Bài toán con quỷ và thầy tu (qua sông)

-> Không quan trọng thứ tự.

1, mô phỏng bài toán

*định nghĩa trạng thái:

- Trạng thái ban đầu: cả ba con quỷ và cả ba nhà sư đều ở bờ phía bên trái. Tương ứng [3,3, 0, 0, 1].

- Thông tin về số lượng nhà sư và quỷ ở hai bên bờ:

[A,B,C,D] trong đó:

+ A,B là số quỷ, số sư ở bờ bên trái

+ C, D là số sư, số quỷ ở bờ bên phải.

- Trạng thái mục tiêu: Cả 3 nhà sư và cả ba con quỷ đều ở bờ bên phải, tương ứng với [0, 0, 3, 3, 0]

2, Các thao tác trạng thái:

- Chuyển một sư từ bờ trái qua bờ phải.

- Chuyển một quỷ từ bờ trái qua bờ phải

- Chuyển hai sư từ bờ trái qua bờ phải.

- Chuyển hai quỷ từ bờ trái qua bờ phải.

- Chuyển một sư và một quỷ từ bờ trái qua bờ phải.

- Chuyển một sư từ bờ phải qua bờ trái.

- Chuyển một quỷ từ bờ phải qua bờ trái

- Chuyển hai sư từ bờ phải qua bờ trái

- Chuyển hai quỷ từ bờ phải qua bờ trái.

- Chuyển một sư và một quỷ từ bờ phải qua bờ trái.

3, Các ràng buộc:

- Số quỷ phải luôn nhỏ hơn hoặc bằng số sư ở mỗi bờ trái hoặc phải.

4, Phương pháp cài đặt:

Biểu diễn trạng thái dưới dạng [A, B, C, D] tương ứng với:

- A là số sư ở bờ bên trái, B là số quỷ ở bờ bên trái, C là số sư ở bờ bên phải, D là số quỷ ở bờ bên phải.

- Trạng thái ban đầu: [3, 3, 0, 0, 1]

- Trạng thái mục tiêu: [0, 0, 3, 3, 0]

5, Chương trình cài đặt (prolog)

```
trans([A,B,C,D,1],[X,Y,Z,T,0]):-A>0,B>0,X is A-1,Y is B-1, Z is C+1, T is D+1.
trans([A,B,C,D,0],[X,Y,Z,T,1]):-C>0,D>0,X is A+1,Y is B+1, Z is C-1, T is D-1.

trans([A,B,C,D,0],[X,B,Z,D,1]):-C>1,X is A+2,Z is C-2.
trans([A,B,C,D,1],[X,B,Z,D,0]):-A>1,X is A-2,Z is C+2.

trans([A,B,C,D,1],[A,Y,C,T,0]):-B>1,Y is B-2,T is D+2.
trans([A,B,C,D,0],[A,Y,C,T,1]):-D>1,Y is B+2,T is D-2.

trans([A,B,C,D,0],[X,B,Z,D,1]):-C>0,X is A+1,Z is C-1.
trans([A,B,C,D,1],[X,B,Z,D,0]):-A>0,X is A-1,Z is C+1.
```

```
trans([A,B,C,D,0],[A,Y,C,T,1]):-D>0,Y is B+1,T is D-1.
trans([A,B,C,D,1],[A,Y,C,T,0]):-B>0,Y is B-1,T is D+1.

dangers([A,B,C,D,_]):-A>=B,C>=D.
dangers([A,B,0,_,_]):-A>=B.
dangers([0,_,A,B,_]):-A>=B.

goal([0,0,3,3,0]).

dfs(X,P,[X|P]):-goal(X),!.
dfs(X,P,L):-trans(X,Y),X\=Y,dangers(Y),\+member(Y,P),dfs(Y,[X|P],L).
```

Bài toán qua sông (nông dân, cáo ,ngỗng và hạt đậu):

-> quan trọng thứ tự.

(Cáo ăn ngỗng, ngỗng ăn đậu nếu k có nông dân ở cùng)

1, Mô phỏng bài toán:

Định nghĩa trạng thái:

- Trạng thái ban đầu nông dân, cáo,ngỗng và hạt đậu đều ở bờ bên trái. Tương ứng: [1,1,1,1,0,0,0,0]

- Tráng thái mục tiêu: nông dân, cáo,ngỗng và hạt đậu đều ở bờ bên phải, tương ứng [0,0,0,0,1,1,1,1]

- 4 vị trí đầu tiên tương ứng với bờ bên trái, 4 vị trí còn lại tương ứng với bờ bên phải.

-Thuyền sức chứa được 2 đối tượng.

2, Các tháo tác trạng thái:

- Chuyển nông dân từ bờ trái qua bờ phải.
- Chuyển nông dân với cáo từ bờ trái qua bờ phải.
- Chuyển nông dân với ngỗng từ bờ trái qua bờ phải.
- Chuyển nông dân với hạt đậu từ bờ trái qua bờ phải.
- Chuyển nông dân từ bờ phải qua bờ trái.
- Chuyển nông dân với cáo từ bờ phải qua bờ trái.
- Chuyển nông dân với ngỗng từ bờ phải qua bờ trái.
- Chuyển nông dân với hạt đậu từ bờ phải qua bờ trái.

3, Các ràng buộc:

- Ngỗng không được ở chung với cáo khi không có nông dân ở cùng ở mỗi bờ.
- Hạt đậu không được ở chung với ngỗng khi không có nông dân ở cùng.

4, Phương pháp cài đặt:

Biểu diễn trạng thái dưới dạng [A, B, C, D, E, F, G, H] tương ứng với:

A, B, C, D lần lượt ứng với nông dân, cáo, ngỗng, hạt đậu ở bờ bên trái.

E, F, G, H lần lượt ứng với nông dân, cáo, ngỗng, hạt đậu ở bờ bên phải.

Trạng thái ban đầu: [1, 1, 1, 1, 0, 0, 0, 0]

Trạng thái mục tiêu: [0, 0, 0, 0, 1, 1, 1, 1]

5, Chương trình cài đặt (prolog)

```
goal([0,0,0,0,1,1,1,1]).
```

```
dangerous([0,1,1,_,1,0,0,_]).
dangerous([0,_,1,1,1,_,0,0]).
dangerous([1,0,0,_,0,1,1,_]).
dangerous([1,_,0,0,0,_,1,1]).

famlr([1,F_l,G_l,B_l,0,F_R,G_R,B_R],[0,F_l,G_l,B_l,1,F_R,G_R,B_R]).
famrl([0,F_l,G_l,B_l,1,F_R,G_R,B_R],[1,F_l,G_l,B_l,0,F_R,G_R,B_R]).

famFoxlr([1,1,G_l,B_l,0,0,G_R,B_R],[0,0,G_l,B_l,1,1,G_R,B_R]).
famFoxrl([0,0,G_l,B_l,1,1,G_R,B_R],[1,1,G_l,B_l,0,0,G_R,B_R]).

famGoslr([1,F_l,1,B_l,0,F_R,0,B_R],[0,F_l,0,B_l,1,F_R,1,B_R]).
famGosrl([0,F_l,0,B_l,1,F_R,1,B_R],[1,F_l,1,B_l,0,F_R,0,B_R]).

famBeanlr([1,F_l,G_l,1,0,F_R,G_R,0],[0,F_l,G_l,0,1,F_R,G_R,1]).
famBeanrl([0,F_l,G_l,0,1,F_R,G_R,1],[1,F_l,G_l,1,0,F_R,G_R,0]).

trans(X,Y):-famlr(X,Y).
trans(X,Y):-famrl(X,Y).
trans(X,Y):-famFoxlr(X,Y).
trans(X,Y):-famFoxrl(X,Y).
trans(X,Y):-famGoslr(X,Y).
trans(X,Y):-famGosrl(X,Y).
trans(X,Y):-famBeanrl(X,Y).
```

```
trans(X,Y):-famBeanlr(X,Y).


dfs(X,P,[X|P]):-goal(X),!.
dfs(X,P,L):-trans(X,Y),X\=Y,\+member(Y,P),\+dangerous(Y),dfs(Y,[X|P],L).
```

https://www.facebook.com/thanhtiem.le

PROLOG

1. Đếm số lượng phần tử trong mảng A

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

2. Đến số lượng phần tử có giá trị là số lẽ

```
demle([],0).
demle([X|T],N):-X mod 2 =\= 0,
   demle(T,M),
   N is M+1.
demle([X|T], N):-X mod 2 =:= 0,
   demle(T,N).
```

3. Đến số lượng phần tử có giá trị là số chẵn

```
demchan([], 0).
demchan([X|T],N):-X mod 2 =:= 0,
   demchan(T,M),
   N is M+1.
demchan([X|T], N):-X mod 2 =\= 0,
   demchan(T,N).
```

4. Tìm phần tử có giá trị là số chẵn

```
timchan([],0).
timchan([X|T]):-X mod 2 =:= 0,write(X),timchan(T).
timchan([X|T]):-X mod 2 =\= 0,timchan(T).
```

5. Tìm phần tử có giá trị là số lẽ

```
timle([],0).
timle([X|T]):-X mod 2 =\= 0,write(X),timle(T).
timle([X|T]):-X mod 2 =:= 0,timle(T).
```

6. Tìm phần tử có giá trị lớn nhất

```
timmax([],0).
timmax([X|T],MAX):-timmax(T,A),X>=A,MAX is X .
timmax([X|T],MAX):-timmax(T,A),X<A,MAX is A.
```

7. Tìm phần tử có giá trị nhỏ nhất

```
timmin([],0).
timmin([X|T],MIN):-timmax(T,A),X>=A,MIN is A.
timmin([X|T],MIN):-timmax(T,A),X<A,MIN is X.
```

8. Tính tổng các phần tử trong danh sách

```
tong([],0).
tong([X|L],N):- tong(L,M),N is M+X.
```

9.Sắp xếp các phần tử trong danh sách theo thứ tự giảm dần

```
giam(L,KQ):-a_sort(L,[],KQ).
a_sort([],Acc,Acc).
a_sort([H|T],Acc,Sorted):-insert(H,Acc,NAcc),a_sort(T,NAcc,Sorted).
insert(X,[Y|T],[Y|NT]):-X<Y,insert(X,T,NT).
insert(X,[Y|T],[X,Y|T]):-X>=Y.
insert(X,[],[X]).
```

10.Sắp xếp các phần tử trong danh sách theo thứ tự tăng dần

```
tang(L,KQ):-b_sort(L,[],KQ).
b_sort([],Acc,Acc).
b_sort([H|T],Acc,Sorted):-b_insert(H,Acc,NAcc),b_sort(T,NAcc,Sorted).
b_insert(X,[Y|T],[Y|NT]):-X>Y,b_insert(X,T,NT).
b_insert(X,[Y|T],[X,Y|T]):-X=<Y.
b_insert(X,[],[X]).
```

1.01 (*) Tìm phần tử cuối cùng của danh sách.

```
my_last(X,[X]).
my_last(X,[_|L]) :- my_last(X,L).
```

1.02 (*) Tìm phần tử cuối cùng nhưng một trong danh sách.

```
last_but_one(X,[X,_]).
last_but_one(X,[_,Y|Ys]) :- last_but_one(X,[Y|Ys]).
```

1.03 (*) Tìm phần tử thứ k của danh sách.

```
element_at(X,[X|_],1).element_at(X,[_|L],K) :- K > 1, K1 is K - 1, element_at(X,L,K1).
```

1,04 (*) Tìm số phần tử của danh sách.

```
my_length([],0).

my_length([_|L],N) :- my_length(L,N1), N is N1 + 1.
```

1,05 (*) Đảo ngược danh sách.

```
my_reverse(L1,L2) :- my_rev(L1,L2,[]).

my_rev([],L2,L2) :- !.
```

```
my_rev([X|Xs],L2,Acc) :- my_rev(Xs,L2,[X|Acc]).
```

1.06 (*) Tìm hiểu xem danh sách có phải là bảng màu không. ví dụ [x, a, m, a, x].

```
is_palindrome(L) :- reverse(L,L).
```

1.07 (**) Làm phẳng cấu trúc danh sách lồng nhau.

Ví dụ :

```
? - my_flatten ([a, [b, [c, d], e]], X).
X = [a, b, c, d, e]
my_flatten(X,[X]) :- \+ is_list(X).
my_flatten([],[]).

my_flatten([X|Xs],Zs) :- my_flatten(X,Y), my_flatten(Xs,Ys), append(Y,Ys,Zs).
```

1.08 (**):Loại bỏ các bản sao liên tiếp của các thành phần danh sách.

```
 ? - nén ([a, a, a, a, b, c, c, a, a, d, e, e, e, e], X).
X = [a, b, c, a, d, e]
compress([],[]).
compress([X],[X]).
compress([X,X|Xs],Zs) :- compress([X|Xs],Zs).
compress([X,Y|Ys],[X|Zs]) :- X \= Y, compress([Y|Ys],Zs).
```

1.09 . Gói các bản sao liên tiếp của các thành phần danh sách vào danh sách phụ.

```
? - pack ([a, a, a, a, b, c, c, a, a, d, e, e, e, e], X).
X = [[a, a, a, a], [b], [c, c], [a, a], [d], [e, e, e, e]]
pack([],[]).
pack([X|Xs],[Z|Zs]) :- transfer(X,Xs,Ys,Z), pack(Ys,Zs).
transfer(X,[],[],[X]).
transfer(X,[Y|Ys],[Y|Ys],[X]) :- X \= Y.
transfer(X,[X|Xs],Ys,[X|Zs]) :- transfer(X,Xs,Ys,Zs)
```

1.10 (*)Mã hóa độ dài chạy của danh sách.

```
? - mã hóa ([a, a, a, a, b, c, c, a, a, d, e, e, e, e], X).
X = [[4, a], [1, b], [2, c], [2, a], [1, d], [4, e]]
:- ensure_loaded(p1_09).
encode(L1,L2) :- pack(L1,L), transform(L,L2).
transform([],[]).
transform([[X|Xs]|Ys],[[N,X]|Zs]) :- length([X|Xs],N), transform(Ys,Zs).
```

1.11 Mã hóa độ dài chạy được sửa đổi.

```
 ? - encode_modified ([a, a, a, a, b, c, c, a, a, d, e, e, e, e], X).
X = [[4, a], b, [2, c], [2, a], d, [4, e]]
:- ensure_loaded(p1_10).
encode_modified(L1,L2) :- encode(L1,L), strip(L,L2).
```

```
strip([],[]).
strip([[1,X]|Ys],[X|Zs]) :- strip(Ys,Zs).
strip([[N,X]|Ys],[[N,X]|Zs]) :- N > 1, strip(Ys,Zs).
```

1.12 (**) Giải mã danh sách được mã hóa theo chiều dài.

```
decode([],[]).
decode([X|Ys],[X|Zs]) :- \+ is_list(X), decode(Ys,Zs).
decode([[1,X]|Ys],[X|Zs]) :- decode(Ys,Zs).
decode([[N,X]|Ys],[X|Zs]) :- N > 1, N1 is N - 1, decode([[N1,X]|Ys],Zs).
```

1.13 (**) Mã hóa độ dài chạy của danh sách (giải pháp trực tiếp).

```
? - encode_direct ([a, a, a, a, b, c, c, a, a, d, e , e, e, e], X).
X = [[4, a], b, [2, c], [2, a], d, [4, e]]
encode_direct([],[]).
encode_direct([X|Xs],[Z|Zs]) :- count(X,Xs,Ys,1,Z), encode_direct(Ys,Zs).
count(X,[],[],1,X).
count(X,[],[],N,[N,X]) :- N > 1.
count(X,[Y|Ys],[Y|Ys],1,X) :- X \= Y.
count(X,[Y|Ys],[Y|Ys],N,[N,X]) :- N > 1, X \= Y.
count(X,[X|Xs],Ys,K,T) :- K1 is K + 1, count(X,Xs,Ys,K1,T).
```

1.14 (*) Sao y các thành phần của danh sách.

```
? - dupli ([a, b, c, c, d], X).
X = [a, a, b, b, c, c, c, c, d, d]
dupli([],[]).
dupli([X|Xs],[X,X|Ys]) :- dupli(Xs,Ys).
```

1.15 (**) Sao y các thành phần của danh sách một số lần nhất định.

```
? - dupli ([a, b, c], 3, X).
X = [a, a, a, b, b, b, c, c, c]
Kết quả của mục tiêu là gì ?? - dupli (X, 3, Y).
dupli(L1,N,L2) :- dupli(L1,N,L2,N).
dupli([],_,[],_).
dupli([_|Xs],N,Ys,0) :- dupli(Xs,N,Ys,N).
dupli([X|Xs],N,[X|Ys],K) :- K > 0, K1 is K - 1, dupli([X|Xs],N,Ys,K1).
```

1.16 (**) Thả mọi phần tử thứ n.x khỏi danh sách.

```
? - drop ([a, b, c, d, e, f, g, h, i, k], 3, X).
X = [a, b, d, e, g, h, k]
drop(L1,N,L2) :- drop(L1,N,L2,N).
drop([],_,[],_).
drop([_|Xs],N,Ys,1) :- drop(Xs,N,Ys,N).
drop([X|Xs],N,[X|Ys],K) :- K > 1, K1 is K - 1, drop(Xs,N,Ys,K1)
```

1.17 (*) Chia danh sách thành hai phần; chiều dài của phần đầu tiên được đưa ra.

```
? - split ([a, b, c, d, e, f, g, h, i, k], 3, L1, L2).
```

```
L1 = [a, b, c]
L2 = [d, e, f, g, h, i, k]
split(L,0,[],L).
split([X|Xs],N,[X|Ys],Zs) :- N > 0, N1 is N - 1, split(Xs,N1,Ys,Zs).
```
1.18 (**) Trích xuất một lát từ danh sách.
```
? - lát ([a, b, c, d, e, f, g, h, i, k], 3,7, L).
 L = [c, d, e, f, g]
slice([X|_],1,1,[X]).
slice([X|Xs],1,K,[X|Ys]) :- K > 1,
   K1 is K - 1, slice(Xs,1,K1,Ys).
slice([_|Xs],I,K,Ys) :- I > 1,
   I1 is I - 1, K1 is K - 1, slice(Xs,I1,K1,Ys).
```
1.19 (**) Xoay danh sách N vị trí sang trái.
```
? - xoay ([a, b, c, d, e, f, g, h], 3, X).
X = [d, e, f, g, h, a, b, c]
:- ensure_loaded(p1_17).
rotate([],_,[]) :- !.
rotate(L1,N,L2) :-
   length(L1,NL1), N1 is N mod NL1, split(L1,N1,S1,S2), append(S2,S1,L2).
```
1.20 (*) Xóa phần tử k khỏi danh sách.
```
? - remove_at (X, [a, b, c, d], 2, R).
X = b
R = [a, c, d]
remove_at(X,[X|Xs],1,Xs).
remove_at(X,[Y|Xs],K,[Y|Ys]) :- K > 1,
   K1 is K - 1, remove_at(X,Xs,K1,Ys).
```
1.21 (*) Chèn một phần tử tại một vị trí nhất định vào danh sách.
```
? - insert_at (alfa, [a, b, c, d], 2, L).
L = [a, alfa, b, c, d]
:- ensure_loaded(p1_20).
insert_at(X,L,K,R) :- remove_at(X,R,K,L).
```
1.22 (*) Tạo danh sách chứa tất cả các số nguyên trong một phạm vi nhất định.
```
? - phạm vi (4,9, L).
L = [4,5,6,7,8,9]
range(I,I,[I]).
range(I,K,[I|L]) :- I < K, I1 is I + 1, range(I1,K,L).
```
23. Trích xuất một số yếu tố được chọn ngẫu nhiên từ danh sách.Các mục đã chọn sẽ được đưa vào danh sách kết quả.
```
? - rnd_select ([a, b, c, d, e, f, g, h], 3, L).
L = [e, d, a]
:- ensure_loaded(p1_20).
rnd_select(_,0,[]).
rnd_select(Xs,N,[X|Zs]) :- N > 0,
```

```
    length(Xs,L),
    I is random(L) + 1,
    remove_at(X,Xs,I,Ys),
    N1 is N - 1,
    rnd_select(Ys,N1,Zs).
```

1.24 (*) Xổ số: Vẽ N số ngẫu nhiên khác nhau từ tập 1..M.Các số được chọn sẽ được đưa vào một danh sách kết quả.

```
? - xổ số (6,49, L).
L = [23,1,17,33,21,37]
:- ensure_loaded(p1_22).
:- ensure_loaded(p1_23).
lotto(N,M,L) :- range(1,M,R), rnd_select(R,N,L).
```

1,25 (*) Tạo hoán vị ngẫu nhiên các yếu tố của danh sách.

```
? - rnd_permu ([a, b, c, d, e, f], L).
L = [b, a, d, c, e, f]
:- ensure_loaded(p1_23).
rnd_permu(L1,L2) :- length(L1,N), rnd_select(L1,N,L2).
```

1.26 (**) Tạo kết hợp K đối tượng riêng biệt được chọn từ các thành phần N của danh sáchCó bao nhiêu cách để một ủy ban gồm 3 người được chọn từ một nhóm 12 người? Chúng ta đều biết rằng có
C (12,3) = 220 khả năng (C (N, K) biểu thị các hệ số nhị thức nổi tiếng). Đối với các nhà toán học thuần túy, kết quả này có thể là tuyệt vời. Nhưng *chúng tôi* muốn thực sự tạo ra tất cả các khả năng (thông qua quay lui).

```
? - kết hợp (3, [a, b, c, d, e, f], L).
L = [a, b, c];
L = [a, b, d];
L = [a, b, e];
combination(0,_,[]).

combination(K,L,[X|Xs]) :- K > 0,

  el(X,L,R), K1 is K-1, combination(K1,R,Xs).

el(X,[X|L],L).

el(X,[_|L],R) :- el(X,L,R).
```

2.01 (**) Xác định xem một số nguyên đã cho có phải là số nguyên tố hay không.

```
is_prime(2).
is_prime(3).
is_prime(P) :- integer(P), P > 3, P mod 2 =\= 0, \+ has_factor(P,3).
has_factor(N,L) :- N mod L =:= 0.
has_factor(N,L) :- L * L < N, L2 is L + 2, has_factor(N,L2).
```

2.02 (**) Xác định các thừa số nguyên tố của một số nguyên dương đã cho.Xây dựng một danh sách phẳng chứa các thừa số nguyên tố theo thứ tự tăng dần.

```
? - Prime_factors (315, L).
L = [3,3,5,7]
prime_factors(N,L) :- N > 0,  prime_factors(N,L,2).
prime_factors(1,[],_) :- !.
prime_factors(N,[F|L],F) :- R is N // F, N =:= R * F, !, prime_factors(R,L,F).
prime_factors(N,L,F) :- next_factor(N,F,NF), prime_factors(N,L,NF).
next_factor(_,2,3) :- !.
next_factor(N,F,NF) :- F * F < N, !, NF is F + 2.
next_factor(N,_,N).
```

2.03 (**) Xác định các thừa số nguyên tố của một số nguyên dương đã cho (2).Xây dựng một danh sách chứa các thừa số nguyên tố và bội số của chúng.

```
? - Prime_factors_mult (315, L).
L = [[3,2], [5,1], [7.1]]
:- ensure_loaded(p2_02).

prime_factors_mult(N,L) :- N > 0, prime_factors_mult(N,L,2).

prime_factors_mult(1,[],_) :- !.

prime_factors_mult(N,[[F,M]|L],F) :- divide(N,F,M,R), !, % F divides N
next_factor(R,F,NF), prime_factors_mult(R,L,NF).

prime_factors_mult(N,L,F) :- !, next_factor(N,F,NF),prime_factors_mult(N,L,NF).

divide(N,F,M,R) :- divi(N,F,M,R,0), M > 0.

divi(N,F,M,R,K) :- S is N // F, N =:= S * F, !,  K1 is K + 1, divi(S,F,M,R,K1).

divi(N,_,M,N,M).
```

2.04 (*) Một danh sách các số nguyên tố.Đưa ra một phạm vi số nguyên theo giới hạn dưới và trên của nó, xây dựng một danh sách tất cả các số nguyên tố trong phạm vi đó.

```
:- ensure_loaded(p2_01).

prime_list(A,B,L) :- A =< 2, !, p_list(2,B,L).

prime_list(A,B,L) :- A1 is (A // 2) * 2 + 1, p_list(A1,B,L).

p_list(A,B,[]) :- A > B, !.

p_list(A,B,[A|L]) :- is_prime(A), !,   next(A,A1), p_list(A1,B,L).

p_list(A,B,L) :-    next(A,A1), p_list(A1,B,L).
```

```
next(2,3) :- !.

next(A,A1) :- A1 is A + 2.
```

5. Phỏng đoán của Goldbach nói rằng mọi số chẵn dương lớn hơn 2 là tổng của hai số nguyên tố. Viết một vị ngữ để tìm hai số nguyên tố có tổng bằng một số nguyên chẵn.

```
? - goldbach (28, L).
L = [5,23]

:- ensure_loaded(p2_01).

goldbach(4,[2,2]) :- !.

goldbach(N,L) :- N mod 2 =:= 0, N > 4, goldbach(N,L,3).

goldbach(N,[P,Q],P) :- Q is N - P, is_prime(Q), !.

goldbach(N,L,P) :- P < N, next_prime(P,P1), goldbach(N,L,P1).

next_prime(P,P1) :- P1 is P + 2, is_prime(P1), !.

next_prime(P,P1) :- P2 is P + 2, next_prime(P2,P1).
```

2.06 (**) Một danh sách các chế phẩm Goldbach.

Đưa ra một loạt các số nguyên theo giới hạn dưới và trên của nó, in một danh sách tất cả các số chẵn và thành phần Goldbach của chúng.

```
? - goldbach_list (9,20).
10 = 3 + 7
12 = 5 + 7
14 = 3 + 11
16 = 3 + 13
18 = 5 + 13
20 = 3 + 17

:- ensure_loaded(p2_05).

goldbach_list(A,B) :- goldbach_list(A,B,2).

goldbach_list(A,B,L) :- A =< 4, !, g_list(4,B,L).

goldbach_list(A,B,L) :- A1 is ((A+1) // 2) * 2, g_list(A1,B,L).

g_list(A,B,_) :- A > B, !.

g_list(A,B,L) :-  goldbach(A,[P,Q]),  print_goldbach(A,P,Q,L),  A2 is A + 2,
g_list(A2,B,L).
```

```
print_goldbach(A,P,Q,L) :- P >= L, !,  writef('%t = %t + %t',[A,P,Q]),
nl.print_goldbach(_,_,_,_).
```

2.07 (**) Xác định ước số chung lớn nhất của hai số nguyên dương.Sử dụng thuật toán của Euclid.

```
? - gcd (36, 63, G).
G = 9
gcd(X,0,X) :- X > 0.
gcd(X,Y,G) :- Y > 0, Z is X mod Y, gcd(Y,Z,G).
:- arithmetic_function(gcd/2).
```

2.08 (*) Xác định xem hai số nguyên dương có phải là số nguyên hay không.

Hai số là số nguyên tố nếu ước số chung lớn nhất của chúng bằng 1.

```
? - coprime (35, 64).
Vâng

:- ensure_loaded(p2_07).

coprime(X,Y) :- gcd(X,Y,1).
```

2.09 (**) Tính hàm tổng của Euler phi (m).

m = 10: r = 1,3,7,9; do đó phi (m) = 4. Lưu ý trường hợp đặc biệt: phi (1) = 1.

```
? - Phi là totient_phi (10).
Phi = 4
:- ensure_loaded(p2_08).
:- arithmetic_function(totient_phi/1).
totient_phi(1,1) :- !.
totient_phi(M,Phi) :- t_phi(M,Phi,1,0).
t_phi(M,Phi,M,Phi) :- !.
t_phi(M,Phi,K,C) :-   K < M, coprime(K,M), !,  C1 is C + 1, K1 is K + 1,
t_phi(M,Phi,K1,C1).
t_phi(M,Phi,K,C) :-  K < M, K1 is K + 1, t_phi(M,Phi,K1,C).
```

2.10 (**) Tính hàm tổng của Euler phi (m) (2).

phi (m) = (p1 - 1) * p1 ** (m1 - 1) * (p2 - 1) * p2 ** (m2 - 1) * (p3 - 1) * p3 ** (m3 - 1) * ... Lưu ý rằng a ** b là viết tắt của sức mạnh thứ b của a.

```
:- ensure_loaded(p2_03).
totient_phi_2(N,Phi) :- prime_factors_mult(N,L), to_phi(L,Phi).
to_phi([],1).
to_phi([[F,1]|L],Phi) :- !,
   to_phi(L,Phi1), Phi is Phi1 * (F - 1).
to_phi([[F,M]|L],Phi) :- M > 1,
   M1 is M - 1, to_phi([[F,M1]|L],Phi1), Phi is Phi1 * F.
```

2.11 (*) So sánh hai phương pháp tính hàm tổng của Euler.

Sử dụng các giải pháp cho các vấn đề 2.09 và 2.10 để so sánh các thuật toán.

```
:- ensure_loaded(p2_09).
```

:- ensure_loaded(p2_10).

totient_test(N) :-write('totient_phi (p2_09):'), time(totient_phi(N,Phi1)), write('result = '), write(Phi1), nl, write('totient_phi_2 (p2_10):') time(totient_phi_2(N,Phi2)), write('result = '), write(Phi2), nl.

3.01 (**) Bảng chân lý cho các biểu thức logic.
and(A,B) :- A, B.or(A,_) :- A.or(_,B) :- B.

equ(A,B) :- or(and(A,B), and(not(A),not(B))).

xor(A,B) :- not(equ(A,B)).nor(A,B) :- not(or(A,B)).

nand(A,B) :- not(and(A,B)).impl(A,B) :- or(not(A),B).bind(true).bind(fail).

table(A,B,Expr) :- bind(A), bind(B), do(A,B,Expr), fail.

do(A,B,_) :- write(A), write(' '), write(B), write(' '), fail.

do(_,_,Expr) :- Expr, !, write(true), nl.

do(_,_,_) :- write(fail), nl.

3.02 (*) Bảng chân lý cho các biểu thức logic (2).
? - bảng (A, B, A và (A hoặc không B)).
đúng thật đúng
thất bại đúng
thất bại đúng thất
bại thất bại thất bại thất bại
:- ensure_loaded(p3_01).
:- op(900, fy,not).
:- op(910, yfx, and).
:- op(910, yfx, nand).
:- op(920, yfx, or).
:- op(920, yfx, nor).
:- op(930, yfx, impl).
:- op(930, yfx, equ).
:- op(930, yfx, xor).
4.01 (*) Kiểm tra xem một thuật ngữ đã cho có đại diện cho cây nhị phân không
istree(nil).
istree(t(_,L,R)) :- istree(L), istree(R).
tree(1,t(a,t(b,t(d,nil,nil),t(e,nil,nil)),t(c,nil,t(f,t(g,nil,nil),nil)))).
tree(2,t(a,nil,nil)).
tree(3,nil).
4.02 (**) Xây dựng cây nhị phân cân bằng hoàn toàn

```
cbal_tree(0,nil) :- !.
cbal_tree(N,t(x,L,R)) :- N > 0,N0 is N - 1, N1 is N0//2, N2 is N0 -
N1,distrib(N1,N2,NL,NR),cbal_tree(NL,L), cbal_tree(NR,R).
distrib(N,N,N,N) :- !.
distrib(N1,N2,N1,N2).
distrib(N1,N2,N2,N1).
```

4.03 (**) Cây nhị phân đối xứng

```
symmetric(nil).
symmetric(t(_,L,R)) :- mirror(L,R).
mirror(nil,nil).
mirror(t(_,L1,R1),t(_,L2,R2)) :- mirror(L1,R2), mirror(R1,L2).
```

4.08 (*) Đếm số lá của cây nhị phân

```
:- ensure_loaded(p4_01).

count_leaves(nil,0).

count_leaves(t(_,nil,nil),1).

count_leaves(t(_,L,nil),N) :- L = t(_,_,_), count_leaves(L,N).

count_leaves(t(_,nil,R),N) :- R = t(_,_,_), count_leaves(R,N).

count_leaves(t(_,L,R),N) :- L = t(_,_,_), R = t(_,_,_),   count_leaves(L,NL),
count_leaves(R,NR), N is NL + NR.

count_leaves1(nil,0).

count_leaves1(t(_,nil,nil),1) :- !.

count_leaves1(t(_,L,R),N) :-  count_leaves1(L,NL), count_leaves1(R,NR), N is NL+NR.
```

4.09 (*) Thu thập lá của cây nhị phân trong danh sách

```
:- ensure_loaded(p4_01).
leaves(nil,[]).
leaves(t(X,nil,nil),[X]).
leaves(t(_,L,nil),S) :- L = t(_,_,_), leaves(L,S).
leaves(t(_,nil,R),S) :- R = t(_,_,_), leaves(R,S).
leaves(t(_,L,R),S) :- L = t(_,_,_), R = t(_,_,_),leaves(L,SL), leaves(R,SR), append(SL,SR,S).
leaves1(nil,[]).
leaves1(t(X,nil,nil),[X]) :- !.
leaves1(t(_,L,R),S) :- leaves1(L,SL), leaves1(R,SR), append(SL,SR,S).
nnodes(nil,0) :- !.
nnodes(t(_,L,R),N) :- N > 0, N1 is N-1,   between(0,N1,NL), NR is N1-NL,  nnodes(L,NL),
nnodes(R,NR).
leaves2(T,S) :- leaves2(T,S,0).
```

```
leaves2(T,S,N) :- nnodes(T,N), leaves1(T,S).
leaves2(T,S,N) :- N1 is N+1, leaves2(T,S,N1).
```

4.10 (*) Thu thập các nút bên trong của cây nhị phân trong danh sách

```
:- ensure_loaded(p4_01).
internals(nil,[]).
internals(t(_,nil,nil),[]).
internals(t(X,L,nil),[X|S]) :- L = t(_,_,_), internals(L,S).
internals(t(X,nil,R),[X|S]) :- R = t(_,_,_), internals(R,S).
internals(t(X,L,R),[X|S]) :- L = t(_,_,_), R = t(_,_,_),   internals(L,SL), internals(R,SR),
append(SL,SR,S).
internals1(nil,[]).
internals1(t(_,nil,nil),[]) :- !.
internals1(t(X,L,R),[X|S]) :-  internals1(L,SL), internals1(R,SR), append(SL,SR,S).
internals2(nil,[]).
internals2(t(X,L,R),[X|S]) :-    append(SL,SR,S), internals2(L,SL), internals2(R,SR).
```

4.11 (*) Thu thập các nút ở một mức nhất định trong danh sách

```
:- ensure_loaded(p4_01).
atlevel(nil,_,[]).
atlevel(t(X,_,_),1,[X]).
atlevel(t(_,L,R),D,S) :- D > 1, D1 is D-1,   atlevel(L,D1,SL), atlevel(R,D1,SR),
append(SL,SR,S).
levelorder(T,S) :- levelorder(T,S,1).
levelorder(T,[],D) :- atlevel(T,D,[]), !.
levelorder(T,S,D) :- atlevel(T,D,SD),   D1 is D+1, levelorder(T,S1,D1), append(SD,S1,S).
```

LISP

P01 (*) Tìm hộp cuối cùng của danh sách.

* (cuối cùng của tôi '(abcd))

```
(defun my_last (lista)
  (if (null lista)
      nil
    (if (null (rest lista))
       lista                    ; testa se a lista so tem um elemento
      (my_last (rest lista))          ; recursao no resto da lista
      )))
```

P02 (*) Tìm hộp cuối cùng nhưng một hộp của danh sách.

* (my-but-last '(abcd))

```
(defun penultimo (lista)
  (let ((reverso (reverse lista)))     ; coloca na variavel reverso o
                                       ; reverse de lista
    (cond
     ((null reverso) nil)
     ;; se a lista tem 2 ou menos elementos, retorno a propria
     ((<= (length reverso) 2)Â  lista)
     ;; se tiver mais de 2 elementos, construo uma lista
     (t (list (second reverso) (first reverso)))
     )))
```

P03 (*) Tìm phần tử thứ K của danh sách.

Phần tử đầu tiên trong danh sách là số 1.

```
(defun element-at (org-list pos &optional (ini 1))
    (if (eql ini pos)
        (car org-list)
        (element-at (cdr org-list) pos (+ ini 1))))

;;; Outra solucao

(defun element-at (lista n)
  (if (= n 1)
      ;; o primeiro elemento esta na posicao 1
      (first lista)
    (element-at (rest lista) (1- n))))
```

P04 (*) Tìm số phần tử của danh sách.

```
(defun comprimento (lista)
  (if (null lista)
      0
    (1+ (comprimento (rest lista)))))
```

P05 (*) Đảo ngược danh sách.

```
(defun inverte (lista)
  (inverte-aux lista () )
  )

(defun inverte-aux (lista resto)
  (if (null lista)
      resto
    (inverte-aux (rest lista) (cons (first lista) resto) )))
```

P06 (*) Tìm hiểu xem danh sách có phải là bảng màu không.ví dụ (xamax).

```
(defun palin (lista)
  (equal lista (reverse lista))
  )
```

P07 (**) Làm phẳng cấu trúc danh sách lồng nhau.

(my-flatten '(a (b (cd) e))) (ABCDE)

```
(defun flatten (orig-list)
    (if (eql orig-list nil)
```

```
        nil
        (let ((elem (car orig-list)) (resto-list (cdr orig-list)))
            (if (listp elem)
                (append (flatten elem) (flatten resto-list))
                (append (cons elem nil) (flatten resto-list))))))
```

P08 (**) Loại bỏ các bản sao liên tiếp của các thành phần danh sách.

* (nén '(aaaabccaadeeee)) (ABCADE)

```
(defun compress (lista)
  (cond
   ((null lista) nil)
   ((null (cdr lista)) lista)
   ;; se o primeiro elemento (de lista) e' igual ao consecutivo
   ;; (primeiro do resto)
   ((eql (first lista) (first (rest lista)))
    ;; entao ignora-se o primeiro da lista e continua recursivamente no resto
    (compress (rest lista)))
   (t (cons (first lista) (compress (rest lista))))) )
```

P09 (**) Gói các bản sao liên tiếp của các thành phần danh sách vào danh sách phụ.

* (gói '(aaaabccaadeeee)) ((AAAA) (B) (CC) (AA) (D) (EEEE))

```
(defun pack (lista)
        (if (eql lista nil)
                nil
                (cons (pega lista) (pack (tira lista)))))
(defun pega (lista)
    (cond ((eql lista nil) nil)
          ((eql (cdr lista) nil) lista)
          ((equal (car lista) (cadr lista))
              (cons (car lista) (pega (cdr lista))))
          (t (list (car lista)))))
(defun tira (lista)
    (cond ((eql lista nil) nil)
          ((eql (cdr lista) nil) nil)
          ((equal (car lista) (cadr lista))
              (tira (cdr lista)))
          (t (cdr lista))))
```

P10 (*) Mã hóa độ dài chạy của danh sách.

* (mã hóa '(aaaabccaadeeee)) ((4 A) (1 B) (2 C) (2 A) (1 D) (4 E))

```
(defun encode (lista)
        (if (eql lista nil)
                nil
                (cons (list (length (pega lista)) (car lista)) (encode (tira lista)))))
(defun pega (lista)
    (cond ((eql lista nil) nil)
          ((eql (cdr lista) nil) lista)
          ((equal (car lista) (cadr lista))
              (cons (car lista) (pega (cdr lista))))
          (t (list (car lista)))))
(defun tira (lista)
    (cond ((eql lista nil) nil)
          ((eql (cdr lista) nil) nil)
          ((equal (car lista) (cadr lista))
              (tira (cdr lista)))
          (t (cdr lista))))
```

P11 (*) Mã hóa độ dài chạy được sửa đổi.

* (đã sửa đổi mã hóa '(aaaabccaadeeee)) ((4 A) B (2 C) (2 A) D (4 E))

```
(defun encode (lista)
        (if (eql lista nil)
                nil
                (if (= (length (pega lista)) 1 )
```

```
                    (cons (car lista) (encode (tira lista)))
                    (cons (list (length (pega lista)) (car lista)) (encode (tira
lista))))
        )
)
(defun pega (lista)
    (cond ((eql lista nil) nil)
          ((eql (cdr lista) nil) lista)
          ((equal (car lista) (cadr lista))
              (cons (car lista) (pega (cdr lista))))
          (t (list (car lista)))
    )
)
(defun tira (lista)
    (cond ((eql lista nil) nil)
          ((eql (cdr lista) nil) nil)
          ((equal (car lista) (cadr lista))
              (tira (cdr lista)))
          (t (cdr lista))))
```

P14 (*) Sao y các thành phần của danh sách.

* (dupli '(abccd)) (AABBCCCCDD)

```
(defun dupli (lista)
    (if (eql lista nil)
        nil
    (append (list (car lista) (car lista)) ( dupli (cdr lista)))))
(dupli '(1 2 3 4 5 6 7 8))
```

P15 (**) Sao chép các phần tử của danh sách một số lần nhất định.

* (repli '(abc) 3) (AAABBBCCC)

```
(defun dupli (lista int &optional (ini int))
    (cond ((eql lista nil) nil)
          ((<= int 0) (dupli (cdr lista) ini ini))
          (t (cons (car lista) (dupli lista (1- int) ini )))))
```

P17 (*) Chia danh sách thành hai phần; chiều dài của phần đầu tiên được đưa ra.

* (split '(abcdefghik) 3) ((ABC) (DEFGHIK))

```
(defun split (pos org-list &optional (ini 0))
    (if (> pos ini)
        (cons (car org-list) (split pos (cdr org-list) (+ ini 1)))))
(defun split-after (pos org-list &optional (ini 0))
    (if (> pos ini)
        (split-after pos (cdr org-list) (+ ini 1))
        org-list))
```

P20 (*) Xóa phần tử K'th khỏi danh sách.

* (remove-at '(abcd) 2) (ACD)

```
(defun remove-at (org-list pos &optional (ini 1))
    (if (eql pos ini)
        (cdr org-list)
        (cons (car org-list) (remove-at (cdr org-list) pos (+ ini 1)))))
```

P21 (*) Chèn một phần tử tại một vị trí nhất định vào danh sách.

* (insert-at 'alfa' (abcd) 2) (A ALFA BCD)

```
(defun insert-at (elem org-list pos)
    (if (or (eql pos 1) (eql org-list nil))
            (cons elem org-list)
         (cons (car org-list) (insert-at elem (cdr org-list) (- pos 1)))))
```

P22 (*) Tạo một danh sách chứa tất cả các số nguyên trong một phạm vi nhất định.

* (phạm vi 4 9) (4 5 6 7 8 9)

```
(defun range (ini fim)
    (if (> ini fim)
        ;; Se a lista é de um número maior para um menor ;;
```

```
        (if (eql ini fim)
            (cons fim nil)
            (cons ini (range (- ini 1) fim)))

        ;; Se a lista é de um número menor para um maior ;;
        (if (eql ini fim)
            (cons fim nil)
            (cons ini (range (+ ini 1) fim)))))
```

P23 (**) Trích xuất một số yếu tố được chọn ngẫu nhiên từ danh sách.

* (rnd-select '(abcdefgh) 3) (EDA)

```
(load "p03.lisp")
(load "p20.lisp")
(defun rnd-select (org-list num &optional (selected 0))
    (if (eql num selected)
        nil
        (let ((rand-pos (+ (random (length org-list)) 1)))
           (cons (element-at org-list rand-pos) (rnd-select (remove-at org-list rand-
pos) num (+ selected 1))))))
```

Xổ số P24 (*): Vẽ N số ngẫu nhiên khác nhau từ tập 1..M.

* (lotto-select 6 49) (23 1 17 33 21 37)

```
(load "p22.lisp")
(load "p23.lisp")
(defun lotto-select (num-elem max-elem)
    (rnd-select (range 1 max-elem) num-elem))
```

P25 (*) Tạo hoán vị ngẫu nhiên các yếu tố của danh sách.

* (rnd-permu '(abcdef)) (BADCEF)

```
(load "p20.lisp")
(load "p23.lisp")
(defun rnd-permu (org-list)
  (if (null org-list)
      ()
    (let ((rand-pos (+ (random (length org-list) 1))))
      (cons (element-at org-list rand-pos)
            (rnd-permu (org-list)))
```

P26 (**) Tạo kết hợp K đối tượng riêng biệt được chọn từ các thànphần Ncủadanh sách

* (kết hợp 3 '(abcdef)) ((ABC) (ABD) (ABE) ...)

```
(load "p17.lisp")
 (defun combination (n-elem org-list)
    (let ((num-elem (- n-elem 1)))
        (if (or (> num-elem (length org-list)) (< n-elem 2))
            nil
            (let ( (elem-list (split num-elem org-list)) (rest-list (split-after num-
elem org-list)))
                (append (combination-lista-elem elem-list rest-list) (combination n-
elem (cdr org-list)))))))
(defun combination-lista-elem (elem-list org-list)
    (if (eql org-list nil)
        nil
        (append (list (append elem-list (list (car org-list)))) (combination-lista-elem
elem-list (cdr org-list)))))
```

P27 (**) Nhóm các thành phần của tập hợp thành các tập hợp rời rạc.

a) Có bao nhiêu cách để một nhóm 9 người làm việc trong 3 nhóm nhỏ khác nhau gồm 2, 3 và 4 người? Viết một hàm tạo ra tất cả các khả năng và trả về chúng trong một danh sách.

b) Tổng quát hóa vị ngữ trên theo cách mà chúng ta có thể chỉ định một danh sách các kích cỡ nhóm và vị ngữ sẽ trả về một danh sách các nhóm.

```
(load "p07.lisp")
(load "p26.lisp")
(defun remove-lista (orig-list elem-list)
```

```
    (if (eql elem-list nil)
        orig-list
        (remove-lista (remove (car elem-list) orig-list) (cdr elem-list))))
(defun group-n (num-elem res-parc orig-list)
    (combination-lista-elem res-parc (combination num-elem (remove-lista orig-list
(flatten res-parc)))))

(defun group-n-list (num-elem res-parc-list orig-list)
    (if (eql res-parc-list nil)
        nil
        (let ((res-parc (car res-parc-list)) (resto (cdr res-parc-list)))
            (append (group-n num-elem res-parc orig-list) (group-n-list num-elem resto
orig-list)))))
(defun group3 (orig)
    (group-n-list 5 (group-n-list 2 (group-n 2 nil orig) orig) orig))
(defun group (orig-list elem-list)
    (if (eql (cdr elem-list) nil)
        (group-n (car elem-list) nil orig-list)
        (let ((elem-list (reverse elem-list)))
            (group-n-list (car elem-list) (group orig-list (reverse (cdr elem-list)))
orig-list))))
```

P28 (**) Sắp xếp danh sách danh sách theo độ dài của danh sách phụ

a). Ví dụ: danh sách ngắn trước, danh sách dài hơn sau hoặc ngược lại.

* (lsort '((abc) (de) (fgh) (de) (ijkl) (mn) (o))) (()) IJKL))

```
(defun lsort (orig-list)
    (sort orig-list #'> :key #'length))
 (defun lfmark (rest-list &optional (orig-list rest-list))
    (if (eql rest-list nil)
        nil
        (cons (list (lfreq orig-list (length (car rest-list))) (car rest-list)) (lfmark
(cdr rest-list) orig-list))))
 (defun lfunmark (orig-list)
    (if (eql orig-list nil)
        nil
        (cons (second (car orig-list)) (lfunmark (cdr orig-list)))))
 (defun lfreq (orig-list e-length)
    (if (eql orig-list nil)
        0
        (if (eql (length (car orig-list)) e-length)
            (+ (lfreq (cdr orig-list) e-length) 1)
            (lfreq (cdr orig-list) e-length))))
 (defun lfsort (orig-list)
    (let ((marked-list (lfmark orig-list)))
        (lfunmark (sort marked-list #'> :key #'(lambda (x) (car x))))))
```