BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC CẦN THƠ**

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

**□ & □**



**BÁO CÁO MÔN HỌC**

**TIN HỌC LÝ THUYẾT (CT12102)**

**Đề tài**

**CHUYỂN NFAԑ SANG DFA**

**TƯƠNG ĐƯƠNG**

**NHÓM BÁO CÁO: 33**

|  |  |  |
| --- | --- | --- |
| **STT** | **MSSV** | **HỌ TÊN** |
| **33** | **B2113327** | **Nguyễn Thanh Bình** |
| **34** | **B2113328** | **Lê Tuấn Đạt** |
| **36** | **B2113337** | **Nguyễn Thị Minh Nguyệt** |

**HK 2, NH 2023 – 2024**

Cần Thơ, 4/2024

MỤC LỤC

[MỤC LỤC 2](#_Toc164672131)

[DANH MỤC HÌNH 4](#_Toc164672132)

[DANH MỤC BẢNG 5](#_Toc164672133)

[TÓM TẮT 6](#_Toc164672134)

[ABSTRACT 7](#_Toc164672135)

[DANH MỤC TỪ VIẾT TẮT 8](#_Toc164672136)

[PHẦN I: GIỚI THIỆU 9](#_Toc164672137)

[1. Đặt vấn đề 9](#_Toc164672138)

[2. Lịch sử giải quyết vấn đề 9](#_Toc164672139)

[3. Mục tiêu đề tài 9](#_Toc164672140)

[4. Đối tượng và phạm vi nghiên cứu 10](#_Toc164672141)

[4.1. Đối tượng nghiên cứu 10](#_Toc164672142)

[4.2. Phạm vi nghiên cứu 10](#_Toc164672143)

[5. Phương pháp nghiên cứu 10](#_Toc164672144)

[5.1. Kiến thức 10](#_Toc164672145)

[5.2. Công cụ hỗ trợ 10](#_Toc164672146)

[5.3. Phương pháp giải quyết vấn đề 10](#_Toc164672147)

[6. Kết quả đạt được 10](#_Toc164672148)

[7. Bố cục bài báo cáo 10](#_Toc164672149)

[PHẦN II: NỘI DUNG 12](#_Toc164672150)

[CHƯƠNG 1 12](#_Toc164672151)

[GIỚI THIỆU TỔNG QUAN VỀ BÀI TOÁN 12](#_Toc164672152)

[1.1. Tại sao phải chuyển đổi NFAԑ sang DFA tương đương? 12](#_Toc164672153)

[1.2. Mô tả chi tiết về bài toán 12](#_Toc164672154)

[1.3. Giải thuật và phương pháp giải quyết bài toán 13](#_Toc164672155)

[1.3.1. Chi tiết hướng giải quyết 13](#_Toc164672156)

[1.3.2. Phân tích 13](#_Toc164672157)

[1.3.3. Giải thuật 14](#_Toc164672158)

[CHƯƠNG 2 15](#_Toc164672159)

[THIẾT KẾ VÀ CÀI ĐẶT 15](#_Toc164672160)

[2.1. Kiến trúc tổng thể của hệ thống. 15](#_Toc164672161)

[2.2. Chức năng của từng phần trong hệ thống. 15](#_Toc164672162)

[2.2.1. Xây dựng tập trạng thái DFA từ NFAԑ 15](#_Toc164672163)

[2.2.2. Ví dụ minh họa 15](#_Toc164672164)

[2.3. Cài đặt chương trình 17](#_Toc164672165)

[2.3.1. Phương thức khởi tạo DFA 17](#_Toc164672166)

[2.3.2. Phương thức khởi tạo NFAԑ 18](#_Toc164672167)

[2.3.3. Phương thức đọc NFAԑ đầu vào 19](#_Toc164672168)

[2.3.4. Phương thức hiển thị NFAԑ 20](#_Toc164672169)

[2.3.5. Phương thức tính ԑ-closure 21](#_Toc164672170)

[2.3.6. Phương thức dịch chuyển 21](#_Toc164672171)

[2.3.7. Phương thức chuyển đổi 22](#_Toc164672173)

[2.3.8. Giao diện hệ thống: 23](#_Toc164672174)

[CHƯƠNG 3 25](#_Toc164672175)

[KIỂM THỬ VÀ ĐÁNH GIÁ 25](#_Toc164672176)

[3.1. Mục tiêu kiểm thử 25](#_Toc164672177)

[3.2. Kịch bản kiểm thử 25](#_Toc164672178)

[3.3. Kết quả kiểm thử 25](#_Toc164672179)

[PHẦN III: KẾT LUẬN 26](#_Toc164672180)

[1. Kết luận 26](#_Toc164672181)

[2. Hướng phát triển đề tài 26](#_Toc164672182)

[TÀI LIỆU THAM KHẢO 27](#_Toc164672183)

DANH MỤC HÌNH

[Hình 2.1 NFA với ԑ - dịch chuyển 16](#_Toc164752479)

[Hình 2.2 Sơ đồ chuyển của DFA 17](#_Toc164752480)

[Hình 2.3 Phương thức khởi tạo DFA 18](#_Toc164752481)

[Hình 2.4 Phương thức khởi tạo NFAԑ 19](#_Toc164752482)

[Hình 2.5 Phương thức đọc NFAԑ đầu vào 19](#_Toc164752483)

[Hình 2.6 Ví dụ file dữ liệu đầu vào 20](#_Toc164752484)

[Hình 2.7 Phương thức hiển thị NFAԑ 20](#_Toc164752485)

[Hình 2.8 Phương thức tính ԑ-closure 21](#_Toc164752486)

[Hình 2.9 Chuyển trạng thái với epsilon 22](#_Toc164752487)

[Hình 2.10 Chuyển trạng thái với nhãn 22](#_Toc164752488)

[Hình 2.11 Phương thức chuyển đổi 23](#_Toc164752489)

[Hình 2.12. Giao diện chuyển NFAԑ sang DFA 24](#_Toc164752490)

[Hình 2.13. Giao diện hiển thị các file dữ liệu 25](#_Toc164752491)

[Hình 2.14. Giao diện hiển thị thông tin NFAԑ 25](#_Toc164752492)

[Hình 2.15. Giao diện hoàn chỉnh sau khi chuyển sang DFA 26](#_Toc164752493)

DANH MỤC BẢNG

[Bảng 2.1 Bảng hàm chuyển của DFA 18](#_Toc150964940)

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 phần quan trọng trong học phần Tin Học Lý Thuyết. Mặc dù cả hai loại đều có khả năng chấp nhận các ngôn ngữ, nhưng cấu trúc và tính chất của chúng khác nhau, mở ra nhiều ứng dụng và trường hợp sử dụng khác nhau. Việc chuyển đổi giữa chúng là cần thiết để phân tích và hiểu rõ về bản chất của các ngôn ngữ và tự động hóa quá trình xử lý thông tin.

Chuyển đổi từ NFAԑ sang DFA đòi hỏi kiến thức vững về cả hai loại, cũng như kỹ năng áp dụng kiến thức để giải quyết các vấn đề liên quan đến việc biểu diễn và xử lý ngôn ngữ.

Việc viết một công cụ tự động để thực hiện quá trình chuyển đổi từ NFAԑ sang DFA tương đương sẽ giúp cải thiện hiểu biết và kỹ năng của người học về hai loại Automata hữu hạn này, cũng như hỗ trợ trong quá trình nghiên cứu và phát triển ứng dụng trong các lĩnh vực như xử lý ngôn ngữ tự nhiên, kiểm thử phần mềm, và các vấn đề liên quan đến tự động hóa.

2. Lịch sử giải quyết vấn đề

Vào năm 1959, Thuật toán Subset Construction được giới thiệu bởi R. E. Moore. Thuật toán này là một trong những phương pháp đầu tiên để chuyển đổi một NFAe thành một DFA. Ý tưởng chính của thuật toán này là sử dụng tập hợp con của tất cả các trạng thái có thể đạt được từ một trạng thái ban đầu thông qua các epsilon-chuyển đổi.

Tiếp sau đó lần lượt có sự đóng góp của Michael O.Rabin(1960s), Edward F.Moore(1962), và từ đó cho đến nay lĩnh vực chuyển đổi NFAԑ thành DFA đã trải qua sự phát triển và tối ưu hóa liên tục và 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 đề tài

* Hiểu biết sâu hơn về NFAԑ và DFA: Nắm vững kiến thức về cả hai loại máy trạng thái hữu hạn và nhận biết sự khác biệt giữa chúng.
* Phát triển kỹ năng lập trình và thiết kế thuật toán: Áp dụng kiến thức lý thuyết để thiết kế thuật toán chuyển đổi từ NFAԑ sang DFA và triển khai nó trong một chương trình máy tính.
* Xây dựng công cụ tự động: Phát triển một ứng dụng hoặc công cụ trực quan cho phép người dùng nhập vào một mô hình NFAԑ và tự động chuyển đổi thành một mô hình DFA tương đương.
* Kiểm tra và đánh giá: Kiểm thử ứng dụng để đảm bảo tính đúng đắn và hiệu quả của quá trình chuyển đổi. Đánh giá sự linh hoạt và hiệu suất của công cụ trước các tình huống và mô hình khác nhau.
* Tăng cường hiểu biết và ứng dụng trong thực tế: Sử dụng công cụ để giải quyết các vấn đề thực tế hoặc thử nghiệm trên các bài toán phức tạp trong lĩnh vực xử lý ngôn ngữ tự nhiên, tự động hóa kiểm thử, và các ứng dụng khác của máy trạng thái hữu hạn.

4. Đối tượng và phạm vi nghiên cứu

4.1. Đối tượng nghiên cứu

NFAԑ: Bao gồm cấu trúc, thuật toán và tính chất của NFAԑ.

DFA: Bao gồm cấu trúc, thuật toán và tính chất của DFA.

4.2. Phạm vi nghiên cứu

Nghiên cứu thuật toán và phương pháp chuyển đổi từ NFAԑ sang DFA, bao gồm các kỹ thuật biểu diễn, xử lý.

Đánh giá hiệu suất, tính chính xác và tính ứng dụng của các phương pháp chuyển đổi trên các bộ dữ liệu thử nghiệm.

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 “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 có hiệu suất vận hành cao hơn NFAԑ vì DFA mỗi trạng thái chỉ có thể chuyển đổi duy nhất đến một trạng thái khác từ đó giúp giảm thiểu sự không xác định trong quá trình xứ lý. Điều này dẫn đến việc giảm thiểu thời gian và tài nguyên tính toán cần thiết.

-DFA thường dễ triển khai hơn và quá trình xử lý đơn giản hơn so với NFAԑ. Việc loại bỏ ԑ-closure và nhãn rỗng ԑ giúp giảm bớt độ phức tạp và dễ triển khai hơn và tốc độ xử lý nhanh hơn. Điều này là đặc biệt quan trọng khi triển khai áp dụng vào thực tế.

-DFA có tập trạng thái đơn giản và dễ hiểu hơn NFAԑ điều này giúp giảm bớt tính phức tạp trong việc quản lý các trạng thái và tăng tính chính xác của quá trình kiểm

-Một số thuật toán như Minization(Tối ưu hóa), Hopcroft’s Alforithm, Cocke-Younger-Kasami(CYK) chỉ hoạt động trên DFA. Do đó việc sử dụng DFA sẽ thuận tiện hơn cho việc phát triển và triển khai hệ thống hơn so với NFAԑ.

Từ đó, ta thấy được việc chuyển đổi từ NFAԑ thành DFA mang lại nhiều lợi ích về mặt hiệu suất, tính đơn giản hóa, sự hiệu quả cao hơn là sử dụng NFAԑ.

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 luỹ 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 NFAԑ.

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

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

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

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

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

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

- δ(T, a): là tập trạng thái của NFAԑ 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 NFAԑ, gọi trạng thái này là T. Giả 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, NFAԑ 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 nhưu sau:

- Mỗi trạng thái của DFA tượng trưng bởi một tập trạng thái của NFAԑ mà NFAԑ 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 NFAԑ.

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 NFAԑ. 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 thập trạnh 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. Kiến trúc tổng thể của hệ thống.

Hệ thống bao gồm hai phần chính: chuyển đổi từ NFAԑ sang DFA và ứng dụng giao diện người dùng.

Phần chuyển đổi sẽ nhận đầu vào là một mô hình NFAԑ và tạo ra một mô hình tương ứng DFA.

Ứng dụng giao diện người dùng sẽ cung cấp giao diện để người dùng tải lên mô hình NFAԑ và hiển thị kết quả DFA tương ứng.

2.2. Chức năng của từng phần trong hệ thống.

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.

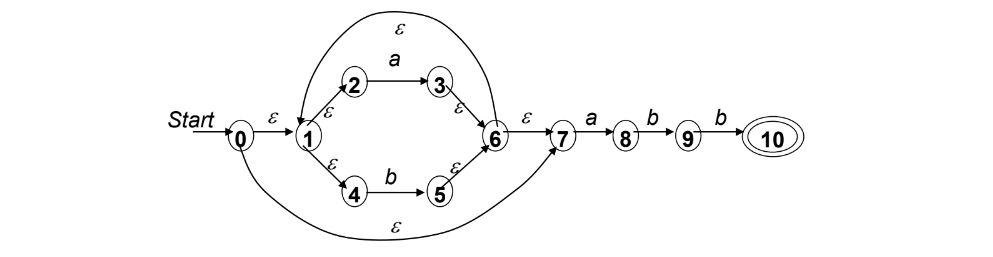
**Bước 3:** Lặp lại quá trình trên và xét trạng thái kết thúc

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(q0) = 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

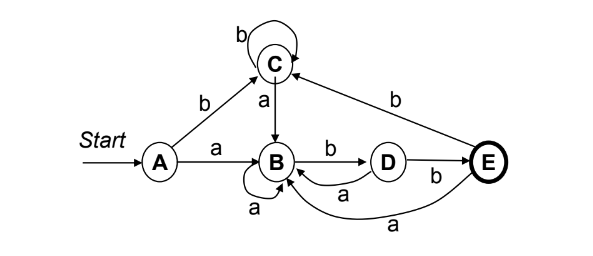
11) ε-closure(δ(E, b)) = ε-closure({5}) = C

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:

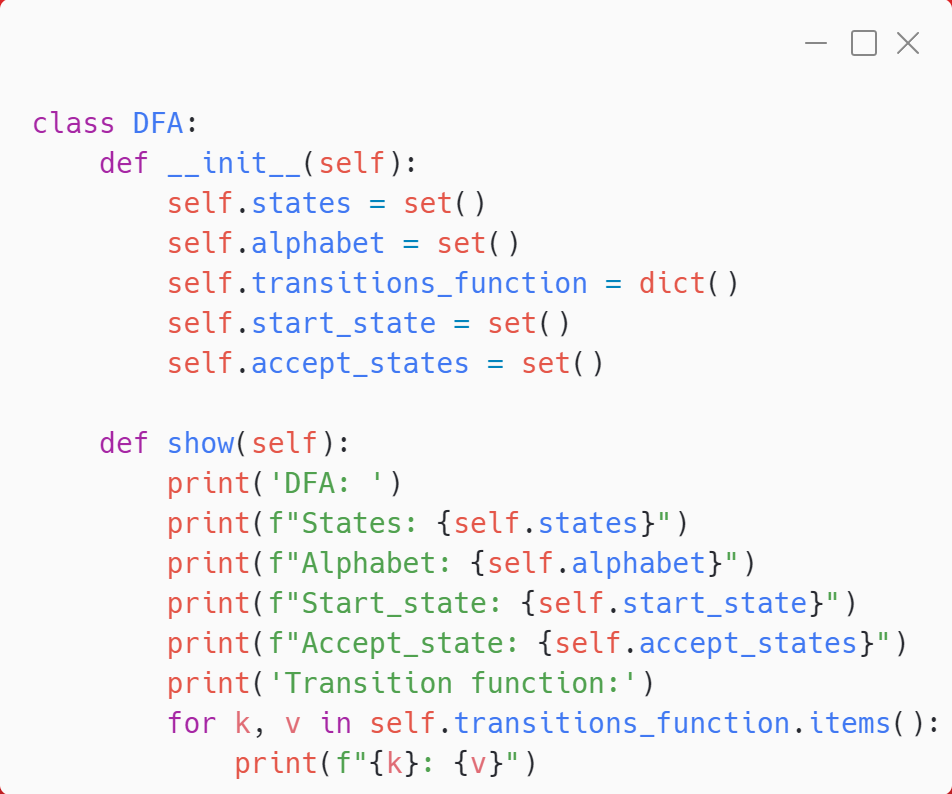


Hình 2.2 Sơ đồ chuyển của DFA

**Nhận xét**: Mặc dù có sự khác biệt trong định nghĩa, ta thấy dạng không đơn định NFAԑ được định nghĩa tổng quát hơn dạng đơn định DFA, nhưng rõ ràng khả năng nhận dạng cùng một lớp ngôn ngữ của chúng là tương đương nhau. 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



Hình 2.3 Phương thức khởi tạo DFA

- State: có kiểu dữ liệu là set, lưu các trạng thái của DFA.

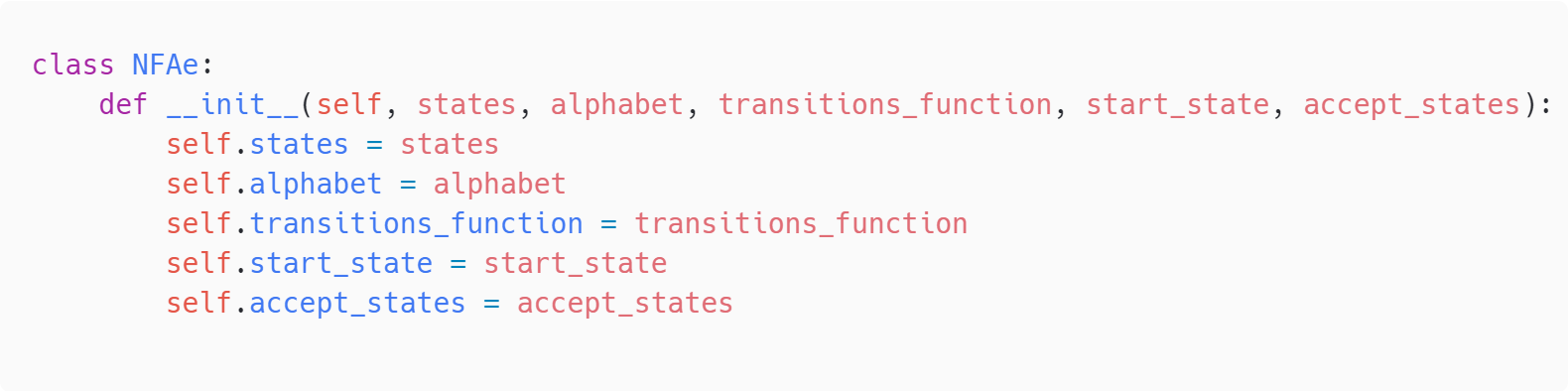
*-* Transition function: có kiểu dữ liệu là từ điển, chứa các hàm xác định trạng thái tiếp theo sau khi DFA chuyển từ một trạng thái hiện tại qua ký hiệu nhập.

*-* Start\_state*:* là trạng thái ban đầu của DFA.

- Accept\_state: 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 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.

2.3.2. Phương thức khởi tạo NFAԑ



Hình 2.4 Phương thức khởi tạo NFAԑ

- State: có kiểu dữ liệu là set, lưu các trạng thái của NFAԑ.

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

*-* Start\_state*:* là trạng thái ban đầu của NFAԑ.

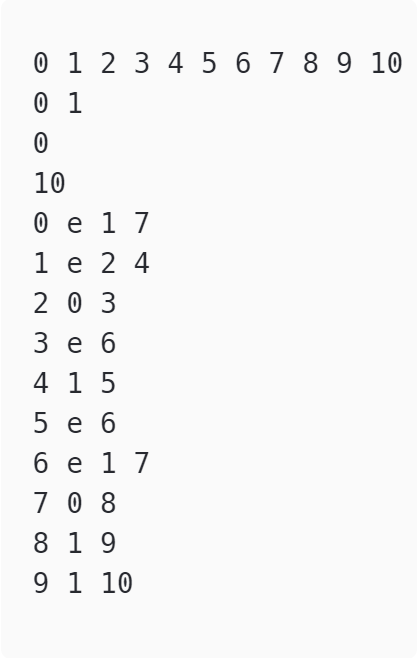
- Accept\_state: một tập hợp, lưu các trạng thái được coi là trạng thái “chấp nhận” hoặc “cuối cùng”. Nếu NFAԑ kết thúc ở một trong những trạng thái này sau khi xử lý một đầu vào, thì đầu vào được coi là được chấp nhận.

2.3.3. Phương thức đọc NFAԑ đầu vào



Hình 2.5 Phương thức đọc NFAԑ đầu vào

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

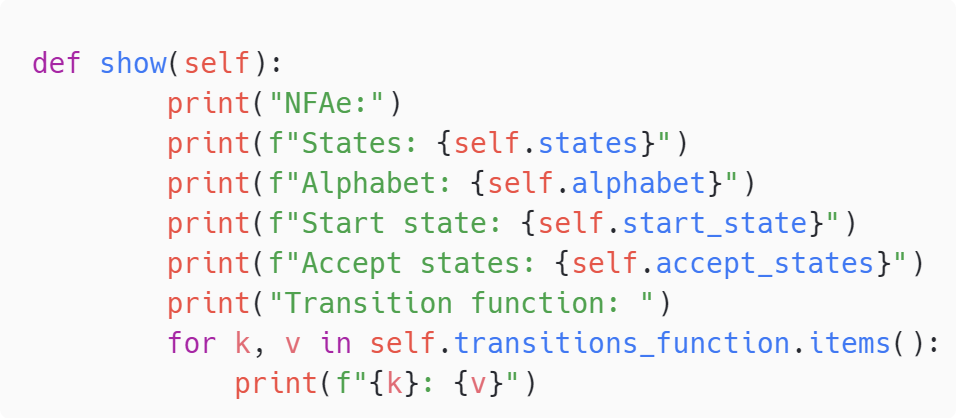


Hình 2.6 Ví dụ file dữ liệu đầu vào

- Hàng đầu tiên, chứa tất cả các trạng thái. Hàng thứ hai chứa các kí tự nhập, gọi chung là bộ nhập. Hàng thứ ba và hàng thứ tư lần lượt là trạng thái bắt đầu và trạng thái 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.

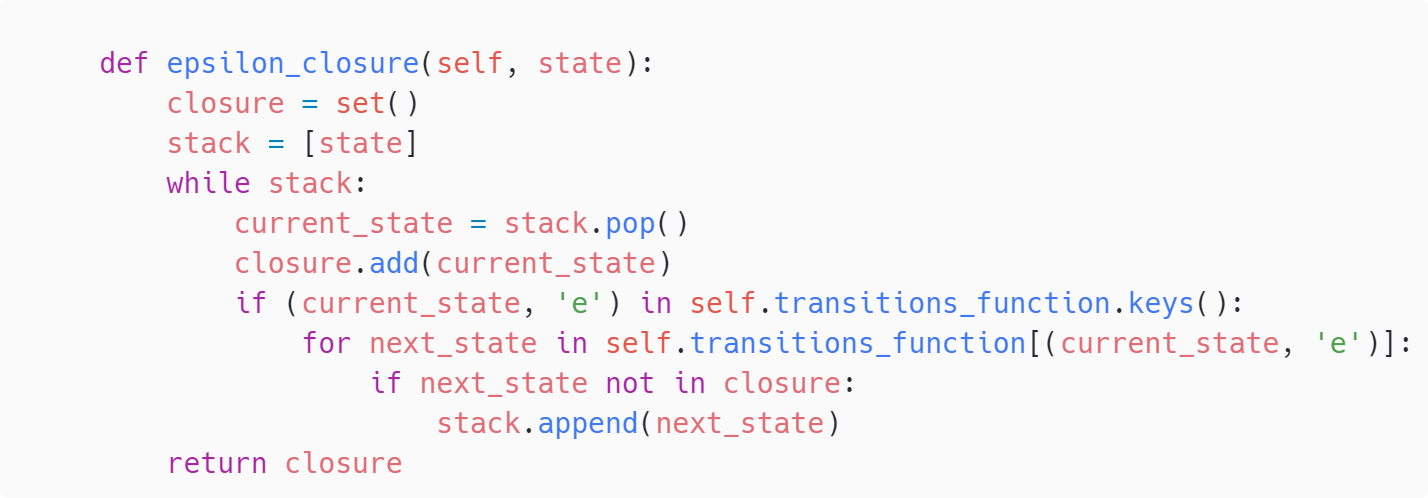
2.3.4. Phương thức hiển thị NFAԑ



Hình 2.7 Phương thức hiển thị NFAԑ

- Phương thức này tương tự với phương thức hiển thị của DFA 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.

2.3.5. Phương thức tính ԑ-closure



Hình 2.8 Phương thức tính ԑ-closure

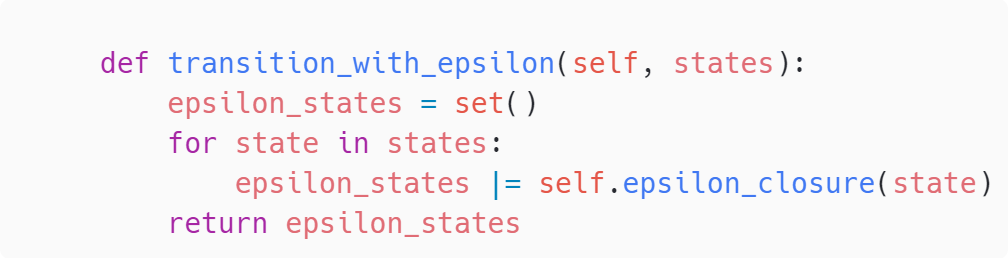
- Phương thức epsilon\_closure nhận đầu vào là một trạng thái và trả về tập hợp các trạng thái tiếp theo sau khi thực hiện phép dịch chuyển ԑ.

- Phương thức bắt đầu bằng việc khởi tạo closure là một tập hợp chứa các trạng thái tiếp theo sau khi dịch chuyển qua ԑ, stack là ngăn xếp chứa các trạng thái đang chờ xét duyệt. Bắt đầu một vòng lặp while để duyệt qua từng trạng thái trong stack, lấy ra trạng thái hiện tại từ đỉnh của stack, thêm trạng thái đang xét vào closure. Nếu tồn tại một chuyển đổi bằng nhãn ԑ từ trạng thái hiện tại, thêm các trạng thái kế tiếp (không nằm trong closure) vào stack để xét những trạng thái tiếp theo. Kết quả sẽ trả về tập hợp closure chứa các trạng thái kết quả từ trạng thái đầu vào sau khi dịch chuyển qua ԑ.

- Vòng lặp kết thúc khi không còn phần tử nào trong stack, điều này xảy ra khi đã duyệt qua tất cả các trạng thái kề của trạng thái hiện tại và không còn trạng thái nào mới để xét.

2.3.6. Phương thức dịch chuyển

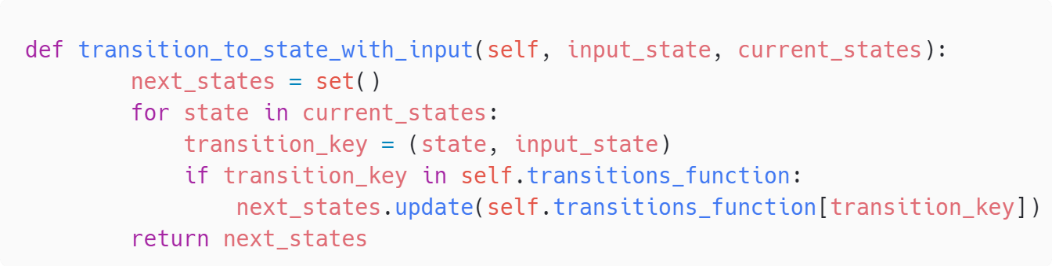
2.3.6.1. Chuyển trạng thái với epsilon



Hình 2.9 Chuyển trạng thái với epsilon

* Khởi tạo tập hợp trống epsilon\_states để lưu trữ các trạng thái mà NFAe có thể đạt được thông qua các bước epsilon.
* Duyệt qua từng trạng thái ban đầu trong tập hợp states.
* Gọi phương thức epsilon\_closure(state) để lấy tập hợp các trạng thái mà NFAe có thể đạt được từ trạng thái hiện tại thông qua các bước epsilon.
* Gộp các trạng thái mới vào epsilon\_states bằng cách sử dụng toán tử ‘ |= ’ để đảm bảo không có trạng thái nào bị trùng lặp.
* Trả về tập hợp epsilon\_states chứa tất cả các trạng thái mà NFAe có thể đạt được thông qua các bước epsilon từ tập hợp các trạng thái ban đầu.

2.3.6.2. Chuyển trạng thái với nhãn



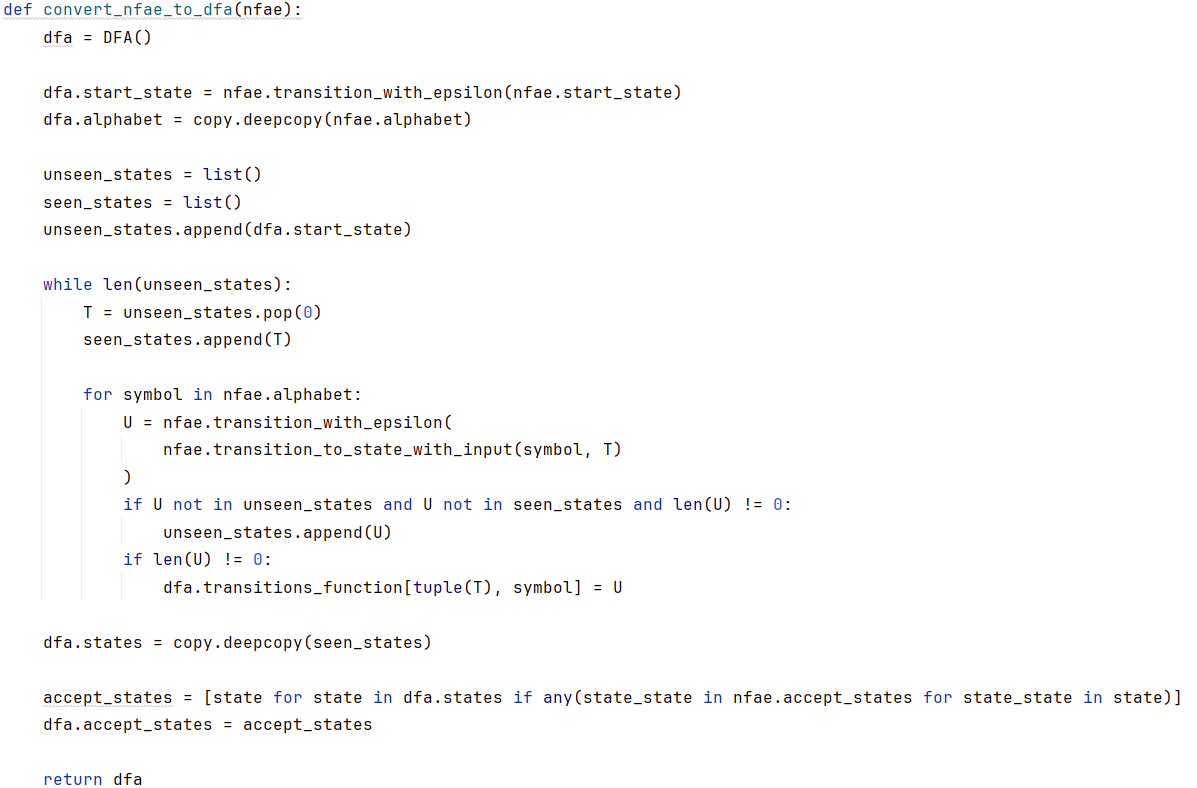
Hình 2.10 Chuyển trạng thái với nhãn

* Khởi tạo một tập hợp rỗng next\_states để lưu trữ các trạng thái tiếp theo.
* Duyệt qua từng trạng thái trong current\_states.
* Tạo khóa chuyển trạng thái (transition\_key) từ trạng thái hiện tại và ký tự đầu vào.
* Kiểm tra nếu transition\_key có trong hàm chuyển trạng thái (transitions\_function):

+ Nếu có, thêm các trạng thái mà NFAe có thể chuyển đến từ transition\_key vào next\_states.

* Trả về next\_states, tập hợp các trạng thái mà NFAe có thể chuyển đến từ trạng thái hoặc tập hợp các trạng thái hiện tại khi nhận ký tự đầu vào.

2.3.7. Phương thức chuyển đổi



Hình 2.11 Phương thức chuyển đổi

- Hàm convert\_nfae\_to\_dfa nhận đầu vào là một NFAԑ và sau những phép biến đổi, kết quả trả về là một DFA tương đương với NFAԑ đầu vào.

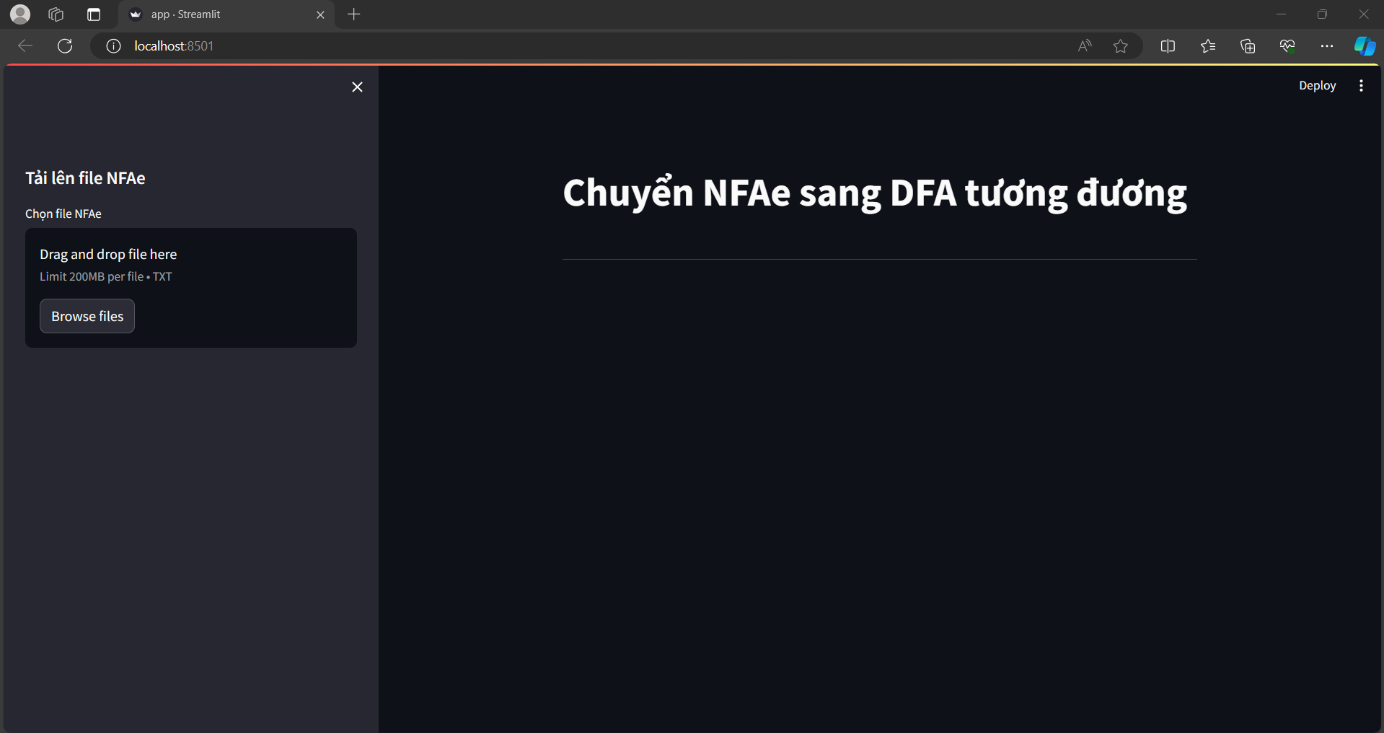
- Tiến hành xây dựng trạng thái bắt đầu của DFA bằng cách sử dụng phương thức transition\_with\_epsilon để lấy tập hợp trạng thái NFAԑ có thể chuyển từ trạng thái bắt đầu khi gặp epsilon.

- Câu lệnh ‘dfa.alphabet = copy.deepcopy(nfae.alphabet)’ dùng để sao chép bộ nhập từ NFAԑ sang DFA.

- unseen\_states là một danh sách lưu các trạng thái chưa được xét, còn seen\_states cũng là một danh sách, nhưng nó lưu các trạng thái đã duyệt qua và nó cũng là tập các trạng thái của DFA.

- Sử dụng dòng lặp while để duyệt qua các trạng thái chưa được xét (unseen\_states). Với mỗi trạng thái, chúng ta duyệt qua mỗi ký tự trong bộ chữ cái của DFA và tính toán tập hợp các trạng thái mới mà NFAe có thể chuyển đến. Nếu tập hợp các trạng thái mới chưa được xem xét, chúng ta thêm nó vào danh sách các trạng thái chưa xem xét và gán hàm chuyển của DFA từ trạng thái hiện tại với ký tự hiện tại đến tập hợp các trạng thái mới tính được.

2.3.8. Giao diện hệ thống:

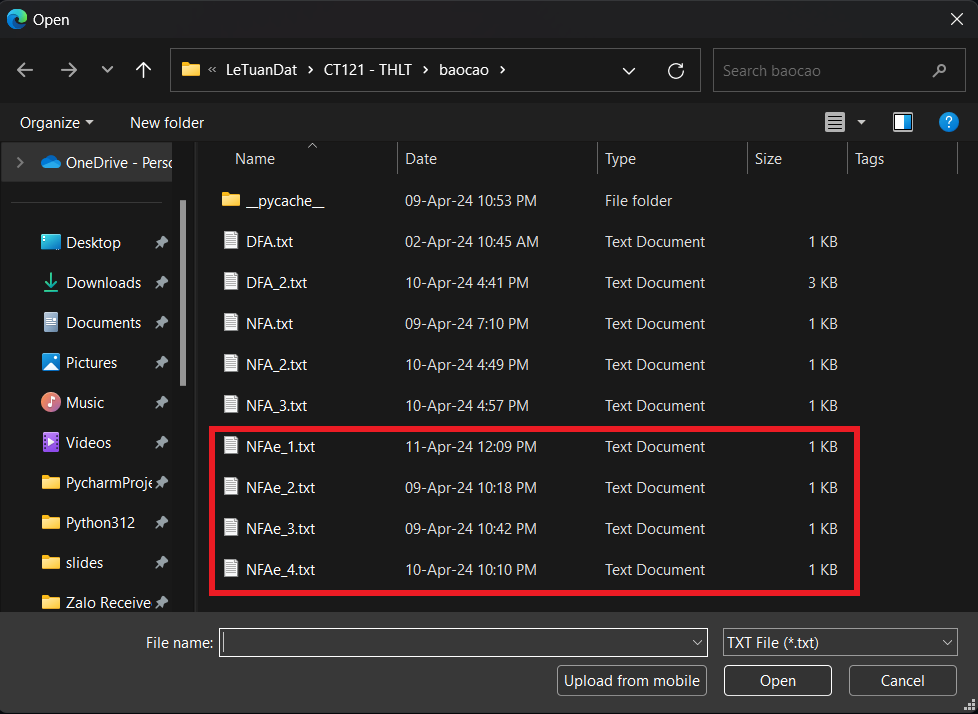


Hình 2.12. Giao diện chuyển NFAԑ sang DFA

- Streamlit là một thư viện Python mã nguồn mở giúp xây dựng các ứng dụng web dễ dàng. Nó có giao diện trực quan và khá đơn giản nhưng cũng rất bắt mắt.

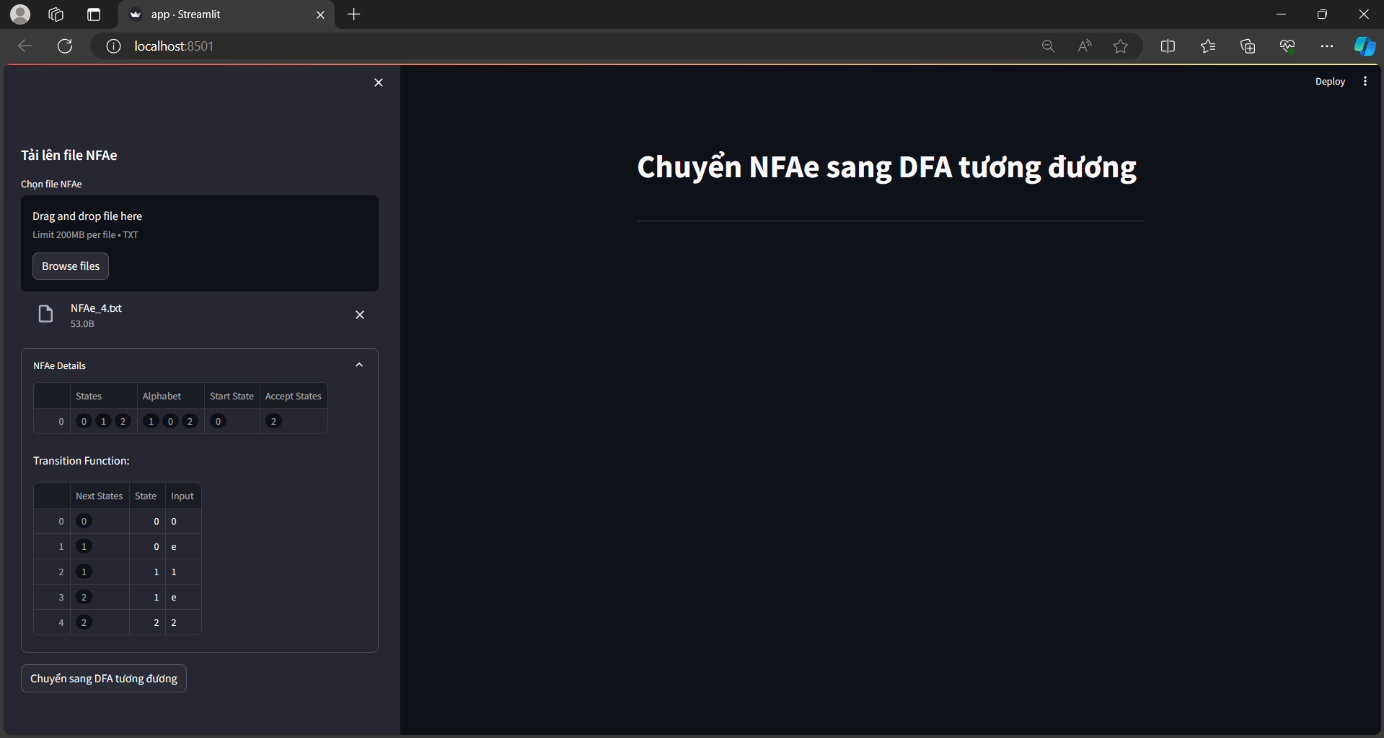
- Phía bên trái là khung để tải dữ liệu NFAԑ lên từ file, cụ thể:

+ Nhấn nút “Browse files”, sau đó, giao diện sẽ hiển thị các file dữ liệu từ máy tính:



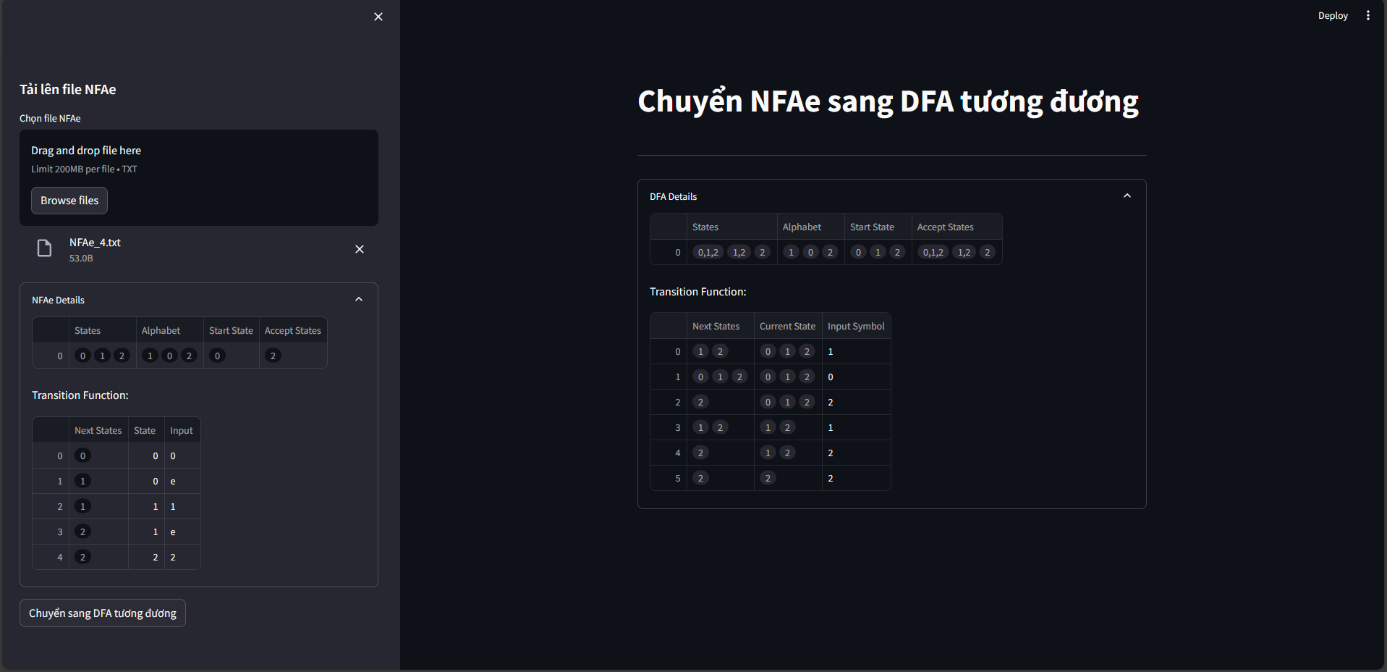
Hình 2.13. Giao diện hiển thị các file dữ liệu

+ Sau khi chọn, ví dụ đã chọn file NFAe\_4.txt, giao diện sẽ hiển thị như sau:



Hình 2.14. Giao diện hiển thị thông tin NFAԑ

+ Tiếp theo, nhấn vào nút "Chuyển sang DFA tương đương" để hệ thống sử dụng dữ liệu được cung cấp và áp dụng hàm **convert\_nfae\_to\_dfa** để tạo ra DFA tương đương từ NFAԑ. Kết quả DFA sẽ được hiển thị trên màn hình.



Hình 2.15. Giao diện hoàn chỉnh sau khi chuyển sang 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\_nfae\_to\_dfa(), 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

- Chương trình đã chạy thành công và hiển thị đúng thông tin của NFAe và quá trình chuyển đổi sang DFA.

- Hiển thị được danh sách hàm chuyển của DFA mới, danh sách hàm chuyển, trạng thái ban đầu, trạng thái chấp nhận và tập hợp các trạng thái.

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 lơn 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.

--- HẾT ---

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.