TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO HỢP GIẢI LOGIC MỆNH ĐỀ

MÔN: CƠ SỞ TRÍ TUỆ NHÂN TẠO

Giảng viên: Nguyễn Khánh Duy

Sinh viên thực hiện: Lê Nguyên Khang

MSSV: 20120113

Lớp: 20_21

I. Tiêu chí đánh giá :

STT	Đặc tả tiêu chí	Điểm	Hoàn thành
1	Đọc dữ liệu đầu vào và lưu trong cấu trúc dữ liệu phù hợp	0.5	100%
2	Cài đặt giải thuật hợp giải trên logic mệnh đề	1	100%
3	Các bước suy diễn phát sinh đủ mệnh đề và kết luận đúng	2.5	100%
4	Tuân thủ mô tả định dạng của đề bài	0.5	100%
5	Báo cáo test case và đánh giá	0.5	100%

II. Giải thích code:

1. ***Các hàm phụ trợ:***

- Khởi tạo list used[]: như là một database lưu lại tất cả các mệnh đề đã được tạo từ đó tránh tạo lập lại.
- def split_String(s): Hàm xử lý nhiễu các string do input đầu vào, loại bỏ kí tự xuống dòng và các khoảng trắng thừa để dễ xử lý.

```
def split_String(s):# Xử lý nhiễu trong input
    s_new = s
    s_new = s_new.replace(" ", "")
    s_new = s_new.replace("\n", "")
    return s_new
```

- def readFile(file_name): Hàm đọc file input (input có thể có nhiều khoảng trắng giữ các mệnh đề và từ khóa)

```
def readFile(file_name): # Đọc File
    s = []
    f = open(file_name, 'r')
    alpha = f.readline()
    alpha = split_String(alpha)
    n = f.readline()
    n = split_String(n)
    for i in f:
        ss = i
        ss = split_String(ss)
        s.append(ss)
    return alpha, n, s # trả về alpha, n và list s các mệnh đề
```

- def sort_ascii(list): Hàm sắp xếp chuỗi kí tự các literal lại dù literal có dấu âm phía trước('-')

```python
# Hàm sắp xếp theo kí tự a-z bubble sort, vì chuỗi có phần tử có dấu - ở đầu nên cần viết chương trình xử lý riêng
def sort_ascii(list):
    ls = list.copy()
    n = len(ls)
    for i in range(0, n - 1):
        for j in range(i + 1, n):
            if (len(ls[i]) > 1 ): x = ls[i][1] # TH chuỗi có dấu - (vd : -A) ta so sánh theo chuỗi không có dấu -
            else: x = ls[i]
            if (len(ls[j]) > 1 ): y = ls[j][1] #
            else: y = ls[j]

            if(x > y):
                tmp_p = ls[i]
                ls[i] = ls[j]
                ls[j] = tmp_p
    return ls
```

- def add_On(ans, sub, pos1, pos2): Hàm mục đích bổ trợ cho hàm uinon tạo mệnh đề mới sau khi hợp giải một cách dễ dàng hơn.

```python
#Hàm bổ trợ tạo mệnh đề
def add_On(ans, sub, pos1, pos2):
    for i in sub: # xét từng phần tử mệnh đề sub
        tmp = 0 # biến đánh dấu
        if i != pos1[0][0] and i != pos1[0][1]:
            for j in pos2:
                if (j == i):
                    tmp = 1
                    break
            if (tmp == 0):
                ans.append(i)
    return ans
```

- def union(sub1, sub2): Hàm dùng để hợp giải 2 mệnh đề nếu có thể.
  + Hàm tách các mệnh đề thành list các literal(loại các từ khóa OR) cho dễ xử lý.
  + Sau đó duyệt tìm kiếm các cặp đối ngẫu ở cả hai mệnh đề, đồng thời tìm các literal trùng trong cả hai mệnh đề.
  + Tiến hành hợp giải (gọi add_on hổ trợ): gộp hai mệnh đề lại với nhau, loại bỏ phần tử đối ngẫu, các phần tử trùng chỉ giữ lại 1 ở mệnh đề tạo thành.
  + Return True và mệnh đề mới nếu có thể hợp giải, ngược lại return false, [].

```python
# Check và tạo mệnh đề
def union(sub1, sub2):
    ans = [] # Lưu ans là kq hợp giải
    pos2 = [] # Lưu các phần tử trùng ở cả hai mệnh đề
    pos1 = [] # Lưu những cặp đối ngẫu

    sub1 = sub1.split('OR') # tách ra literal riêng để dễ xử lý như: so sánh, sắp xếp
    sub2 = sub2.split('OR') #
    for i in sub1:
        for j in sub2:
            if len(i) - len(j) != 0: # 2 biến chiều dài khác nhau có khả năng đối ngẫu
                if (len(i) > 1):
                    if i[1] == j[0] : # kiểm tra đối ngẫu
                        pos1.append((i, j))
                else:
                    if i[0] == j[1]: #
                        pos1.append((i, j))
            if i == j: # Phần tử trùng ở cả 2 mệnh đề
                pos2.append(i)

    if (len(pos1) == 1): # TH chỉ có 1 biến đối ngẫu -> tiến hành các bước hợp giải
        ans = add_On(ans, sub1, pos1, pos2)
        ans = add_On(ans, sub2, pos1, pos2)
        for i in pos2:
            ans.append(i)
        return True, ans # trả về True là hợp giải thành công
    else:
        return False, ans # ngược lại, ans = []
```

- def Not_alpha(alpha): là hàm phủ định lại alpha, nếu alpha là một mệnh đề thì tách các từ khóa(OR) ra sau đó phủ định từ literal và cho vào list not_alpha.

```python
#Phủ định alpha
def Not_alpha(alpha):
    alpha = alpha.split('OR') # Nếu alpha là mệnh đề thì tách ra các literal
    not_alpha = []
    for i in alpha :
        if len(i) == 2:
            not_alpha.append(i[1])
        else:
            ss = '-' + i
            not_alpha.append(ss)
    return not_alpha
```

- def initialization(not_alpha, arr_cur): Hàm khởi tạo các đối số đầu vào cho thuật toán robinson dễ xử lý hơn. Gồm thêm vào used[], arr_cur[] những mệnh đề đã tồn tại từ alpha và từ giả thuyết các list s.

```
# Khởi tạo các giá trị ban đầu theo giả thuyết và kết luận
def initialization(not_alpha, arr_cur):
    for i in not_alpha: # Thêm dữ liệu từ phủ định alpha
        arr_cur.append(i)
        if i not in used:
            used.append(i)

    for i in s: # Thêm dữ liệu từ giả thuyết
        arr_cur.append(i)
        ss = ''
        s1 = i
        s1 = s1.split('OR')
        for j in s1:
            ss += j
        if ss not in used:
            used.append(ss)
```

2. Giải thuật robinson

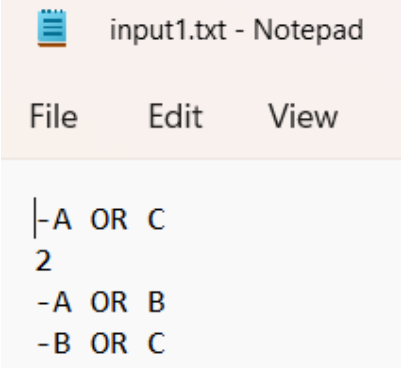
- def PL_Resolution(alpha, n, s): Hàm thực hiện giải thuật hợp giải robinson.
  + arr_cur: lưu các mệnh đề hiện tại đã được tạo, từ list này chọn 2 mệnh đề có thể tạo và tạo mệnh đề mới.
  + add: mảng lưu các mệnh đề mới được tạo thành.
  + Gọi initialization(not_alpha, arr_cur) khởi tại theo giả thuyết và alpha.
  + Tạo một vòng lặp vô tận while và bằng 2 vòng lặp i, j chọn 2 phần tử trong arr_cur tiến hành hợp giải bằng hàm union.
  + Hợp giải thành công, nếu rỗng cho nó bằng '{}' và bật cờ lệnh stopp = 1 ngược lại sort các literal theo a-z .
  + Nếu mệnh đề tạo thành là mới(chưa nằm trong used) thì thêm vào used, add, arr_cur.
  + Sau 2 vòng lặp i, j tiến hành xuất các mệnh đề mới theo định dạng của đề.
  + Nếu stopp = 1 hoặc không tạo được mệnh đề nào mới(len(add) = 0) thì dừng while.

```
def PL_Resolution(alpha, n, s): # Robinson
    arr_cur = [] #arr_cur mảng các mệnh đề hiện tại đã được tạo
    add = [] # mảng lưu các mệnh đề mới được tạo thành từ mỗi vòng lặp
    not_alpha = Not_alpha(alpha)
    initialization(not_alpha, arr_cur)
    file = open("out.txt", mode = 'w', encoding='utf-8-sig')
    loop = 1
    stopp = 0 # đánh dấu xuất hiện mệnh đề rỗng
    while loop == 1 :
        ans = []
        #
        for i in range(len(arr_cur)-1):
            for j in range(i + 1, len(arr_cur)):
                tmp, ans = union(arr_cur[i], arr_cur[j]) # hợp giải 2 mệnh đề
                if (tmp == True): # Hợp giải được
                    if ans == []: # Mệnh đề rỗng
                        ans.append('{}')
                        stopp = 1
                    else:
                        ans = sort_ascii(ans) # Sắp xếp các literal của mệnh đề

                    str1 = '' # chuỗi literal không có từ khóa để dễ lưu
                    for i_ans in ans:
                        str1 += i_ans
                    if str1 not in used:# Nếu mệnh đề chưa tồn tại trước đó
                        used.append(str1)
                        str2 = ''
                        for i_ans in ans:
                            str2 += (i_ans + ' OR ')
                        str2 = str2[0:len(str2)-4]
                        add.append(str2) # thêm vào add

        for i in add:# thêm mệnh đề mới được tạo vào arr_cur
            s_a = split_String(i)
            arr_cur.append(s_a)
```

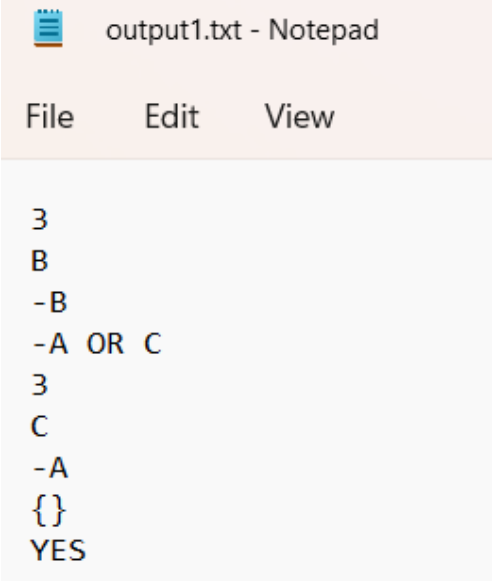
III. Input và output:

1. ***Test 1:*** Test xuất YES có alpha là mệnh đề.

- Input

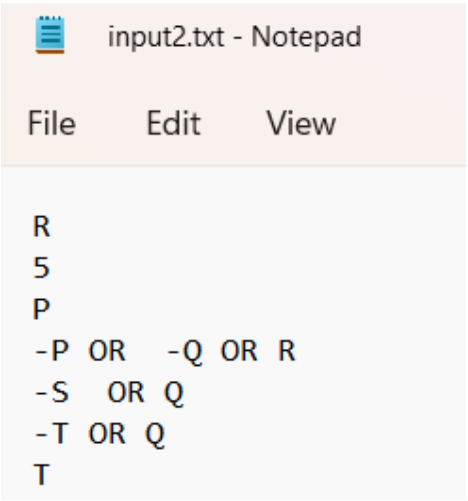


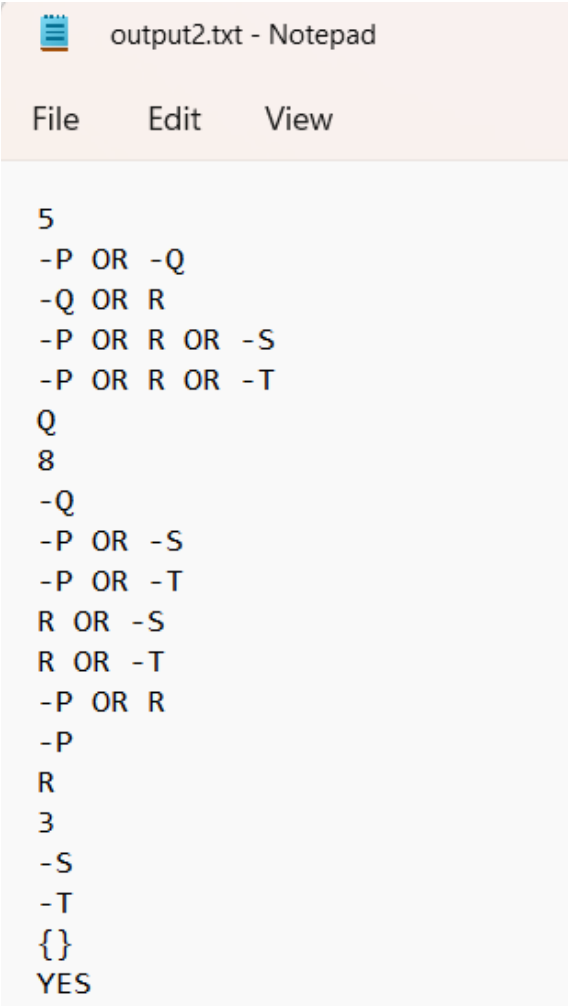
- Output



2. ***Test 2***: Test xuất YES có nhiều khoảng trắng giữa literal và từ khóa

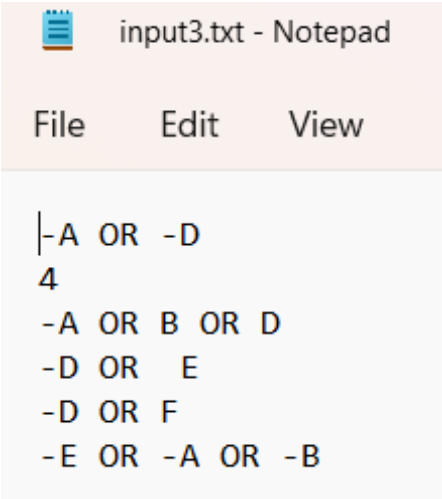
- Input – Output

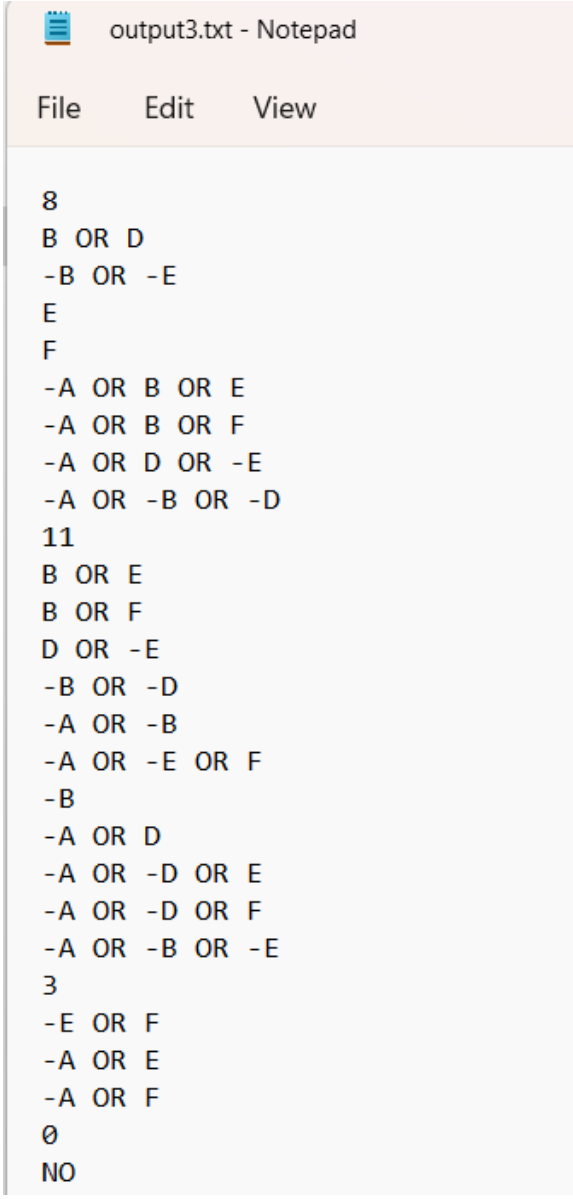




3. ***Test 3***: Test xuất NO có alpha là mệnh đề.

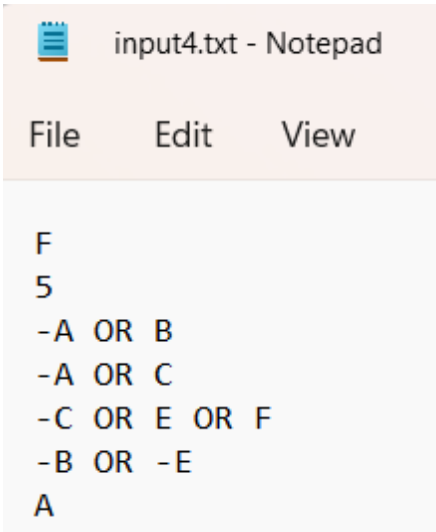
- Input – Output

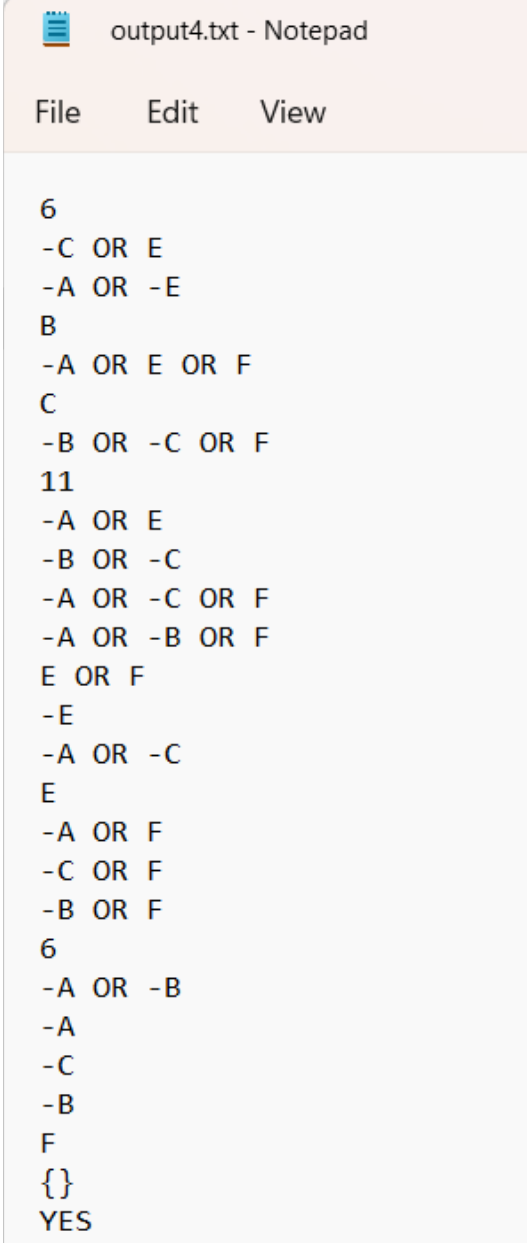




4. ***Test 4:*** Test xuất YES

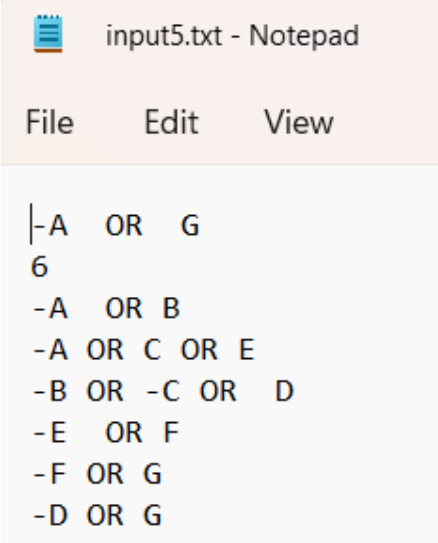
- Input - Output





5. ***Test 5:*** Test xuất YES có alpha là mệnh đề.

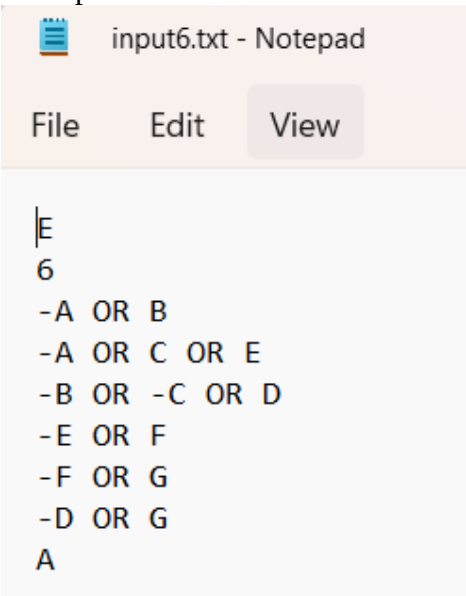
- Input



- Output: Xuất YES khá dài xem ở mục OUTPUT

6. ***Test 6:*** Test xuất NO.

- Input



- Output: Xuất NO khá dài xem ở mục OUTPUT

IV. Đánh giá:

- Ưu điểm: Giải thuật được code phân chia hàm rõ ràng từng phần dễ bảo trì sửa chữa, biến đặt tên rõ ràng có chú thích. Code có sự linh động trong việc đọc file, có thể nhập nhiều khoảng trắng giữa cá literal và từ khóa hay khoảng trắng đầu, cuối mệnh đề đều giải quyết được.
- Nhược điểm: Giải thuật được code dùng mảng arr_cur lưu tất cả các mệnh đề được tạo thành đến hiện tại rồi chọn 2 trong số các mệnh đề đã được tạo thành để tiến hành hợp giải, việc này có phần không tối ưu. Trong số các mệnh đề vòng lặp trước vong lặp hiện

tại đã hợp giải trước rồi thì làm như này sẽ bị lặp lại việc kiểm tra hợp giải các mệnh đề ấy, không cần thiết. Tuy nhiên việc này không quá ảnh hướng nhiều đến code cũng như tốc dộ chạy vì số mệnh đề không quá lớn. Giải phát khắc phục khuyết điểm trên là sử dụng phần tử mảng add để hợp giải với phần tử mảng arr_cur tránh bị lặp lại và tối ưu hơn.