INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO,
CÂMPUS BIRIGUI - SP
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

ISADORA DISPOSTI BUENO DOS SANTOS

EXERCÍCIOS – PROLOG

BIRIGUI - SP
2023

5.1. Defina o predicado último(L,U), que determina o último item U numa lista L. Por exemplo, último([a,b,c],U), resulta em U=c.

```
último([X], X).
último([_|Resto], U) :- último(Resto, U).
```

```
?- último([terra, sol, lua],U).
U = lua ;
```

5.2. Defina o predicado tam(L,N), que determina o número de itens N existente numa lista L. Por exemplo, tam([a,b,c],N), resulta em N=3.

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

```
?- tam([a,b,c,d,e,f,g],N).
N = 7.
```

5.3. Defina o predicado soma(L,S) que calcula a soma S dos itens da lista L. Por exemplo, soma([4,9,1],S) resulta em S=14.

```
soma([], 0).
soma([Cabeca|Cauda], S) :-
    soma(Cauda, S1),
    S is Cabeca + S1.
```

```
?- soma([4,9,1,5,1],S).
S = 20.
```

5.4. Defina o predicado máx(L,M) que determina o item máximo M na lista L. Por exemplo, máx[4,9,1],M) resulta em M=9.

```
máx([X], X).
máx([Cabeca|Cauda], M) :-
    máx(Cauda, M1),
    (Cabeca > M1 ->
        M = Cabeca;
        M = M1
    ).
```

```
?- máx([4,9,1, 15, 20, 2, 3],M).
M = 20 ;
false.
```

5.5. Usando o predicado anexa, defina o predicado inv(L,R) que inverte a lista L. Por exemplo, inv([b, c, a], R) resulta em R=[a,c,b].

```
anexa([], B, B).
anexa([X|A], B, [X|C]) :-
    anexa(A, B, C).

inv([], []).
inv([Cabeca|Cauda], R) :-
    inv(Cauda, R1),
    anexa(R1, [Cabeca], R).
```

```
?- inv([b, c, a,d, q], R).
R = [q, d, a, c, b].
```

5.6. Usando o predicado inv, defina o predicado sim(L) que verifica se uma lista é simétrica. Por exemplo, sim([a,r,a,r,a]) resulta em yes.

```
anexa([], B, B).
anexa([X|A], B, [X|C]) :-
    anexa(A, B, C).

inv([], []).
inv([Cabeca|Cauda], R) :-
    inv(Cauda, R1),
    anexa(R1, [Cabeca], R).

sim(L) :- inv(L, L).
```

```
?- sim([a,r,a,r,a]).
true.
```

5.7. Usando a tabela d(0,zero), d(1,um), ..., d(9,nove), defina o predicado txt(D,P) que converte uma lista de dígitos numa lista de palavras. Por exemplo, txt([7,2,1],P) resulta em P=[sete,dois,um].

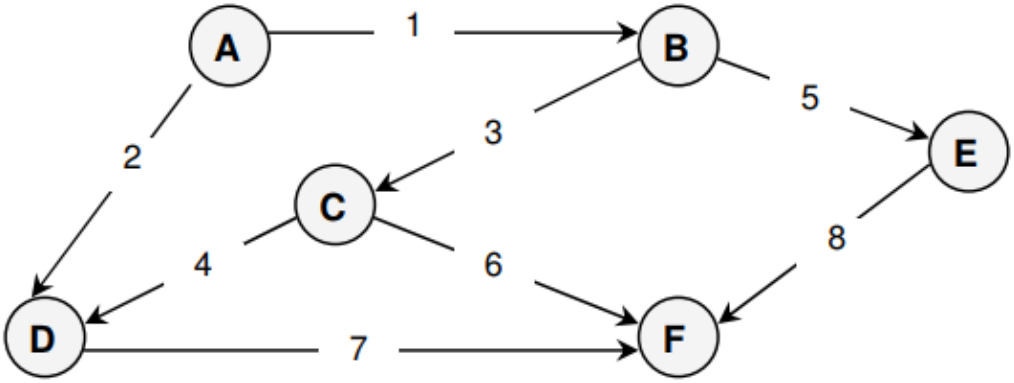
```
d(0, zero).
d(1, um).
d(2, dois).
d(3, tres).
d(4, quatro).
d(5, cinco).
d(6, seis).
d(7, sete).
d(8, oito).
d(9, nove).

txt([], []).
```

```
txt([CabecaDigito|CaudaDigito], [CabecaPalavra|CaudaPalavra]) :-
    d(CabecaDigito, CabecaPalavra),
    txt(CaudaDigito, CaudaPalavra).
```

```
?- txt([7,2,1],P).
P = [sete, dois, um].
```

5.8. O grafo a seguir representa um mapa, cujas cidades são representadas por letras e cujas estradas são representadas por números.



a) Usando o predicado estrada(Número,Origem,Destino), crie um programa para representar esse mapa.

```
estrada(1,a,b).
estrada(2,a,d).
estrada(3,b,c).
estrada(5,b,e).
estrada(4,c,d).
estrada(6,c,f).
estrada(7,d,f).
estrada(8,e,f).
```

b) Defina o predicado rota(A,B,R), que determina uma rota R (lista de estradas) que leva da cidade A até a cidade B.

```
rota(A, B, R) :- rota_aux(A, B, [], R).
rota_aux(A, B, _, [N]) :-
    estrada(N, A, B).
rota_aux(A, B, Visitadas, [N|R]) :-
    estrada(N, A, C),
    C \= B,
    \+ member(C, Visitadas),
    rota_aux(C, B, [C|Visitadas], R).
```

```
?- rota(a, f, R).
R = [1, 3, 6];
R = [1, 3, 4, 7];
R = [1, 5, 8];
R = [2, 7];
false.
```

5.9. Um retângulo é representado pela estrutura retângulo(A,B,C,D), cujos vértices são A, B, C e D, nesta ordem.

a) Defina o predicado regular(R) que resulta em yes apenas se R for um retângulo cujos lados sejam verticais e horizontais.

```
vertical(ponto(X1,_), ponto(X2,_)) :- X1 = X2.
horizontal(ponto(_,Y1), ponto(_,Y2)) :- Y1 = Y2.

regular(retangulo(A,B,C,D)) :-
   (vertical(A,B), horizontal(B,C), vertical(C,D), horizontal(D,A)) ;
   (horizontal(A,B), vertical(B,C), horizontal(C,D), vertical(D,A)).
```

```
?- regular(retangulo(ponto(1,1), ponto(1,4), ponto(5,4), ponto(5,1))).
true;
false.
```

b) Defina o predicado quadrado(R) que resulta em yes apenas se R for um retângulo cujos lados têm as mesmas medidas.

```
vertical(ponto(X1,_), ponto(X2,_)) :- X1 = X2.
horizontal(ponto(_,Y1), ponto(_,Y2)) :- Y1 = Y2.
regular(retangulo(A,B,C,D)) :-
   (vertical(A,B), horizontal(B,C), vertical(C,D), horizontal(D,A)) ;
   (horizontal(A,B), vertical(B,C), horizontal(C,D), vertical(D,A)).
distancia(ponto(X1,Y1), ponto(X2,Y2), D) :-
   D is sqrt((X2-X1)**2 + (Y2-Y1)**2).
quadrado(R) :-
   regular(R),
   R = retangulo(A,B,C,D),
   distancia(A,B,D1),
   distancia(B,C,D2),
   distancia(C,D,D3),
   distancia(D,A,D4),
   D1 = D2, D2 = D3, D3 = D4.
```

```
?- quadrado(retangulo(ponto(1,1), ponto(1,4), ponto(4,4), ponto(4,1))).
true;
false.
```

6.1. Supondo que a base de dados esteja inicialmente vazia, indique qual será o seu conteúdo após terem sido executadas as seguintes consultas.
?- asserta(metal(ferro)).

?- assertz(metal(cobre)).
?- asserta(metal(ouro)).
?- assertz(metal(zinco)).
?- retract(metal(X)).

Após as a execução das consultas teremos a seguinte base de dados:
metal(ferro).
metal(cobre).
metal(zinco).

6.2. Implemente os predicados liga, desliga e lâmpada para que eles funcionem conforme indicado pelos exemplos a seguir:

?- liga, lâmpada(X).
X = acessa
Yes
?- desliga, lâmpada(X).
X = apagada
Yes

```
:- dynamic estado/1.
estado(apagada).

liga :-
    retract(estado(apagada)),  % Remove o estado atual da lâmpada se for 'apagada'.
    asserta(estado(acessa)).   % Adiciona o estado 'acessa' para a lâmpada.

desliga :-
    retract(estado(acessa)),   % Remove o estado atual da lâmpada se for 'acessa'.
    asserta(estado(apagada)).  % Adiciona o estado 'apagada' para a lâmpada.

lâmpada(X) :-
    estado(X).
```

```
?- liga, lâmpada(X).
X = acessa.

?- desliga, lâmpada(X).
X = apagada.
```

6.3. O predicado asserta adiciona um fato à base de dados, incondicionalmente, mesmo que ele já esteja lá. Para impedir essa redundância, defina o predicado memorize, tal que ele seja semelhante a asserta, mas só adicione à base de dados fatos inéditos.

```
memorize(Fato) :-
    clause(Fato, true), !.

memorize(Fato) :-
    asserta(Fato).
```

```
?- memorize(num(1)).
true.

?- memorize(num(1)).
true.

?- listing(num).
:- dynamic num/1.

num(1).

true.

?- memorize(num(2)).
true.

?- listing(num).
:- dynamic num/1.

num(2).
num(1).

true.
```

6.4. Suponha um robô capaz de andar até um certo local e pegar ou soltar objetos. Além disso, suponha que esse robô mantém numa base de dados sua posição corrente e as respectivas posições de uma série de objetos. Implemente os predicados pos(Obj,Loc), ande(Dest), pegue(Obj) e solte(Obj), de modo que o comportamento desse robô possa ser simulado, conforme exemplificado a seguir:
?- pos(O,L).
O = robô L = garagem;
O = tv L = sala;
No
?- pegue(tv), ande(quarto), solte(tv), ande(cozinha).

anda de garagem até sala
pega tv
anda de sala até quarto
solta tv
anda de quarto até cozinha
Yes

```
?- pegue(tv), ande(quarto), solte(tv), ande(cozinha).
anda de garagem até sala
pega tv
anda de sala até quarto
solta tv
anda de quarto até cozinha
true.
```

```
:- dynamic pos/2.
pos(robô, garagem).
pos(tv, sala).
ande(Dest) :-
    retract(pos(robô, Origem)),
    write('anda de '), write(Origem), write(' até '), writeln(Dest),
    asserta(pos(robô, Dest)).
```

```
pegue(Obj) :-
    pos(robô, LocRobo),
    pos(Obj, LocObj),
    (
        LocRobo = LocObj  % Se o robô e o objeto estão no mesmo local
        ->
        retract(pos(Obj, LocObj)),
        write('pega '), writeln(Obj)
        ;
        ande(LocObj),  % Faça o robô ir até o local do objeto
        retract(pos(Obj, LocObj)),
        write('pega '), writeln(Obj)
    ).
solte(Obj) :-
    pos(robô, Loc),
    asserta(pos(Obj, Loc)),
    write('solta '), writeln(Obj).
```

6.5. Modifique o programa desenvolvido no exercício anterior de modo que, quando for solicitado ao robô pegar um objeto cuja posição é desconhecida, ele pergunte ao usuário onde está esse objeto e atualize a sua base de dados com a nova informação. Veja um exemplo:

```
?- pos(O,L).
O = robô L = cozinha;
O = tv L = quarto;
No

?- pegue(lixo), ande(rua), solte(lixo), ande(garagem).
Onde está lixo? quintal
anda de cozinha até quintal
pega lixo
anda de quintal até rua
solta lixo
anda de rua até garagem
Yes

?- pos(O,L).
O = robô L = garagem;
O = lixo L = rua;
O = tv L = quarto;
 No
```

```
?- pegue(lixo), ande(rua), solte(lixo), ande(garagem).
Onde está lixo? quintal.
anda de garagem até quintal
pega lixo
anda de quintal até rua
solta lixo
anda de rua até garagem
true.
```

```
?- pos(O,L).
O = robô,
L = garagem ;
O = lixo,
L = rua ;
O = tv,
L = sala.
```

```
:- dynamic pos/2.
pos(robô, garagem).
pos(tv, sala).
ande(Dest) :-
   retract(pos(robô, Origem)),
   write('anda de '), write(Origem), write(' até '), writeln(Dest),
   asserta(pos(robô, Dest)).
pegue(Obj) :-
   pos(robô, LocRobo),
   (
      pos(Obj, LocObj)
      ->
      (
         LocRobo = LocObj
         ->
         retract(pos(Obj, LocObj)),
         write('pega '), writeln(Obj)
         ;
         ande(LocObj),
         retract(pos(Obj, LocObj)),
         write('pega '), writeln(Obj)
      )
      ;
      write('Onde está '), write(Obj), write('? '), read(NewLoc),
      asserta(pos(Obj, NewLoc)),
      ande(NewLoc),
      retract(pos(Obj, NewLoc)),
      write('pega '), writeln(Obj)
   ).
solte(Obj) :-
   pos(robô, Loc),
   asserta(pos(Obj, Loc)),
   write('solta '), writeln(Obj).
```

6.6. Acrescente também ao programa do robô o predicado leve(Obj,Loc), que leva um objeto até um determinado local. Por exemplo:

```
?- leve(tv,sala).
anda de garagem até quarto
pega tv
```

anda de quarto até sala
solta tv
Yes

```
?- leve(tv, sala).
anda de garagem até sala
pega tv
anda de sala até sala
solta tv
true.
```

```
:- dynamic pos/2.
pos(robô, garagem).
pos(tv, sala).
ande(Dest) :-
    retract(pos(robô, Origem)),
    write('anda de '), write(Origem), write(' até '), writeln(Dest),
    asserta(pos(robô, Dest)).
pegue(Obj) :-
    pos(robô, LocRobo),
    (
        pos(Obj, LocObj)
        ->
        (
            LocRobo = LocObj
            ->
            retract(pos(Obj, LocObj)),
            write('pega '), writeln(Obj)
            ;
            ande(LocObj),
            retract(pos(Obj, LocObj)),
            write('pega '), writeln(Obj)
        )
        ;
        write('Onde está '), write(Obj), write('? '), read(NewLoc),
        asserta(pos(Obj, NewLoc)),
        ande(NewLoc),
        retract(pos(Obj, NewLoc)),
        write('pega '), writeln(Obj)
    ).
solte(Obj) :-
    pos(robô, Loc),
    asserta(pos(Obj, Loc)),
    write('solta '), writeln(Obj).
leve(Obj, Loc) :-
    pegue(Obj),
    ande(Loc),
    solte(Obj).
```