

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN CÁ NHÂN 2:
LOGIC

Họ và tên: Nguyễn Nhật Đăng

MSSV: 20120050

Môn học: Cơ Sở Trí Tuệ Nhân Tạo

Thành phố Hồ Chí Minh – 2023

Contents

1. Ý tưởng thuật toán.....	3
2. Kịch bản kiểm thử.....	8
3. Đánh giá thuật toán:	14

1. Ý tưởng thuật toán

```
for i in range(1,8):
    case = "Test" + str(i)
    in_path = os.path.join(dirname, 'Testcase', case, 'input.txt')
    out_path = os.path.join(dirname, 'Testcase', case, 'output.txt')
    with open(in_path, 'r+') as fin:
        alpha = fin.readline().strip()
        n = int(fin.readline())
        kb = []
        for i in range(n):
            kb.append(fin.readline().strip())
    steps=[]
    entail = pl_resolution(kb, alpha, steps)
```

Sau khi đã đọc alpha và kb từ file “input.txt”, ta sẽ thực hiện việc gọi hàm `pl_resolution` để giải bài toán, các biến truyền vào bao gồm: kb và alpha đã đọc từ file, mảng steps dùng để lưu các mệnh đề được tạo ra sau các vòng lặp.

Trước khi nói về hàm `pl_resolution`, ta sẽ đi qua một số hàm hỗ trợ sau:

- Hàm `delete_And`: sử dụng để xóa bỏ quan hệ AND trong các mệnh đề có sẵn, dựa theo nguyên tắc $A \text{ AND } B \Leftrightarrow A, B$ (Vì theo như lý thuyết, trong một mệnh đề theo chuẩn CNF vẫn có thể tồn tại phép AND)

```
def delete_AND(clause: str)->list:
    """
    xóa And trong các vế, vd từ (A OR B) AND (C OR D) sẽ phân tích thành [A OR B, C OR D]
    """
    clause_new = clause.split(' AND ')
    for i in range(len(clause_new)):
        if '(' in clause or ')' in clause_new[i]:
            clause_new[i] = clause_new[i].replace("(", "")
            clause_new[i] = clause_new[i].replace(")", "")
    return clause_new
```

- Hàm `split_clause`: Sau khi đã xóa bỏ phép AND, trong mệnh đề chỉ còn lại các literal được nối với nhau bởi phép OR, hàm `split_clause` sẽ phân tích các câu mệnh đề dạng string này thành một mảng gồm các literal để thuận tiện hơn cho việc tính toán hợp giải sau này:

```
def split_clause(clause: str) -> list:
    """
    Phân tách các clause (chỉ gồm literal và phép OR) thành các một list gồm các literal
    """
    literals = clause.split(' OR ')
    return literals
```

- Hàm Not và is_opposite: hàm Not nhận vào 1 literal và trả về phủ định của literal đó, Hàm is_opposite nhận vào 2 literal và trả về kết quả xem 2 literal đó có đối nghịch nhau không.

```
def Not(literal: str):
    if '-' in literal:
        return literal[1:]
    else:
        return '-' + literal

def is_opposite(literal1: str, literal2: str):
    """
    Kiểm tra xem liệu 2 literal truyền vào có đối nghịch nhau không
    """
    return Not(literal1) == literal2
```

- Hàm always_true: sau khi hợp giải 2 mệnh đề và sinh ra một mệnh đề mới, sẽ tồn tại khả năng để mệnh đề này có dạng "... OR A OR -A OR ...", lúc này mệnh đề này là luôn đúng và ta có thể bỏ qua nó.

```
def always_true(clause: list):
    """
    Kiểm tra các clause xem chúng có luôn đúng không, vd: A OR B OR -A sẽ luôn đúng
    """
    for i in range(len(clause) - 1):
        for j in range(i + 1, len(clause)):
            if is_opposite(clause[i], clause[j]):
                return True
    return False
```

- Hàm sort_clause: Sau khi các mệnh đề đã qua hàm split_clause ở trên và phân thành mảng các literal, hàm sort_clause sẽ được sử dụng để sắp xếp các literal theo thứ tự bảng chữ cái.

```
def sort_clause(clause: list)->list:
    """
    Sort các literal (sau khi đã qua hàm split clause) theo thứ tự bảng chữ cái, bỏ qua dấu
    """
    return sorted(list(set(clause)), key=lambda x:x[-1])
```

Hàm pl_resolution: sau khi truyền vào kb và alpha, ta sẽ lần lượt phân tích chúng bằng cách sử dụng 2 hàm delete_AND và split_clause. Đặc biệt đối với alpha, ta sẽ thêm việc là phủ định lại toàn bộ các literal với hàm Not();

```
#phân tách các vế của KB đã cho thành các mảng gồm toàn literal
kb_new=[]
for clause in kb:
    for clause_new in delete_AND(clause):
        kb_new.append(split_clause(clause_new))
#phân tách các vế của alpha thành các literal và phủ định lại chúng
alpha_new = []
for clause in delete_AND(alpha):
    # alpha_new.append(Not(literal) for literal in phan_tach(clause))
    for literal in split_clause(clause):
        alpha_new.append([ ])
        alpha_new[-1].append((Not(literal)))
```

Sau đó, ta thêm các phủ định của alpha này vào kb:

```
for literal in alpha_new:
    if literal not in kb_new:
        kb_new.append(literal)
```

Bước vào phần hợp giải, ta thực hiện liên tục các vòng lặp để phân tích, ở mỗi vòng lặp, ta sẽ kiểm tra từng cặp mệnh đề i và j đang có trong kb, và thực hiện ghép chúng với nhau thông qua hàm pl_resolve:

```
for i in range(len(kb_new) - 1):
    for j in range(i + 1, len(kb_new)):
        pl_res = pl_resolve(kb_new[i], kb_new[j])
```

```
def pl_resolve(clause1: list, clause2: list)->list:
    """
    Hợp giải 2 clause để trả về danh sách những mệnh đề có thể được phát sinh ra.
    """
    clauses = []
    for i in range(len(clause1)):
        for j in range(len(clause2)):
            if is_opposite(clause1[i], clause2[j]):
                clause1_new = clause1.copy()
                clause1_new.pop(i)
                clause2_new = clause2.copy()
                clause2_new.pop(j)
                clause = clause2_new + clause1_new
                clauses.append(sort_clause(clause))
    return clauses
```

Ở hàm pl_resolve, khi ở 2 mệnh đề (đã được phân tích thành mảng gồm các literal) tồn tại 1 cặp literal đối nghịch nhau, ta sẽ áp dụng luật hợp giải để tạo ra 1 mệnh đề mới. Vì một cặp mệnh đề có thể có nhiều cặp literal đối nhau, nên ta sẽ thử áp dụng với từng cặp và lưu lại những mệnh đề được sinh ra vào một mảng.

Với mỗi mệnh đề mới sinh ra từ 2 cặp mệnh đề i và j cho trước, sẽ có các trường hợp sau xảy ra:

- Mệnh đề luôn đúng: ta trực tiếp bỏ qua.
- Mệnh đề không phải luôn đúng: Nếu những mệnh đề này chưa tồn tại trong kb và chưa được tạo ra từ cặp i', j' nào khác trong vòng lặp này, ta sẽ thêm nó vào mảng steps[-1] – đây là mảng lưu những mệnh đề mới được tạo ra trong vòng lặp hiện tại – và đồng thời đánh dấu rằng ở vòng lặp này đã tạo ra ít nhất 1 mệnh đề mới.
- Đặc biệt nếu mệnh đề này là mệnh đề rỗng, ta sẽ đánh dấu để đánh giá lại ở cuối vòng lặp.

```
for clause in pl_res:
    #Nếu mệnh đề mới hợp giải được là luôn đúng thì ta bỏ qua nó
    if always_true(clause): continue
    #Nếu mệnh đề này chưa có trong kb và chưa được tìm thấy trong vòng lặp này thì thêm nó vào danh sách.
    if clause not in kb_new and clause not in steps[-1]:
        #is_new_clause sẽ dùng để thể hiện xem trong vòng lặp này có mệnh đề nào mới được tạo ra không.
        is_new_clause = True
        steps[-1].append(clause)
    #Nếu mệnh đề mới này là mệnh đề rỗng, thì ta có thể kết luận được luôn kết quả của bài toán.
    if clause==[]: entail = True
```

Sau khi thực hiện xong việc kiểm tra từng cặp, ta sẽ đánh giá ở cuối vòng lặp:

- Nếu không tạo ra được mệnh đề nào mới, ta trả về kết quả False.
- Nếu tạo ra được ít nhất 1 mệnh đề, ta sẽ thêm những mệnh đề mới này vào kb.
- Nếu trong những mệnh đề mới có mệnh đề rỗng, ta trả kết quả hàm về True.

Cuối cùng, nếu đã thêm mệnh đề mới vào kb và không có mệnh đề nào rỗng, ta sẽ đi tiếp tới vòng lặp sau.

Khi hàm `pl_resolution` chạy xong, ta sẽ có được kết quả xem liệu K có entail alpha không, và đồng thời đó là những mệnh đề được sinh ra trong các vòng lặp được lưu trong mảng `steps`. Việc còn lại chỉ là in ra theo yêu cầu của đề bài, ta sẽ sử dụng hàm `print_step` (để in các bước) và `to_CNF` (từ mảng các literal tạo ra một mệnh đề theo dạng CNF) để in ra kết quả vào file “output.txt”

```
def to_CNF(clause: list) -> str:
    """
    In clause theo chuẩn CNF
    """
    res = ""
    if len(clause) == 0:
        res = "{}"
    else:
        res = clause[0]
        for i in range(1, len(clause)):
            res = res + ' OR ' + clause[i]
    return res

def print_steps(f, steps: list):
    """
    In kết quả của các bước được lưu trong mảng steps
    """
    for step in steps:
        f.write(str(len(step)) + '\n')
        for clause in step:
            f.write(to_CNF(clause) + '\n')
```

2. Kịch bản kiểm thử

Dưới đây sẽ là 5 kịch bản kiểm thử bao gồm 2 file “input.txt” và “output.txt”

Bộ test 1: đây là một bộ test cơ bản mà đề bài cung cấp

- Input:

Input.txt	input.txt - Notepad
-A	-A
4	4
-A OR B	-A OR B
B OR -C	B OR -C
A OR -B OR C	A OR -B OR C
-B	-B

- Output:

Output.txt	output.txt - Notepad
3	3
-A	-A
B	B
-C	-C
4	4
-B OR C	-B OR C
A OR C	A OR C
A OR -B	A OR -B
{}	{}
YES	YES

Dễ thấy, kết quả đúng như kết quả cho sẵn ở đề bài

Bộ test 2: Bộ test được nâng cao hơn bộ test 1, với alpha vẫn giữ nguyên chỉ bao gồm một literal, nhưng số mệnh đề ở kb lại tăng lên.

- Input:



input.txt - Notepad

File Edit Format View Help

E

7

A OR B OR C

B OR C

C OR -D

A OR D

A OR C OR E

B OR -D OR -E

B OR -C OR E

- Output:


 output.txt - Notepad

File Edit Format View Help

8
A OR B OR E
B OR E
A OR C
B OR -D OR E
A OR B OR -E
A OR B OR C OR -D
B OR -C OR -D
B OR -C
6
A OR B OR -D
A OR B
B OR -D
B
A OR B OR -C
A OR B OR -D OR E
Ø
NO

Bộ test 3: tăng số lượng literal ở alpha


- Input:

 input.txt - Notepad

File Edit Format View Help


X OR -Y
5
T OR U OR V OR X OR Y
-T
-V
-U OR X
V OR -Y|

- Output:


 output.txt - Notepad
File Edit Format View Help
8
U OR V OR X OR Y
T OR U OR X OR Y
T OR V OR X OR Y
T OR U OR V OR X
T OR U OR V OR Y
-Y
-U
V
11
U OR X OR Y
V OR X OR Y
U OR V OR X
U OR V OR Y
T OR X OR Y
T OR U OR X
T OR U OR Y
{
T OR V OR X
T OR U OR V
T OR V OR Y
YES

Bộ test 4: Một bộ test có độ phức tạp cao với alpha gồm nhiều literal và kb với số lượng mệnh đề khá lớn

- Input:

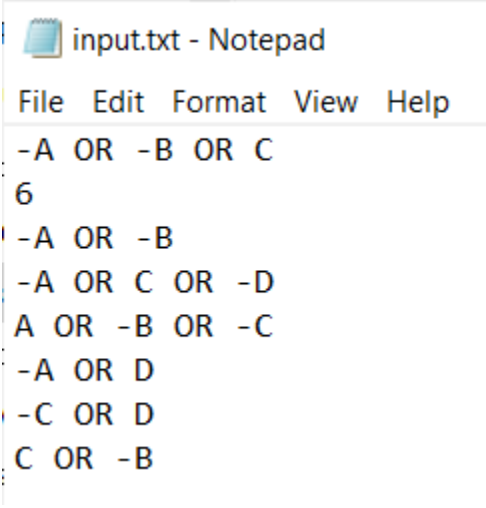
 input.txt - Notepad
File Edit Format View Help
A OR -B OR -C OR D
7
A OR -C
B OR -D
-C OR D
A OR -C OR D
C OR -D
A OR -B
-B OR -C

- Output:

 output.txt - Notepad
File Edit Format View Help
10
A OR -D
-C
A
B OR -C
A OR B OR -C
-C OR -D
D
A OR D
-B OR -D
-B
6
A OR B
A OR -B OR -C
A OR B OR -D
A OR C
{ }
A OR -C OR -D
YES

Bộ test 5: tương tự như bộ test 4

- Input:



```
input.txt - Notepad
File Edit Format View Help
-A OR -B OR C
6
-A OR -B
-A OR C OR -D
A OR -B OR -C
-A OR D
-C OR D
C OR -B
```

- Output:



output.txt - Notepad

File Edit Format View Help

12

-B OR -C

-B

-A

-A OR C

C OR -D

-A OR -D

-B OR -C OR D

A OR -B

A OR -C

D

-B OR D

C

14

-A OR -B OR -D

-B OR C OR -D

-A OR -B OR C

A OR -B OR -D

-B OR -C OR -D

-A OR -C

{ }

-D

-B OR -D

-A OR -B OR D

-B OR C

A OR -D

-A OR -B OR -C

-C OR -D

YES|

3. Đánh giá thuật toán:

- Ưu điểm: đây là một thuật toán dễ hiểu và không quá phức tạp khi cài đặt, các mệnh đề được phát sinh đầy đủ vì chương trình kiểm tra từng cặp mệnh đề trong kb với nhau.
- Nhược điểm: vì phải chạy để kiểm tra với từng cặp mệnh đề trong kb nên độ phức tạp về thời gian là $O(n^2)$, sẽ mất rất lâu để hoàn thành khi n lớn. Đồng thời, do 1 số hạn chế về mặt kỹ thuật trong khâu cài đặt mà ở pl_solution đã không loại bỏ những mệnh đề đã được dùng để hợp giải, dẫn tới ở mỗi vòng lặp, số mệnh đề mới được tạo ra trùng với mệnh đề đã tạo ở những vòng lặp trước sẽ rất nhiều. Để giải quyết, như đã nói thì ta có thể loại bỏ các cặp mệnh đề đã được dùng để hợp giải, ở những vòng lặp sau chỉ còn những mệnh đề chưa được sử dụng hoặc mệnh đề mới, từ đó giảm lượng trùng lặp và giảm số lượng cặp phải xét.