INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui	**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA** **Campus Birigui** **Bacharelado em Engenharia de Computação**

Disciplina: Inteligência Artificial	**Lista 3**
Professor: Prof. Dr. Murilo Varges da Silva	**Data:** 16/10/2023
Nome do Aluno: Henrique Akira Hiraga	**Prontuário:** BI300838X

LISTA 3 – PROLOG

1. O programa a seguir associa a cada pessoa seu esporte preferido.

```
joga(ana,volei).
joga(bia,tenis).
joga(ivo,basquete).
joga(eva,volei).
joga(leo,tenis).
```

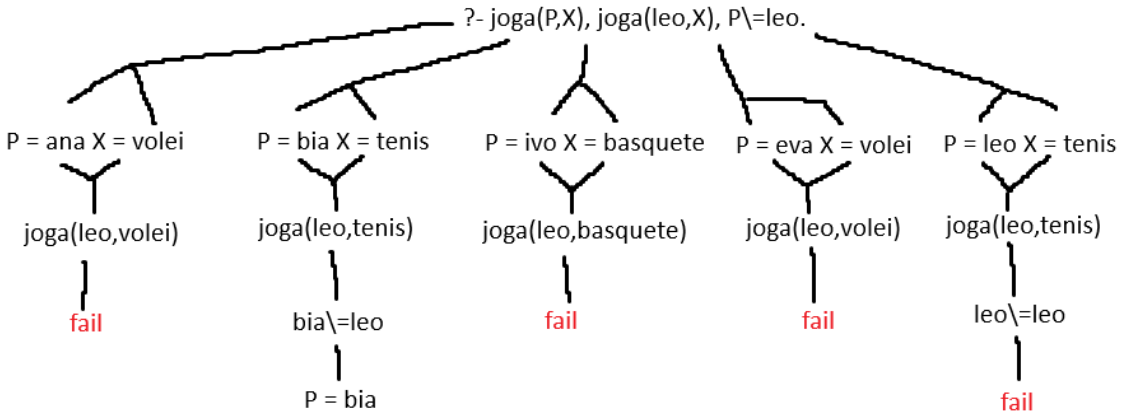
Suponha que desejamos consultar esse programa para encontrar um parceiro P para jogar com Leo. Então, podemos realizar essa consulta de duas formas:

a. `?- joga(P,X), joga(leo,X), P\=leo.`

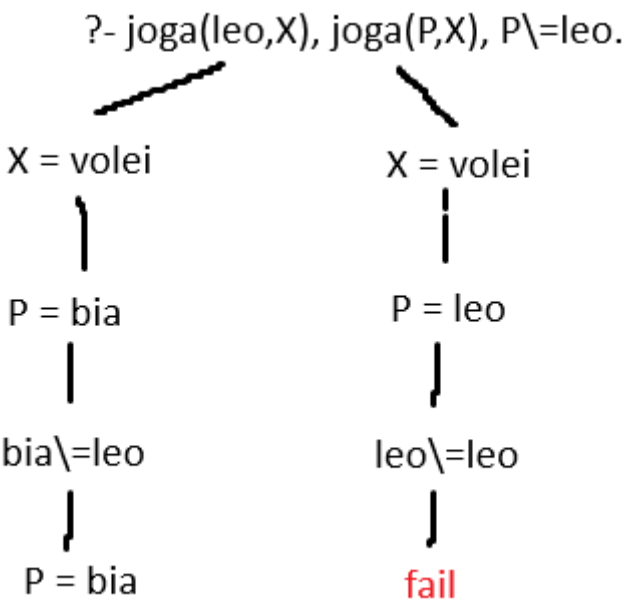
b. `?- joga(leo,X), joga(P,X), P\=leo.`

Desenhe as árvores de busca construídas pelo sistema ao respondercada uma dessas consultas. Qual consulta é mais eficiente, por quê?

Resultado da árvore de busca da pesquisa (a):



Resultado da árvore de busca da pesquisa (b):



A consulta (b) é mais eficiente porque ela começa com uma restrição específica (Leo jogando volei) e, em seguida, encontra um parceiro que atende a essa restrição. Isso reduz a quantidade de backtracking necessário. A consulta (a) começa com a variável P e, em seguida, verifica a restrição em relação a Leo, o que pode levar a mais tentativas de unificação e, portanto, ser menos eficiente.

2. O predicado num classifica números em três categorias: positivos, nulo e negativos. Esse predicado, da maneira como está definido, realiza retrocesso desnecessário. Explique por que isso acontece e, em seguida, utilize cortes para eliminar esse retrocesso.

```
num(N,positivo) :- N>0.
num(0,nulo).
num(N,negativo) :- N<0.
```

Considerando o predicado num, ao fazermos uma consulta como por exemplo num(5, C), ele irá achar a solução (C = positivo) e, em seguida fará o retrocesso para verificar se há outras soluções, o que é desnecessário para esse caso. O mesmo se aplica se entrarmos com um valor de N negativo, após o Prolog encontrar a solução (C = negativo) ele irá retornar e procurar

outras soluções.

Para solucionarmos esses retrocessos desnecessários, basta adicionar o operador de corte ‘!’, que fará com que o Prolog não realize o retrocesso a partir desse ponto. O predicado então ficará da seguinte maneira:

```
num(N,positivo) :- N > 0, !.
num(0,nulo).
num(N,negativo) :- N < 0, !.
```

A adição do corte após as condições que satisfaçam as categorias "positivo" e "negativo" garante que o Prolog não faça retrocesso nessas regras após encontrar uma solução. Assim, o retrocesso desnecessário é eliminado, tornando o predicado mais eficiente. No entanto, a regra para o caso "nulo" não tem um corte, o que permite ao Prolog continuar a procurar por mais soluções se N for igual a zero.

3. Suponha que o predicado fail não existisse em Prolog. Qual das duas definições a seguir poderia ser corretamente usada para causar falhas?

   a. `falha :- (1=1).`

   b. `falha :- (1=2).`

   A definição que ocasionará falha será a (b). Neste caso, a regra falha falhará porque a igualdade 1=2 é avaliada como falsa, o que levará à falha da regra. Isso é equivalente a usar fail para indicar uma falha, mas sem o uso direto do predicado fail.

4. Considere o programa a seguir:

```
animal(cão).
animal(canário).
animal(cobra).
animal(morcego).
animal(gaivota).
voa(canário).
voa(morcego).
voa(gaivota).
dif(X,X) :- !, fail.
```

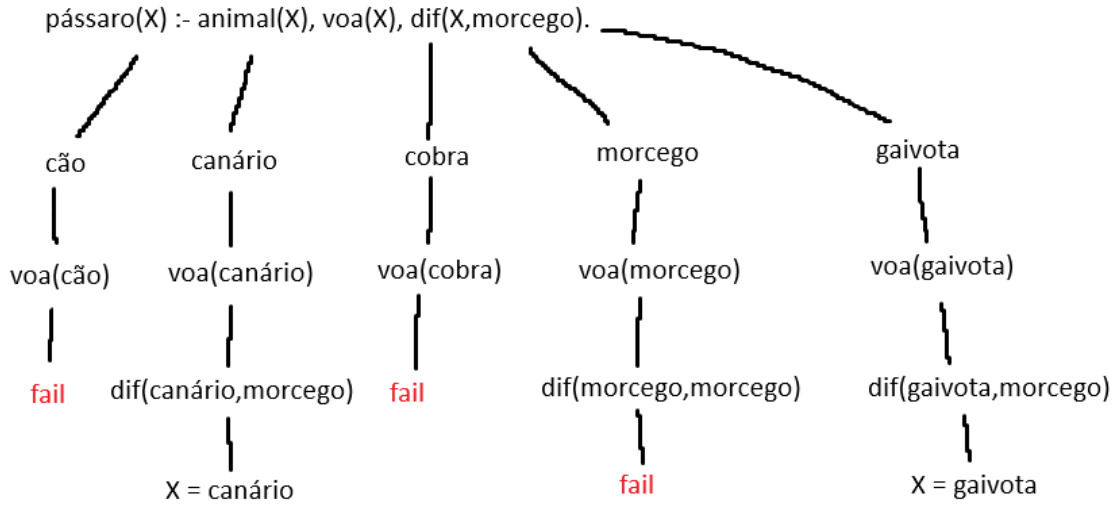
```
dif(_,_).
pássaro(X) :- animal(X), voa(X), dif(X,morcego).
```

Desenhe a árvore de busca necessária para responder a consulta

?- pássaro(X).

Em seguida, execute o programa para ver se as respostas do sistema correspondem àquelas que você encontrou.

Desenho da árvore de busca:



Execução do programa:

```
[trace]  ?- pássaro(X).
   Call: (10) pássaro(_30470) ? creep
   Call: (11) animal(_30470) ? creep
   Exit: (11) animal(cão) ? creep
   Call: (11) voa(cão) ? creep
   Fail: (11) voa(cão) ? creep
   Redo: (11) animal(_30470) ? creep
   Exit: (11) animal(canário) ? creep
   Call: (11) voa(canário) ? creep
   Exit: (11) voa(canário) ? creep
   Call: (11) dif(canário, morcego) ? creep
   Exit: (11) dif(canário, morcego) ? creep
   Exit: (10) pássaro(canário) ? creep
X = canário ;
   Redo: (11) animal(_30470) ? creep
   Exit: (11) animal(cobra) ? creep
   Call: (11) voa(cobra) ? creep
   Fail: (11) voa(cobra) ? creep
   Redo: (11) animal(_30470) ? creep
   Exit: (11) animal(morcego) ? creep
   Call: (11) voa(morcego) ? creep
   Exit: (11) voa(morcego) ? creep
   Call: (11) dif(morcego, morcego) ? creep
   Call: (12) fail ? creep
   Fail: (12) fail ? creep
   Fail: (11) dif(morcego, morcego) ? creep
   Redo: (11) animal(_30470) ? creep
   Exit: (11) animal(gaivota) ? creep
   Call: (11) voa(gaivota) ? creep
   Exit: (11) voa(gaivota) ? creep
   Call: (11) dif(gaivota, morcego) ? creep
   Exit: (11) dif(gaivota, morcego) ? creep
   Exit: (10) pássaro(gaivota) ? creep
X = gaivota.
```

5. Defina um predicado recursivo para calcular o produto de dois números naturais usando apenas soma e subtração.

Predicado recursivo:

```
produto(0, _, 0).  % O produto de 0 com qualquer número é 0.
produto(X, Y, Resultado) :-
    X > 0,  % Certifique-se de que X seja um número natural.
    X1 is X - 1,  % Reduza X em 1.
    produto(X1, Y, ResultadoParcial),  % Chamada recursiva.
    Resultado is ResultadoParcial + Y.  % Adicione Y ao resultado parcial.
```

Resultado:

```
?- produto(5,2,X).
X = 10 ;
false.
```

6. Defina um predicado recursivo exibir um número natural em binário.

Predicado recursivo:

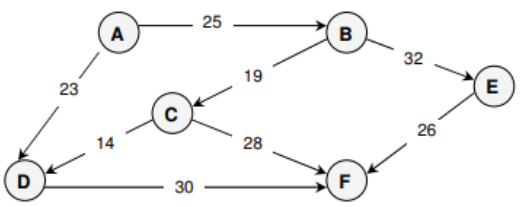
```
decimal_para_binario(0, "0").  % Caso base: 0 em decimal é "0" em binário.
decimal_para_binario(1, "1").  % Caso base: 1 em decimal é "1" em binário.

decimal_para_binario(N, Binario) :-
    N > 1,
    Resto is N mod 2,  % Calcula o resto da divisão por 2.
    Quociente is N // 2,  % Calcula o quociente da divisão por 2.
    decimal_para_binario(Quociente, BinarioAnterior),  % Chamada recursiva.
    atom_concat(BinarioAnterior, Resto, Binario).  % Concatena o resto ao resultado anterior.
```

Resultado:

```
?- decimal_para_binario(10,X).
X = '1010' ;
false.
```

7. O grafo a seguir representa um mapa, cujas cidades são representadas por letras e cujas estradas (de sentido único) são representados por números, que indicam sua extensão em km.



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

```
%estrada(Origem, Destino, Km)
estrada(a, b, 25).
estrada(a, d, 23).
estrada(b, c, 19).
estrada(b, e, 32).
estrada(c, d, 14).
estrada(c, f, 28).
estrada(d, f, 30).
estrada(e, f, 26).
```

```
?- estrada(a, Destino, Distância).
Destino = b,
Distância = 25 ;
Destino = d,
Distância = 23.
```

b. Defina a relação transitiva dist(A,B,D), que determina a distância D entre duas cidades A e B.

```
%estrada(Origem, Destino, Km)
estrada(a, b, 25).
estrada(a, d, 23).
estrada(b, c, 19).
estrada(b, e, 32).
estrada(c, d, 14).
estrada(c, f, 28).
estrada(d, f, 30).
estrada(e, f, 26).

dist(A, B, D) :- estrada(A, B, D).
dist(A, B, D) :- estrada(A, C, D1), dist(C, B, D2), D is D1 + D2.
```

```
?- dist(a,f,D).
D = 72 ;
D = 88 ;
D = 83 ;
D = 53 ;
false.
```