**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

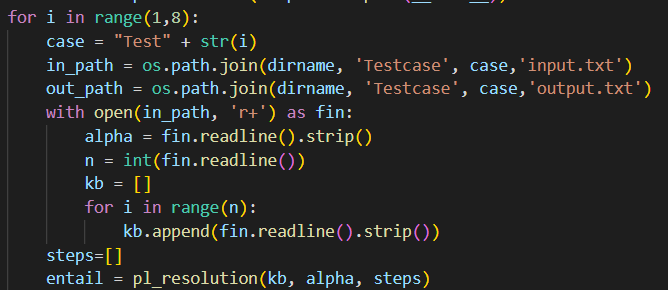
Contents

[**1.** **Ý tưởng thuật toán** 2](#_Toc133153117)

[**2.** **Kịch bản kiểm thử** 7](#_Toc133153118)

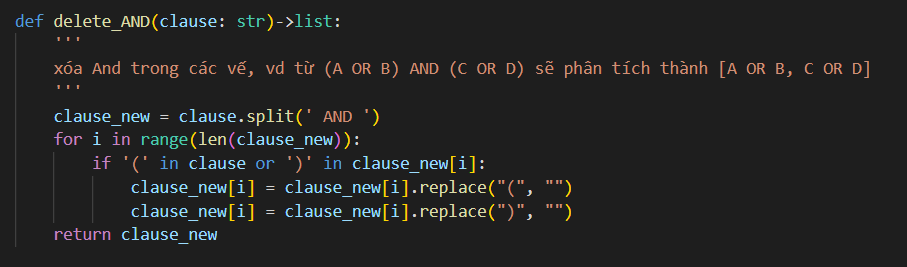
[**3.** **Đánh giá thuật toán:** 14](#_Toc133153119)

1. **Ý tưởng thuật toán**

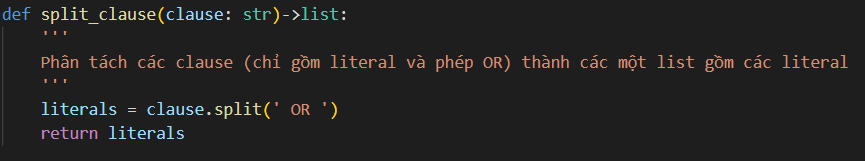


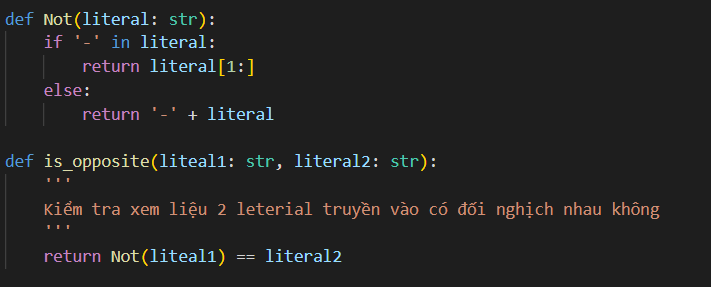
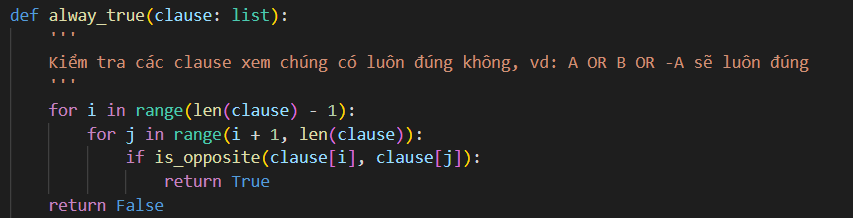
Sau khi đã đọc alpha và kb từ file “input.txt”, ta sẽ thực hiện việc gọi hàm pl\_resolution để giải bài toán, các biến truyền vào bao gồm: kb và alpha đã đọc từ file, mảng steps dùng để lưu các mệnh đề được tạo ra sau các vòng lặp.

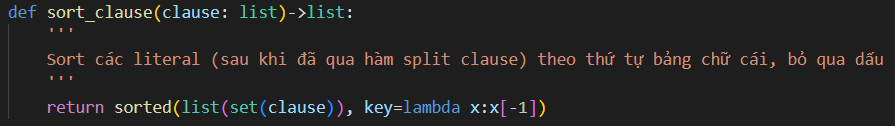
Trước khi nói về hàm pl\_resolution, ta sẽ đi qua một số hàm hỗ trợ sau:

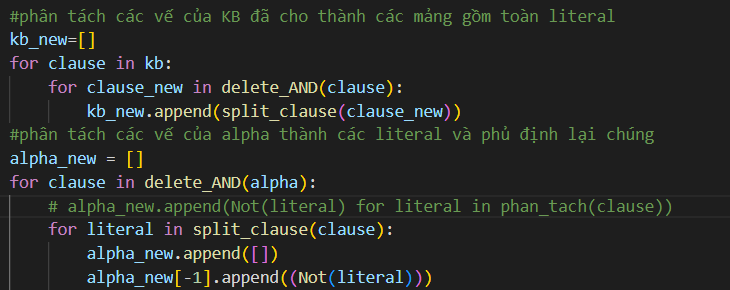
* Hàm delete\_And: sử dụng để xóa bỏ quan hệ AND trong các mệnh đề có sẵn, dựa theo nguyên tắc A AND B <=> A, B (Vì theo như lý thuyết, trong một mệnh đề theo chuẩn CNF vẫn có thể tồn tại phép AND)

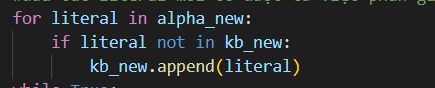
* Hàm split\_clause: Sau khi đã xóa bỏ phép AND, trong mệnh đề chỉ còn lại các literal được nối với nhau bởi phép OR, hàm split\_clause sẽ phân tích các câu mệnh đề dạng string này thành một mảng gồm các literal để thuận tiện hơn cho việc tính toán hợp giải sau này:



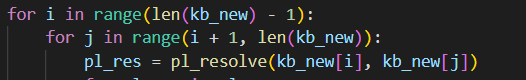
* Hàm Not và is\_opposite: hàm Not nhận vào 1 literal và trả về phủ định của literal đó, Hàm is\_opposite nhận vào 2 literal và trả về kết quả xem 2 literal đó có đối nghịch nhau không.
* Hàm always\_true: sau khi hợp giải 2 mệnh đề và sinh ra một mệnh đề mới, sẽ tồn tại khả năng để mệnh đề này có dạng “… OR A OR -A OR …”, lúc này mệnh đề này là luôn đúng và ta có thể bỏ qua nó.
* 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.

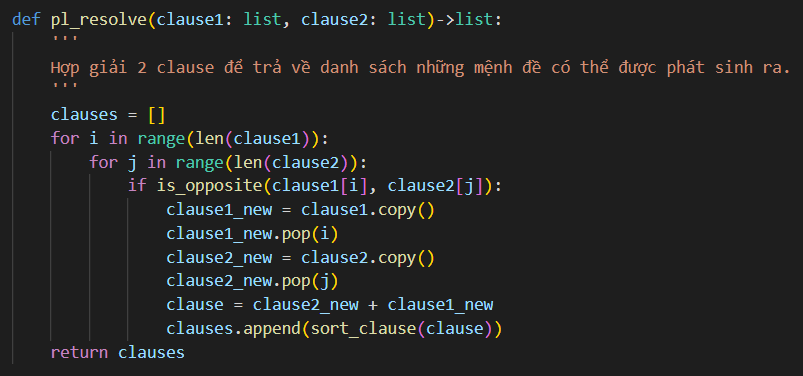


Hàm pl\_resolution: sau khi truyền vào kb và alpha, ta sẽ lần lượt phân tích chúng bằng cách sử dụng 2 hàm delete\_AND và split\_clause. Đặc biệt đối với alpha, ta sẽ thêm việc là phủ định lại toàn bộ các literal với hàm Not();

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

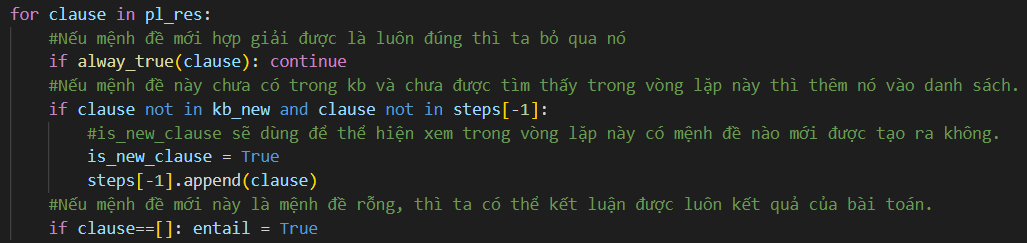
Bước vào phần hợp giải, ta thực hiện liên tục các vòng lặp để phân tích, ở mỗi vòng lặp, ta sẽ kiểm tra từng cặp mệnh đề i và j đang có trong kb, và thực hiện ghép chúng với nhau thông qua hàm pl\_resolve:





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.

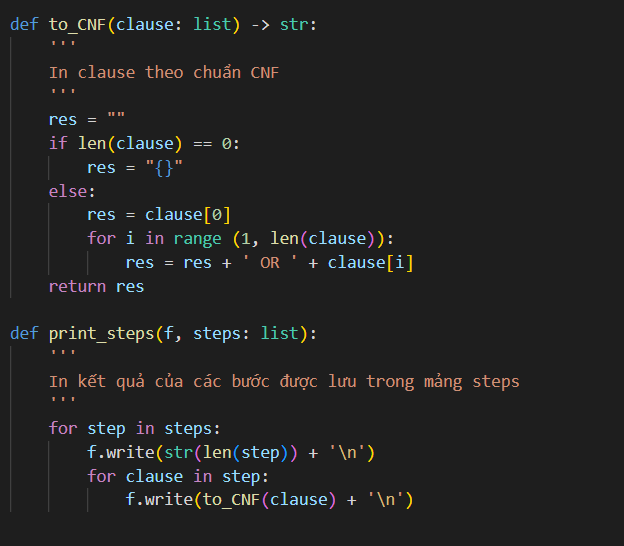
Với mỗi mệnh đề mới sinh ra từ 2 cặp mệnh đề i và j cho trước, sẽ có các trường hợp sau xảy ra:

* 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.

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.

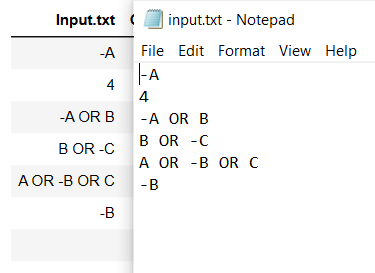
Khi hàm pl\_resolution chạy xong, ta sẽ có được kết quả xem liệu K có entail alpha không, và đồng thời đó là những mệnh đề được sinh ra trong các vòng lặp được lưu trong mảng steps. Việc còn lại chỉ là in ra theo yêu cầu của đề bài, ta sẽ sử dụng hàm print\_step (để in các bước) và to\_CNF (từ mảng các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

1. **Kịch bản kiểm thử**

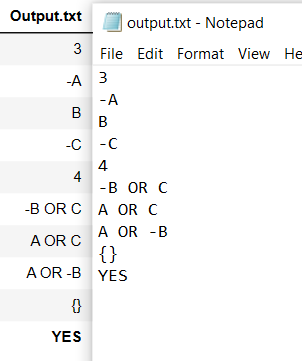
Dưới đây sẽ là 5 kịch bản kiểm thử bao gồm 2 file “input.txt” và “output.txt”

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

* Intput:



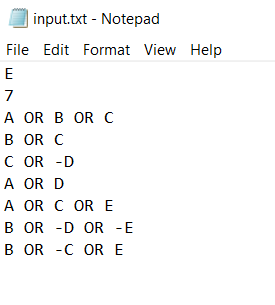
* Output:



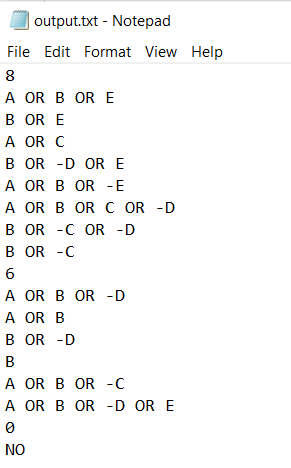
Dễ thấy, kết quả đúng như kết quả cho sẵn ở đề bài

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:

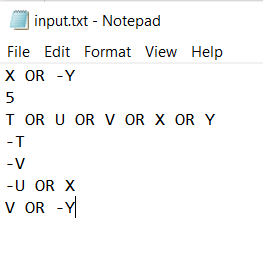
****

* Output:

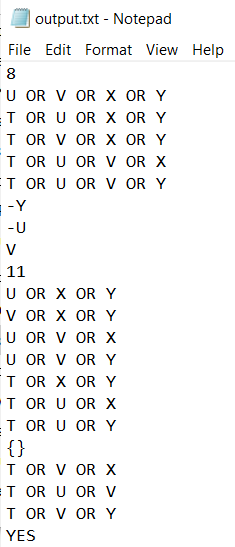


Bộ test 3: tăng số lượng literal ở alpha

* Input:

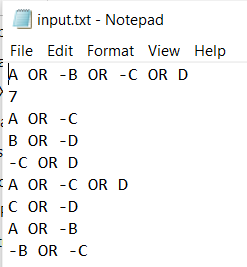


* Output:

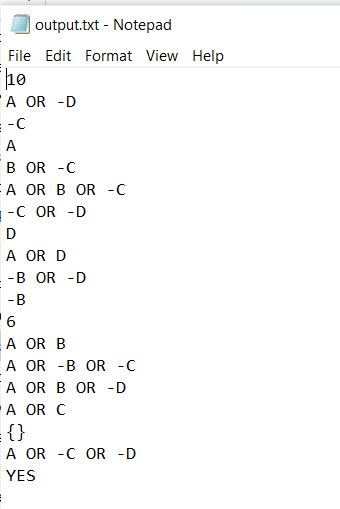


Bộ test 4: Một bộ test có độ phức tạp cao với alpha gồm nhiều literal và kb với số lượng mệnh đề khá lớn

* Input:

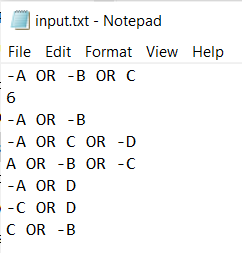


* Output:

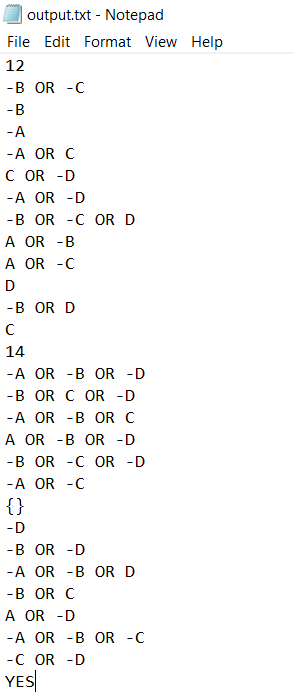


Bộ test 5: tương tự như bộ test 4

* Input:



* Output:



1. **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(n2), 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.