2 PROLOG

Tham khảo tài liệu **Hướng dẫn sử dụng Prolog**.

2.1 Khởi động

Cho các mệnh đề sau:

- Mary thích thức ăn (food).
- Mary thích rượu vang (wine).
- John thích rượu vang.
- John thích Mary.

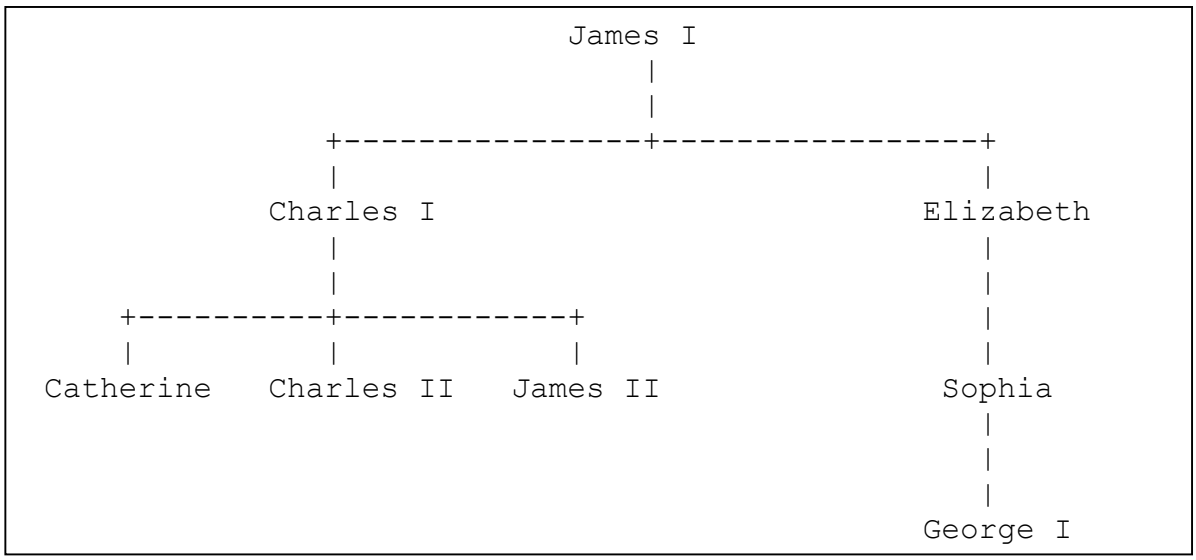
a. Biểu diễn các mệnh trên bằng ngôn ngữ Prolog

b. Thành lập câu truy vấn và ghi nhận kết quả cho các câu hỏi sau:
   - Mary có thích thức ăn không ?
   - John có thích rượu vang không ?
   - John có thích thức ăn không ?

c. Bổ sung các luật sau đây vào chương trình của câu a.
   - John thích những thứ Mary thích.
   - John thích người thích rượu vang
   - John thích những người tự thích chính bản thân họ.

2.2 Cây gia phả

Cho cây gia phả như hình bên dưới:



Giới tính của từng người được cho trong bảng sau:

```
James I   : male
Charle I  : male
Charle II : male
Jame II   : male
George I  : male

Catherine : female
Elizabeth : female
Sophia    : female
```

a. Sử dụng các vị từ:
   - male(X) = "X là nam"
   - female(X) = "X là nữ"
   - parent(X, Y) = "X là cha/mẹ của Y"

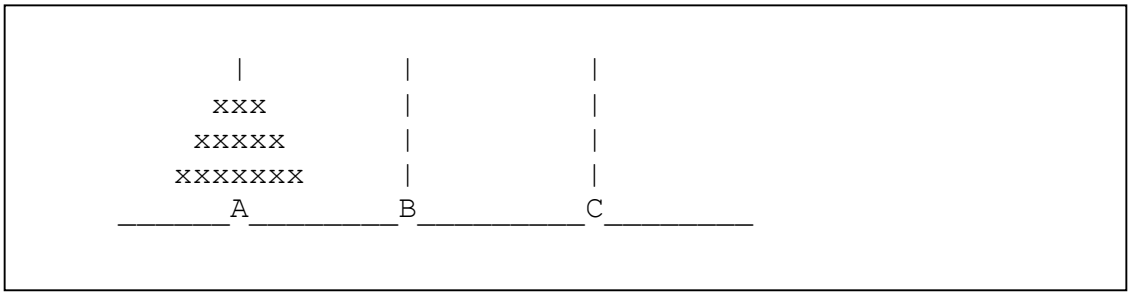
   biểu diễn cây gia phả trên bằng Prolog

b. Thành lập truy vấn và ghi nhận kết quả cho các câu hỏi sau:
   - George I có phải là cha/mẹ của Charle I không ?
   - Cha/mẹ của Charles I là ai ?
   - Ai là con của Charles I ?

c. Thêm các luật sau vào chương trình của câu a:
   - M là mẹ (mother) của X nếu M là cha/mẹ (parent) của X và M là nữ (female).
   - F là cha (father) của X nếu F là cha/mẹ (parent) của X và F là nam (male).
   - X là anh/em (sibling) của Y nếu họ có cùng cha/mẹ (parent).

2.3 Tháp Hà Nội

Bài toán tháp Hà Nội là một bài toán khá quen thuộc: Có 3 cái cọc A, B, C. Một chồng n đĩa có bán kính khác nhau được đặt vào cọc A. Hãy tìm cách di chuyển tất cả chồng đĩa sang cọc B. Quy tắc: mỗi lần di chuyển 1 đĩa trên cùng của một cọc và đặt nó sang 1 cọc khác sao cho đĩa trên phải có bán kính nhỏ hơn đĩa dưới.



Để giải bài toán này, trước hết ta thiết kế vị từ move(N, A, B, C): thực hiện việc di chuyển N đĩa từ cột A sang cột B thông qua cột C.

Nếu N = 1, ta chuyển trực tiếp từ A sang B, ta in ra A --> B.

Nếu N > 1, ta thực hiện move (N-1, A, C, B), move(1, A, B, C) và move(N-1, C, B, A).

Code:

```
move(1, A, B, _) :- print(A), print(' --> '),
                    print(B), nl.
move(N, A, B, C) :- N1 is N-1,
                    move(N1, A, C, B),
                    move(1, A, B, C), move(N1, C, B, A).
```

Trong đó:

- Khi biến xuất hiện trong định nghĩa của vị từ mà ta không sử dụng nó, ta sử dụng dấu gạch dưới _ để tránh cảnh báo (warning):

move(1, A, B, C) :- print(A), print(' --> '), print(B), nl. user:1: warning: singleton variables [C] for move/4

- print(X): in nội dung của biến X.
- nl: xuống dòng.
- 'symbol': biểu diễn ký hiệu tên symbol. symbol được xem như hằng. Do Prolog quy ước hằng phải được viết bằng chữ thường, biến viết bằng chữ hoa nên nếu muốn hằng là chữ hoa ta để nó trong cặp dấu nháy đơn '' và xem nó như symbol.

Chạy thử:

```
?- move(3, 'A', 'B', 'C').
A --> B
A --> C
B --> C
A --> B
C --> A
C --> B
A --> B
```

2.4 Số học

Tham khảo các phép toán số học của GNU Prolog:

http://www.gprolog.org/manual/html_node/gprolog030.html

1. Viết chương trình tính n!: giai_thua(N, R).

Nếu N = 0, kết quả là 1, ngược lại ta tính (N-1)! Và nhân kết quả với N.

giai_thua(0, 1).

giai_thua(N, R) :- N > 0, N1 is N – 1, giai_thua(N1, R1), R is R1*N.

Sử dụng:

giai_thua(5, R).

2. Viết chương trình tính số Fibonaci thứ N biết rằng:

$F_1 = 1$

$F_2 = 1$

$F_n = F_{n-1} + F_{n-2}$ với $n > 2$.

```
fibo(1, 1).
fibo(2, 1).
fibo(N, R) :- N > 2, N1 is N-1, N2 is N-2, fibo(N1, R1), fibo(N2, R2), R is R1 + R2.
```

Sử dụng:

```
fibo(5, R).
```

3. Viết chương trình kiểm tra số P có phải là số nguyên tốt không prime(P).

Ta thiết kế một vị từ phụ để kiểm tra số P có chia hết cho số nào từ 2 đến sqrt(P) không: **divide(N, P)**. Nếu P chia hết cho N => dừng, ngược lại kiểm tra xem N*N có còn nhỏ hơn P không, nếu vẫn còn thì tăng N lên 1 và tiếp tục kiểm tra N+1 với P.

```
divide(N, P) :- P mod N =:= 0.
divide(N, P) :- P mod N \= 0, N*N < P, N1 is N+1, divide(N1, P).
```

Chú ý:

Các phép toán trên số học: =:=, =\=, <, >, =<, >= định trị hai vế và so sánh.

Sau khi đã có vị từ divide, ta định nghĩa hàm prime thông qua vị từ này.

```
prime(P) :- \+divide(2, P).
```

P là số nguyên tố nếu P không chia hết số nào từ 2 đến sqrt(P).

Sử dụng:

```
prime(7).
```

4. Viết chương trình tìm ước chung lớn nhất của 2 số A và B: **gcd(A, B, D)**.
5. Viết chương trình kiểm tra hai số có nguyên tố cùng nhau không: **coprime(A, B)**.
6. Viết chương trình tính hàm φ(m) tìm hàm Euler của m: **phi(M, R)**. Hàm φ(m) được định nghĩa bằng số các số nguyên dương nhỏ hơn m nguyên tố cùng nhau với m. Ví dụ φ(10) = 4. Các số từ 1 đến 9 nguyên tố cùng nhau với 10 là 1, 3, 7, 9. Chú ý: trường hợp đặc biệt: φ(1) = 1.

2.5 Danh sách

Tạo danh sách:

[]: danh sách rỗng

[a, b, c]: danh sách gồm có 3 phần tử a, b và c

[x | [b, c]]: danh sách gồm có 3 phần tử a, b và c

[x | L]: danh sách có phần tử đầu tiên là x và phần đuôi là L

[a, b | [c]]: danh sách gồm có 3 phần tử a, b và c

Truy xuất phần tử của danh sách:

- Sử dụng phép hợp nhất (unification), ví dụ:
- [X, Y] = [1, 2]. Cho kết quả X = 1 và Y = 2
- [X | Y] = [1, 2, 3, 4]. Cho kết quả X = 1, Y = [2, 3, 4]
- [_, X|Y] = [1, 2, 3, 4]. Cho kết quả X = 2, Y = [3, 4]

1. Viết chương trình đếm số phần tử của một danh sách

Danh sách rỗng có số phần tử bằng 0. Nếu danh sách có thể tách thành 1 phần: đầu và danh sách đuôi T, ta đếm số phần tử của T, sau đó lấy kết quả cộng thêm 1.

```
count([], 0).
count([_|T], N) :- count(T, N1), N is N1 + 1.
```

Sử dụng:

```
count([1, 3, 5, 2, 6, 3], N).
```

2. Viết chương trình kiểm tra một phần tử có trong danh sách hay không: member(X, L).

Nếu phần tử đầu của danh sách L giống với X thì dừng.

Ngược lại, tìm X trong phần đuôi của L.

```
member(X, [X|_]).
member(X, [Y|L]) :- X \= Y, member(X, L).
```

Sử dụng:

```
member(2, [3, 4, 2, 5, 7]).
```

Ta có thể sử dụng hàm này để liệt kê các phần tử của 1 danh sách bằng cách gọi !

```
member(X, [1,  2, 3]).
```

Sau đó gõ dấu chấm phẩy ‘;’ sau mỗi kết quả.

member(X, [1, 2, 3]).

X = 1 ? ;

X = 2 ? ;

X = 3

3. Viết chương trình thêm một phần tử X vào cuối danh sách L, kết quả lưu trong R.

push_back(X, L, R).

Nếu L rỗng, kết quả R là danh sách chỉ có 1 phần tử X. Nếu không ta tách L thành hai phần: đầu H và đuôi T; thêm X vào T và sau cùng ghép H vào đầu của kết quả.

push_back(X, [], [X]).

push_back(X, [H|T], R) :- push_back(X, T, R1), R = [H|R1].

Luật cuối cùng có thể viết gọn hơn như sau:

push_back(X, [H|T], [H|R1]) :- push_back(X, T, R1).

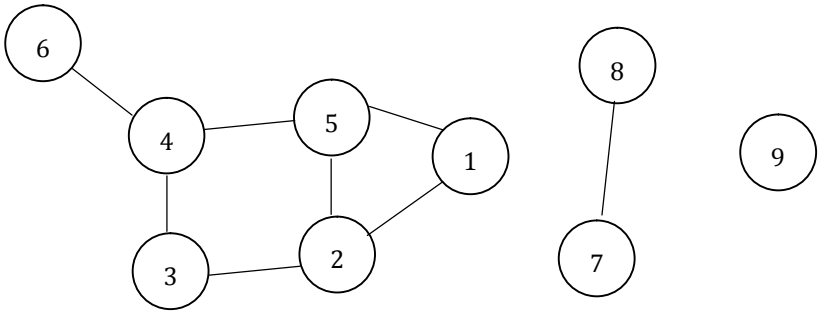
Sử dụng:

push_back(8, [5, 6, 7], R).

4. Viết chương trình nối hai danh sách L1 và L2 lại với nhau: **add_list(L1, L2, R)**. Ví dụ: add_list([1, 2, 3], [4, 5], R) kết quả: [1, 2, 3, 4, 5].
5. Viết chương trình đảo thứ tự của một danh sách: **reverse_list(L, R)**. Ví dụ: reverse_list([1, 2, 3], R) kết quả: [3, 2, 1].
6. Viết chương trình đếm số phần tử của danh sách L có giá trị bằng X: **count(X, L, N)**. Ví dụ count(3, [2, 3, 4, 3, 5], N) kết quả: N = 3.
7. Viết chương trình loại bỏ các phần tử trùng nhau của 1 danh sách: **unique(L, R)**. Ví dụ: unique([1, 2, 1, 4, 3], R) thì R = [1, 2, 4, 3].
8. Viết chương trình tìm tất cả các ước nguyên tốt của n: prime-factors(N, R). Ví dụ prime-factors(315, R) cho kết quả R = [3, 3, 5, 7].

2.6 Đồ thị

Cho đồ thị như hình vẽ. Hãy biểu diễn đồ thị này bằng ngôn ngữ Prolog để có thể truy vấn tính liên thông giữa hai đỉnh bất kỳ.



Ta thiết kế 1 vị từ edge(X, Y) để biểu diễn cạnh nối hai đỉnh X và Y. Sử dụng vị từ edge ta biểu diễn đồ thị trên như sau:

```
edge(1,2).
edge(1,5).
edge(2,3).
edge(2,5).
edge(3,4).
edge(4,5).
edge(4,6).
edge(7,8).
```

Do đồ thị đã cho là đồ thị vô hướng nên edge(1, 2) có 1 cạnh nối giữa **đỉnh 1 và đỉnh 2**. Prolog không chưa thông minh để hiểu cũng có 1 cạnh nối giữa đỉnh 2 và đỉnh 1. Vì thế, nếu ta truy vấn: edge(1,2) sẽ cho kết quả **yes**. Tuy nhiên nếu ta truy vấn edge(2,1) sẽ cho kết quả **no**.

Để giải quyết vấn đề này ta định nghĩa thêm một vị từ canh(X, Y) mô tả có cạnh nối giữa hai đỉnh X và Y thông qua vị từ edge(X, Y).

canh(X, Y) :- edge(X, Y).

canh(X, Y): - edge(Y, X).

Ta có thể gom hai luật này thành một bằng phép toán OR trong Prolog như sau:

canh(X, Y) :- edge(X, Y) ; edge(Y, X).

Để xét xem hai đỉnh có liên thông với nhau hay không ta dựa vào định nghĩa tính liên thông trong đồ thị: "Có đường đi từ đỉnh này tới đỉnh kia".

- Trường hợp đường đi có chiều dài 1 (đi trực tiếp): có cạnh nối giữa X và Y.

  lien_thong(X, Y) :- canh(X, Y).

- Trường hợp đường đi có chiều dài lớn hơn 1: tồn tại 1 đỉnh Z sao cho X, Z liên thông và có 1 cạnh giữa Z và Y.

  lien_thong(X, Y) :- lien_thong(X, Z), canh(Z, Y).

Thử truy vấn: lien_thong(1, 4). Nếu kết quả là true/yes là ok.

Thử tiếp: lien_thong(8, 9).

Chương trình sẽ rơi vào vòng lặp vô tận vì lien_thong(8, 9) sẽ gọi canh(8, 7), canh(7, 8), canh(8, 7), …. mà không tìm được lien_thong(8,9) hoặc lien_thong(7, 9).

=> Hãy tìm cách giải quyết trường hợp này.

Ta rơi vào vòng lặp vô tận là bởi vì các đỉnh bị lặp lại trên đường đi. Để tránh vị lặp vô tận, ta phải tìm đường đi **không lặp** từ X đến Y. Ta bổ sung thêm một danh sách để lưu lại các đỉnh đã đi qua để tránh lặp lại.

```
member(X, [X|_]).
member(X, [Y|T]) :- X \= Y, member(X, T).
co_duong_di(Y, Y, _).
co_duong_di(X, Y, P) :- canh(X, Z), \+member(Z, P),
co_duong_di(Z, Y, [Z|P]).
```

Vị từ member(X, L) kiểm tra xem phần tử X có trong danh sách L không. Vị từ member(X, L) đã được định nghĩa sẵn. Nếu muốn định nghĩa lại, bạn có thể đổi tên thành **member1** chẳng hạn.

Vị từ **co_duong_di(X, Y, P)** sẽ tìm xem có đường đi từ X đến Y với các đỉnh đi qua được lưu trong danh sách P. Nếu X trùng với Y, quá trình tìm đường đi kết thúc. Nếu không, tìm đỉnh Z sao cho có cạnh nối giữa X và Z và Z chưa có trong P, ta thêm Z vào P và tìm đường đi từ Z đến Y.

Sau đó, ta định nghĩa lại vị từ lien_thong(X, Y).

lien_thong(X, Y) :- co_duong_di(X, Y, [X]).

Biểu diễn đồ thị như dữ liệu:

Đồ thị như trên còn có thể được biểu diễn bằng 2 danh sách: một danh sách đỉnh và một danh sách cạnh:

[1, 2, 3, 4, 5, 6, 7, 8, 9]

[[1, 2], [1,5], [2, 3], [2, 5], [3, 4], [4, 5], [4, 6], [7, 8]].

Mỗi cạnh được biểu diễn bằng 1 danh sách có 2 phần tử. Danh sách cạnh là một danh sách các danh sách có 2 phần tử.

Với cách biểu diễn như thế, hãy viết lại hàm lien_thong(V, E, X, Y): nhận vào danh sách đỉnh V, danh sách các cạnh E và 2 đỉnh X, Y, xác định xem đỉnh X có liên thông với đỉnh Y không.

Trước hết ta viết vị từ canh(X, Y) xét xem có 1 cạnh nào nối X và Y không: **canh(X, Y, E)**: Nếu cạnh đầu tiên của danh sách cạnh E chứa [X, Y] hoặc [Y, X] thì có cạnh nối giữa 2 đỉnh X và Y. Nếu không ta tìm tiếp trong phần còn lại của danh sách E.

```
canh(X, Y, [[X, Y] | _]).
canh(X, Y, [[Y, X] | _]).
canh(X, Y, [_ | T]) :- canh(X, Y, T).
```

Kế đến ta thiết kế lại vị từ có đường đi như trên:

```
member1(X, [X|_]).
member1(X, [Y|T]) :- X \= Y, member1(X, T).
co_duong_di(_, Y, Y, _).
co_duong_di(E, X, Y, P) :- canh(X, Z, E),
                           \+member1(Z, P),
                           co_duong_di(E, Z, Y, [Z|P]).

lien_thong(E, X, Y) :- co_duong_di(E, X, Y, [X]).
```

Sử dụng:

| ?- lien_thong([[1, 2], [1,5], [2, 3], [2, 5], [3, 4], [4, 5], [4, 6], [7, 8]], 1, 8).

(1 ms) no
| ?- lien_thong([[1, 2], [1,5], [2, 3], [2, 5], [3, 4], [4, 5], [4, 6], [7, 8]], 7, 8).

true ?

yes
| ?- lien_thong ([[1, 2], [1,5], [2, 3], [2, 5], [3, 4], [4, 5], [4, 6], [7, 8]], 7, 9).

no

Bài tập nâng cao:

1. Viết vị từ **connected_part(V, E, N)** đếm số bộ phận liên thông của đồ thị được biểu diễn bằng danh sách đỉnh V và danh sách cạnh E, kết quả trả về N.

Ví dụ:

connected_parts([1, 2, 3, 4, 5, 6, 7, 8, 9], [[1, 2], [1,5], [2, 3], [2, 5], [3, 4], [4, 5], [4, 6], [7, 8]], N).

Kết quả N = 3.

Gợi ý: chọn 1 đỉnh trong V, tìm tất cả các đỉnh liên thông với V, tăng số thành phần liên thông lên 1, gọi các đỉnh còn lại là V1, đếm số thành phần liên thôn của V1. Quá trình sẽ tiếp tục cho đến khi V rỗng.

2. Viết vị từ **find_path(E, X, Y, P)** tìm đường đi từ X đến Y và lưu kết quả trong P.

Ví dụ:

find_path([[1, 2], [1,5], [2, 3], [2, 5], [3, 4], [4, 5], [4, 6], [7, 8]], 1, 6, P).

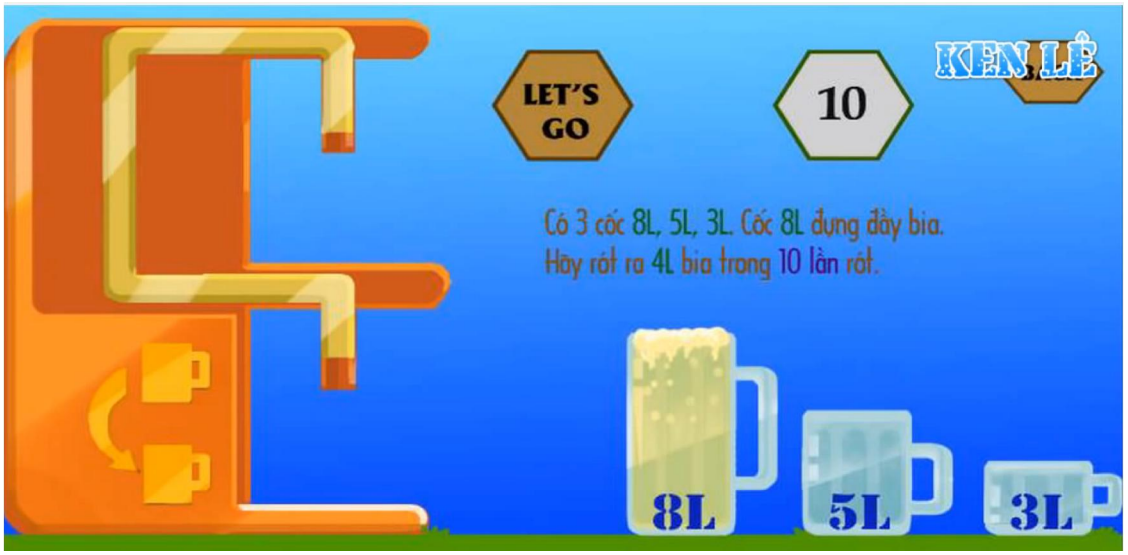
Cho kết quả: P = [1, 2, 3, 4, 6].

2.7 Trí tuệ nhân tạo

Prolog là một ngôn ngữ tuyệt vời để minh hoạ các bài toán trong lĩnh vực trí tuệ nhân tạo. Ta dùng vị từ để mô tả các trạng thái và sử dụng cơ chế suy diễn của Prolog để tìm kiếm lời giải. Ta sẽ bắt đầu bằng bài toán đong sữa/dầu/bia quen thuộc.

2.7.1 Bài toán đong sữa

Có 3 bình dung tích 8L, 5L và 3L. Bình 8L chứa đầy sữa, hai bình còn lại rỗng. Ta có thể đong sữa từ bình này sang bình kia theo quy tắc: đong bình A sang bình B cho đến khi A hết sữa hoặc đến khi B đầy.



Hình chụp từ game "Qua sông" trên các thiết bị di động.

Để giải bài toán này, ta mô hình hoá bài toán về các trạng thái và tìm một chuỗi các trạng thái hợp lệ theo đúng quy tắc đong sữa để đạt được trạng thái mong muốn.

Ta có thể biểu diễn một trạng thái bằng vị từ state(A, B, C): với A, B, C lần lượt là số sữa hiện có trong bình 8L, 5L và 3L. Ví dụ: trạng thái bắt đầu sẽ là:

state(8, 0, 0).

Trạng thái mong muốn có thể là: state(4, 4, 0).

Một phép đong hợp lệ phải tuân theo quy tắc đong sữa. Ta thiết kế vị từ pour(X, Y) để chuyển từ trạng thái X sang trạng thái Y thông qua một bước đong.

1. Đong từ bình 8L sang bình 5L: bình 8L có sữa và bình 5L chưa đầy, bình 3L giữ nguyên.

Trường hợp đổ hết bình 8L qua bình 5L mà bình 5L vẫn chưa đầy:

pour([A1, B1, C1], [A2, B2, C2]) :- A1 > 0, B1 < 5, A2 = 0, B2 is A1 + B1, C2 = C1.

Trường hợp đổ xong bình 5L đầy, bình 8L vẫn còn sữa:

pour([A1, B1, C1], [A2, B2, C2]) :- A1 > 0, B1 < 5, A2 = A1 + B1 - 5, B2 = 5, C2 = C1.

Tương tự, hãy viết thêm một số luật để đong từ:

2. Đong từ bình 8L sang bình 3L

3. Đong từ bình 5L sang bình 3L

4. Đong từ bình 5L sang bình 8L

5. Đong từ bình 3L sang bình 8L

6. Đong từ bình 3L sang bình 5L

Sau khi đã có 6 luật (thực ra là 12) để đong, hãy thử:

pour([8, 0, 0], X).

Ta sẽ được.

X = [3, 5, 0];

X = [5, 0, 3]

Tiếp theo, ta thiết kế vị từ **find_path(S, G, P)** để tìm đường từ trạng thái bắt đầu S đi đến trạng thái mong muốn G, kết quả lưu trong P.

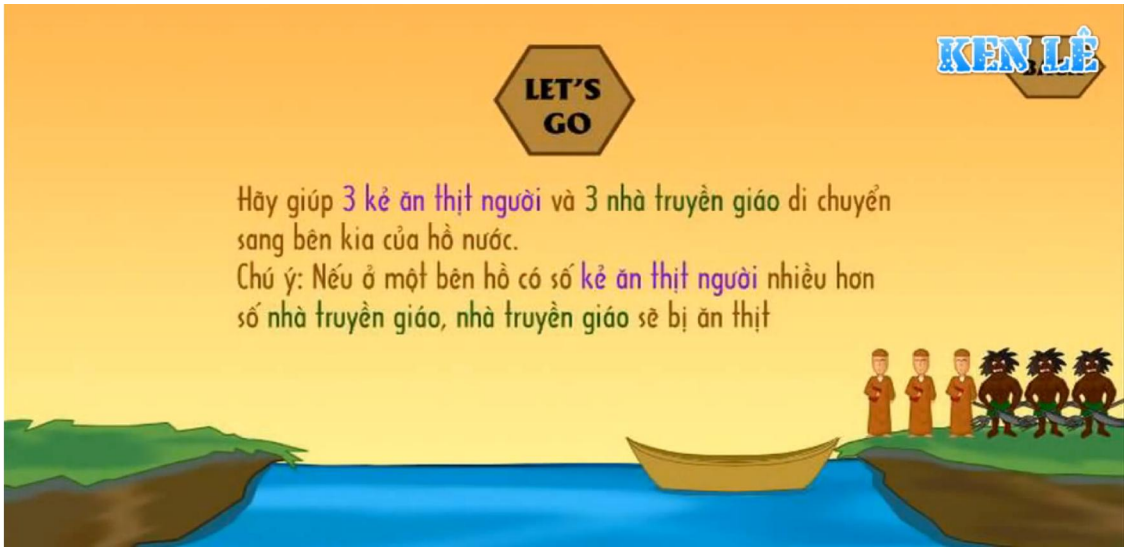
Nếu trạng thái bắt đầu S trùng với G, quá trình tìm kiếm dừng lại và trả về Path = P. Ngược lại, S khác G, tìm một trạng X sao cho có thể chuyển trạng thái từ S sang X, X chưa có trong P, thêm X vào P và tìm đường đi từ X đến G.

```
find_path(S, G, P, Path) :- S = G, Path = P.
find_path(S, G, P, Path) :- S \= G, pour(S, X),
          \+member(X, P), find_path(X, G, [X|P], Path).
```

Sử dụng:

find_path([8, 0, 0], [4, 4, 0]), [[8, 0, 0]], Path).

2.7.2 Bài toán qua sông 1 (3 nhà truyền giáo và 3 kẻ ăn thịt người)



Biểu diễn bài toán:

1 trạng thái được định nghĩa bằng bộ 3 (G, A, T) trong đó G và A: số nhà truyền giáo và số kẻ ăn thịt người đang ở bờ phải, T là vị trí thuyền (0: trái, 1: phải).

Trạng thái bắt đầu:

(3, 3, 1).

Trạng thái mong muốn: tất cả đều ở bờ trái của con sông.

(0, 0, 0).

Các phép di chuyển:

1. Đưa 2 nhà truyền giáo

Từ bờ phải sang trái

move([G1, A, T1], [G2, A, T2]) :- G1 > 1, T1 = 1, G2 is G1 – 2, valid(G2, A), T2 = 0.

Với valid(G, A) là vị từ kiểm tra xem trạng thái G, A có hợp lệ không. Các trạng thái hợp lệ:

```
valid(0, _).
valid(3, _).
valid(G, A) :- G > 0, G < 3, G >= A, (3 - G) >= (3 - A).
```

Số nhà truyền giáo phải bằng hoặc lớn hơn số kẻ ăn thịt người.

Từ bờ trái sang phải

move([G1, A, T1], [G2, A, T2]) :- G1 < 2, T1 = 0, G2 is G1 + 2, valid(G2, A), T2 = 1.

Làm tương tự như thế cho các phép toán còn lại.

3. **Đưa 1 nhà truyền giáo**

move([G1, A, T1], [G2, A, T2]) :- G1 > 0, T1 = 1, G2 is G1 – 1, valid(G2, A2), T2 = 0.

move([G1, A, T1], [G2, A, T2]) :- G1 < 3, T1 = 0, G2 is G1 + 1, valid(G2, A2), T2 = 1.

4. **Đưa 1 nhà truyền giáo và 1 kẻ ăn thịt người**

move([G1, A1, T1], [G2, A2, T2]) :- G1 > 0, A1 > 0, T1 = 1,
G2 is G1 – 1, A2 is A1 -1, valid(G2, A2), T2 = 0.

move([G1, A1, T1], [G2, A2, T2]) :- G1 < 3, A1 < 3, T1 = 0,
G2 is G1 + 1, A2 is A1 + 1, valid(G2, A2), T2 = 1.

5. **Đưa 2 kẻ ăn thịt người**

6. **Đưa 1 kẻ ăn thịt người**

Phần tìm đường đi giống hệt như bài đong sữa.

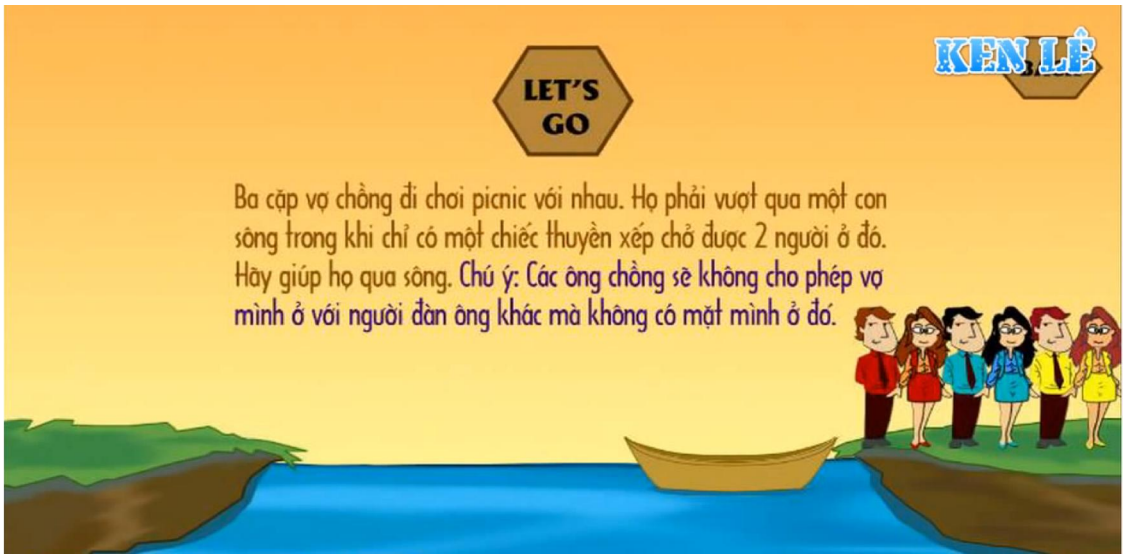
Kết quả:

find_path([3,3,1], [0, 0, 0], [[3,3,1]], P).

P =
[[0,0,0],[1,1,1],[0,1,0],[0,3,1],[0,2,0],[2,2,1],[1,1,0],[3,1,1],[3,0,0],[3,2,1],[2,2,0],[3,3,1]]
?

2.7.3 Bài toán qua sông (3 ông chồng ghen)

Có 3 cặp vợ chồng muốn qua sông. Chiếc thuyền chỉ chở được tối đa 2 người. Do 3 ông chồng đều ghen nên không để cho vợ mình ở với người đàn ông khác mà không có mặt mình.



Biểu diễn bài toán:

Gọi một trạng thái là một danh sách có 7 phần tử tương ứng với vị trí 6 người và vị trí của chiếc thuyền:

(C1, V1, C2, V2, C3, V3, T).

Nếu gọi 0: bờ trái và 1: bờ phải ta có trạng thái bắt đầu là:

(1, 1, 1, 1, 1, 1, 1).

Trạng thái kết thúc: cả 6 người đều ở bờ trái:

(0, 0, 0, 0, 0, 0, 0).

Các phép di chuyển: move(S, X).

1. Đưa C1, V1 từ bờ phải sang bờ trái: vị trí các người khác giữ nguyên, thuyền phải ở cùng bờ với C1 và V1.

move([C1, V1, C2, V2, C3, V3, T], [C11, V11, C2, V2, C3, V3, TT]) :- …

Tương tự định nghĩa các luật còn lại.

…