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

BÀI TẬP TIỂU LUẬN

MÔN CẤU TRÚC RỜI RẠC

Chuyển biểu thức logic trung tố sang hậu tố (Infix to Postfix) và lập bảng chân trị cho biểu thức.

Người hướng dẫn: **Cô Võ Hoàng Anh**

Người thực hiện: **Tô Vĩnh Khang – 51800408**

Bùi Quang Khải – 51800785

Lớp : **18050203**

Khoá : **22**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2019

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

BÀI TẬP TIỂU LUẬN

MÔN CẤU TRÚC RỜI RẠC

Chuyển biểu thức logic trung tố sang hậu tố (Infix to Postfix) và lập bảng chân trị cho biểu thức.

Người hướng dẫn: **Cô Võ Hoàng Anh**
Người thực hiện: **Tô Vĩnh Khang - 51800408**
Bùi Quang Khải - 51800785
Lớp : **18050203**
Khoá : **22**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2019

LỜI CẢM ƠN

Em xin chân thành cảm ơn cô Võ Hoàng Anh đã giảng dạy và hướng dẫn tận tình cho em trong suốt thời gian học môn Thực hành Cấu trúc rời rạc này. Giúp em có thêm kiến thức về các thuật toán hữu ích cũng như ứng dụng sử dụng những hàm có sẵn trong ngôn ngữ lập trình Python.

Cuối cùng , em xin cảm ơn Trường Đại học Tôn Đức Thắng và Khoa Công nghệ thông tin đã tạo điều kiện cho em được học tập và hoàn thành tốt môn học này.

TÓM TẮT

Ứng dụng giải thuật RPN (Reverse Polish Notation) để chuyển đổi các biểu thức logic từ trung tố (Infix) sang hậu tố (Postfix) và sử dụng các biểu thức logic cơ bản , các giải thuật con kèm theo để tính bảng chân trị cho biểu thức. Giải thích ý nghĩa và chạy thực nghiệm code cho các testcase.

MỤC LỤC

DANH MỤC HÌNH

DANH MỤC BẢNG

CHƯƠNG 1 – GIỚI THIỆU

1.1 Giới thiệu và phân công công việc

1.1.1 Thành viên nhóm

- Họ tên: Tô Vĩnh Khang
  MSSV: 51800408
  Lớp: 18050203
- Họ tên: Bùi Quang Khải
  MSSV: 51800785
  Lớp: 18050203

1.1.2 Bảng phân công

Tên	Công việc	Bắt đầu – Kết thúc
Tô Vĩnh Khang	Tìm hiểu thuật toán Infix to Postfix cho Python: Kí hiệu biểu thức logic cơ bản , độ ưu tiên của biểu thức, tìm hiểu các hàm trong thư việc Stack.	07/10/2019 – 11/10/2019
	Thảo luận và đưa ra ý tưởng.	11/10/2019 – 12/10/2019
	Hiện thực bằng code Python.	12/10/2019 – 19/10/2019
	Trao đổi và kiểm thử code.	20/10/2019 – 24/10/2019
	Báo cáo tiểu luận.	25/10/2019 – 31/10/2019
Bùi Quang Khải	Tìm hiểu thuật toán lập bảng chân trị bằng các biểu thức logic từ dạng Postfix.	07/10/2019 – 11/10/2019
	Thảo luận và đưa ra ý tưởng.	11/10/2019 – 12/10/2019
	Hiện thực bằng code Python.	12/10/2019 – 19/10/2019
	Trao đổi và kiểm thử code.	20/10/2019 – 24/10/2019
	Báo cáo tiểu luận.	25/10/2019 – 31/10/2019

Bảng 1.1 Phân công công việc nhóm

1.2 Công việc cá nhân đã làm

[07/10/2019 18:00]: Phân công công việc với Khải tại The Coffee House Quận 11.

[07/10/2019 21:05]: Tìm hiểu giải thuật Infix to Postfix bằng Python.

[08/10/2019 08:52]: Tiếp tục tìm hiểu giải thuật Infix to Postfix bằng Python.

[08/10/2019 10:39]: Tìm hiểu biểu thức logic cơ bản.

[08/10/2019 13:17]: Tìm hiểu thư viện Stack của Python.

[08/10/2019 13:34]: Tìm hiểu các hàm trong thư viện Stack.

[08/10/2019 13:58]: Code thử từng hàm.

[08/10/2019 19:41]: Tìm cách gán độ ưu tiên (Precedence) cho các toán tử (Operator).

[09/10/2019 07:56]: Tìm cách để nhận dạng phần tử truyền vào là chữ cái của alphabet.

[09/10/2019 09:01]: Xem lại bài tập lớn Cấu trúc dữ liệu giải thuật bằng ngôn ngữ Java để lấy ý tưởng.

[09/10/2019 14:22]: Kết quả đoạn code chưa như mong muốn.

[09/10/2019 14:48]: Chỉnh sửa và chạy lại.

[09/10/2019 14:55]: Kết quả output trả về đúng mong muốn. Kiểm thử lại lần cuối.

[09/10/2019 14:59]: Hoàn thành công việc sớm hơn tiến độ phân công.

[10/10/2019 20:46]: Tìm hiểu trước Postfix to Truthtable.

[10/10/2019 22:09]: Nghiên cứu đoạn code Testrun.py

[11/10/2019 19:00]: Thảo luận tại Garage Coffee Lữ Gia Quận 11 về các đoạn code và nêu ý tưởng với Khải.

[12/10/2019 20:11]: Chỉnh sửa đoạn code trong hàm Infix2Postfix() trả về kết quả phù hợp với tham số của hàm Postfix2Truthtable() đang cần.

[12/10/2019 20:29]: Tìm hiểu được cách sắp xếp thứ tự bảng chữ cái alphabet.

[12/10/2019 20:33]: Tạo bảng chân trị tạm để append các kết quả trả về vào bảng đó.

[12/10/2019 21:56]: Bảng chân trị chưa trả về đúng kết quả mong muốn và chưa thể xuống hàng.

[13/10/2019 19:42]: Phát hiện ra cần viết thêm hàm cho biểu thức "=" để trả về kết quả đúng nhất.

[13/10/2019 20:11]: Nãy ra ý tưởng về lồng ghép nhiều mảng lại.

[13/10/2019 00:09]: Kết quả không tiến triển thêm. Tiếp tục tìm hiểu.

[14/10/2019 19:55]: Kết quả không tiến triển thêm. Tiếp tục tìm hiểu.

[20/10/2019 14:04]: Thảo luận về cách xuống dòng cho bảng chân trị tại Garage Coffee Lữ Gia Quận 11 với Khải

[20/10/2019 18:38]: Kết quả không tiến triển thêm. Tiếp tục tìm hiểu.

[20/10/2019 20:14]: Phát hiện ra định dạng kiểu tuple và bảng chân trị đã xuống hàng.

[20/10/2019 22:08]: Sau một vài lần chỉnh sửa thì kết quả trả về đúng theo yêu cầu.

[20/10/2019 22:10]: Chạy file Testrun.py thì testcase trả về kết quả True cho toàn bộ.

[20/10/2019 22:10]: Tạo thêm các testcase khác để kiểm thử.

[26/10/2019 17:05]: Thảo luận về nội dung và cách trình bày báo cáo tiểu luận tại The Coffee House Q11 với Khải.

[29/10/2019 18:11]: Đã hoàn thành chương 1,2,4. Tiếp tục suy nghĩ và trình bày nội dung chương 3.

[29/10/2019 21:11]: Hoàn thành chương 3 cùng với việc đánh số tự động cho mục lục.

[30/10/2019 17:54]: Kiểm tra lại lần cuối toàn bộ tại Garage Coffee Lữ Gia Quận 11 với Khải.

[30/10/2019 19:22]: Hoàn thành bài báo cáo tiểu luận.

CHƯƠNG 2 – CƠ SỞ LÝ THUYẾT

2.1 Lý thuyết về Reverse Polish Notation (RPN)

- Reverse Polish Notation (RPN) hay còn gọi là kí hiệu Ba Lan ngược. Là một ký hiệu toán học trong đó dấu đi theo toán hạng. Ví dụ:

• Trong ký hiệu thông thường, bài toán logic được viết như sau:

P & Q

• Trong ký hiện Ba Lan ngược, bài toán được viết thành:

P Q &

- Biểu thức Ba Lan ngược là biểu thức được thể hiện ở dạng hậu tố (postfix). Ví dụ:

• Biểu thức infix như sau:

(P | Q & R) > K

• Được chuyển sang dạng postfix như thành:

P Q | R & K >

2.2 Lý thuyết về các biểu thức logic cơ bản

- Logic là nội dung trung tâm của khoa học máy tính, về các định lý về sự không toàn vẹn. Logic nghiên cứu và phân loại cấu trúc của các khẳng định và các lý lẽ, cả hai đều thông qua việc nghiên cứu các hệ thống hình thức của việc suy luận và qua sự nghiên cứu lý lẽ trong ngôn ngữ tự nhiên.

- Ngày nay, logic được sử dụng phổ biến trong lý thuyết lý luận, ứng dụng rộng rãi trong các lãnh vực của trí tuệ nhân tạo, chứng minh định lí tự động,..

- Các biểu thức logic cơ bản như:

~	Phép phủ định
V	Phép tuyển
Λ	Phép hội
→	Phép kéo theo
↔	Phép tương đương

Bảng 2.1 Chú thích về các biểu thức logic cơ bản

• Phép phủ định:

Phủ định của mệnh đề P là một mệnh đề. Nó đúng khi P sai và sai khi P đúng.

• Phép tuyển:

Tuyển của hai mệnh đề P,Q là một mệnh đề. Nó sai khi cả hai mệnh đề P,Q cùng sai và đúng cho các trường hợp còn lại.

• Phép hội:

Hội của hai mệnh đề P,Q là một mệnh đề. Nó đúng khi cả hai mệnh đề P,Q cùng đúng và sai cho các trường hợp còn lại.

• Phép kéo theo:

P kéo theo Q là một mệnh đề. Nó chỉ sai khi P đúng nhưng Q sai. Các trường hợp còn lại luôn đúng.

• Phép tương đương:

P tương đương Q là một mệnh đề. Nó đúng khi cả hai mệnh đề P,Q cùng đúng hoặc cùng sai.

2.3 Sử dụng để tính bảng chân trị

• Phép phủ định:

P	~P
True	False
False	True

Bảng 2.2 Bảng chân trị của phép phủ định

• Phép tuyển:

P	Q	P V Q
True	True	True
True	False	True
False	True	True
False	False	False

Bảng 2.3 Bảng chân trị của phép tuyển

• Phép hội:

P	Q	P ∧ Q
True	True	True
True	False	False
False	True	False
False	False	False

Bảng 2.4 Bảng chân trị của phép hội

• Phép kéo theo:

P	Q	P → Q
True	True	True
True	False	False
False	True	True
False	False	True

Bảng 2.5 Bảng chân trị của phép kéo theo

• Phép tương đương:

P	Q	P ↔ Q
True	True	True
True	False	False
False	True	False
False	False	True

Bảng 2.6 Bảng chân trị của phép tương đương

CHƯƠNG 3 – THỰC NGHIỆM

3.1 Một số hàm cần thiết được ứng dụng để hiện thực bài toán

-Hàm **join():**

Trả về một chuỗi được nối với các phần tử của một lần lặp.

-Hàm **split():**

Chia một chuỗi thành một danh sách, có thể chỉ định dấu phân cách, dấu phân cách mặc định là bất kỳ khoảng trắng.

-Hàm **isalpha():**

Trả về True nếu tất cả các ký tự trong chuỗi thuộc bảng chữ cái alphabet. Nếu không, nó trả về False.

-Hàm **append():**

Lấy một phần từ duy nhất và thêm nó vào cuối danh sách.

-Hàm **push():**

Đẩy phần tử vào trong ngăn xếp.

-Hàm **pop():**

Xóa phần tử tại vị trí đã cho trong ngăn xếp.

-Hàm **isEmpty():**

Kiểm tra xem ngăn xếp đó có rỗng hay không. Trả về True nếu rỗng. Nếu không , nó trả về False.

-Hàm **peek():**

Trả về giá trị top trong ngăn xếp mà không xóa phần tử khỏi ngăn xếp. Do đó, nó trả về cùng một giá trị như các hoạt động như pop() nhưng không sửa đổi dữ liệu.

-Hàm **sorted():**

Sắp xếp chuỗi kiểu list hoặc tuple và trả về một danh sách với các thành phần theo cách sắp xếp (từ nhỏ đến lớn, từ A đến Z theo bảng chữ cái alphabet).

-Hàm **Lambda arguments : expression**

Là một hàm ẩn danh nhỏ, có thể nhận bất kỳ số lượng đối số, nhưng chỉ có thể có một biểu thức.

-Hàm **bin():**

Chuyển các giá trị thập phân sang giá trị nhị phân.

-Hàm **rjust():**

Trả về chuỗi mới với chiều dài được hàm nhận vào sau khi thay thế các giá trị ở bên trái chuỗi gốc.

-Hàm **map():**

Hàm nhận vào 2 tham số là phần tính toán và list, set hoặc tuple. Hàm sẽ dùng phần tính toán để tính toán cho phần list, set hoặc tuple.

-Hàm **dict():**

Tạo một danh sách từ điển.

-Hàm **zip():**

Gộp từng phần tử của nhiều list lại dựa trên thứ tự từng phần tử trong những list đó.

-Hàm **tuple():**

Tạo một chuỗi gồm nhiều phần tử.

3.2 Chạy tay theo code cho Testcase dòng 1

-Trong hàm **Infix2Postfix(Infix):**

• Đầu vào: R|(P&Q)

Tạo một từ điển mang tên **Operator** để gán các biểu thức logic với độ ưu tiên của nó.

=> Kết quả: {'=': 1, '>': 2, '|': 3, '&': 4, '~': 5, '(': 0, ')': 0}

Dùng ***Infix = " ".join(Infix)*** để nối cái phần tử khoảng trắng vào ***Infix.***

=> Kết quả: R | (P & Q)

Dùng vòng lặp ***for i in tokens:*** để duyệt từng phần tử trong danh sách

Nếu duyệt đến phần tử trong danh sách là kí tự trong bảng chữ cái alphabet thì ghi vào chuỗi mang tên ***Postfix.***

Nếu duyệt đến phần tử trong danh sách là "(" hoặc biểu thức logic thì chuyển vào ngăn xếp mang tên ***temp.***

Nếu duyệt đến phần tử trong danh sách là ")" thì đưa hết phần tử trong ngăn xếp ***temp*** ghi vào chuỗi ***Postfix*** đến khi gặp dấu "(" và xóa dấu "(" ra khỏi ngăn xếp.

Sẽ được viết như sau:

```
if (i.isalpha()):
    Postfix.append(i)
```

```
elif (i=="("):
        temp.push(i)
elif (i==")"):
        top = temp.pop()
        while (top != "("):
                Postfix.append(top)
                top = temp.pop()
```

Trong quá trình thêm biểu thức logic vào Stack, nếu biểu thức logic đang thêm vào có độ ưu tiên thấp hơn toán tử ở đỉnh thì ta phải lấy hết các biểu thức logic có độ ưu tiên cao hơn ra để ghi vào chuỗi trước rồi mới thêm biểu thức logic đang xét vào Stack thông qua dòng code:

```
while ((not temp.isEmpty()) and (Operator[temp.peek()] >= Operator[i])):
        Postfix.append(temp.pop())
temp.push(i)
```

Khi đã duyệt xong không còn kí tự nào thì sẽ ghi tất cả những gìcòn lại trong ngăn xếp ***temp*** vào chuỗi ***Postfix*** bằng dòng code bên ngoài vòng lặp for:

```
while (not temp.isEmpty()):
        Postfix.append(temp.pop())
```

Ký tự từ chuỗi infix	Bước thực hiện	Stack hiện tại	Ghi chuỗi postfix
R	Alphabet – Ghi vào chuỗi		R
|	Dấu – Thêm vào Stack	|	R
(	Dấu "(" – Thêm vào Stack	| (	R
P	Alphabet – Ghi vào chuỗi	| (	R P
&	Dấu – Thêm vào Stack.	| (&	R P
Q	Alphabet – Ghi vào chuỗi	| (&	R P Q
)	Dấu ")" – Đưa hết phần tử trong Stack ghi vào chuỗi đến khi gặp dấu (và xóa dấu (ra khỏi Stack	|	R P Q &
Không còn kí tự	Ghi tất cả những dấu còn lại trong Stack vào chuỗi		R P Q & |

Bảng 3.1 Chạy tay theo code chi tiết Testcase dòng 1 - Hàm Infix2Postfix(Infix)

- **Đầu ra: ['R', 'P', 'Q', '&', '|']**

-Trong hàm **Postfix2Truthtable(Postfix):**

• Đầu vào: ['R', 'P', 'Q', '&', '|']

Khi truyền biểu thức ***Postfix*** vào hàm thì vòng lặp ***for i in Postfix*** sẽ lấy các biến chữ cái của biểu thức Postfix và lưu vào list ***inp***. Phần ***lệnh not i in inp*** giúp tránh lưu các biến chữ cái trùng nhau.

```
for i in Postfix:
    if i.isalpha() and not i in inp:
        inp.append(i)
```

Sau đó sắp xếp các biến chữ cái theo thứ tự alphabet.

```
sort_characters=sorted(inp)
```

Tiếp đến, vòng lặp ***for i in range(2**len((inp)))*** sẽ dựa vào số lượng các chữ cái trong Postfix để tạo bảng chân trị với số trường hợp bằng 2^N (với N là số chữ cái)

Biến ***ListBinary*** sẽ khởi tạo một dãy số nhị phân tương ứng với các biến chữ cái.

```
ListBinary = bin(i)[2:].rjust(len((inp)), '0')
```

Sau đó ***BooleansInp*** sẽ tạo ra một list các giá trị logic True hoặc False tương ứng với các giá trị 0 hoặc 1 của dãy số nhị phân được tạo ra ở ***ListBinary***.

```
BooleansInp = list(map((lambda Postfix: True if Postfix == '1' else False), ListBinary))
```

Tiếp đến ta tạo ra một từ điển ***k*** để gán các giá trị logic True hoặc False tương ứng với các biến chữ cái.

```
k = dict(zip((inp), BooleansInp))
```

Vòng lặp ***for m in BooleansInp*** sẽ lưu các giá trị logic True hoặc False tương ứng của các biến chữ cái vào list ***TempTable***.

```
for m in BooleansInp:
        TempTable.append(m)
```

Tiếp đến vòng ***lặp for j in Postfix*** sẽ dò các phần tử trong ***Postfix***, nếu là các biến chữ cái thì sẽ lưu vào list Truthtable các giá trị logic True hoặc False tương ứng với các biến chữ cái trong từ điển ***k***. Nếu là các toán tử logic thì sẽ tiến hành tính toán logic theo hai giá trị logic cuối và kế cuối của các biến chữ cái trong ***Postfix*** là ***Truthtable[-1]*** và ***Truthtable[-2]***. Khi tính xong, ngoài trường hợp phép phủ định thì các trường hợp còn lại giá trị mới sẽ được lưu lại vào ***Truthtable[-2]*** và cho vào list ***TempTable*** sau đó tiến hành xóa bỏ đi vị trí cuối cũ để không gây ra các tính toán sai lệch ở những trường hợp khác. Đoạn code như sau:

```
for j in Postfix:
   if j.isalpha():
      Truthtable.append(k[j])
   else:
      if  j == '~':
         Truthtable[-1] = not Truthtable[-1]
         TempTable.append(Truthtable[-1])
```

```
elif j == '&':
    Truthtable[-2] = Truthtable[-2] & Truthtable[-1]
    TempTable.append(Truthtable[-2])
    del Truthtable[-1]
elif j == '|':
    Truthtable[-2] = Truthtable[-2] | Truthtable[-1]
    TempTable.append(Truthtable[-2])
    del Truthtable[-1]
elif j == '>':
    Truthtable[-2] = Truthtable[-2] <= Truthtable[-1]
    TempTable.append(Truthtable[-2])
    del Truthtable[-1]
elif j == '=':
    Truthtable[-2] = Equal(Truthtable[-2],Truthtable[-1])
    TempTable.append(Truthtable[-2])
    del Truthtable[-1]
```

Kết thúc vòng lặp ***for j in Postfix***, lúc này list *TempTable* sẽ lưu giữ các giá trị logic True hoặc False tương ứng với số cột một trường hợp của bảng chân trị. Sau đó ta sẽ đổi list ***TempTable*** thành dạng ***tuple*** để hàm ***writeTruthtable(table)*** xem nó là một chuỗi để khi xuất ra file được một trường hợp thì nó sẽ xuống dòng. Tiếp đến lưu các trường hợp của list TempTable vào list DataAll. Vì list ***TempTable*** sau mỗi bước lặp đều tái khởi tạo để lưu từng trường hợp nên sau mỗi bước lặp ta lưu từng trường hợp của bảng chân trị vào list ***DataAll***. Cuối cùng, ta được một bảng chân trị gói gọn trong list ***DataAll*** thông qua dòng code:

```
DataAll.append(tuple(TempTable))
```

Phần chạy code tay chi tiết cho hàm Postfix2Truthtable(Postfix):

Kiểm tra phần tử là chữ cái alphabet thì cho vào ***inp***

Bước	Phần tử Postfix	Điều kiện	inp
1	R	True	[‘R’]
2	|	False	[‘R’]
3	P	True	[‘R’, ‘P’]
4	&	False	[‘R’, ‘P’]
5	Q	True	[‘R’, ‘P’, ‘Q’]

Bảng 3.2 Chạy tay theo code Testcase dòng 1 – List inp

Tiếp đến sắp xếp list trên thành [‘P’, ‘Q’, ‘R’]. Ta có 2^3 số trường hợp.

	Bước	Vòng lặp i	Vòng lặp m	Vòng lặp j
TH 1	1	→000 →[False, False, False] →{‘P’: False, ‘Q’: False, ‘R’: False}	TempTable = [False]	Truthtable = [False]
	2		TempTable = [False, False]	Truthtable = [False, False]
	3		TempTable = [False, False, False]	Truthtable = [False, False, False]
	4			TempTable = [False, False, False, False]
	5			TempTable = [False, False, False, False, False]

Bảng 3.3 Chạy tay theo code Testcase dòng 1 – Các vòng lặp trong TH1

	Bước	Vòng lặp i	Vòng lặp m	Vòng lặp j
TH 2	1	→001 →[False, False, True] →{'P': False, 'Q': False, 'R': True}	TempTable = [False]	Truthtable = [True]
	2		TempTable = [False, False]	Truthtable = [True, False]
	3		TempTable = [False, False, True]	Truthtable = [True, False, False]
	4			TempTable = [False, False, True, False]
	5			TempTable = [False, False, True, False, True]

Bảng 3.4 Chạy tay theo code Testcase dòng 1 – Các vòng lặp trong TH2

	Bước	Vòng lặp i	Vòng lặp m	Vòng lặp j
TH 3	1	→010 →[False, True, False] →{'P': False, 'Q': True, 'R': False}	TempTable = [False]	Truthtable = [False]
	2		TempTable = [False, True]	Truthtable = [False, False]

	3		TempTable = [False, True, False]	Truthtable = [False, False, True]
	4			TempTable = [False, True, False, False]
	5			TempTable = [False, True, False, False, False]

Bảng 3.5 Chạy tay theo code Testcase dòng 1 – Các vòng lặp trong TH3

	Bước	**Vòng lặp i**	**Vòng lặp m**	**Vòng lặp j**
TH 4	1	→011 →[False, True, True] →{'P': False, 'Q': True, 'R': True}	TempTable = [False]	Truthtable = [True]
	2		TempTable = [False, True]	Truthtable = [True, False]
	3		TempTable = [False, True, True]	Truthtable = [True, False, True]
	4			TempTable = [False, True, True, False]
	5			TempTable = [False, True, True, False, True]

Bảng 3.6 Chạy tay theo code Testcase dòng 1 – Các vòng lặp trong TH4

	Bước	Vòng lặp i	Vòng lặp m	Vòng lặp j
TH 5	1	→100 →[True, False, False] →{'P': True, 'Q': False, 'R': False}	TempTable = [True]	Truthtable = [False]
	2		TempTable = [True, False]	Truthtable = [False, True]
	3		TempTable = [True, False, False]	Truthtable = [False, True, False]
	4			TempTable = [True, False, False, False]
	5			TempTable = [True, False, False, False, False]

Bảng 3.7 Chạy tay theo code Testcase dòng 1 – Các vòng lặp trong TH5

	Bước	Vòng lặp i	Vòng lặp m	Vòng lặp j
TH 6	1	→101 →[True, False, True] →{'P': True, 'Q': False, 'R': True}	TempTable = [True]	Truthtable = [True]
	2		TempTable = [True,	Truthtable = [True,

			False]	True]
	3		TempTable = [True, False, True]	Truthtable = [True, True, False]
	4			TempTable = [True, False, True, False]
	5			TempTable = [True, False, True, False, True]

Bảng 3.8 Chạy tay theo code Testcase dòng 1 – Các vòng lặp trong TH6

	Bước	Vòng lặp i	Vòng lặp m	Vòng lặp j
TH 7	1	→110 →[True, True, False] →{'P': True, 'Q': True, 'R': False}	TempTable = [True]	Truthtable = [False]
	2		TempTable = [True, True]	Truthtable = [False, True]
	3		Temptable = [True, True, False]	Truthtable = [False, True, True]
	4			TempTable = [True, True, False, True]
	5			TempTable = [True, True, False, True, True]

Bảng 3.9 Chạy tay theo code Testcase dòng 1 – Các vòng lặp trong TH7

	Bước	Vòng lặp i	Vòng lặp m	Vòng lặp j
TH 8	1	→111 →[True, True, True] →{'P': True, 'Q': True, 'R': True}	TempTable = [True]	Truthtable = [True]
	2		TempTable = [True, True]	Truthtable = [True, True]
	3		TempTable = [True, True, True]	Truthtable = [True, True, True]
	4			TempTable = [True, True, True, True]
	5			TempTable = [True, True, True, True, True]

Bảng 3.10 Chạy tay theo code Testcase dòng 1 – Các vòng lặp trong TH8

TH 1: TempTable = [False, False, False, False, False]

TH 2: TempTable = [False, False, True, False, True]

TH 3: TempTable = [False, True, False, False, False]

TH 4: TempTable = [False, True, True, False, True]

TH 5: TempTable = [True, False, False, False, False]

TH 6: TempTable = [True, False, True, False, True]

TH 7: TempTable = [True, True, False, True, True]

TH 8: TempTable = [True, True, True, True, True]

Cuối cùng mọi ***TempTable*** được thêm vào ***DataAll*** dưới dạng tuple tạo thành bảng chân trị hoàn chỉnh.

• Đầu ra:

DataAll = [(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)]

3.3 Chạy tay theo code cho Testcase dòng 2

-Trong hàm **Infix2Postfix(Infix):**

• Đầu vào: ~P|(Q&R)>R

Ý nghĩa các bước chạy tương tự như đã nêu ở phần **chạy code tay Testcase dòng 1.**

Dùng ***Infix = " ".join(Infix)*** để nối cái phần tử khoảng trắng vào ***Infix***.

=> Kết quả: ~ P | (Q & R) > R

Ký tự từ chuỗi infix	Bước thực hiện	Stack hiện tại	Ghi chuỗi postfix
~	Dấu – Thêm vào Stack	~	
P	Alphabet – Ghi vào chuỗi	~	P
|	Dấu – Thêm vào Stack. Xét độ ưu tiên ("~" cao hơn "|" nên ghi "~" vào chuỗi)	|	P ~
(	Dấu "(" – Thêm vào Stack.	| (	P ~

Q	Alphabet – Ghi vào chuỗi	| (	P ~ Q
&	Dấu – Thêm vào Stack.	| (&	P ~ Q
R	Alphabet – Ghi vào chuỗi	| (&	P ~ Q R
)	Dấu “)” – Đưa hết phần tử trong Stack ghi vào chuỗi đến khi gặp dấu (và xóa dấu (ra khỏi Stack	|	P ~ Q R &
>	Alphabet – Ghi vào chuỗi. Xét độ ưu tiên. (“|” cao hơn “>” nên ghi “|” vào chuỗi)	>	P ~ Q R & |
R	Alphabet – Ghi vào chuỗi	>	P ~ Q R & | R
Không còn kí tự	Ghi tất cả những dấu còn lại trong Stack vào chuỗi		P ~ Q R & | R >

Bảng 3.11 Chạy tay theo code chi tiết Testcase dòng 2 - Hàm Infix2Postfix(Infix)

- **Đầu ra: ['P', '~', 'Q', 'R', '&', '|', 'R', '>']**

-Trong hàm **Postfix2Truthtable(Postfix):**

• Đầu vào: ['P', '~', 'Q', 'R', '&', '|', 'R', '>']

Ý nghĩa các bước chạy tương tự như đã nêu ở phần **chạy code tay Testcase dòng 1.**

Bước	Phần tử Postfix	Điều kiện	inp
1	P	True	[‘P’]
2	~	False	[‘P’]
3	Q	True	[‘P’, ‘Q’]
4	R	True	[‘P’, ‘Q’, ‘R’]
5	&	False	[‘P’, ‘Q’, ‘R’]
6	|	False	[‘P’, ‘Q’, ‘R’]
7	R	False	[‘P’, ‘Q’, ‘R’]
8	>	False	[‘P’, ‘Q’, ‘R’]

Bảng 3.12 Chạy tay theo code Testcase dòng 2 – List inp

List inp = [‘P’, ‘Q’, ‘R’]. Ta có 2^3 số trường hợp của bảng chân trị.

	Bước	Vòng lặp i	Vòng lặp m	Vòng lặp j
TH 1	1	→000 →[False, False, False] →{‘P’: False, ‘Q’: False, ‘R’: False}	TempTable = [False]	Truthtable = [False]
	2		TempTable = [False, False]	TempTable = [False, False, False, True]
	3		TempTable = [False, False, False]	Truthtable = [True, False]

	4			Truthtable = [True, False, False]
	5			TempTable = [False, False, False, True, False]
	6			TempTable = [False, False, False, True, False, True]
	7			Truthtable = [True, False]
	8			TempTable = [False, False, False, True, False, True, False]

Bảng 3.13 Chạy tay theo code Testcase dòng 2 – Các vòng lặp trong TH1

	Bước	**Vòng lặp i**	**Vòng lặp m**	**Vòng lặp j**
TH 2	1	→001 →[False, False, True] →{'P': False, 'Q': False, 'R': True}	TempTable = [False]	Truthtable = [False]
	2		TempTable = [False, False]	TempTable = [False, False, True, True]
	3		TempTable = [False, False, True]	Truthtable = [True, False]

	4			Truthtable = [True, False, True]
	5			TempTable = [False, False, True, True, False]
	6			TempTable = [False, False, True, True, False, True]
	7			Truthtable = [True, True]
	8			TempTable = [False, False, True, True, False, True, True]

Bảng 3.14 Chạy tay theo code Testcase dòng 2 – Các vòng lặp trong TH2

	Bước	**Vòng lặp i**	**Vòng lặp m**	**Vòng lặp j**
TH 3	1	→010 →[False, True, False] →{'P': False, 'Q': True, 'R': False}	TempTable = [False]	Truthtable = [False]
	2		TempTable = [False, True]	TempTable = [False, True, False, True]
	3		TempTable = [False, True, False]	Truthtable = [True, True]

	4			Truthtable = [True, True, False]
	5			TempTable = [False, True, False, True, False]
	6			TempTable = [False, True, False, True, False, True]
	7			Truthtable = [True, False]
	8			TempTable = [False, True, False, True, False, True, False]

Bảng 3.15 Chạy tay theo code Testcase dòng 2 – Các vòng lặp trong TH3

	Bước	**Vòng lặp i**	**Vòng lặp m**	**Vòng lặp j**
TH 4	1	→011 →[False, True, True] →{'P': False, 'Q': True, 'R': True}	TempTable = [False]	Truthtable = [False]
	2		TempTable = [False, True]	TempTable = [False, True, True, True]
	3		TempTable = [False, True, True]	Truthtable = [True, True]

	4			Truthtable = [True, True, True]
	5			TempTable = [False, True, True, True, True]
	6			TempTable = [False, True, True, True, True, True]
	7			Truthtable = [True, True]
	8			TempTable = [False, True, True, True, True, True, True]

Bảng 3.16 Chạy tay theo code Testcase dòng 2 – Các vòng lặp trong TH4

	Bước	**Vòng lặp i**	**Vòng lặp m**	**Vòng lặp j**
TH 5	1	→100 →[True, False, False] →{'P': True, 'Q': False, 'R': False}	TempTable = [True]	Truthtable = [True]
	2		TempTable = [True, False]	TempTable = [True, False, False, False]
	3		TempTable = [True, False, False]	Truthtable = [False, False]

	4			Truthtable = [False, False, False]
	5			TempTable = [True, False, False, False, False]
	6			TempTable = [True, False, False, False, False, False]
	7			Truthtable = [False, False]
	8			TempTable = [True, False, False, False, False, False, True]

Bảng 3.17 Chạy tay theo code Testcase dòng 2 – Các vòng lặp trong TH5

	Bước	**Vòng lặp i**	**Vòng lặp m**	**Vòng lặp j**
TH 6	1	→101 →[True, False, True] →{'P': True, 'Q': False, 'R': True}	TempTable = [True]	Truthtable = [True]
	2		TempTable = [True, False]	TempTable = [True, False, True, False]
	3		TempTable = [True, False, True]	Truthtable = [False, False]

	4			Truthtable = [False, False, True]
	5			TempTable = [True, False, True, False, False]
	6			TempTable = [True, False, True, False, False, False]
	7			Truthtable = [False, True]
	8			TempTable = [True, False, True, False, False, False, True]

Bảng 3.18 Chạy tay theo code Testcase dòng 2 – Các vòng lặp trong TH6

	Bước	**Vòng lặp i**	**Vòng lặp m**	**Vòng lặp j**
TH 7	1	→110 →[True, True, False] →{'P': True, 'Q': True, 'R': False}	TempTable = [True]	Truthtable = [True]
	2		TempTable = [True, True]	TempTable = [True, True, False, False]
	3		TempTable = [True, True, False]	Truthtable = [False, True]

	4			Truthtable = [False, True, False]
	5			TempTable = [True, True, False, False, False]
	6			TempTable = [True, True, False, False, False, False]
	7			Truthtable = [False, False]
	8			TempTable = [True, True, False, False, False, False, True]

Bảng 3.19 Chạy tay theo code Testcase dòng 2 – Các vòng lặp trong TH7

	Bước	**Vòng lặp i**	**Vòng lặp m**	**Vòng lặp j**
TH 8	1	→111 →[True, True, True] →{'P': True, 'Q': True, 'R': True}	TempTable = [True]	Truthtable = [True]
	2		TempTable = [True, True]	TempTable = [True, True, True, False]
	3		TempTable = [True, True, True]	Truthtable = [False, True]

	4			Truthtable = [False, True, True]
	5			TempTable = [True, True, True, False, True]
	6			TempTable = [True, True, True, False, True, True]
	7			Truthtable = [True, True]
	8			TempTable = [True, True, True, False, True, True, True]

Bảng 3.20 Chạy tay theo code Testcase dòng 2 – Các vòng lặp trong TH8

TH 1: TempTable = [False, False, False, True, False, True, False]

TH 2: TempTable = [False, False, True, True, False, True, True]

TH 3: TempTable = [False, True, False, True, False, True, False]

TH 4: TempTable = [False, True, True, True, True, True, True]

TH 5: TempTable = [True, False, False, False, False, False, True]

TH 6: TempTable = [True, False, True, False, False, False, True]

TH 7: TempTable = [True, True, False, False, False, False, True]

TH 8: TempTable = [True, True, True, False, True, True, True]

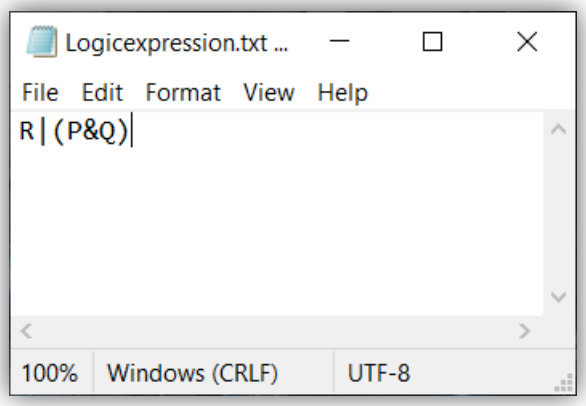
Cuối cùng mọi ***TempTable*** được thêm vào ***DataAll*** dưới dạng tuple tạo thành bảng chân trị hoàn chỉnh.

• Đầu ra:

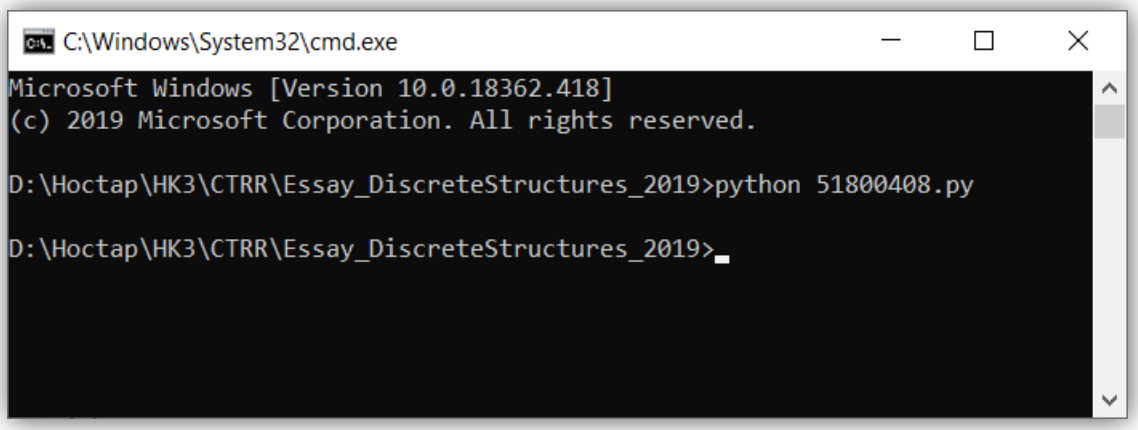
DataAll = [(False, False, False, True, False, True, False), (False, False, True, True, False, True, True), (False, True, False, True, False, True, False), (False, True, True, True, True, True, True), (True, False, False, False, False, False, True), (True, False, True, False, False, False, True), (True, True, False, False, False, False, True), (True, True, True, False, True, True, True)]

CHƯƠNG 4 – KẾT QUẢ

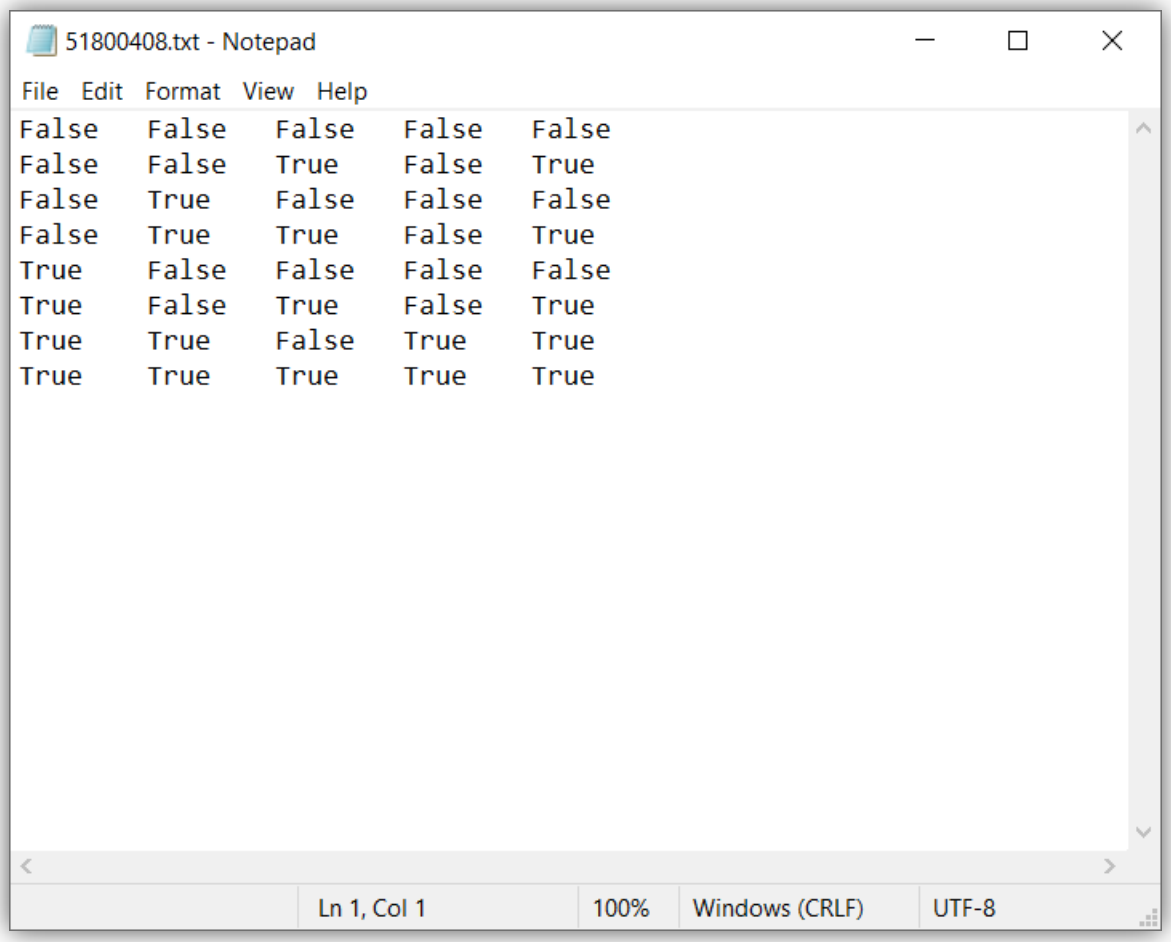
4.1 Testcase 1



Hình 4.1 File input của Testcase 1

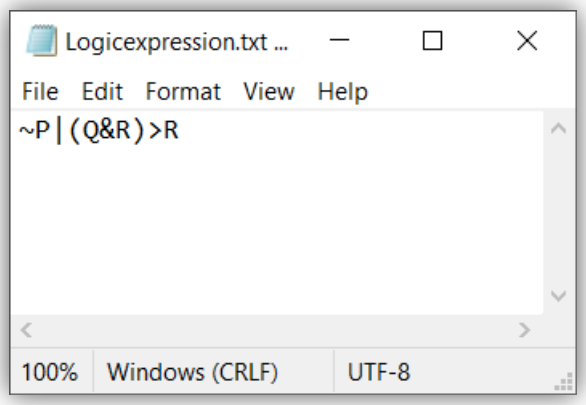


Hình 4.2 Dòng lệnh chạy code Testcase 1

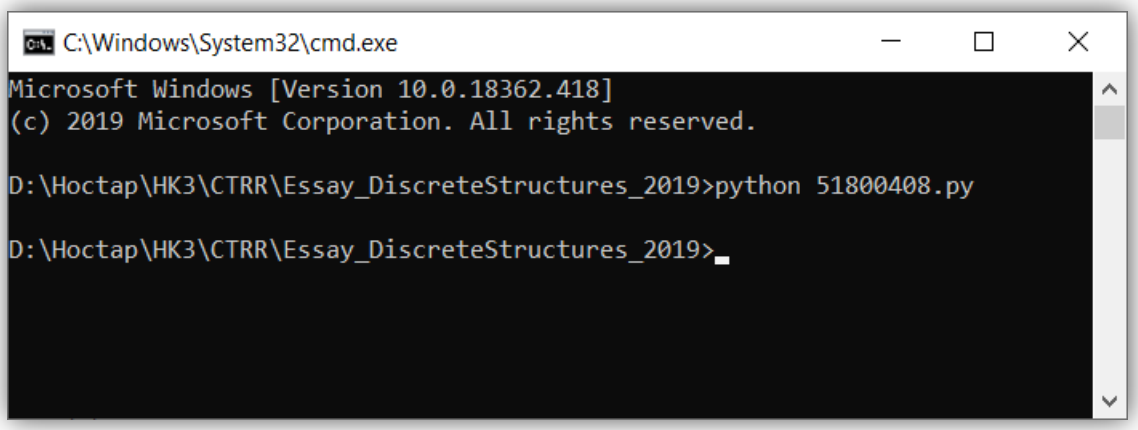


Hình 4.3 Kết quả file output của Testcase 1

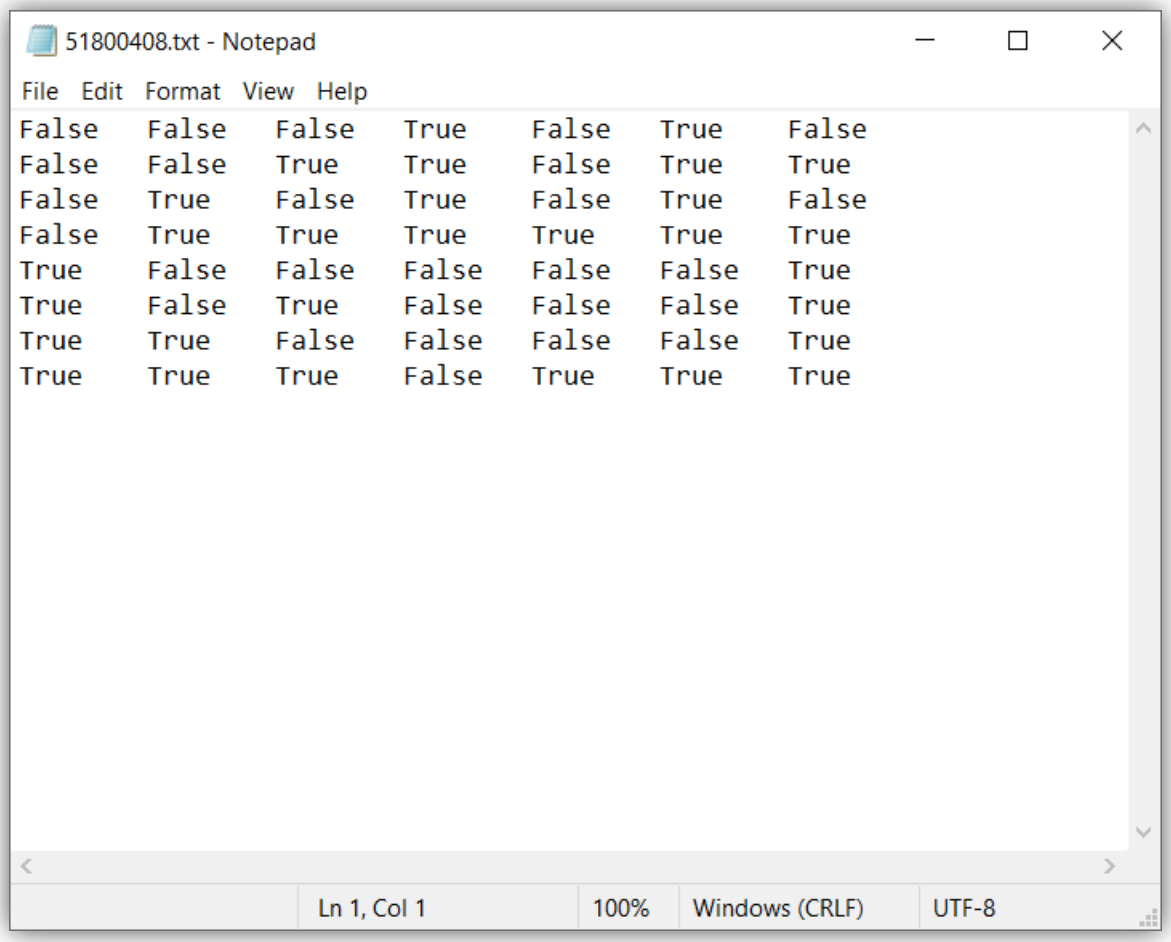
4.2 Testcase 2



Hình 4.4 File input của Testcase 2

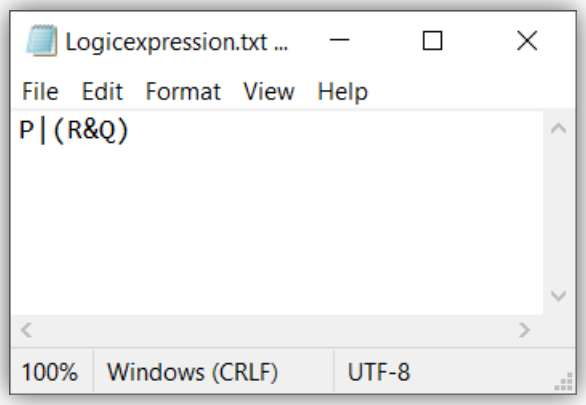


Hình 4.5 Dòng lệnh chạy code Testcase 2

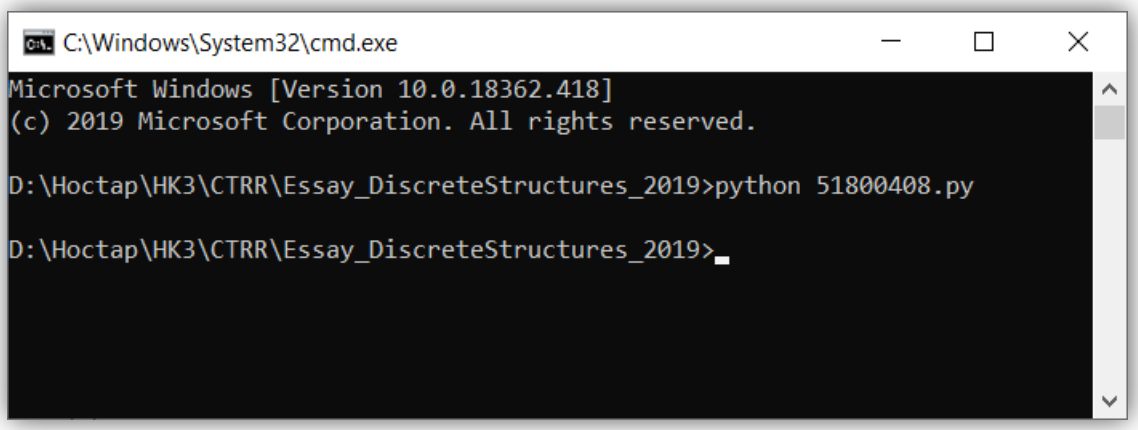


Hình 4.6 Kết quả file output của Testcase 2

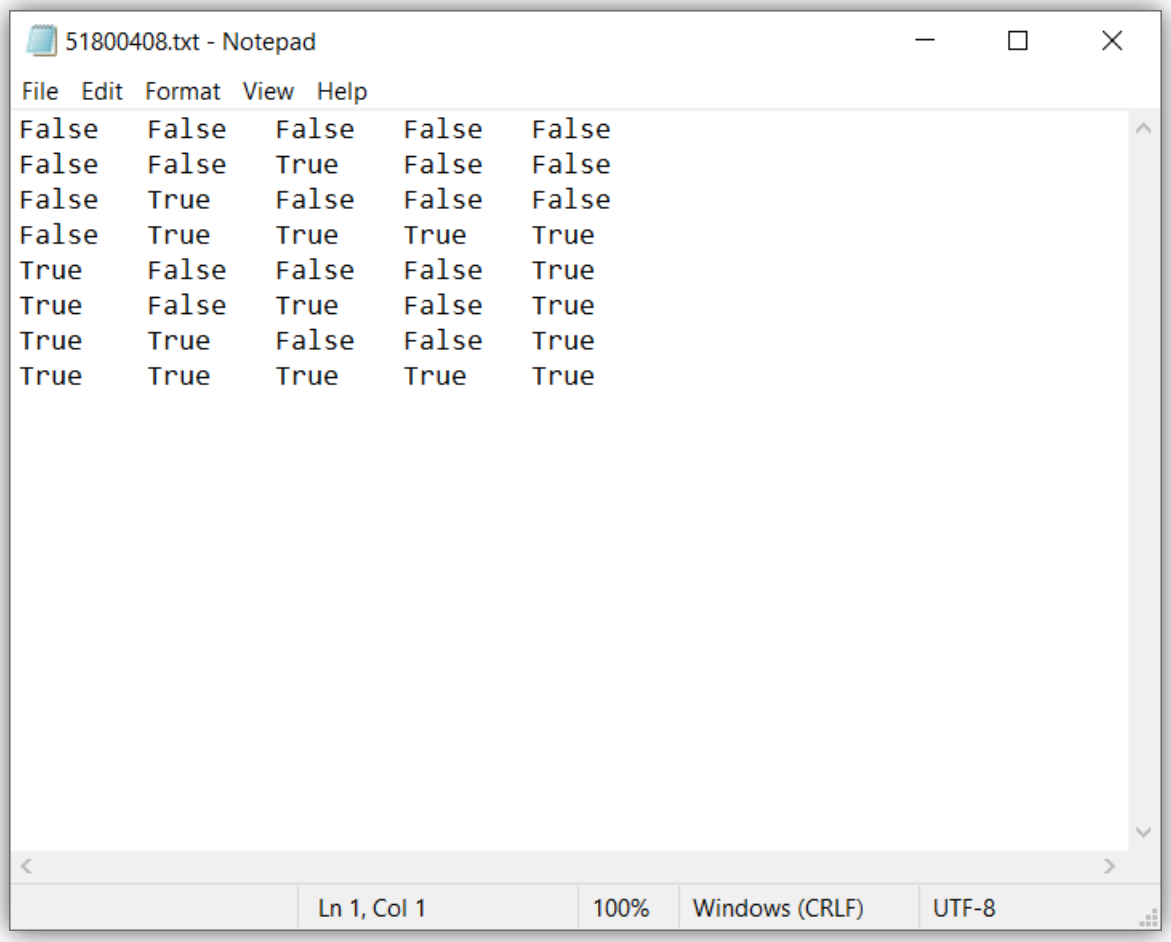
4.3 Testcase 3



Hình 4.7 File input của Testcase 3

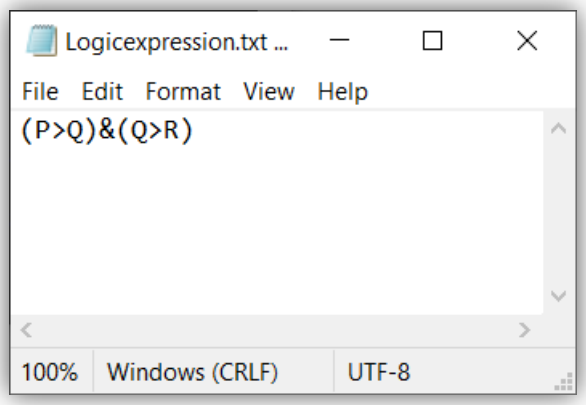


Hình 4.8 Dòng lệnh chạy code Testcase 3

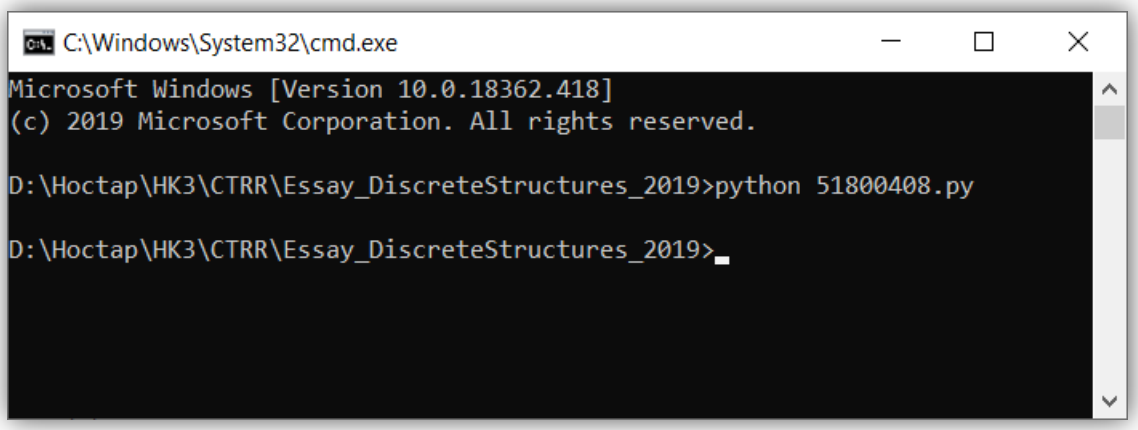


Hình 4.9 Kết quả file output của Testcase 3

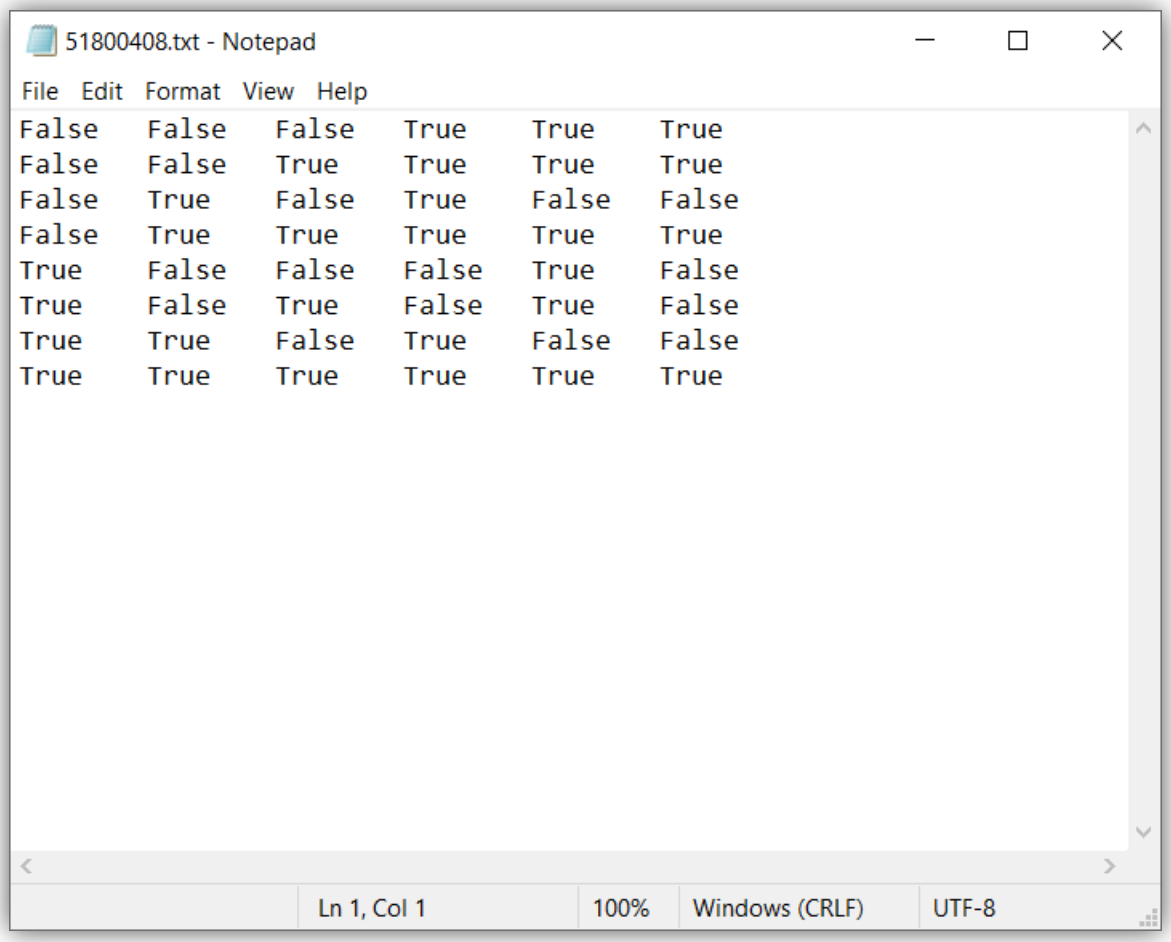
4.4 Testcase 4



Hình 4.10 File input của Testcase 4

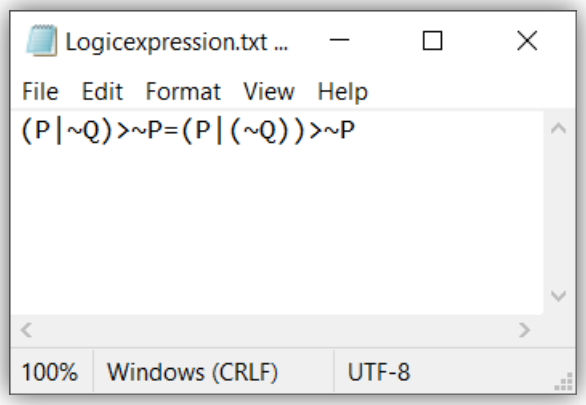


Hình 4.11 Dòng lệnh chạy code Testcase 4

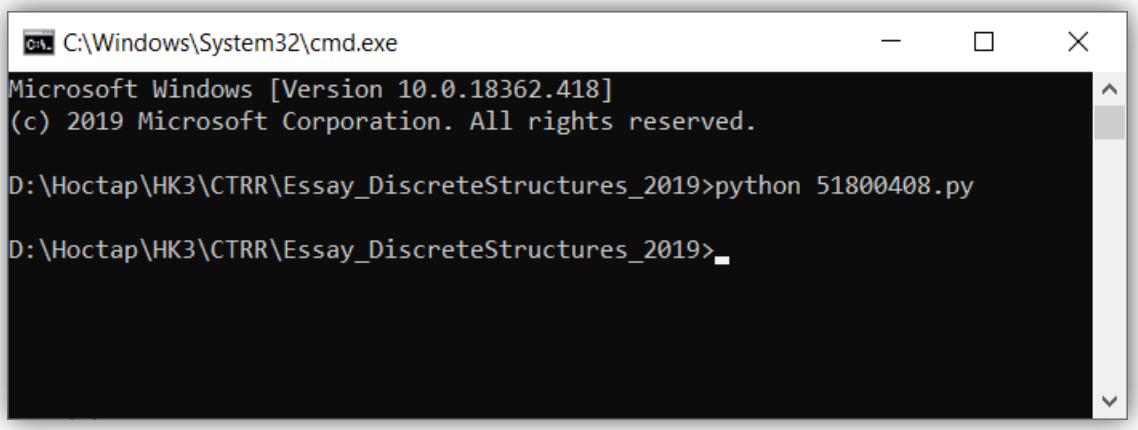


Hình 4.12 Kết quả file output của Testcase 4

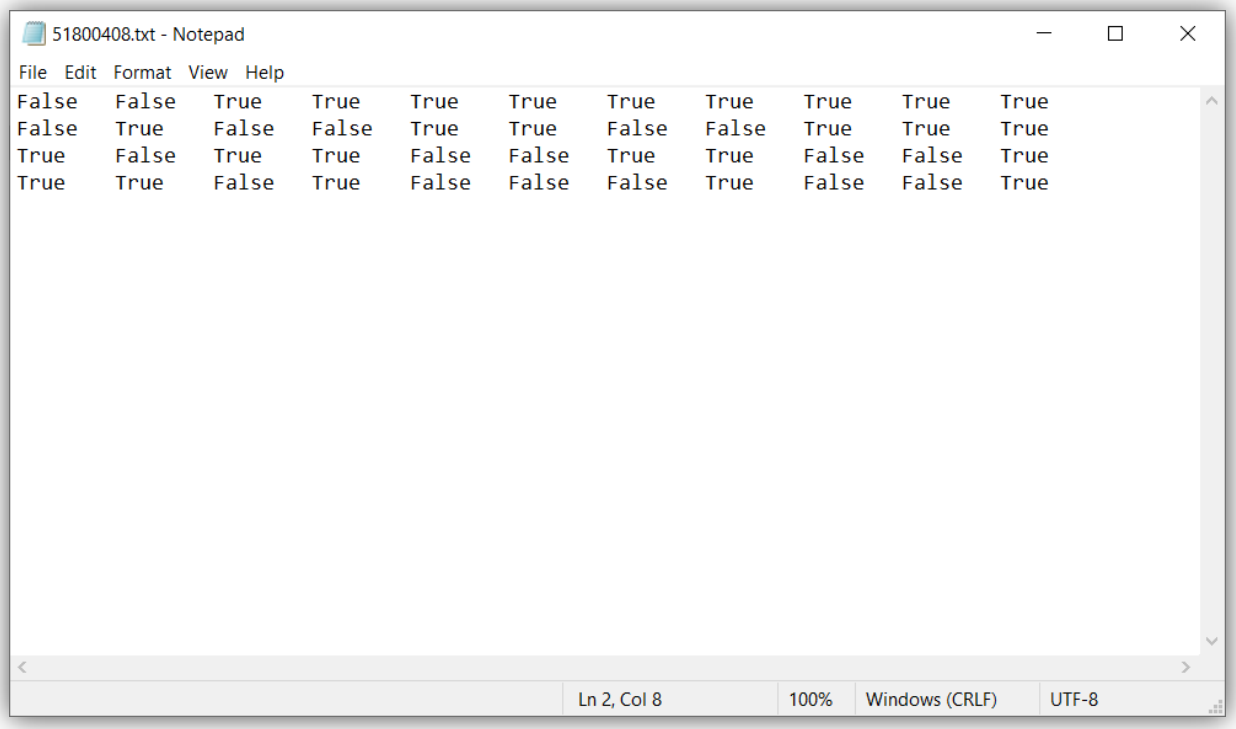
4.5 Testcase 5



Hình 4.13 File input của Testcase 5



Hình 4.14 Dòng lệnh chạy code Testcase 5



Hình 4.15 Kết quả file output của Testcase 5

TÀI LIỆU THAM KHẢO

[1] [08/10/2019 08:52] https://vi.wikipedia.org/wiki/Reverse_Polish_notation

[2] [08/10/2019 10:39] https://vi.wikipedia.org/wiki/Logic

[3] [08/10/2019 13:17] https://realpython.com/how-to-implement-python-stack/

[4] [08/10/2019 13:34] https://www.geeksforgeeks.org/using-list-stack-queues-python/

[5] [08/10/2019 13:58] https://www.geeksforgeeks.org/join-function-python/

[6] [09/10/2019 07:56] https://www.geeksforgeeks.org/python-string-isalpha-application/

[7] [09/10/2019 09:01] Data Structures and Algorithms 1 – Assignment_DSA1.pdf

[8] [10/10/2019 22:09] https://realpython.com/python-lambda/

[9] [10/10/2019 22:13] https://www.geeksforgeeks.org/zip-in-python/

[10] [10/10/2019 22:20] https://www.programiz.com/python-programming/methods/built-in/dict

[11] [10/10/2019 22:17] https://www.geeksforgeeks.org/string-rjust-ljust-python/

[12] [12/10/2019 20:29] https://www.geeksforgeeks.org/sorted-function-python/

[17] [12/10/2019 21:56] https://www.programiz.com/python-programming/methods/built-in/map

[18] [12/10/2019 21:59]: https://www.programiz.com/python-programming/methods/built-in/bin

[13] [13/10/2019 00:09] https://docs.python.org/2/tutorial/introduction.html?fbclid=IwAR35uMSsNmHSuLLEWrKXmLbfa6RmYnABS1NXC1mzUs9m5MDmD9Fo6Q2_vfk

[14] [20/10/2019 20:14] https://www.tutorialspoint.com/python/python_tuples.htm

[15] [26/10/2019 17:05] https://viblo.asia/p/reserve-polish-notation-924lJrG8lPM

[16] [26/10/2019 17:09] https://stackoverflow.com/questions/432842/how-do-you-get-the-logical-xor-of-two-variables-in-python