BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC CẦN THƠ**

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

**🙞 🕮 🙜**



**BÁO CÁO**

**TIN HỌC LÝ THUYẾT (CT12101)**

**Đề tài**

**DEMO CHUYỂN NFAԑ SANG DFA TƯƠNG ĐƯƠNG**

**NHÓM BÁO CÁO 24**

|  |  |  |
| --- | --- | --- |
| **SỐ THỨ TỰ TRONG DANH SÁCH** | **MSSV** | **HỌ TÊN** |
| **24** | **B2113329** | **Nguyễn Hoàng Điển** |
| **27** | **B2113341** | **Phan Minh Tài** |
| **30** | **B2113347** | **Nguyễn Trần Tuyết Vân** |

**HK 2, NH 2023 - 2024**

Cần Thơ, 4/2024

NHẬN XÉT CỦA GIÁO VIÊN

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

LỜI CẢM ƠN

Cần thơ, ngày tháng năm

Người viết

Họ tên sinh viên

MỤC LỤC

[PHẦN I: GIỚI THIỆU 12](#_Toc164868614)

[1. Đặt vấn đề 12](#_Toc164868615)

[2. Lịch sử giải quyết vấn đề 12](#_Toc164868616)

[3. Mục tiêu nghiên cứu 12](#_Toc164868617)

[4. Đối tượng và phạm vi nghiên cứu 12](#_Toc164868618)

[4.1. Đối tượng nghiên cứu 12](#_Toc164868619)

[4.2. Phạm vi nghiên cứu 13](#_Toc164868620)

[5. Phương pháp nghiên cứu 13](#_Toc164868621)

[5.1. Kiến thức 13](#_Toc164868622)

[5.2. Công cụ hỗ trợ 13](#_Toc164868623)

[5.3. Phương pháp giải quyết vấn đề 13](#_Toc164868624)

[6. Kết quả đạt được 13](#_Toc164868625)

[7. Bố cục bài báo cáo 13](#_Toc164868626)

[PHẦN II: NỘI DUNG 14](#_Toc164868627)

[CHƯƠNG 1 14](#_Toc164868628)

[GIỚI THIỆU TỔNG QUAN VỀ BÀI TOÁN 14](#_Toc164868629)

[1.1. Tại sao phải chuyển đổi NFAԑ sang DFA tương đương? 14](#_Toc164868630)

[1.2. Mô tả chi tiết về bài toán 14](#_Toc164868631)

[1.3. Giải thuật và phương pháp giải quyết bài toán 15](#_Toc164868632)

[1.3.1. Chi tiết hướng giải quyết 15](#_Toc164868633)

[1.3.2. Phân tích 15](#_Toc164868634)

[1.3.3. Giải thuật 16](#_Toc164868635)

[CHƯƠNG 2 17](#_Toc164868636)

[THIẾT KẾ VÀ CÀI ĐẶT 17](#_Toc164868637)

[2.1. Ngôn ngữ lập trình và thư viện 17](#_Toc164868638)

[2.1.1. Ngôn ngữ lập trình Python là gì? 17](#_Toc164868639)

[2.1.2. Đặc điểm của ngôn ngữ Python 17](#_Toc164868640)

[2.1.3. Ứng dụng của ngôn ngữ Python 17](#_Toc164868641)

[2.1.4. Thư viện Streamlit 17](#_Toc164868642)

[2.1.5. Thư viện Automathon 18](#_Toc164868643)

[2.1.6. Thư viện Graphviz 18](#_Toc164868644)

[2.1.7. Thư viện Pandas 18](#_Toc164868645)

[2.2. Thiết kế chương trình 18](#_Toc164868646)

[2.2.1. Xây dựng tập trạng thái DFA từ NFAԑ 19](#_Toc164868647)

[2.2.2. Ví dụ minh họa 19](#_Toc164868648)

[2.3. Cài đặt chương trình 21](#_Toc164868649)

[2.3.1. Phương thức khởi tạo DFA 21](#_Toc164868650)

[2.3.2. Phương thức khởi tạo NFAԑ 22](#_Toc164868651)

[2.3.3. Phương thức đọc NFAԑ đầu vào 22](#_Toc164868652)

[2.3.4. Phương thức hiển thị NFAԑ 24](#_Toc164868653)

[2.3.5. Phương thức tính ԑ-closure 26](#_Toc164868654)

[2.3.6. Phương thức dịch chuyển 27](#_Toc164868655)

[2.3.7. Phương thức chuyển đổi 27](#_Toc164868656)

[2.3.8. Hàm sử dụng các phương thức 29](#_Toc164868657)

[2.3.8. Các hàm chuyển đổi kiểu dữ liệu để xây dựng đồ thị 30](#_Toc164868658)

[2.4. Giao diện hệ thống 31](#_Toc164868659)

[2.4.1. Giới thiệu hệ thống 31](#_Toc164868660)

[2.4.2. Cài đặt hệ thống 33](#_Toc164868661)

[CHƯƠNG 3 37](#_Toc164868662)

[KIỂM THỬ VÀ ĐÁNH GIÁ 37](#_Toc164868663)

[3.1. Mục tiêu kiểm thử 37](#_Toc164868664)

[3.2. Kịch bản kiểm thử 37](#_Toc164868665)

[3.3. Kết quả kiểm thử 37](#_Toc164868666)

[PHẦN III: KẾT LUẬN 38](#_Toc164868667)

[1. Kết luận 38](#_Toc164868668)

[2. Hướng phát triển đề tài 38](#_Toc164868669)

[TÀI LIỆU THAM KHẢO 39](#_Toc164868670)

DANH MỤC HÌNH

[Hình 2.1 NFA với ԑ - dịch chuyển 19](#_Toc164786182)

[Hình 2.2 Sơ đồ chuyển của DFA 20](#_Toc164786183)

[Hình 2.3 Phương thức khởi tạo DFA 21](#_Toc164786184)

[Hình 2.4 Phương thức khởi tạo NFAԑ 22](#_Toc164786185)

[Hình 2.5 Phương thức đọc NFAԑ đầu vào từ file 23](#_Toc164786186)

[Hình 2.6 Ví dụ file dữ liệu đầu vào 23](#_Toc164786187)

[Hình 2.7 Đọc NFAԑ từ bàn phím 24](#_Toc164786188)

[Hình 2.8 Phương thức hiển thị NFAԑ 25](#_Toc164786189)

[Hình 2.9 Phương thức tính ԑ-closure 26](#_Toc164786190)

[Hình 2.10 Phương thức dịch chuyển 27](#_Toc164786191)

[Hình 2.11 Phương thức chuyển đổi 27](#_Toc164786192)

[Hình 2.13 Hàm sử dụng các phương thức 29](#_Toc164786193)

[Hình 2.14 Hàm sử dụng các phương thức 29](#_Toc164786194)

[Hình 2.15 Hàm sử dụng các phương thức 30](#_Toc164786195)

[Hình 2.16 Hàm sử dụng các phương thức 31](#_Toc164786196)

[Hình 2.17 Giao diện hệ thống 32](#_Toc164786197)

[Hình 2.18 Upload file NFAε 32](#_Toc164786198)

[Hình 2.19 Ô nhập các thành phần của NFAε 33](#_Toc164786199)

[Hình 2.20 Hệ thống thông báo lỗi khi nhập thiếu Q0 33](#_Toc164786200)

[Hình 2.21 Hệ thống thông báo lỗi nhập sai định dạng 34](#_Toc164786201)

[Hình 2.22 Nhập NFAε từ bàn phím 34](#_Toc164786202)

[Hình 2.23 Đồ thị của NFAε 35](#_Toc164786203)

[Hình 2.24 Các bước xây dựng trạng thái cho DFA 35](#_Toc164786204)

[Hình 2.25 Các thành phần của DFA 36](#_Toc164786205)

[Hình 2.26 Đồ thị của DFA 36](#_Toc164786206)

DANH MỤC BẢNG

[Bảng 2.1 Bảng hàm chuyển của DFA 18](#_Toc164467270)

TÓM TẮT

ABSTRACT

DANH MỤC TỪ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| Từ viết tắt | Từ tiếng anh | Nghĩa tiếng việt |
|  |  |  |
| FA | Finite Automata | Ôtômát hữu hạn |
| DFA | Deterministic Finite Automata | Ôtômát hữu hạn đơn định |
| NFA | Nondeterministic Finite Automata | Ôtômát hữu hạn không đơn định |
| NFAԑ | Nondeterministic Finite Automata with  epsilon-transitions | NFA với ԑ-dịch chuyển |
| RE | Regular Expressions | Biểu thức chính quy |
| VSC | Visual Studio Code | Phần mềm lập trình Visual Studio Code |

PHẦN I: GIỚI THIỆU

1. Đặt vấn đề

Chuyển đổi từ NFAԑ sang DFA là một kiến thức quan trọng trong học phần tin học lý thuyết. Mặc dù lớp ngôn ngữ điều có thể được chấp nhận bởi NFAԑ và DFA nhưng cấu trúc và tính chất là khác nhau nên có thể vận dụng cho nhiều trường hợp khác nhau. Đây là bài toán đòi hỏi có kiến thức về DFA, NFA và NFAԑ, cùng với kỹ năng vận dụng các kiến thức trên nhằm xử lý các bài toán khác nhau về việc chuyển đổi từ NFAԑ sang DFA tương đương.

2. Lịch sử giải quyết vấn đề

Trong cuốn sách có tên “Introduction to Automata Theory, Languages and Computation” của John E. Hopcroft và Jeffrey D. Ullman được xuất bản vào năm 1979 đã đề cập đến vấn đề chuyển đổi từ NFAԑ sang DFA tương đương cũng như đưa ra phương pháp chuyển đổi.

Ngày nay, với sự phát triển của công nghệ thông tin, các phương pháp chuyển đổi từ NFAε sang DFA cũng như là các FA hay RE đã được tích hợp vào nhiều công cụ phần mềm và thư viện, giúp cho quá trình này diễn ra một cách dễ dàng và hiệu quả hơn.

3. Mục tiêu nghiên cứu

Mục tiêu của đề tài là dựa vào các ngôn ngữ lập trình đã học, xây dựng lên hệ thống chuyển đổi từ NFAε sang DFA tương đương, có thể đón nhận ngôn ngữ mà NFAε đã đón nhận trước đó.

4. Đối tượng và phạm vi nghiên cứu

4.1. Đối tượng nghiên cứu

DFA gồm một tập hữu hạn các trạng thái và một tập các phép chuyển từ trạng thái này tới trạng thái khác trên các ký hiệu nhập (input symbols) được chọn từ tập một chữ cái Σ nào đó. Mỗi ký hiệu có đúng một phép chuyển khỏi mỗi trạng thái (có thể quay về chính nó). Một trạng thái, thường được gọi là q0, gọi là trạng thái bắt đầu (trạng thái automata bắt đầu). Một số trạng thái được thiết kế như trạng thái kết thúc hay trạng thái chấp nhận.

NFAε là mô hình được mở rộng từ mô hình DFA cho phép chấp nhận 0, 1, 2,… các phép chuyển từ một trạng thái trên cùng một ký hiệu nhập và cho phép các phép chuyển trên nhãn rỗng ԑ.

4.2. Phạm vi nghiên cứu

Đề tài tập trung vào nghiên cứu và xây dựng thuật toán trực quan và hiệu quả trong quá trình thực hiện chuyển đổi NFAε sang DFA tương đương.

5. Phương pháp nghiên cứu

5.1. Kiến thức

- Kiến thức về phương pháp chuyển đổi NFAε sang DFA tương đương.

- Kiến thức về ngôn ngữ lập trình Python.

5.2. Công cụ hỗ trợ

Demo nghiên cứu được viết trên công cụ Visual Studio Code.

5.3. Phương pháp giải quyết vấn đề

- Dựa vào các giải thuật đã được học trong giáo trình thiết kế giải thuật bài toán.

- Sử dụng công cụ VSC để lập trình giải thuật trên và chạy thử nghiệm kết quả.

6. Kết quả đạt được

Sử dụng ngôn ngữ lập trình Python được viết trên công cụ VSC để tạo demo xây dựng giải thuật chuyển đổi NFAε sang DFA tương đương. Trong đó bao gồm tập các trạng thái bắt đầu, trạng thái kết thúc, ký hiệu nhập, tập hợp các trạng thái và hàm chuyển các trạng thái qua ký hiệu nhập của NFAԑ; từ đó xây dựng DFA bao gồm tập các trạng thái bắt đầu, trạng thái kết thúc, ký hiệu nhập, tập hợp các trạng thái và hàm chuyển các trạng thái qua ký hiệu nhập có thể nhận cùng một ngôn ngữ dựa trên NFAԑ đã cho từ trước.

7. Bố cục bài báo cáo

* **Phần I: Giới thiệu.**

Giới thiệu tổng quan về đề tài “Demo chuyển NFAԑ sang DFA tương đương”.

* **Phần II: Nội dung**

- Chương 1: Mô tả chi tiết, phân tích bài toán.

- Chương 2: Thiết kế, cài đặt bài toán, trình bày từng bước xây dựng chương trình.

- Chương 3: Kiểm thử chương trình, đánh giá độ chính xác và sự tối ưu của chương trình.

* **Phần III: Kết luận**

Chạy demo giải thuật, trình bày kết quả đạt được và đưa ra hướng phát triển cho đề tài trong tương lai.

PHẦN II: NỘI DUNG

CHƯƠNG 1

GIỚI THIỆU TỔNG QUAN VỀ BÀI TOÁN

1.1. Tại sao phải chuyển đổi NFAԑ sang DFA tương đương?

Qua khảo sát các dạng mở rộng từ mô hình automata hữu hạn ban đầu, ta thấy DFA thực chất là trường hợp đặc biệt của NFAԑ nhưng với mỗi trạng thái hiện tại khi qua một ký hiệu nhập, chỉ có duy nhất một đường truyền dẫn đến một trạng thái khác và không có sự truyền trên nhãn rỗng (truyền trên nhãn ԑ).

Việc chuyển đổi NFAԑ sang DFA tương đương là cần thiết vì những lý do sau:

- DFA thường hiệu quả hơn trong khi thực hiện vì trạng thái là duy nhất và xác định, khi gặp nhãn đầu vào, chỉ có một con đường duy nhất để tiếp tục, do đó giảm bớt sự không chắc chắn trong việc xác định trạng thái tiếp theo.

- DFA thường có cấu trúc đơn giản hơn NFAԑ, số lượng trạng thái trong DFA thường ít hơn hoặc bằng số lượng trạng thái trong NFAԑ, điều này giúp giảm bớt không gian lưu trữ cũng như tăng tốc độ xử lý.

- DFA dễ hiểu hơn vì nguyên tắc chuyển đổi rõ ràng và không có không gian cho sự mơ hồ hoặc nhiều sự lựa chọn như trong NFAԑ, cho nên việc kiểm tra chuỗi đầu vào trở nên dễ dàng hơn và ít phức tạp hơn.

Như vậy, việc chuyển đổi từ NFAԑ sang DFA tương đương là cần thiết và quan trọng trong một số tình huống cụ thể, điều này giúp cho khả năng giải quyết bài toán được tối ưu hóa đến mức có thể.

1.2. Mô tả chi tiết về bài toán

Giả sử mỗi trạng thái của DFA là một tập trạng thái của NFA, DFA dùng trạng thái của mình để lưu giữ tất cả các trạng thái của NFA đạt được sau khi NFA đọc một ký hiệu nhập. Như vậy sau khi đọc các ký hiệu nhập a1, a2, …, an, DFA ở trạng thái là tập con của các trạng thái thuộc NFA, đạt được khi NFA đi từ trạng thái bắt đầu theo một con đường nào đó có tên a1a2 … an. Số trạng thái của DFA lúc đó phải bằng số phần tử trong tập lũy thừa của số trạng thái NFA. Song, trên thực tế trường hợp xấu nhất này thường ít khi xảy ra. Các trạng thái thật sự dùng trong sơ đồ chuyển cho một DFA sẽ được xác định theo các phép chuyển trạng thái trên nhãn là mọi ký hiệu từ trạng thái bắt đầu của DFA và sau đó lần lượt bổ sung thêm vào tập trạng thái nếu như nó chưa có trong đó.

1.3. Giải thuật và phương pháp giải quyết bài toán

1.3.1. Chi tiết hướng giải quyết

**Đầu vào:** Một automata hữu hạn không đơn định, cho phép di chuyển trên nhãn rỗng NFA.

**Đầu ra:** Một automata hữu hạn đơn định DFA nhận dạng cùng ngôn ngữ như NFA.

**Phương pháp:** Xây dựng bảng hàm chuyển cho DFA mô phỏng đồng thời tất cả các chuyển dịch của NFA trên chuỗi nhập cho trước.

Ta dùng các tác vụ sau để lưu giữ các tập trạng thái của NFA:

(q: là một trạng thái của NFA; T: là tập trạng thái của NFA)

- ԑ-closure(q): là tập trạng thái của NFA đạt được từ trạng thái q trên sự truyền rỗng.

- ԑ-closure(T): là tập trạng thái của NFA đạt được từ tất cả các trạng thái q thuộc tập T trên sự truyền rỗng.

- δ(T, a): là tập trạng thái của NFA đạt được từ tất cả các trạng thái q thuộc tập T trên sự truyền bằng ký hiệu nhập a.

1.3.2. Phân tích

Trước khi đọc vào một ký tự nhập, DFA có thể ở một trạng thái bất kỳ trong các trạng thái thuộc ԑ-closure(q0) với q0 là trạng thái bắt đầu của NFA, gọi trạng thái này là T. Giải sử các trạng thái của T là các trạng thái đạt được từ q0 trên các ký hiệu nhập và giả sử a là ký hiệu nhập kế tiếp. Khi đọc a, NFA có thể chuyển đến một trạng thái bất kỳ trong tập trạng thái δ(T, a). Khi chúng cho phép sự truyền rỗng, NFA có thể ở bất kỳ trạng thái nào trong ԑ-closure(δ(T, a)) sau khi đã đọc a.

Ta xây dựng các trạng thái và bảng hàm chuyển cho DFA theo cách như sau:

- Mỗi trạng thái của DFA tượng trưng bởi một tập trạng thái của NFA mà NFA có thể di chuyển đến sau khi đọc một chuỗi ký hiệu nhập gồm: tất cả sự truyền rỗng có thể xảy ra trước hoặc sau các ký hiệu nhập được đọc.

- Trạng thái bắt đầu của DFA là ԑ-closure(q0).

- Các trạng thái và hàm chuyển sẽ được thêm vào D bằng giải thuật trên.

- Một trạng thái của DFA là trạng thái kết thúc nếu nó là tập các trạng thái của NFA chứa ít nhất một trạng thái kết thúc của NFA.

Việc tính toán ԑ-closure(T) có thể xem như quá trình tìm kiếm một đồ thị của các nút từ các nút cho trước và đồ thị bao gồm toàn những cạnh có nhãn ԑ của NFA. Giải thuật đơn giản để tìm ԑ-closure(T) là dùng Stack để lưu giữ các trạng thái mà cạnh của chúng chưa được kiểm tra cho sự truyền rỗng.

1.3.3. Giải thuật

|  |
| --- |
| Trạng thái bắt đầu ԑ-closure(q0) chỉ là một trạng thái trong các trạng thái của DFA và trạng thái này chưa được đánh dấu;  ***While*** Có một trạng thái T của DFA chưa được đánh dấu ***do***  ***Begin***  Đánh dấu T; {xét trạng thái T}  ***For*** Với mỗi ký hiệu nhập a ***do***  ***Begin***  U := ԑ-closure(δ(T, a))  ***If*** U không có trong tập trạng thái của DFA ***then***  ***Begin***  Thêm U vào tập các trạng thái của DFA và trạng thái này chưa được đánh dấu  Ꟙ[T, a] := U; {Ꟙ[T, a] là phần tử của bảng chuyển DFA}  ***End;***  ***End;***  ***End;*** |

CHƯƠNG 2

THIẾT KẾ VÀ CÀI ĐẶT

2.1. Ngôn ngữ lập trình và thư viện

2.1.1. Ngôn ngữ lập trình Python là gì?

Python là một ngôn ngữ lập trình bậc cao, mã nguồn mở và đa nền tảng. Đồng thời là ngôn ngữ lập trình thông dịch cho phép người dùng có thể viết code ngay trên trình thông dịch hoặc viết vào file sau đó chạy chúng.

2.1.2. Đặc điểm của ngôn ngữ Python

Một ngôn ngữ thông dịch: Python chạy trực tiếp trên từng dòng mã, nếu có lỗi trong mã chương trình thì nó sẽ ngừng chạy.Khác với những ngôn ngữ lập trình khác, Python không sử dụng cặp dấu ‘{}’, mà sử dụng thụt đầu dòng, việc này giúp cho lập trình viên dễ dàng sử dụng hơn.

Một ngôn ngữ linh hoạt, không cần phải khai báo loại biến khi viết mã vì Python sẽ xác định chúng vào thời điểm chạy chương trình. Python gần gũi với ngôn ngữ của con người hơn các ngôn ngữ lập trình khác. Do đó, người lập trình không cần lo lắng về kiến trúc và quản lý bộ nhớ.

2.1.3. Ứng dụng của ngôn ngữ Python

Python được dùng để phát triển web phía máy chủ bao gồm những hàm backend phức tạp mà các trang web thực hiện để hiển thị thông tin người dùng.

Tự động hoá bằng các tập lệnh Python như: gửi email, tải xuống nội dung, tìm kiếm lỗi trong nhiều tệp,… Có nhiều thư viện hỗ trợ cho các hoạt động khác nhau như, xử lý dữ liệu lớn, xử lý hình ảnh, nhận diện giọng nói…

2.1.4. Thư viện Streamlit

Cách cài đặt trên trình dòng lệnh:

**pip install streamlit**

Streamlit là một công cụ mã nguồn mở giúp bạn tạo ứng dụng web dễ dàng từ các đoạn mã Python. Nó cho phép bạn tạo các ứng dụng tương tác một cách nhanh chóng và dễ dàng, không cần phải có kiến thức sâu về web development.

Với Streamlit, bạn có thể tạo các ứng dụng web để trực quan hóa dữ liệu, xây dựng các bảng điều khiển, hoặc thậm chí là tạo các ứng dụng máy học (machine learning) mà người dùng có thể tương tác trực tiếp. Điểm mạnh của Streamlit là sự đơn giản và dễ sử dụng, giúp các nhà phát triển tập trung vào việc phát triển chức năng của ứng dụng mà không cần lo lắng quá nhiều về phần giao diện.

2.1.5. Thư viện Automathon

Cách cài đặt trên trình dòng lệnh:

**pip install automathon**

Automathon là một khái niệm mới kết hợp giữa "automation" (tự động hóa) và "hackathon" (cuộc thi lập trình). Trong automathon, các đội hoặc cá nhân cạnh tranh để tạo ra các giải pháp tự động hóa cho các vấn đề cụ thể. Thí sinh sử dụng kỹ năng lập trình và sáng tạo để xây dựng các hệ thống tự động hóa cho các quy trình và nhiệm vụ, từ tự động hóa quy trình làm việc hàng ngày đến tạo ra các công cụ tự động hóa trong nhiều lĩnh vực khác nhau. Automathon cũng có thể được tổ chức để tạo ra sự nhận thức và quan tâm đến tự động hóa trong cộng đồng lập trình, đồng thời khuyến khích sự học hỏi và hợp tác giữa các nhà phát triển và chuyên gia trong lĩnh vực này.

2.1.6. Thư viện Graphviz

Để cài đặt thư viện Graphviz, hãy truy cập vào [**https://graphviz.org/download**](https://graphviz.org/download)**,** để tải phiên bản thư viện tương ứng về máy sau đó tiến hành cài đặt.

Graphviz là một công cụ phổ biến được sử dụng để tạo ra các biểu đồ và đồ thị từ dữ liệu cấu trúc. Được phát triển bởi AT&T Labs, Graphviz cung cấp một bộ công cụ mạnh mẽ để biểu diễn thông tin theo các dạng đồ thị như đồ thị hướng, đồ thị vô hướng, cây, và nhiều dạng khác.

2.1.7. Thư viện Pandas

Cách cài đặt trên trình dòng lệnh gõ:

**pip install pandas**

Pandas là một thư viện mã nguồn mở trong Python được sử dụng rộng rãi cho việc xử lý và phân tích dữ liệu. Được xây dựng trên nền tảng của NumPy, Pandas cung cấp các cấu trúc dữ liệu linh hoạt như DataFrame và Series, giúp cho việc làm việc với dữ liệu trở nên đơn giản và hiệu quả hơn.

Các đặc điểm chính của Pandas bao gồm khả năng đọc và ghi dữ liệu từ và vào nhiều định dạng tệp tin khác nhau như CSV, Excel, SQL, và nhiều định dạng khác; khả năng thao tác và làm sạch dữ liệu bằng các phương thức như lọc, sắp xếp, và thay đổi dữ liệu; khả năng thực hiện các phép tính toán và thống kê trên dữ liệu; và khả năng tương tác với dữ liệu chuỗi thời gian và chuỗi thời gian.

2.2. Thiết kế chương trình

2.2.1. Xây dựng tập trạng thái DFA từ NFAԑ

Ta xây dựng DFA từ NFAԑ bằng các bước sau:

**Bước 1:** Xác định tập trạng thái mới

Từ trạng thái bắt đầu của NFAԑ, tính ԑ-closure của trạng thái đó, kết quả là tập hợp các trạng thái có thể đạt được từ các trạng thái ban đầu thông qua các phép chuyển trên nhãn rỗng (dịch chuyển trên ԑ). Gán nhãn cho từng tập hợp thu được và xem như trạng thái ban đầu của DFA.

**Bước 2:** Dịch chuyển trên một ký hiệu nhập

- Từ trạng thái đã có, lần lượt chuyển từng trạng thái trong tập trạng thái trên qua ký hiệu nhập, thu được tập trạng thái mới. Tính e-closure của tập các trạng thái vừa nhận được, kết quả có được là tập hợp mới các trạng thái.

- Kiểm tra tập trạng thái mới với các tập đã có từ trước, nếu giống với tập đã có thì bỏ qua, nếu là tập trạng thái mới (không giống với các tập đã có) thì gán nhãn và xem như là một trạng thái mới của DFA.

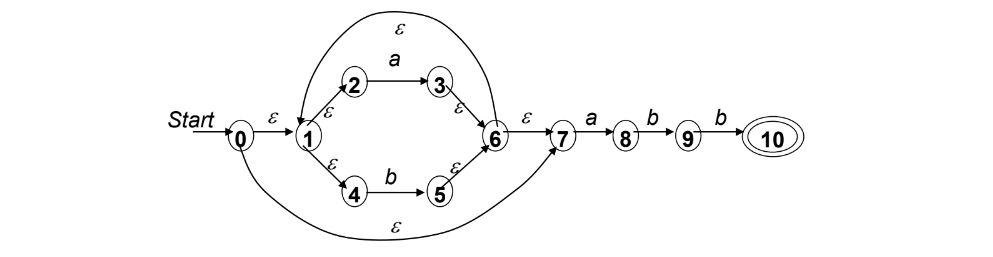
**Bước 3:** Lập lại quá trình trên và xét trạng thái kết thúc

Thực hiện bước 2 cho đến khi không sinh ra trạng thái mới nào khác (các trạng thái hiện tại giống với trạng thái trước) và kết thúc chương trình.

- Một trạng thái của DFA là trạng thái kết thúc nếu nó là tập các trạng thái của NFAε chứa ít nhất một trạng thái kết thúc của NFAε.

2.2.2. Ví dụ minh họa

Tạo DFA từ NFAԑ sau:



Hình 2.1 NFA với ԑ - dịch chuyển

Các bước xây dựng trạng thái cho DFA:

1) Trạng thái bắt đầu của DFA :

ԑ-closure(0) = {0, 1, 2, 4, 7} = A\*

2) ε-closure(δ(A, a)) = ε-closure({3, 8}) = {1, 2, 3, 4, 6, 7, 8} = B\*

3) ԑ-closure(δ(A, b)) = ε-closure({5}) = {1, 2, 4, 5, 6, 7} = C\*

4) ε-closure(δ(B, a)) = ε-closure({3, 8}) = B

5) ε-closure(δ(B, b)) = ε-closure({5, 9}) = {1, 2, 4, 5, 6, 7, 9} = D\*

6) ε-closure(δ(C, a)) = ε-closure({3, 8}) = B

7) ԑ-closure(δ(C, b)) = ε-closure({5}) = C

8) ε-closure(δ(D, a)) = ε-closure({3, 8}) = B

9) ε-closure(δ(D, b)) = ε-closure({5, 10}) = {1, 2, 4, 5, 6, 7, 10} = E\*

10) ε-closure(δ(E, a)) = ε-closure({3, 8}) = B

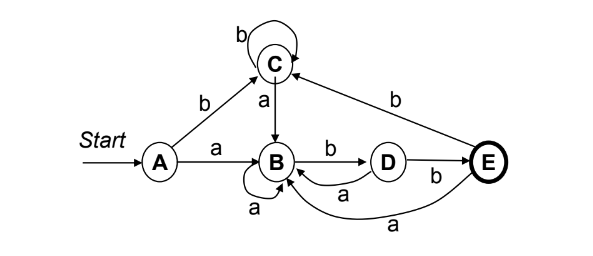
11) ε-closure(δ(E, b)) = ε-closure({5}) = C

Từ các tập trạng thái này, ta xác định được A là trạng thái bắt đầu, E là trạng thái kết thúc (vì trong E có chứa trạng thái 10 là trạng thái kết thúc của NFAԑ) và bảng hàm chuyển của DFA như sau:

|  |  |  |
| --- | --- | --- |
| **Trạng thái** | **Ký hiệu nhập** | |
| **a** | **b** |
| A | B | C |
| B | B | D |
| C | B | C |
| D | B | E |
| E | B | C |

Bảng 2.1 Bảng hàm chuyển của DFA

Từ bảng chuyển như trên, ta xây dựng sơ đồ chuyển trạng thái cho DFA tương đương nhận dạng cùng ngôn ngữ có dạng như sau:



Hình 2.2 Sơ đồ chuyển của DFA

**Nhận xét**: Ta thấy dạng không đơn định NFA được định nghĩa tổng quát hơn dạng đơn định DFA. Trong thực tế, các máy tính số hoàn toàn là đơn định, trạng thái của chúng tại mỗi thời điểm là xác định được duy nhất từ một chuỗi nhập bất kỳ và trạng thái bắt đầu.

2.3. Cài đặt chương trình

2.3.1. Phương thức khởi tạo DFA

A computer screen shot of a program

Description automatically generated

Hình 2.3 Phương thức khởi tạo DFA

- Q: có kiểu dữ liệu là danh sách, lưu các trạng thái của DFA.

*-* Σ: là tập hợp, lưu các ký hiệu nhập mà DFA đọc cho hàm chuyển trạng thái.

*-* Ꟙ: có kiểu dữ liệu là từ điển, chứa các hàm xác định trạng thái tiếp theo sau khi DFA chuyển từ một trạng thái hiện tại qua ký hiệu nhập.

*-* Q0*:* là trạng thái ban đầu của DFA.

- F: một tập hợp, lưu các trạng thái được coi là trạng thái “chấp nhận” hoặc “cuối cùng”. Nếu DFA kết thúc ở một trong những trạng thái này sau khi xử lý một đầu vào, thì đầu vào được coi là được chấp nhận.

- Phương thức Show\_DFA dùng để hiển thị danh sách hàm chuyển của DFA, tập các ký hiệu nhập, trạng thái bắt đầu, trạng thái kết thúc và tập các trạng thái.

A computer screen shot of a math equation

Description automatically generated2.3.2. Phương thức khởi tạo NFAԑ

Hình 2.4 Phương thức khởi tạo NFAԑ

- Q: có kiểu dữ liệu là tập hợp, lưu các trạng thái của NFAε.

*-* Σ: là tập hợp, lưu các ký hiệu nhập mà NFAε đọc cho hàm chuyển trạng thái.

*-* Ꟙ: có kiểu dữ liệu là từ điển, chứa các hàm xác định trạng thái tiếp theo sau khi NFAε chuyển từ một trạng thái hiện tại qua ký hiệu nhập hoặc trạng thái ε.

*-* Q0*:* là trạng thái ban đầu của NFAε.

- F: một tập hợp, lưu các trạng thái được coi là trạng thái “chấp nhận” hoặc “cuối cùng”. Nếu NFAε kết thúc ở một trong những trạng thái này sau khi xử lý một đầu vào, thì đầu vào được coi là được chấp nhận.

2.3.3. Phương thức đọc NFAԑ đầu vào

A computer screen shot of a program

Description automatically generated2.3.3.1 Đọc NFAԑ từ file

Hình 2.5 Phương thức đọc NFAԑ đầu vào từ file

A number on a white background

Description automatically generated- Phương thức thực hiện việc đọc dữ liệu từ file ban đầu, như ví dụ dưới đây:

Hình 2.6 Ví dụ file dữ liệu đầu vào

- Hàng đầu tiên, chứa một ký tự bắt đầu và các ký tự kết thúc. Trong đó ký tự đầu tiên là ký tự bắt đầu, các ký tự còn lại là ký tự kết thúc.

- Các hàng còn lại là những hàm chuyển, với ký tự đầu tiên là trạng thái hiện tại, ký tự tiếp theo là ký hiệu nhập và các ký tự còn lại là các trạng trái chuyển đến sau khi chuyển qua ký hiệu nhập.

A computer screen shot of a program

Description automatically generated2.3.3.2. Đọc NFAԑ từ bàn phím

Hình 2.7 Đọc NFAԑ từ bàn phím

- Định dạng dữ liệu đọc từ bàn phím tương tự định dạng dữ liệu đọc từ file

2.3.4. Phương thức hiển thị NFAԑ

A computer screen shot of a program

Description automatically generated- Phương thức này hiển thị danh sách hàm chuyển của NFAԑ, tập các ký hiệu nhập, trạng thái bắt đầu, trạng thái kết thúc và tập các trạng thái.

Hình 2.8 Phương thức hiển thị NFAԑ

A screenshot of a computer code

Description automatically generated2.3.5. Phương thức tính ԑ-closure

Hình 2.9 Phương thức tính ԑ-closure

- Giá trị đầu vào của phương thức là tập các trạng thái hiện tại (States) để thực hiện phép dịch chuyển qua ԑ, kết quả trả về là một tập hợp chứa các trạng thái tiếp theo sau khi thực hiện phép dịch chuyển.

- Phương thức bắt đầu bằng việc khởi tạo Result là một tập hợp chứa các trạng thái tiếp theo sau khi dịch chuyển qua ԑ, Queue là hàng đợi chứa các trạng thái đang chờ xét duyệt. Đầu tiên, đưa đồng thời các trạng thái hiện tại vào Result và Queue. Sau đó, tạo vòng lặp duyệt qua từng từng trạng thái trong hàng đợi, gán q là trạng thái đang duyệt, nếu tồn tại một khóa (q, ‘ԑ’) trong từ điển Ꟙ thì gán kết quả hàm chuyển vào Value, nếu Value chưa có trong tập Result thì lưu vào Result, đồng thời đưa Value vào hàng đợi làm trạng thái tiếp theo cho phép chuyển. Kết quả trả về là tập hợp các trạng thái tiếp theo (Result) của trạng thái hiện tại sau khi dịch chuyển qua ԑ.

- Vòng lặp kết thúc khi không còn phần tử nào trong hàng đợi hay Queue rỗng, điều này xảy ra khi sử dụng phương thức pop(index), lấy ra và xóa một phần tử có vị trí là position.

A computer screen shot of a code

Description automatically generatedHình 2.10 Phương thức dịch chuyển

2.3.6. Phương thức dịch chuyển

- Phương thức này đọc các giá trị đầu vào (States) là tập các trạng thái hiện tại, thực hiện phép dịch chuyển qua ký hiệu nhập c. Đầu tiên, khởi tạo Result là tập hợp các trạng thái tiếp theo sau khi dịch chuyển, tạo vòng lặp duyệt qua từng trạng thái hiện tại, kiểm tra cặp (q, c) có là một khóa trong từ điển Ꟙ, nếu đúng gán Value là từng kết quả của hàm chuyển, sau đó lưu Value vào Result. Vòng lặp kết thúc khi đã duyệt qua hết các trạng thái trong States và trả về Result.

2.3.7. Phương thức chuyển đổi

A computer screen shot of a computer code

Description automatically generated

Hình 2.11 Phương thức chuyển đổi

A computer code on a white background

Description automatically generatedHình 2.12 Phương thức chuyển đổi

- Phương thức thực hiện phép chuyển đổi từ một NFAԑ sang DFA tương đương, phép chuyển sẽ trả về là một DFA đã được định nghĩa từ ban đầu.

- Phương thức bắt đầu bằng việc khởi tạo ra danh sách IsOpen chứa các trạng thái chưa duyệt qua ký hiệu nhập, Closed là danh sách chứa các trạng thái tiếp theo sau khi duyệt qua ký hiệu nhập, Result là một DFA. Khởi tạo Start bằng giá trị sau khi dịch chuyển trạng thái bắt đầu của NFAԑ qua ԑ, Start là trạng thái bắt đầu của DFA.

- Thêm Start vào IsOpen, tạo vòng lặp lần lượt duyệt qua từng phần tử trong IsOpen, lấy ra và xóa phần tử đầu tiên bằng phương thức pop(index) gán bằng q. Thêm q vào Closed, xét từng phần thử trong tập ký hiệu nhập Σ gán c là ký tự đang xét, sử dụng phương thức dịch chuyển trạng thái q qua ký hiệu nhập c Move(q, c), tính ԑ\_closure tập trạng thái kết quả của Move(q, c), gán giá trị là một trạng thái vừa tìm được của DFA từ trạng thái hiện tại (q) qua ký hiệu nhập (c) vào Next. Kiểm tra Next có trong danh sách IsOpen và Close, nếu Next không tồn tại trong 2 danh sách kể trên, đưa Next vào IsOpen và lưu giá trị Next là kết quả của hàm chuyển từ trạng thái q qua ký hiệp nhập c vào từ điển hàm chuyển của DFA (Result.Ꟙ[tuple(q), c] = Next). Vòng lặp kết thúc khi không còn phần tử chờ xét duyệt hay danh sách IsOpen rỗng.

- Tất cả các trạng thái của DFA đã được duyệt qua sẽ lưu vào Closed, gán Closed vào danh sách các trạng thái của DFA (Result.Q) bằng phương thức deepcopy(input) của thư viện copy. Sau đó, xét trạng thái kết thúc, duyệt từng tập trạng thái trong Closed, nếu có một phần tử trở lên trong tập trạng thái đang xét là phần tử thuộc trạng thái kết thúc của NFAԑ, thêm tập trạng thái đó vào danh sách các trạng thái kết thúc của DFA. Lưu tập hợp các ký hiệu nhập của NFAԑ vào tập các ký hiệu nhập của DFA, kết thúc chương trình và trả về kết quả Result sau khi chuyển đổi NFAԑ là một DFA tương đương.

A screenshot of a computer program

Description automatically generated2.3.8. Hàm sử dụng các phương thức

A screenshot of a computer program

Description automatically generatedHình 2.13 Hàm sử dụng các phương thức

Hình 2.14 Hàm sử dụng các phương thức

A computer screen shot of a program code

Description automatically generated

Hình 2.15 Hàm sử dụng các phương thức

- Hiển thị một hướng dẫn để chọn file NFAε hoặc nhập thông số từ bàn phím. Cho phép người dùng tải lên một file txt chứa định dạng của NFAε.

- Yêu cầu người dùng nhập trạng thái ban đầu (Q0), các trạng thái kết thúc (F), và chuyển trạng thái (δ) nếu họ không tải file lên. Có một nút "Convert" để bắt đầu quá trình chuyển đổi. Nếu người dùng nhấn nút "Convert", chương trình sẽ thực hiện chuyển đổi từ NFAε sang DFA và hiển thị kết quả DFA tương ứng. Nếu người dùng không cung cấp đủ thông tin, sẽ hiển thị một cảnh báo yêu cầu nhập đầy đủ thông tin.

2.3.8. Các hàm chuyển đổi kiểu dữ liệu để xây dựng đồ thị

A screenshot of a computer program

Description automatically generatedHình 2.16 Hàm sử dụng các phương thức

2.4. Giao diện hệ thống

2.4.1. Giới thiệu hệ thống

- Để khởi động hệ thống, trên trình dòng lệnh gõ:

**streamlit run <đường dẫn đến file chương trình>**

A screenshot of a computer

Description automatically generated- Giao diện hệ thống sử dụng thư viện Streamlit của Python, Streamlit là một framework Python mã nguồn mở được sử dụng để xây dựng ứng dụng web dựa trên dữ liệu (data-driven web applications) một cách nhanh chóng và dễ dàng. Nó được thiết kế để giúp các nhà phân tích dữ liệu và nhà phát triển tạo ra các ứng dụng tương tác với người dùng mà không cần phải sử dụng các công nghệ phức tạp như HTML, CSS hoặc JavaScript.

Hình 2.17 Giao diện hệ thống

- Hệ thống cho phép người dùng chuyển từ dạng NFAε sang dạng DFA, có 2 cách nhập NFAε từ người dùng:

* A white rectangular object with a black border

  Description automatically generatedNhập bằng file: người dùng nhấn button chọn file NFAε đã tạo sẵn từ trước.

Hình 2.18 Upload file NFAε

* Nhập từ bàn phím: người dùng nhập trực tiếp cách thành phần của NFAε vào các ô tương ứng.

A screenshot of a computer

Description automatically generatedHình 2.19 Ô nhập các thành phần của NFAε

2.4.2. Cài đặt hệ thống

A screenshot of a computer

Description automatically generated- Khi người dùng nhập thiếu thông tin và nhấn button Convert hệ thống sẽ thông báo như sau:

Hình 2.20 Hệ thống thông báo lỗi khi nhập thiếu Q0

Các trường hợp nhập thiếu các thông tin khác thì hệ thống sẽ thông báo tương ứng.

- Khi người dùng nhập sai định dạng được quy ước ở 2.3.3.1

Hình 2.A screenshot of a chat

Description automatically generated21 Hệ thống thông báo lỗi nhập sai định dạng

A screenshot of a computer

Description automatically generated- Khi người dùng nhập đầy đủ và đúng định dạng, sau đó nhấn nút Convert hệ thống sẽ hiển thị ra DFA như sau:

Hình 2.22 Nhập NFAε từ bàn phím

A diagram of a diagram

Description automatically generatedHình 2.23 Đồ thị của NFAε

A table of equations

Description automatically generated with medium confidenceHình 2.24 Các bước xây dựng trạng thái cho DFA

A diagram of a diagram

Description automatically generatedA screenshot of a math test

Description automatically generatedHình 2.25 Các thành phần của DFA

Hình 2.26 Đồ thị của DFA

CHƯƠNG 3

KIỂM THỬ VÀ ĐÁNH GIÁ

3.1. Mục tiêu kiểm thử

Mục tiêu của phần kiểm thử này là xác định độ chính xác và tối ưu của demo chuyển đổi NFAԑ sang DFA tương đương. Từ NFAԑ ban đầu chuyển thành DFA tương đương với các trạng thái mới cho DFA và bảng hàm chuyển của DFA.

3.2. Kịch bản kiểm thử

- Đầu tiên sẽ đọc một danh sách đầu vào của một NFAԑ, hiển thị chi tiết ra danh sách hàm chuyển của NFAԑ, tập ký hiệu nhập, tập trạng thái bắt đầu, tập trạng thái kết thúc và tập hợp các trạng thái có trong NFAԑ.

- Tiếp theo gọi hàm Convert(), hiển thị quá trình chuyển đổi các trạng thái từ NFAԑ sang DFA, từng trạng thái được chuyển qua từng ký hiệu nhập và kiểm tra sinh ra trạng thái mới cho DFA theo các bước xây dựng DFA từ NFAԑ đã nêu ở Chương 2.

- Cuối cùng, hiển thị kết quả là danh sách hàm chuyển của DFA sinh ra khi chuyển từ NFAԑ sang DFA, tập ký hiệu nhập, tập trạng thái bắt đầu, tập trạng thái kết thúc và tập hợp các trạng thái có trong DFA. Tạo bảng hàm chuyển cho DFA mới.

3.3. Kết quả kiểm thử

- Phần demo đã chạy được chương trình, in ra đúng kết quả của các phép tính và xây dựng được mô hình DFA từ NFAԑ đã cho.

- Nhưng vì chế về kiến thức nên vẫn chưa thể gán được giá trị trạng thái cho DFA mới, cũng như chưa tạo được bảng hàm chuyển của DFA vừa tạo ra.

- Nhìn chung, phần demo đã giải quyết gần 90% các vấn đề đã đặt ra ở phần đầu của đề tài này.

PHẦN III: KẾT LUẬN

1. Kết luận

Trong bài báo cáo này chúng ta đã tìm hiểu về quá trình chuyển đổi từ một automata hữu hạn không đơn định có epsilon dịch chuyển (NFAԑ) sang automato hữu hạn đơn định (DFA). Quá trình này là quan trọng trong lý thuyết ngôn ngữ hình thức và xử lý chuỗi, khi mà sự đơn giản hóa trong việc xác định các trạng thái hữu hạn là cần thiết.

Cũng trong bài báo cáo, chúng ta đã đi sâu vào các bước trong quá trình chuyển đổi từ NFAԑ sang DFA. Bắt đầu với việc xác định trạng thái bắt đầu, tìm tập trạng thái của DFA qua các phép dịch chuyển trên ký hiệu nhập và xây dựng một DFA hoàn chỉnh với hàm chuyển, ký hiệu nhập, tập trạng thái, trạng thái bắt đầu và trạng thái kết thúc.

Việc chuyển đổi từ NFA sang DFA mang lại nhiều lợi ích, bao gồm hiệu quả trong quá trình thực hiện, đơn giản hóa cấu trúc, thuận tiện cho việc phân tích và tối ưu hóa. Điều này giúp cho việc hiểu rõ cấu trúc các ngôn ngữ dễ dàng hơn từ đó tăng tốc độ xử lý khi thực hiện các việc như kiểm tra chuỗi, tìm kiếm theo mẫu,…

2. Hướng phát triển đề tài

Trong quá trình thực hiện đề tài, nhóm đã tìm ra được các hướng phát triển sau:

- Tối ưu hóa quá trình chuyển đổi từ NFAԑ sang DFA tương đương: có thể cắt giảm các trạng thái dư thừa không cần thiết để giảm thiểu không gian trạng thái của DFA, như vậy có thể tăng tốc độ xử lý ngôn ngữ cũng như giải quyết bài toán cho.

- Xử lý ngôn ngữ tự nhiên: kết hợp quá trình chuyển đổi từ NFAԑ sang DFA tương đương trong ngữ cảnh xử lý ngôn ngữ và trí tuệ nhân tạo, hỗ trợ trong việc phân tích và xử lý các ngôn ngữ phức tạp như siêu văn bản.

- Xây dựng công cụ phân tích: việc chuyển đổi từ NFAԑ sang DFA tương đương giúp giảm bớt sự phức tạp của ngôn ngữ từ đó có thể dễ dàng hơn trong việc kiểm tra cũng như tối ưu hóa cho các biểu thức chính quy.

- Xử lý dữ liệu lớn và phân tán: nghiên cứu mở rộng quá trình chuyển đổi để xử lý dữ liệu lớn và phân tán, có thể áp dụng trong các ứng dụng liên quan đến xử lý ngôn ngữ tự nhiên trực tuyến hoặc môi trường tính toán phân tán.

- Ứng dụng trong Internet of Things (IoT): Sử dụng automata để mô hình hóa và kiểm soát các trạng thái của các thiết bị IoT, giúp quản lý mạng và tương tác giữa các thiết bị, hỗ trợ quản lý trong môi trường phức tạp.

TÀI LIỆU THAM KHẢO

[1] Võ Huỳnh Trâm – Giáo trình tin học lý thuyết – Nhà xuất bản Cần Thơ – 2009.

[2] John E. Hopcroft, Jeffrey D. Ullman – Introductions to Automata Theory, Languages and Computation – Wesley Publising Company, Inc – 1979.