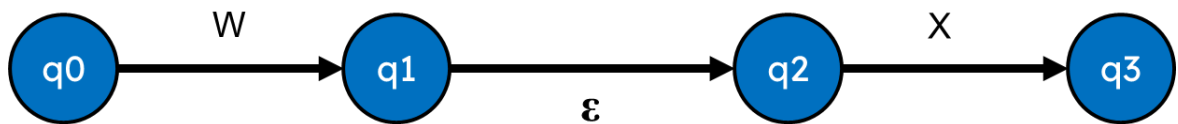
- Nếu là ký tự '*' (phép bao đóng):



- Nếu là ký tự '|' (phép hợp):



- Nếu là ký tự '.' (phép nối kết):



5. Giao diện thực hiện chức năng chương trình

Chương trình sẽ thực hiện việc kiểm tra chuỗi đầu vào có được NFA_{ϵ} 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 NFA_{ϵ} :** 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 NFA_{ϵ} :** sau khi người dùng nhập regex và xác nhận, chương trình sẽ khởi tạo NFA_{ϵ} và cho phép thực hiện kiểm tra chuỗi.
- **Chức năng xem đồ thị NFA_{ϵ} :** 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 NFA_{ϵ} hoạt động.

IV. 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 NFA ϵ 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 NFA ϵ một cách trực quan.

1. 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 NFA ϵ .
- **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 NFA ϵ . 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 NFA ϵ , hỗ trợ trực quan hóa các trạng thái và chuyển đổi của NFA ϵ . 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 NFA ϵ 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.

2. Cấu trúc lưu trữ NFA ϵ

Cấu trúc lưu trữ của NFA ϵ 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.

```
1 States: q0, q1, q2
2 Alphabet: 0, 1, 2, e
3 Start: q0
4 Accept: q2
5 Transition:
6 q0 0 q0
7 q0 e q1
8 q1 1 q1
9 q1 e q2
10 q2 2 q2
```

3. Lớp State

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

```
1 # Class State để tạo các trạng thái khác nhau
2 class State:
3     counter = 0
4
5     def __init__(self):
6         self.id = f"q{State.counter}"
7         State.counter += 1
8
9     @classmethod
10    def reset_counter(cls):
11        # Reset counter về 0
12        cls.counter = 0
13
14    def __str__(self):
15        return str(self.id)
16
17    def __repr__(self):
18        return str(self.id)
19
20    def __hash__(self):
21        return hash(self.id)
22
23    def __eq__(self, other):
24        return isinstance(other, State) and self.id == other.id
```

4. Lớp NFA ϵ

Lớp NFA ϵ là thành phần cốt lõi của chương trình, chịu trách nhiệm xử lý liên quan đến NFA ϵ . 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 NFA ϵ hay không.

- **Các thuộc tính:**

- **states:** Tập hợp tất cả các trạng thái của NFA ϵ .
- **alphabet:** Tập hợp các ký tự đầu vào mà NFA ϵ 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 δ , 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).

```
1 class NFAe:
2     def __init__(self, steps_display):
3         self.states = set() # Tập trạng thái Q
4         self.alphabet = set() # Ký tự nhập
5         self.start_state = None # Trạng thái bắt đầu
6         self.accept_states = set() # Trạng thái kết thúc
7         self.transitions = {} # Hàm chuyển đổi delta
8         self.steps_display = steps_display # Log các bước thực hiện
```

- **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.

```
1 # Tìm E-closure của 1 trạng thái
2 def e_closure(self, states):
3     closure = set(states)
4     stack = list(states)
5
6     while stack:
7         current = stack.pop()
8         for next_state in self.transitions.get((current, 'e'), []):
9             if next_state not in closure:
10                 closure.add(next_state)
11                 stack.append(next_state)
12
13     self.steps_display.insert(
14         tk.END,
15         f"Trạng thái mới sau khi tính e-closure: {closure if closure else '∅'}\n"
16     )
17     return closure
```

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

```
1 # Chuyển trạng thái hiện tại sang trạng thái mới trên nhãn symbol
2 def move(self, states, symbol):
3     new_states = set()
4     for state in states:
5         new_states.update(self.transitions.get((state, symbol), []))
6     self.steps_display.insert(
7         tk.END,
8         f"Trạng thái mới trên nhãn {symbol}: {new_states if new_states else '∅'}\n"
9     )
10    return new_states
```

- **accepts(self, string):** Kiểm tra xem chuỗi đầu vào có được chấp nhận bởi NFA ϵ không. Phương thức này sử dụng epsilon_closure và move để duyệt qua các trạng thái.

```
def accepts(self, input_string):
    # Tính delta* của q0 trên nhãn e
    current_states = self.e_closure({self.start_state})
    self.steps_display.delete(1.0, tk.END)
    self.steps_display.insert(tk.END, f"Trạng thái bắt đầu: {current_states}\n")

    # Chuyển trạng thái trên từng ký tự đầu vào
    for symbol in input_string:
        self.steps_display.insert(tk.END, f"Ký tự đang xét: {symbol}\n")
        current_states = self.e_closure(self.move(current_states, symbol))
    # Kiểm tra trạng thái cuối cùng
    if not current_states.isdisjoint(self.accept_states):
        self.steps_display.insert(
            tk.END,
            f"Trạng thái cuối cùng: {current_states if current_states else ''}\nChuỗi được chấp nhận vì trạng thái cuối cùng ({current_states}) chứa trạng thái kết thúc {current_states & self.accept_states}.\n"
        )
        return True
    else:
        self.steps_display.insert(
            tk.END,
            f"Trạng thái cuối cùng: {current_states if current_states else ''}\nChuỗi không được chấp nhận vì trạng thái cuối cùng không chứa trạng thái kết thúc {self.accept_states}.\n"
        )
    return False
```

5. Hàm hỗ trợ

- **Hàm `read_NFAe_from_file`:** Đọc dữ liệu từ file và khởi tạo đối tượng NFA ϵ .

```
1 def read_NFAe_from_file(filename, steps_display):
2     nfae = NFAe(steps_display=steps_display)
3     with open(filename, 'r') as file:
4         lines = file.readlines()
5
6         states = lines[0].split(":")[1].strip().split(",")
7         alphabet = lines[1].split(":")[1].strip().split(",")
8         start_state = lines[2].split(":")[1].strip()
9         accept_states = lines[3].split(":")[1].strip().split(",")
10
11         # Nạp trạng thái
12         for state in states:
13             nfae.add_state(state.strip())
14         # Nạp ký tự đầu vào
15         nfae.alphabet = alphabet
16         # Nạp trạng thái bắt đầu
17         nfae.add_state(start_state, is_start=True)
18         # Nạp trạng thái kết thúc
19         for state in accept_states:
20             nfae.add_state(state, is_accept=True)
21         # Nạp hàm chuyển delta
22         for line in lines[5:]:
23             from_state, symbol, to_state = line.split()
24             nfae.add_transition(from_state, symbol, to_state)
25
26     return nfae
```

- **Hàm `draw_nfae`:** Vẽ sơ đồ trạng thái NFA ϵ và lưu thành file ảnh bằng thư viện graphviz.

```
1 def draw_NFAe(nfae):
2     dot = Digraph(format="png")
3     dot.attr(rankdir="LR")
4     for state in nfae.states:
5         if state in nfae.accept_states:
6             dot.node(str(state), shape="doublecircle")
7         else:
8             dot.node(str(state), shape="circle")
9
10    dot.node("start", shape="point")
11    dot.edge("start", str(nfae.start_state))
12
13    for (from_state, symbol), to_states in nfae.transitions.items():
14        for to_state in to_states:
15            label = " $\epsilon$ " if symbol == "" else symbol
16            dot.edge(str(from_state), str(to_state), label=label)
17
18    output_file = r"./diagram"
19    try:
20        dot.render(output_file, cleanup=True)
21        return output_file + ".png"
22    except Exception as e:
23        print("Lỗi vẽ đồ thị: ", e)
24        return None
```

- **Hàm `regex_to_postfix`:** Chuyển regex từ infix sang postfix.

```
1 def regex_to_postfix(regex):
2     """Chuyển regex từ infix sang postfix (RPN)"""
3     precedence = {'|': 1, '.': 2, '*': 3} # Xếp các toán tử theo thứ tự ưu tiên
4     output = []
5     operators = []
6
7     def add_concat_operator(regex):
8         """Thêm phép nối kết cho regex"""
9         result = []
10        for i, token in enumerate(regex):
11            result.append(token)
12            if token not in '(|' and i + 1 < len(regex) and regex[i + 1] not in '|)*':
13                result.append('.')
14        return result
15
16    regex = add_concat_operator(regex)
17
18    for char in regex:
19        if char.isalnum(): # Toán hạng
20            output.append(char)
21        elif char == '(':
22            operators.append(char)
23        elif char == ')':
24            while operators and operators[-1] != '(':
25                output.append(operators.pop())
26            operators.pop() # Xóa '('
27        else: # Toán tử
28            while (operators and operators[-1] != '(' and
29                  precedence[operators[-1]] >= precedence[char]):
30                output.append(operators.pop())
31            operators.append(char)
32
33    while operators:
34        output.append(operators.pop())
35
36    return ''.join(output)
```

- **Hàm `create_NFAe_from_regex_postfix`:** Chuyển regex ở dạng postfix sang NFA ϵ .

```
1 def create_NFAe_from_regex_postfix(postfix, steps_display) -> NFAe:
2     nfa_stack = []
3     for char in postfix:
4         if char.isalnum(): # Ký tự thuần
5             nfa = NFAe(steps_display)
6             start = State()
7             accept = State()
8             nfa.add_state(start, is_start=True)
9             nfa.add_state(accept, is_accept=True)
10            nfa.add_transition(start, char, accept)
11            nfa_stack.append(nfa)
12        elif char == '*': # Phép bao đóng
13            nfa = nfa_stack.pop()
14            start = State()
15            accept = State()
16            nfa.add_transition(start, "e", nfa.start_state)
17            nfa.add_transition(start, "e", accept)
18            for accept_state in nfa.accept_states:
19                nfa.add_transition(accept_state, "e", nfa.start_state)
20                nfa.add_transition(accept_state, "e", accept)
21
22            nfa.add_state(start, is_start=True)
23            nfa.reset_accept_states()
24            nfa.add_state(accept, is_accept=True)
25            nfa_stack.append(nfa)
```

```
1     elif char == '|': # Phép hợp
2         nfa2 = nfa_stack.pop()
3         nfa1 = nfa_stack.pop()
4         nfa = NFAe(steps_display)
5         start = State()
6         accept = State()
7         nfa.add_state(start, is_start=True)
8         nfa.add_state(accept, is_accept=True)
9         nfa.add_transition(start, "e", nfa2.start_state)
10        nfa.add_transition(start, "e", nfa1.start_state)
11        for accept_state in nfa1.accept_states:
12            nfa.add_transition(accept_state, "e", accept)
13        for accept_state in nfa2.accept_states:
14            nfa.add_transition(accept_state, "e", accept)
15        nfa.states.update(nfa1.states)
16        nfa.states.update(nfa2.states)
17        nfa.transitions.update(nfa1.transitions)
18        nfa.transitions.update(nfa2.transitions)
19        nfa_stack.append(nfa)
20
21    elif char == '.': # Phép nối kết
22        nfa2 = nfa_stack.pop()
23        nfa1 = nfa_stack.pop()
24        for accept_state in nfa1.accept_states:
25            nfa1.add_transition(accept_state, "e", nfa2.start_state)
26        nfa1.states.update(nfa2.states)
27        nfa1.transitions.update(nfa2.transitions)
28        nfa1.accept_states = nfa2.accept_states
29        nfa_stack.append(nfa1)
30
31    return nfa_stack.pop()
```

6. 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.

```
1 class NFAeMenu:
2     def __init__(self, root):
3         self.root = root
4         self.root.title("Kiểm tra NFAe")
5
6         self.nfae = None
7         self.root.configure(bg=BG_COLOR)
8         self.root.geometry(f"{DEFAULT_WIDTH}x{DEFAULT_HEIGHT}") # Kích thước cửa sổ
9
10        # Main Label
11        self.label = tk.Label(root, text="Kiểm tra NFAe", font=("Helvetica", 30, "bold"))
12        self.label.pack(pady=10)
13
14        # Load NFAe Button
15        self.load_NFAe_button = tk.Button(
16            root, text="Nạp NFAe từ File", font=BUTTON_FONT,
17            bg=BUTTON_COLOR, fg=BUTTON_TEXT_COLOR,
18            command=self.load_NFAe
19        )
20        self.load_NFAe_button.pack(pady=10)
21
22        # Regex String Entry
23        frame = tk.Frame(root)
24        frame.pack(pady=10)
25        self.regex_label = tk.Label(frame, text="Hoặc nhập biểu thức chính quy:", font=NORMAL_FONT)
26        self.regex_label.pack(padx=5)
27
28        self.regex_entry = tk.Entry(frame, width=25, font=NORMAL_FONT)
29        self.regex_entry.pack(side=tk.LEFT, padx=5)
30
31        submit_button = tk.Button(
32            frame,
33            text="Submit",
34            font=BUTTON_TEXT_COLOR,
35            bg=BUTTON_COLOR,
36            fg=BUTTON_TEXT_COLOR,
37            command=self.submit_regex)
38        submit_button.pack(side=tk.LEFT, padx=5)
```



```
1 # View NFAe Button
2 self.view_NFAe_button = tk.Button(
3     root, text="Xem NFAe", font=BUTTON_FONT,
4     bg=BUTTON_COLOR, fg=BUTTON_TEXT_COLOR,
5     command=self.show_NFAe_diagram
6 )
7 self.view_NFAe_button.pack(pady=10)
8
9 # Input String Entry
10 self.input_label = tk.Label(root, text="Nhập chuỗi để kiểm tra:", font=NORMAL_FONT)
11 self.input_label.pack(pady=20)
12
13 self.input_entry = tk.Entry(root, width=25, font=NORMAL_FONT)
14 self.input_entry.pack(pady=5)
15
16 # Check Button
17 self.check_button = tk.Button(
18     root, text="Kiểm tra", font=BUTTON_FONT,
19     bg=BUTTON_COLOR, fg=BUTTON_TEXT_COLOR,
20     command=self.check_string
21 )
22 self.check_button.pack(pady=10)
23
24 # Result Label
25 self.result_label = tk.Label(root, text="", font=NORMAL_FONT, fg="blue")
26 self.result_label.pack(pady=10)
27
28 # Step display Text
29 self.steps_display = tk.Text(root, height=10, width=60, font=NORMAL_FONT, state="disabled")
30 self.steps_display.pack(pady=10)
```

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

```
1 def load_NFAe(self):
2     file_path = filedialog.askopenfilename(filetypes=[("Text Files", "*.txt")])
3     if file_path:
4         try:
5             self.nfae = read_NFAe_from_file(file_path, self.steps_display)
6             messagebox.showinfo("Success", "Nạp thành công NFAe!")
7             draw_nfae(self.nfae)
8         except Exception as e:
9             traceback.print_exception(e)
10            messagebox.showerror("Error", f"Nạp thất bại NFAe: {e}")
```

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

```
1 def submit_regex(self):
2     regex = self.regex_entry.get()
3     if not regex:
4         messagebox.showwarning("Warning", "Biểu thức chính quy không được rỗng!")
5         return
6     try:
7         State.reset_counter()
8         self.nfae = regex_to_NFAe(regex, self.steps_display)
9         self.reset_GUI()
10        messagebox.showinfo("Success", "Nạp biểu thức chính quy thành công!")
11        draw_NFAe(self.nfae)
12    except Exception as e:
13        traceback.print_exception(e)
14        messagebox.showerror("Error", f"Nạp biểu thức chính quy thất bại: {e}")
```

- **Hàm check_string:** Hàm này kiểm tra xem một chuỗi nhập vào có được NFA ϵ chấp nhận hay không và hiển thị kết quả cho người dùng thông qua giao diện.

```
1 def check_string(self):
2     self.steps_display.config(state="normal")
3     if not self.nfae:
4         messagebox.showwarning("Warning", "Hãy nạp NFAe trước!")
5         return
6
7     input_string = self.input_entry.get()
8     if self.nfae.accepts(input_string):
9         self.result_label.config(text="NFAe CHẤP NHẬN chuỗi đã nhập.", fg="green")
10    else:
11        self.result_label.config(text="NFAe KHÔNG CHẤP NHẬN chuỗi đã nhập.", fg="red")
12    self.steps_display.config(state="disabled")
```

- **Hàm show_NFAe_diagram:** có nhiệm vụ hiển thị sơ đồ trạng thái của NFA ϵ dưới dạng hình ảnh cho người dùng xem trên giao diện.

Đề tài: Xây dựng NFA ϵ và kiểm tra một chuỗi có thuộc NFA ϵ đã cho không

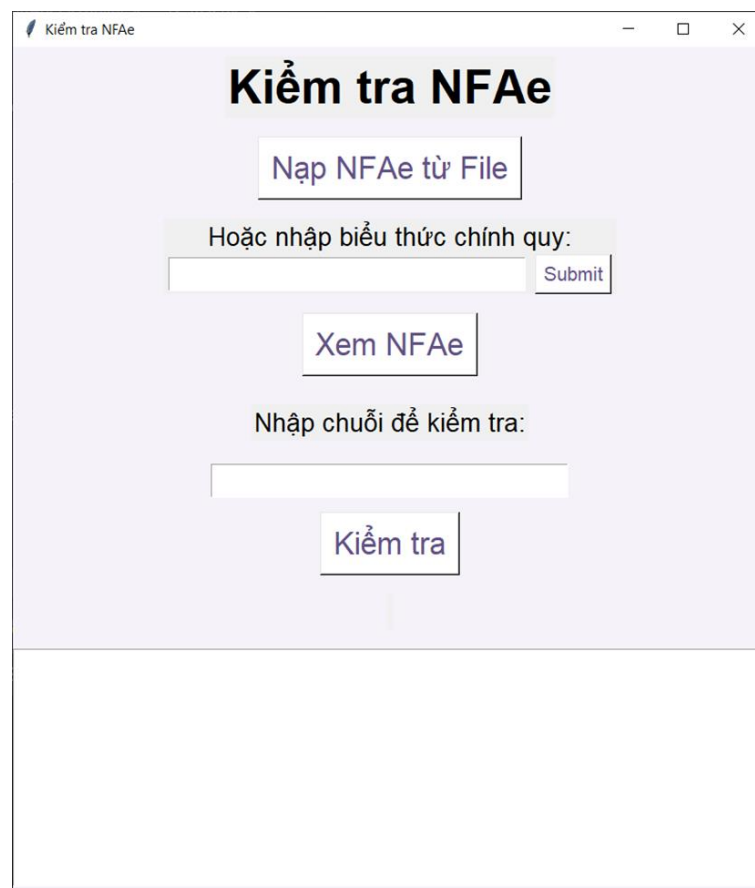
```
1 def show_NFAe_diagram(self, image_path="./diagram.png"):
2     if not self.nfae:
3         messagebox.showwarning("Warning", "Hãy nạp NFAe trước!")
4         return
5
6     diagram_window = tk.Toplevel(self.root)
7     diagram_window.title("NFAe Diagram")
8     try:
9         if not os.path.exists(image_path):
10             raise FileNotFoundError(f"Không tồn tại hình ảnh tại đường dẫn: {image_path}")
11
12         # Mở hình ảnh với Pillow
13         img = Image.open(image_path)
14
15         # Chuyển đổi thành đối tượng mà Tkinter có thể sử dụng
16         img = ImageTk.PhotoImage(img)
17
18         # Tạo Label để hiển thị hình ảnh
19         label = tk.Label(diagram_window, image=img)
20         label.image = img # Lưu tham chiếu để tránh bị thu hồi
21         label.pack()
22
23     except Exception as e:
24         print(f"Lỗi tải hình ảnh: {e}")
25         error_label = tk.Label(diagram_window, text=f"Không thể tải hình ảnh: {e}")
26         error_label.pack()
```

V. 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 NF ϵ , kiểm tra chuỗi đầu vào, và xem sơ đồ trạng thái của NF ϵ .

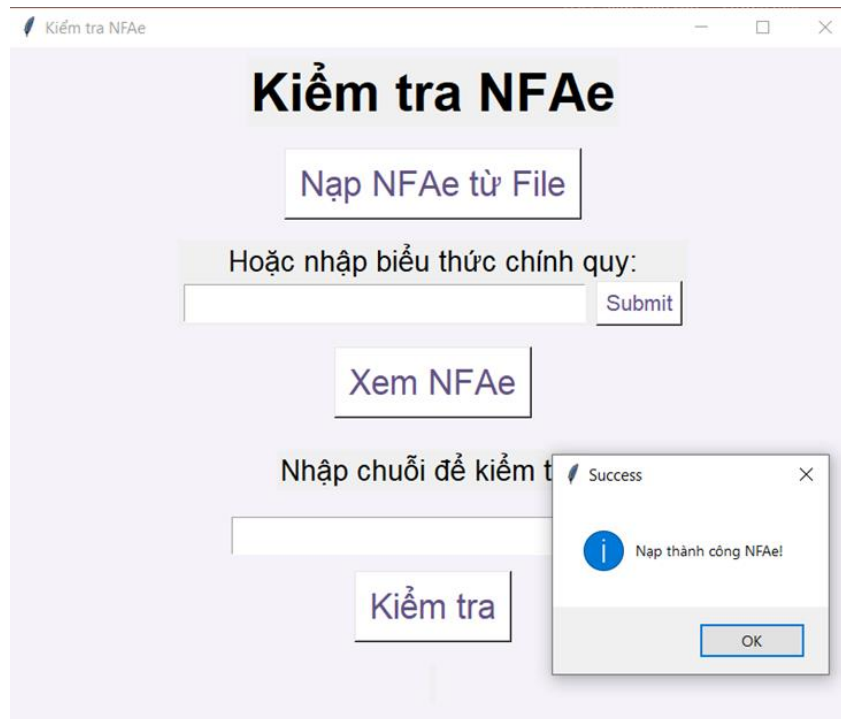
Các chức năng chính như tải file NF ϵ , kiểm tra chuỗi, hiển thị bước kiểm tra chuỗi, và vẽ sơ đồ trạng thái đều được tích hợp và hoạt động mượt mà trên giao diện.



2. Xử lý file dữ liệu

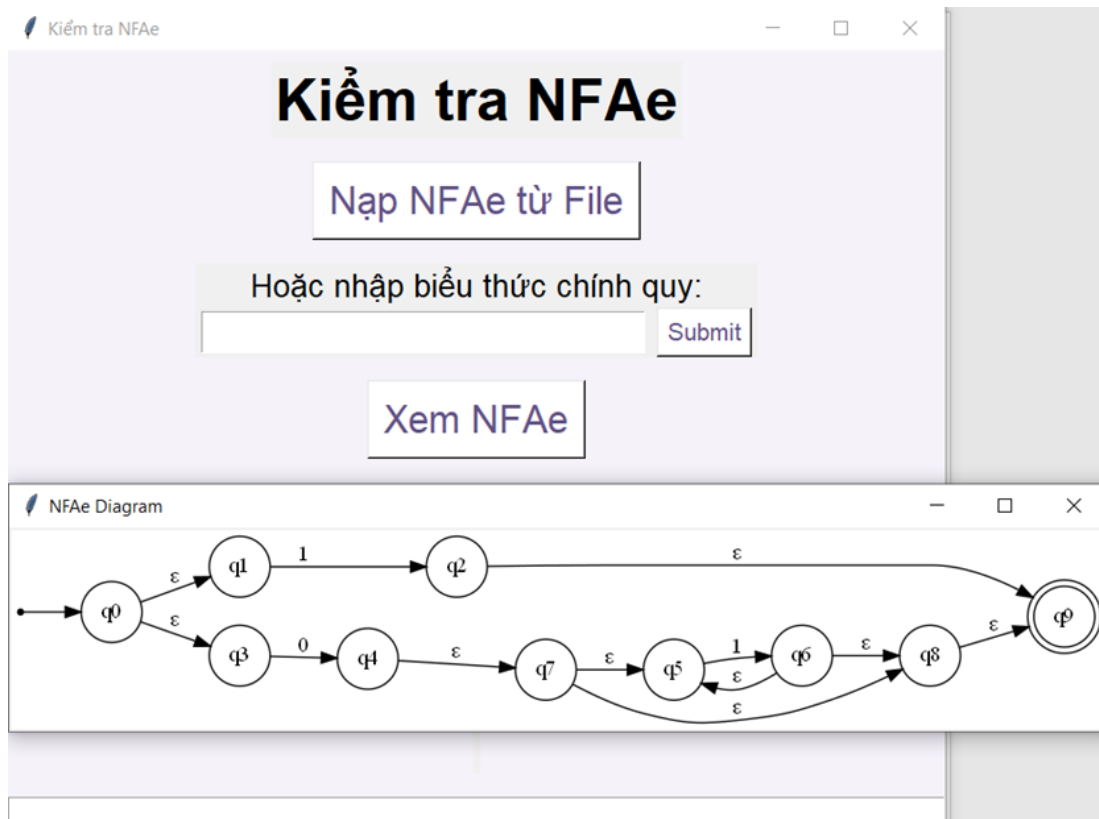
Người dùng có thể tải một file dữ liệu chứa mô tả NF \mathcal{A}_ϵ (theo định dạng txt), chương trình sẽ đọc và phân tích file này để xây dựng đối tượng NF \mathcal{A}_ϵ .

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. Hiện thị sơ đồ trạng thái NFA ϵ

Chương trình có thể vẽ sơ đồ trạng thái của NFA ϵ 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 NFA ϵ .



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 NFA ϵ đã cho hay không.

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

- **Chuỗi được chấp nhận:**

The screenshot shows a web application titled "Kiểm tra NFAε". It has a light purple background. At the top, there's a header "Kiểm tra NFAε" with a small icon. Below the header, there are two main input methods: "Nạp NFAε từ File" (Load NFAε from File) and "Hoặc nhập biểu thức chính quy:" (Or enter regular expression:). The regular expression input field contains "01*1" and has a "Submit" button next to it. Below this, there's a "Xem NFAε" (View NFAε) button. Further down, there's a section "Nhập chuỗi để kiểm tra:" (Enter string to check:) with an input field containing "01" and a "Kiểm tra" (Check) button. At the bottom, a green message states "NFAε CHẤP NHẬN chuỗi đã nhập." (NFAε ACCEPTS the entered string.).

Trạng thái bắt đầu: {q8, q6, q0}

Ký tự đang xét: 0

Trạng thái mới trên nhãn 0: {q1}

Trạng thái mới sau khi tính e-closure: {q1, q5, q4, q9, q2}

Ký tự đang xét: 1

Trạng thái mới trên nhãn 1: {q3}

Đề tài: Xây dựng NFA ϵ và kiểm tra một chuỗi có thuộc NFA ϵ đã cho không

- **Chuỗi bị từ chối:**

Kiểm tra NFA ϵ

Kiểm tra NFA ϵ

Nạp NFA ϵ từ File

Hoặc nhập biểu thức chính quy:
01*11

Submit

Xem NFA ϵ

Nhập chuỗi để kiểm tra:
010

Kiểm tra

NFA ϵ KHÔNG CHẤP NHẬN chuỗi đã nhập.

Trạng thái bắt đầu: {q8, q6, q0}
Ký tự đang xét: 0
Trạng thái mới trên nhãn 0: {q1}
Trạng thái mới sau khi tính e-closure: {q1, q5, q4, q9, q2}
Ký tự đang xét: 1
Trạng thái mới trên nhãn 1: {q3}

VI. 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 NFA_ϵ :** Hiện tại, hệ thống chỉ hỗ trợ đọc dữ liệu NFA_ϵ 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 NFA_ϵ 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 máy tự động khác ngoài NFA_ϵ , như DFA (Deterministic Finite Automata), hoặc hỗ trợ việc chuyển đổi giữa các loại tự động hóa này.

VII. 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

([Reverse Polish Notation](#))