

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN



BÀI TẬP LỚN MÔN CẤU TRÚC RỜI RẠC

CHUYỂN BIỂU THỨC LOGIC TRUNG TỔ SANG HẬU TỔ VÀ LẬP BẢNG CHÂN TRỊ (Infix to Postfix)

Người hướng dẫn: Cô NGUYỄN THỊ HUỲNH TRÂM

Người thực hiện: NGUYỄN THỊ ANH THU' – 51900564

Lớp : 19050401

Khoá : 23

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN



BÀI TẬP LỚN MÔN CẤU TRÚC RỜI RẠC

CHUYỂN BIỂU THỨC LOGIC TRUNG TỔ SANG HẬU TỔ VÀ LẬP BẢNG CHÂN TRỊ (Infix to Postfix)

Người hướng dẫn: Cô NGUYỄN THỊ HUỲNH TRÂM

Người thực hiện: NGUYỄN THỊ ANH THU' – 51900564

Lớp : 19050401

Khoá : 23

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021

LỜI CẢM ƠN

Vì tình hình diễn biến phức tạp của dịch COVID nên quá trình học tập và thi cuối kì ít nhiều bị ảnh hưởng. Đầu tiên em xin gửi lời cảm ơn chân thành đến cô Nguyễn Thị Huỳnh Trâm và khoa Công Nghệ Thông Tin đã tạo điều kiện cho chúng em có thể làm báo cáo để hoàn thành kết thúc môn học, cũng như đã giúp đỡ và hướng dẫn em trong suốt quá trình học tập và làm báo cáo đề em và các bạn không phải bị trễ tiến độ.

Việc viết báo cáo đã giúp em rèn luyện sự thêm kỹ năng trình bày, cũng như một số kỹ năng khác. Do chưa có nhiều kinh nghiệm viết báo cáo cũng như giới hạn về mặt kiến thức và khả năng lập luận nên trong bài báo cáo này chắc chắn sẽ không tránh khỏi sai sót, rất mong nhận được nhận xét và đóng góp ý kiến từ phía thầy cô để em hoàn thiện hơn.

Cuối cùng kính chúc Quý thầy cô và các bạn luôn tràn đầy năng lượng và sức khoẻ thật tốt trong mùa dịch COVID này.

Em xin chân thành cảm ơn.

TÓM TẮT

Biểu thức đại số ở dạng thông thường đều được biểu diễn dưới dạng trung tố, là toán tử hai ngôi và chúng phân cách giữa hai toán hạng với nhau. Một biểu thức số học dạng trung tố có thể bao gồm nhiều toán tử và hơn hai toán hạng, ví dụ $(A + B) * C (D / (J+D))$. Tuy nhiên đối với máy tính, để tính được giá trị của một biểu thức đại số theo dạng này không đơn giản như ta vẫn làm. Vì hệ thống máy tính chỉ có thể hiểu và hoạt động trên ngôn ngữ nhị phân, nên nó giả định rằng một phép toán số học chỉ có thể diễn ra trong hai toán hạng, như: $A - B$, $C * D$, v.v.

Biểu thức Ba Lan ngược (Reverse Polish notation - RPN) là biểu thức được thể hiện dưới dạng hậu tố (Postfix). Nếu là biểu thức bình thường thì các toán hạng sẽ nằm giữa các toán tử ($5 + 7$, toán hạng 5 và 7, toán tử là +), đây được gọi là biểu thức thể hiện ở dạng trung tố, cách thể hiện như thế này giúp con người dễ hình dung và tính toán, nhưng máy tính thì khó nhận ra vấn đề, đây là nguyên nhân biểu thức hậu tố Ba Lan ngược ra đời giúp máy tính có thể hiểu và thực hiện nhanh và chính xác. Trong bài tiểu luận này, vấn đề thực hiện một biểu thức logic phức tạp trên máy tính sẽ được giải quyết bằng ngôn ngữ lập trình Python, dựa trên kỹ thuật chuyển đổi từ biểu thức trung tố (Infix) sang biểu thức Biểu thức Ba Lan ngược (Reverse Polish notation – RPN) hay gọi là biểu thức hậu tố (Postfix).

Ý tưởng đó là áp dụng cơ chế LIFO (Last In First Out) của stack để chuyển đổi, ban đầu ta xét một biểu thức từ trái sang phải, nếu gặp toán hạng (phép toán hoặc biến) thì push vào chuỗi kết quả, nếu gặp toán tử hoặc dấu ngoặc thì push vào ngăn xếp stack, sau đó dựa vào độ ưu tiên giữa các toán tử để pop (lấy ra) thêm vào chuỗi hoặc push (đưa vào) ngăn xếp cho hợp lý. Kết quả cuối cùng, ta sẽ được một chuỗi hậu tố.

Ví dụ: $R | (P \& Q)$ là biểu thức ở dạng trung tố

Kết quả sau khi chuyển sang hậu tố: $R P Q \& |$

Thuật toán là:

Nếu là toán hạng thì cho ra output.

Nếu là dấu “(“ thì cho vào stack.

Nếu là dấu “)” thì lấy các toán tử trong stack ra và cho vào output cho đến khi gặp dấu mở ngoặc “(“. Dấu “(“ cũng phải đưa ra khỏi stack.

Nếu là toán tử thì:

- Chùng nào ở đỉnh stack là toán tử và toán tử đó có độ ưu tiên lớn hơn hoặc bằng độ ưu tiên toán tử hiện tại thì lấy toán tử đó ra khỏi stack và cho ra output.
- Đưa toán tử hiện tại vào stack.

Sau khi duyệt hết chuỗi infix, nếu trong stack còn toán tử thì sẽ đưa hết ra chuỗi kết quả output.

MỤC LỤC

LỜI CẢM ƠN	i
TÓM TẮT	ii
MỤC LỤC	1
DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ.....	3
CHƯƠNG 1 – GIỚI THIỆU	5
1.1. Chi tiết công việc cá nhân.....	5
1.2. Sơ lược các chương tiếp theo.....	5
CHƯƠNG 2 – CƠ SỞ LÝ THUYẾT	6
2.1. Biểu thức Ba Lan ngược (Reverse Polish notation)	6
2.1.1. Sơ lược về biểu thức Ba Lan:	6
2.1.2. Biểu thức Ba Lan ngược:.....	6
2.1.3. Thuật toán chuyển đổi từ Infix sang Postfix:.....	8
2.1.4. Thuật toán tính biểu thức logic Postfix:	9
2.2. Logic cơ bản và sử dụng để tính bảng chân trị.	9
2.2.1. Phép phủ định.	10
2.2.2. Phép hội.	10
2.2.3. Phép tuyển.	10
2.2.4. Phép kéo theo.....	11
2.2.5. Phép tương đương.....	12
2.2.6. Độ ưu tiên của các phép toán logic và các kí hiệu quy ước.....	12

2.3 Basic logic dùng bảng chân trị:.....	12
2.3.1 Logic mệnh đề:	13
2.3.2 Logic Vị Từ:	13
CHƯƠNG 3 – THỰC NGHIỆM	14
3.1. Chạy tay testcase 1 với Infix = ‘R (P&Q)’	14
3.1.1. Chuyển đổi Infix sang Postfix	14
3.1.2. Tính list Postfix xuất bảng chân trị.....	14
3.2. Chạy tay testcase 2 với Infix = “~P (Q&R)>R”	16
3.2.1. Chuyển đổi Infix sang Postfix	16
3.2.2. Tính list Postfix xuất bảng chân trị.....	17
3.3 Code và giải thích	18
CHƯƠNG 4 – KẾT QUẢ.....	21
4.1. Kết quả testcase 1	21
4.2. Kết quả testcase 2	22
4.3. Kết quả testcase 3	23
4.4. Kết quả testcase 4	23
4.5. Kết quả testcase 5	24
TÀI LIỆU THAM KHẢO	25

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ

DANH MỤC HÌNH

<i>Hình 3. Code demo</i>	<i>21</i>
<i>Hình 4.1 Postfix ở testcase 1</i>	<i>21</i>
<i>Hình 4.2 Bảng chân trị cho testcase 1</i>	<i>22</i>
<i>Hình 4.3 Postfix ở testcase 2.....</i>	<i>22</i>
<i>Hình 4.4 Bảng chân trị cho testcase 2</i>	<i>22</i>
<i>Hình 4.5 Postfix ở testcase 3.....</i>	<i>23</i>
<i>Hình 4.6 Bảng chân trị cho testcase3</i>	<i>23</i>
<i>Hình 4.7 Postfix ở testcase 4.....</i>	<i>23</i>
<i>Hình 4.8 Bảng chân trị cho testcase 4</i>	<i>24</i>
<i>Hình 4.9 Postfix ở testcase 5.....</i>	<i>24</i>
<i>Hình 4.10 Bảng chân trị cho testcase 5</i>	<i>24</i>

DANH MỤC BẢNG

<i>Bảng 2.1 Bảng chân trị phép phủ định.</i>	<i>10</i>
<i>Bảng 2.2 Bảng chân trị phép hội.</i>	<i>10</i>
<i>Bảng 2.3 Bảng chân trị phép tuyển.....</i>	<i>11</i>
<i>Bảng 2.4 Bảng chân trị phép kéo theo.....</i>	<i>11</i>
<i>Bảng 2.5 Bảng chân trị phép tương đương.....</i>	<i>12</i>
<i>Bảng 2.7 Bảng quy ước các toán tử logic cơ bản.</i>	<i>12</i>
<i>Bảng 3.1 Từng bước chuyển Infix sang Postfix testcase 1.....</i>	<i>14</i>
<i>Bảng 3.2 Thực hiện tính bảng chân trị từ Postfix testcase 1.</i>	<i>15</i>

<i>Bảng 3.3 Tiêu đề bảng chân trị testcase 1</i>	<i>15</i>
<i>Bảng 3.4 TruthTable testcase 1</i>	<i>15</i>
<i>Bảng 3.5 Từng bước chuyển Infix sang Postfix testcase 2.....</i>	<i>16</i>
<i>Bảng 3.6 Thực hiện tính bảng chân trị từ Postfix testcase 2.</i>	<i>17</i>
<i>Bảng 3.7 Tiêu đề bảng chân trị testcase 2</i>	<i>18</i>
<i>Bảng 3.8 TruthTable testcase 2</i>	<i>18</i>

CHƯƠNG 1 – GIỚI THIỆU

1.1. Chi tiết công việc cá nhân

- Tìm các hàm liên quan đến list, dictionary, tuple.
- Tìm cách thực hiện các toán tử thông qua thư viện operator.
- Tìm các thao tác trên chuỗi.
- Tìm cách sử dụng hàm thông qua dictionary.
- Tìm cách thêm 1 phần tử từ 1 list vào 1 tuple trong 1 list.
- Thực hiện code từ những thông tin tìm được.
- Kiểm tra code bằng các testcase có sẵn.
- Chú thích các dòng code.

1.2. Sơ lược các chương tiếp theo

Các lý thuyết trong bài được thể hiện ở chương 2, với khái niệm về biểu thức Ba Lan ngược, thuật toán chuyển đổi và tính toán Postfix cùng những phép toán logic cơ bản được sử dụng để tính bảng chân trị. Chương tiếp theo là các bước thực hiện việc chuyển đổi biểu thức và tính bảng chân trị trên 2 ví dụ bằng phần code trên ngôn ngữ python đã làm trước đó. Cuối cùng, chương 4 là kết quả sau khi chạy code theo các testcase cho sẵn.

1.2.1 Chương 2 -Phân lý thuyết

1.2.2 Chương 3-Chuyển đổi biểu thức và tính bảng chân trị

1.2.3 Chương 4- Kết quả thực nghiệm

1.2.4 Chương 5- Tài liệu tham khảo

CHƯƠNG 2 – CƠ SỞ LÝ THUYẾT

2.1. Biểu thức Ba Lan ngược (Reverse Polish notation)

2.1.1. Sơ lược về biểu thức Ba Lan:

Có thể phân biệt ba dạng biểu diễn của biểu thức toán học qua ví dụ sau:

- Biểu thức trung tố: $P \mid Q$; Toán tử đặt giữa.
- Biểu thức tiền tố: $\mid P Q$; Toán tử đặt trước (Ký pháp Ba Lan).
- Biểu thức hậu tố: $P Q \mid$; Toán tử đặt sau (Ký pháp Ba Lan ngược).

Biểu thức Ba Lan (Polish notation) hay biểu thức tiền tố (Prefix notation), là một cách viết biểu thức toán học thuận lợi cho việc thực hiện tính toán trên hệ thống máy tính, được phát minh vào những năm 1920 bởi nhà toán học người Ba Lan Jan Lucasiewicz.

Điểm cơ bản của cách viết này là không cần dùng đến các dấu ngoặc và luôn thực hiện từ trái sang phải. Trong đó các toán tử đi trước toán hạng của chúng, trái ngược với biểu thức Infix, trong đó các toán tử được đặt giữa các toán hạng.

Ví dụ: Infix: “ $A - B * C$ ” sang biểu thức Prefix: “ $- A * B C$ ”.

Tuy nhiên, trong biểu thức Prefix khi toán tử tính từ trái sang phải, chúng sử dụng các giá trị ở bên phải và nếu chính các giá trị này liên quan đến tính toán thì điều này sẽ thay đổi thứ tự mà các toán tử phải được xét đến. Xét phép tính “ $A - B * C$ ”, biểu thức Prefix sẽ biểu diễn “ $- A * B C$ ” với phép $* B C$ sẽ được tính trước trong khi dấu “ $+$ ” vẫn còn nằm ở phía trái phép tính. Điều này làm phá vỡ thứ tự bài toán dẫn đến tăng truy cập bộ nhớ máy tính, làm thao tác lập trình trở nên phức tạp hơn.

2.1.2. Biểu thức Ba Lan ngược:

Vào cuối những năm 1950, nhà triết học và nhà khoa học máy tính người Úc Charles L. Hamblin đã đề nghị giải quyết các thuật toán dựa trên các đặc điểm của biểu thức Ba Lan. Nhưng thay vào đó, các toán tử được đặt sau các toán hạng và từ đó ra đời

biểu thức Ba Lan ngược (Reverse Polish notation) hay còn gọi là biểu thức hậu tố (Postfix notation).

Ví dụ: Infix: “ $A - B * C$ ” sang biểu thức Postfix: “ $A B C * -$ ”.

Với biểu thức Postfix, vì các toán tử sử dụng các giá trị ở bên trái của chúng, bất kỳ giá trị nào liên quan đến tính toán sẽ được tính toán khi đi từ trái sang phải. Do đó thứ tự đánh giá của các toán tử sẽ liền mạch, không bị phá vỡ theo cách tương tự như trong các biểu thức Tiền tố. Từ đó, việc ứng dụng biểu thức Postfix sẽ đơn giản hóa, giảm truy cập tài nguyên bộ nhớ trong việc xử lý bài toán trên máy tính.

❖ Ví dụ: Trong kí hiệu thông thường, một bài toán được biểu diễn theo thứ tự các số hạng toán tử như sau: $(1+5) * (5-2)$

- Theo nguyên tắc thông thường, dấu ngoặc sẽ được ưu tiên, sau đó đến phép nhân và phép chia (theo thứ tự lần lượt, cái nào có trước thực hiện trước), cuối cùng sẽ là phép cộng và phép trừ.
 - Còn theo kí pháp nghịch đảo Ba Lan, các toán tử sẽ được viết: $1\ 5\ +\ 5\ 2\ -\ *$
- ⇒ Tiến trình thực hiện:
- Đẩy 1 lên ngăn xếp
 - Đẩy 5 lên ngăn xếp, ngăn xếp hiện tại chứa (5, 1)
 - Thao tác cộng: lấy hai số trên cùng ra khỏi ngăn xếp (5, 1), cộng chúng với nhau và đưa kết quả trở vào lại ngăn xếp, bây giờ ngăn xếp sẽ chứa số 6
 - Đẩy 5 lên ngăn xếp
 - Đẩy 2 lên ngăn xếp, ngăn xếp hiện tại chứa (6 5 2)
 - Thao tác trừ: lấy hai số trên cùng ra khỏi ngăn xếp (5 2) trừ đi số trên cùng từ số bên dưới, đưa kết quả trở lại ngăn xếp, bây giờ ngăn xếp sẽ chứa (6 3)

- Thao tác nhân: lấy hai số trên cùng ra khỏi ngăn xếp, nhân chúng lại với nhau và đưa kết quả về lại ngăn xếp. Bây giờ ngăn xếp chứa số 18, là kết quả cuối cùng của phép toán trên.

2.1.3. Thuật toán chuyển đổi từ Infix sang Postfix:

Trong khoa học máy tính, có một dạng câu trúc dữ liệu trừu tượng với phương pháp quản lý dữ liệu LIFO (last in first out) - ngăn xếp stack hỗ trợ cho việc chuyển đổi từ Infix sang Postfix.

Đóng vai trò là tập hợp các phần tử, có hai thao tác chính trên stack:

- Thêm một phần tử vào stack (push).
- Loại bỏ phần tử (pop) được thêm gần nhất ra khỏi stack (đỉnh stack).

Ứng dụng stack vào thuật toán chuyển đổi biểu thức Infix, ta có các bước sau:

- Khởi tạo một stack rỗng.
- Khởi tạo một list rỗng chứa Postfix trả về.
- Cho chạy vòng lặp xét từng phần tử trong Infix:
 - Nếu là 1 toán hạng: đưa vào Postfix.
 - Nếu là 1 toán tử:

(1) Nếu stack rỗng hoặc độ ưu tiên của toán tử đó cao hơn so với đỉnh stack: thêm toán tử vào stack.

(2) Ngược lại nếu độ ưu tiên của toán tử đó thấp hơn so với đỉnh stack: loại bỏ đỉnh stack thêm vào Postfix cho đến khi thỏa (1), thêm toán tử vào stack.

- Nếu là dấu '(': đưa vào stack.
- Nếu là dấu ')':

(1) Loại bỏ đỉnh stack đưa vào Postfix, cho tới khi đỉnh stack là '('.

(2) Loại bỏ '(' khỏi đỉnh stack.

- Sau cùng loại bỏ các phần tử còn lại trong stack đưa vào Postfix.

Kết quả cuối cùng của thuật toán là Postfix tương ứng với Infix đưa vào.

2.1.4. Thuật toán tính biểu thức logic Postfix:

Tương tự với thuật toán chuyển đổi từ Infix sang Postfix, stack cũng được sử dụng để tính biểu thức Postfix cho ra kết quả bài toán.

Đối với biểu thức logic, các chữ cái biểu thị cho các mệnh đề đóng vai trò là các toán hạng trong biểu thức. Các toán tử được xét đến là các toán tử logic (NOT, AND, OR ...).

Thuật toán giải quyết cho biểu thức logic Postfix được đưa ra như sau:

- Khởi tạo một stack rỗng.
- Khởi tạo một danh sách mô tả bảng chân trị có các trường hợp của các mệnh đề trong Postfix.
- Cho chạy vòng lặp xét từng phần tử trong Postfix:
 - Nếu là chữ cái (mệnh đề): đưa vị trí cột giá trị tương ứng vào stack
 - Nếu là toán tử logic (ngoại trừ NOT):
 - (1) Lấy 2 giá trị đỉnh stack (a) và kế đỉnh stack (b) là vị trí 2 cột trong bảng chân trị và thực hiện phép tính logic (b <toán tử> a) đó cho từng hàng.
 - (2) Đưa kết quả vào 1 cột mới tiếp theo trong bảng chân trị.
 - (3) Loại bỏ giá trị vị trí 2 cột đã tính khỏi stack và thêm vị trí cột kết quả mới vào stack.
 - Nếu là toán tử NOT: bước (1) chỉ lấy 1 giá trị đỉnh stack để tính và bước (3) loại bỏ nó, các bước còn lại tương tự.

Kết quả của biểu thức logic sẽ là một bảng chân trị có các cột là giá trị các mệnh đề trong Postfix, các cột sau là kết quả của các phép tính chân trị theo thứ tự của list Postfix. Các hàng của bảng là các trường hợp của các mệnh đề trong Postfix dẫn đến mệnh đề kết quả bài toán.

2.2. Logic cơ bản và sử dụng để tính bảng chân trị.

Các phép toán logic cơ bản sử dụng để tính bảng chân trị gồm: phép phủ định, phép hội, phép tuyển, phép kéo theo, phép tương đương.

2.2.1. Phép phủ định.

Phép phủ định của một mệnh đề P được kí hiệu là $\sim P$ hay $\neg P$, thường thêm từ “không” hoặc “phủ định” vào mệnh đề.

Bảng chân trị biểu diễn phép phủ định:

P	$\sim P$
True	False
False	True

Bảng 2.1 Bảng chân trị phép phủ định.

Ví dụ, mệnh đề “5 là số nguyên tố” phủ định thành “5 không là số nguyên tố”.

2.2.2. Phép hội.

Giữa hai mệnh đề P và Q , phép hội kí hiệu là $P \wedge Q$, đọc là P “và” Q . Phép hội P , Q sẽ đúng khi cả hai mệnh đề P , Q cùng đúng.

Bảng chân trị biểu diễn phép hội:

P	Q	$P \wedge Q$
True	True	True
True	False	False
False	True	False
False	False	False

Bảng 2.2 Bảng chân trị phép hội.

Ví dụ, mệnh đề “5 là số nguyên tố và 3 không là số nguyên tố” là mệnh đề sai.

2.2.3. Phép tuyển.

Giữa hai mệnh đề P và Q , phép tuyển kí hiệu là $P \vee Q$, đọc là P “hoặc” Q . Phép tuyển P, Q sai khi cả hai mệnh đề P, Q cùng sai.

Bảng chân trị biểu diễn phép tuyển:

P	Q	$P \vee Q$
True	True	True
True	False	True
False	True	True
False	False	False

Bảng 2.3 Bảng chân trị phép tuyển.

Ví dụ, mệnh đề “5 là số nguyên tố hoặc 3 không là số nguyên tố” là đúng.

2.2.4. Phép kéo theo.

Giữa hai mệnh đề P và Q , phép kéo theo kí hiệu là $P \rightarrow Q$, đọc là P “kéo theo” Q hay “nếu” P “thì” Q . Phép kéo theo P, Q sai chỉ khi P đúng mà Q sai.

Bảng chân trị biểu diễn phép tuyển:

P	Q	$P \rightarrow Q$
True	True	True
True	False	False
False	True	True
False	False	True

Bảng 2.4 Bảng chân trị phép kéo theo.

Ví dụ, mệnh đề “nếu 5 là số nguyên tố thì 3 không là số nguyên tố” là sai.

2.2.5. Phép tương đương.

Giữa hai mệnh đề P và Q, phép tương đương kí hiệu là $P \leftrightarrow Q$, đọc là P “tương đương” Q hay P “nếu và chỉ nếu” Q. Phép tương đương P, Q sẽ đúng khi cả hai mệnh đề P, Q cùng đúng hoặc cùng sai.

Bảng chân trị biểu diễn phép tương đương:

P	Q	$P \leftrightarrow Q$
True	True	True
True	False	False
False	True	False
False	False	True

Bảng 2.5 Bảng chân trị phép tương đương.

Ví dụ, mệnh đề “5 là số nguyên tố nếu và chỉ nếu 3 là số nguyên tố” là đúng.

2.2.6. Độ ưu tiên của các phép toán logic và các kí hiệu quy ước.

Độ ưu tiên của các phép toán logic và các kí hiệu quy ước trong bài được thể hiện qua bảng sau:

Phép toán	Kí hiệu toán tử	Độ ưu tiên
Phép phủ định	\sim	1
Phép hội	$\&$	2
Phép tuyển	$ $	3
Phép kéo theo	$>$	4
Phép tương đương	$=$	5

Bảng 2.7 Bảng quy ước các toán tử logic cơ bản.

Trong đó, độ ưu tiên cao nhất là phép phủ định (ưu tiên 1), thấp nhất là phép tương đương (ưu tiên 5).

2.3 Basic logic dùng bảng chân trị:

Cơ sở logic gồm:

2.3.1 Logic mệnh đề:

- Là logic rất đơn giản, tuy khả năng biểu diễn của nó còn hạn chế nhưng thuận tiện để đưa vào nhiều khái niệm quan trọng trong logic.
- Giá trị chân trị của mệnh đề: mỗi mệnh đề chỉ có một trong hai giá trị “đúng” kí hiệu là “T” và “Sai” kí hiệu là “F”. Không thể có đồng thời vừa đúng vừa sai.
- Các mệnh đề không thể tách ra thành các mệnh đề đơn giản hơn gọi là mệnh đề đơn. Các mệnh đề có thể tách thành các mệnh đề đơn giản hơn gọi là mệnh đề phức. Nói cách khác, mệnh đề phức được tạo thành từ các mệnh đề đơn.
- Dạng mệnh đề: là biểu thức được cấu tạo từ các mệnh đề, các biến mệnh đề (q, p, r, \dots), các phép toán và dấu đóng mở ngoặc.

2.3.2 Logic Vị Từ:

- Cho phép mô tả thế giới với các đối tượng, các thuộc tính của đối tượng, các quan hệ giữa các đối tượng đó.
- Logic vị từ có thêm một số quy tắc diễn biến khác với logic mệnh đề, chủ yếu được dùng với câu có chứa lượng từ, cho phép biến đổi những câu này thành câu không có lượng từ
- Ví dụ:
 - + Đối tượng: một cái ghế, một cái cây, một cây viết, một sinh viên
 - + Tính chất: cái ghế làm bằng gỗ, cái cây màu xanh lá, cây viết bằng nhựa, đẹp trai, ...
 - + Quan hệ: bạn bè, mẹ con, ông cháu, bên trong, bên ngoài, nằm trên, nằm dưới, nằm giữa, song song
 - + Hàm: là một trường hợp tách riêng của quan hệ, với mỗi đầu vào ta chỉ có một giá trị hàm duy nhất.
- Logic vị từ có cú pháp và ngữ nghĩa được xây dựng dựa trên khái niệm đối tượng. Hệ thống logic này đóng vai trò quan trọng trong việc biểu diễn tri thức do có khả năng biểu diễn phong phú và tự nhiên, đồng thời là cơ sở cho nhiều hệ thống logic khác.

CHƯƠNG 3 – THỰC NGHIỆM

3.1. Chạy tay testcase 1 với Infix = 'R|(P&Q)'

3.1.1. Chuyển đổi Infix sang Postfix

Stack được khởi tạo ban đầu rỗng và giá trị Postfix là 1 list rỗng.

i	Infix[i]	stack	Postfix	Explain
		[]	[]	
1	R	[]	['R']	c is an operand, push c into Postfix
2		[' ']	['R']	c is an operator and stack is empty, then push c into stack
3	([' ', '(']	['R']	c is '(', push c into stack
4	P	[' ', '(']	['R', 'P']	c is an operand, push c into Postfix
5	&	[' ', '(', '&']	['R', 'P']	c is an operator, push c into stack
6	Q	[' ', '(', '&']	['R', 'P', 'Q']	c is an operand, push c into Postfix.
7)	[' ']	['R', 'P', 'Q', '&']	c is “)”, push operator form stack to Postfix until encounter “(”. Then pop off “(”.
8		[]	['R', 'P', 'Q', '&', ' ']	Push the remaining elements from stack to Postfix

Bảng 3.1 Từng bước chuyển Infix sang Postfix testcase 1.

Kết quả Postfix tương ứng là [R, P, Q, &, |].

3.1.2. Tính list Postfix xuất bảng chân trị

Giá trị alphas lúc này là {'P': 0, 'Q': 1, 'R' : 2}, stack rỗng.

Postfix = [R, P, Q, & |].

Truth table ban đầu có 3 cột đánh từ 0 đến 2 dành cho các mệnh đề PQR.

Giá trị stack (1) là trước phép tính, stack (2) là sau khi thực hiện phép tính.

i	Postfix[i]	Alphas tương ứng	stack(1)	Phép tính	Vị trí cột mới	stack(2)
			[]			
0	R	2	[2]			
1	P	0	[2 , 0]			
2	Q	1	[2 , 0 , 1]			
3	&		[2]	Cột 0 & Cột 1	3	[2 , 3]
4			[2 , 3]	Cột 2 Cột 3	4	[4]
			[4]			

Bảng 3.2 Thực hiện tính bảng chân trị từ Postfix testcase 1.

TruthTable thể hiện trên một list với các phần tử tuple là các hàng của bảng chân trị. Kết quả của phép tính trên có phần tiêu đề là:

Cột	0	1	2	3	4
Mệnh đề	P	Q	R	P&Q	R (P&Q)

Bảng 3.3 Tiêu đề bảng chân trị testcase 1

Position	0	1	2	3	4
Statement	P	Q	R	P&Q	R (P&Q)
Value	False	False	False	False	False
	False	False	True	False	True
	False	True	False	False	False
	False	True	True	False	True
	True	False	False	False	False
	True	False	True	False	True
	True	True	False	True	True
	True	True	True	True	True

Bảng 3.4 TruthTable testcase 1

3.2. Chạy tay testcase 2 với Infix = “~P|(Q&R)>R”

3.2.1. Chuyển đổi Infix sang Postfix

Stack được khởi tạo ban đầu rỗng và giá trị Postfix là 1 list rỗng.

i	Infix[i]	stack	Postfix	Explain
		[]	[]	
1	~	['~']	[]	C is an operator, push it into stack.
2	P	['~']	['P']	C is an operand, push it into Postfix.
3		[' ']	['P', '~']	precedence of “~” is greater than “ ”’s, push “~” into Postfix before push “ ” into stack.
4	([' ', '(']	['P', '~']	c is "(", push it into stack.
5	Q	[' ', '(']	['P', '~', 'Q']	
6	&	[' ', '(', '&']	['P', '~', 'Q']	
7	R	[' ', '(', '&']	['P', '~', 'Q', 'R']	
8)	[' ']	['P', '~', 'Q', 'R', '&']	C is “)”, push “&” into Postfix and pop off “(“.
9	>	['>']	['P', '~', 'Q', 'R', '&', ' ']	Precedence of “ ” is greater than “>”’s, push “ ” into Postfix, then push “>” into stack.
10	R	['>']	['P', '~', 'Q', 'R', '&', ' ', 'R']	
11		[]	['P', '~', 'Q', 'R', '&', ' ', 'R', '>']	Push the remaining operator from stack to Postfix.

Bảng 3.5 Từng bước chuyển Infix sang Postfix testcase 2.

Kết quả Postfix tương ứng là [P, ~, Q, R, &, |, R, >].

3.2.2. Tính list Postfix xuất bảng chân trị

Giá trị alphas lúc này là { 'P': 0, 'Q' : 1 , 'R' : 2}, stack rỗng, Postfix = [P, ~, Q, R, &, |, R, >].

Truthtable (bảng chân trị) ban đầu có 3 cột đánh từ 0 đến 2 dành cho các mệnh đề PQR.

Giá trị stack (1) là giá trị stack trước phép tính, stack(2) là sau khi thực hiện phép tính.

i	Postfix[i]	Alphas tương ứng	stack(1)	Phép tính	Vị trí cột mới	stack(2)		
			[]					
0	P	0	[0]					
1	~		[0]	~ Cột 0	3	[3]		
2	Q	1	[3, 1]					
3	R	2	[3, 1, 2]					
4	&			[3, 1, 2]		Cột 1 & Cột 2	4	[3, 4]
5				[3, 4]		Cột 3 Cột 4	5	[5]
6	R	2		[5, 2]				
7	>			[5, 2]		Cột 5 > Cột 2	6	[6]

Bảng 3.6 Thực hiện tính bảng chân trị từ Postfix testcase 2.

Truth table thể hiện trên một list với các phần tử tuple là các hàng của bảng chân trị. Kết quả của phép tính trên có phần tiêu đề là:

Cột	0	1	2	3	4	5	6
Mệnh đề	P	Q	R	$\sim P$	$Q \& R$	$\sim P \mid (Q \& R)$	$\sim P \mid (Q \& R) > R$

Bảng 3.7 Tiêu đề bảng chân trị testcase 2

Position	0	1	2	3	4	5	6
Statement	P	Q	R	$\sim P$	$Q \& R$	$\sim P \mid (Q \& R)$	$\sim P \mid (Q \& R) > R$
Value	False	False	False	True	False	True	False
	False	False	True	True	False	True	True
	False	True	False	True	False	True	False
	False	True	True	True	True	True	True
	True	False	False	False	False	False	True
	True	False	True	False	False	False	True
	True	True	False	False	False	False	True
	True	True	True	False	True	True	True

Bảng 3.8 TruthTable testcase 2

3.3 Code và giải thích

```

8  # độ ưu tiên các toán tử
9  # toán tử có độ ưu tiên càng cao thì chữ số càng nhỏ
10 def priorityOfOperator(op):
11     switcher = {
12         '~': 1,
13         '&': 2,
14         '|': 3,
15         '>': 4,
16         '=': 5,
17     }
18     return switcher.get(op, 10)
19

```

```

20 # hàm lấy giá trị cuối của stack
21 def peek(stack):
22     if stack:
23         return stack[len(stack)-1]
24     return 0
25
26 def Infix2Postfix(Infix):
27
28     #bắt đầu khởi tạo
29     isWord = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz' #dãy chữ để kiểm tra chữ cái trong infix
30     stack = [] #khởi tạo stack để tính toán postfix
31     postfix = [] #khởi tạo postfix
32     #kết thúc khởi tạo
33
34     #chạy vòng lặp lấy từng giá trị trong infix đã cho và tính toán
35     for i in Infix:
36         if isWord.find(i) != -1:
37             #nếu giá trị là chữ cái thì bỏ vào postfix đã khởi tạo
38             postfix.append(i)
39         elif i == '(':
40             #nếu giá trị là '(' thêm vào stack
41             stack.append(i)
42         elif i == ')':
43             #gặp toán tử ), bắt đầu kiểm tra stack rỗng hay không
44             #nếu stack không rỗng, bắt đầu pop từ stack ra append vào postfix đến khi nào gặp giá trị ) trong stack
45             if stack:
46                 while peek(stack) != '(':
47                     postfix.append(stack.pop())
48                 stack.pop()
49             else:
50                 #những trường hợp ưu tiên tiếp theo
51                 op = priorityOfOperator(i) #tính toán độ ưu tiên toán tử theo hàm đã viết
52                 if op == 1:
53                     #nếu độ ưu tiên là 1 -> toán tử NOT
54                     if stack:
55                         #kiểm tra stack có rỗng hay không
56                         #nếu không rỗng thì bắt đầu so sánh với giá trị cuối của stack
57                         #trúng trường hợp thì pop khỏi stack và append vào postfix
58                         #những trường hợp khác thì append vào stack
59                         if priorityOfOperator(peek(stack)) == 1:
60                             postfix.append(stack.pop())
61                             stack.append(i)
62                         else:
63                             stack.append(i)

```



```

64         else:
65             stack.append(i)
66     else:
67         if stack:
68             if priorityOfOperator(peek(stack)) == op:
69                 postfix.append(stack.pop())
70                 while stack and priorityOfOperator(peek(stack)) <= op:
71                     postfix.append(stack.pop())
72                 stack.append(i)
73             elif priorityOfOperator(peek(stack)) < op:
74                 postfix.append(stack.pop())
75                 while stack and priorityOfOperator(peek(stack)) <= op:
76                     postfix.append(stack.pop())
77                 stack.append(i)
78             else:
79                 stack.append(i)
80         else:
81             stack.append(i)
82     #sau khi chạy vòng lặp hoàn tất của infix
83     #kiểm tra stack rỗng hay không, nếu chưa rỗng thì pop hết từ stack, append vào postfix
84     while stack:
85         postfix.append(stack.pop())
86     return postfix #kết quả trả về là postfix dạng list
87
88 def Postfix2Truthtable(Postfix):
89
90     # bắt đầu khởi tạo
91     list_var=[]           #danh sách các biến trong postfix
92     truth_table=[]        #bảng chân trị
93     row_truth_table = [] #hàng trong bảng chân trị
94     dict_var={}           #từ điển tìm kiếm kết quả của mỗi biến
95     var_truth_table=[]    #các biến hiển thị trong bảng chân trị
96     stack=[]              # khởi tạo stack để tính toán postfix
97     tempValue=''          #biến lưu trữ kết quả tạm của bảng
98     #kết thúc khởi tạo
99
100     # tìm tổng cộng các biến trong postfix
101     isWord='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
102     for i in Postfix:
103         if isWord.find(i) != -1: #nếu phần tử trong postfix là chữ cái thì thêm vào list_var
104             list_var.append(i)
105     list_var= list(set(list_var)) #lọc các biến trùng nhau và sắp xếp theo bảng chữ cái
106     list_var.sort()
107
108     # thêm biến vào thư viện biến khởi tạo các biến bằng rỗng
109     for i in list_var:
110         dict_var[i]=''
111
112     # tạo tổ hợp kết quả True False cho các biến trong bảng chân trị
113     var_truth_table=list(itertools.product([False, True], repeat=len(list_var)))
114

```

```

115 # chạy vòng lặp để tính kết quả cho bảng, #
116 # lấy từng hàng của tổ hợp đã được tập hợp sẵn cho var_truth_table chạy vòng lặp để tính kết quả từ postfix
117 # sau đó lưu từng hàng kết quả vào row_truth_table và đưa chúng vào truth_table
118 # mỗi lần chạy vào lặp, dùng thư viện để xác định giá trị của các biến sau đó sẽ lưu vào stack thực hiện xử lý toán tử sau đó
119 for row in var_truth_table: #từng hàng của bảng
120     for i in range(len(row)): #chạy vòng lặp lấy kết quả của từng biến trong tổ hợp gán vào thư viện biến
121         dict_var[list_var[i]]=row[i]
122         row_truth_table.append(row[i])
123
124     for i in Postfix: #chạy vòng lặp xử lý những toán tử trong postfix
125         if isWord.find(i) !=-1:
126             stack.append(dict_var[i])
127         else:
128             if i=='~':
129                 tmp=stack.pop()
130                 tempValue=not(tmp)
131             else:
132                 b=stack.pop()
133                 a=stack.pop()
134                 if i=='&':
135                     tempValue=a and b
136                 elif i=='|':
137                     tempValue=a or b
138                 elif i=='>':
139                     tempValue=not(a) or b
140                 elif i=='=':
141                     tempValue=a == b
142                 stack.append(tempValue)
143                 row_truth_table.append(tempValue)
144     truth_table.append(tuple(row_truth_table)) #thêm từng hàng kết quả vào bảng chân trị
145     row_truth_table=[] #khởi tạo lại từng hàng rỗng để lưu kết quả của hàng tiếp theo
146 return truth_table

```

Hình 3. Code demo

CHƯƠNG 4 – KẾT QUẢ

4.1. Kết quả testcase 1

- Infix = “R|(P&Q)”
- Postfix xuất ra màn hình là:

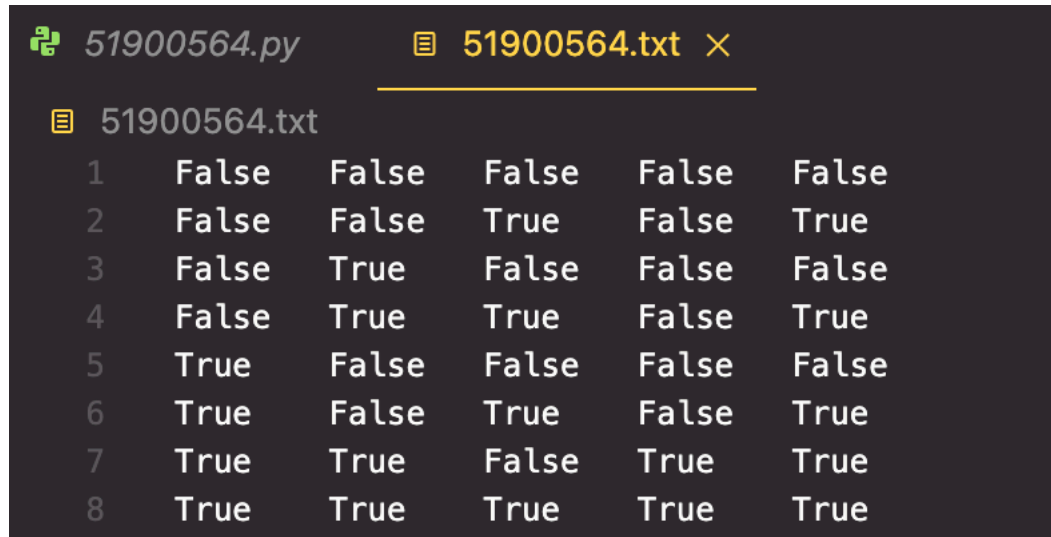
```

nguyenthianhthu@NguyenThiAnhThu-MacBook-Pro btl % python3 51900564.py
['R', 'P', 'Q', '&', '|']
nguyenthianhthu@NguyenThiAnhThu-MacBook-Pro btl %

```

Hình 4.1 Postfix ở testcase 1

- Bảng chân trị của bài toán:



1	False	False	False	False	False
2	False	False	True	False	True
3	False	True	False	False	False
4	False	True	True	False	True
5	True	False	False	False	False
6	True	False	True	False	True
7	True	True	False	True	True
8	True	True	True	True	True

Hình 4.2 Bảng chân trị cho testcase 1

4.2. Kết quả testcase 2

- Infix = “ $\sim P|(Q \& R) > R$ ”
- Postfix xuất ra màn hình là:

```
[Running] python -u "/Users/nguyenthianhthu/Documents/CTRR/btl/51900564.py"
['P', '~', 'Q', 'R', '&', '|', 'R', '>']
```

Hình 4.3 Postfix ở testcase 2

- Bảng chân trị của bài toán:



1	False	False	False	True	False	True	False
2	False	False	True	True	False	True	True
3	False	True	False	True	False	True	False
4	False	True	True	True	True	True	True
5	True	False	False	False	False	False	True
6	True	False	True	False	False	False	True
7	True	True	False	False	False	False	True
8	True	True	True	False	True	True	True

Hình 4.4 Bảng chân trị cho testcase 2

4.3. Kết quả testcase 3

- Infix = “P|(R&Q)”
- Postfix xuất ra màn hình là:

```
[Running] python -u "/Users/nguyenthianhthu/Documents/CTRR/btl/51900564.py"
['P', 'R', 'Q', '&', '|']
```

Hình 4.5 Postfix ở testcase 3

- Bảng chân trị của bài toán:



	P	R	Q	&	
1	False	False	False	False	False
2	False	False	True	False	False
3	False	True	False	False	False
4	False	True	True	True	True
5	True	False	False	False	True
6	True	False	True	False	True
7	True	True	False	False	True
8	True	True	True	True	True
9					

Hình 4.6 Bảng chân trị cho testcase3

4.4. Kết quả testcase 4

- Infix = “(P>Q) &(Q>R)”
- Postfix xuất ra màn hình là:

```
[Running] python -u "/Users/nguyenthianhthu/Documents/CTRR/btl/51900564.py"
['P', 'R', 'Q', '&', '|']
```

Hình 4.7 Postfix ở testcase 4

- Bảng chân trị của bài toán:

51900564.txt ✕

51900564.txt

1	False	False	False	True	True	True
2	False	False	True	True	True	True
3	False	True	False	True	False	False
4	False	True	True	True	True	True
5	True	False	False	False	True	False
6	True	False	True	False	True	False
7	True	True	False	True	False	False
8	True	True	True	True	True	True

Hình 4.8 Bảng chân trị cho testcase 4

4.5. Kết quả testcase 5

- Infix = “ $(P|\sim Q) > \sim P = (P|(\sim Q)) > \sim P$ ”
- Postfix xuất ra màn hình là:

```
[Running] python -u "/Users/nguyenthianhthu/Documents/CTRR/btl/51900564.py"
['P', 'Q', '~', '|', 'P', '~', '>', 'P', 'Q', '~', '|', 'P', '~', '>', '=']
```

Hình 4.9 Postfix ở testcase 5

- Bảng chân trị của bài toán:

51900564.py 51900564.txt ✕

51900564.txt

1	False	False	True	True	True	True	True	True	True	True	True
2	False	True	False	False	True	True	False	False	True	True	True
3	True	False	True	True	False	False	True	True	False	False	True
4	True	True	False	True	False	False	False	True	False	False	True
5											

Hình 4.10 Bảng chân trị cho testcase 5

TÀI LIỆU THAM KHẢO

Tiếng Việt

- [1] Nguyễn Thị Huỳnh Trâm, truy cập ngày 28/11/2021, [Tài liệu môn học](#).
- [2] Ngô Xuân Bách (2021) , truy cập ngày 10/12/2021, [Một số kiến thức cơ bản](#).
- [3] Từ Minh Phương (2021), truy cập ngày 10/12/ 2021, [Logic vị từ](#).
- [4] Miracleandeffort.LĐT (2016), truy cập ngày 11/12/ 2021, [Thuật toán tính biểu thức bằng biến đổi Ba Lan ngược](#).

Tiếng Anh

- [1] Wiki, truy cập ngày 11/12/2021, [Reverse Polish notation](#)
- [2] Abhishek Jain, on June 14, 2017, truy cập ngày 12/12/2021, [Infix to Postfix conversion using stack](#)