

Inteligência Artificial

Programação em Lógica III

Licenciatura em Engenharia Informática
2024/2025

_recursividade

- Recursão/Recursividade: função que se chama a si própria para resolver um determinado problema
- O uso da recursividade é muito comum na linguagem PROLOG
- Na implementação de um predicado recursivo devemos ter em atenção que deverá haver sempre uma alternativa para finalizar as chamadas recursivas (condição/condições de paragem)
 - Por uma regra, ou facto, que não efectua essa chamada
 - Por uma das alternativas de uma disjunção

_recursividade

```
contagem_decrescente(0):-write(0),!.  
contagem_decrescente(X):- X<0, fail.  
contagem_decrescente(X):- X>0, write(X), nl, X1 is X - 1,  
    contagem_decrescente(X1).
```

- Os primeiros dois casos são casos de paragem ou casos em que a função não está definida
- O último caso, é o caso base

_recursividade

- **Exercício:** como reescrever o código abaixo para que os números sejam escritos por ordem crescente?

```
contagem_decrescente(0):-write(0),!.  
contagem_decrescente(X):- X<0, fail.  
contagem_decrescente(X):- X>0, write(X), nl, X1 is X - 1,  
    contagem_decrescente(X1).
```

_recursividade

- **Exercício:** como reescrever para que os números sejam escritos por ordem crescente?

```
contagem_decrescente(0):-write(0),!.  
contagem_decrescente(X):- X<0, fail.  
contagem_decrescente(X):- X>0, write(X), nl, X1 is X - 1,  
    contagem_decrescente(X1).
```

```
contagem_crescente(0):-write(0),!.  
contagem_crescente(X):- X<0, fail.  
contagem_crescente(X):- X>0, X1 is X - 1,  
    contagem_crescente(X1), nl, write(X).
```

_recursividade

$$n! = n \times (n - 1) \times (n - 2) \times (n - 3) \times \dots \times 3 \times 2 \times 1$$

$$= n \times (n - 1)!$$

```
factorial(0,1):-!.
```

```
factorial(N,F):-N1 is N-1,factorial(N1,F1),F is N*F1.
```

Vejamos o que acontece quando se efetua a chamada ?-factorial(3,F).

sucesso (F ← 6)

```
factorial(0,1) falha
```

```
factorial(3,F):- N1 <- 2, factorial(2,F1), F is 3*2.
```

sucesso (F ← 6)

```
factorial(0,1) falha
```

```
factorial(2,F):-N1 <- 2-1, factorial(1,F1), F is 2*1.
```

sucesso (F ← 2)

```
factorial(0,1) falha
```

```
factorial(1,F):-N1 <- 1-1,factorial(0,F1), F is 1*1.
```

sucesso (F ← 1)

```
factorial(0,1):-!.
```

sucesso (F ← 1)

_listas

- Em PROLOG as listas podem ser:
 - Não vazias, tendo uma cabeça (1º elemento da lista) e uma cauda (lista com os restantes elementos)
 - Vazias, quando não têm nenhum elemento
- As listas podem ser representadas
 - Pela enumeração dos seus elementos separados por vírgulas e envolvidos por [e] (por exemplo [a,b,c,d])
 - Pela **notação cabeça-cauda** separadas pelo | e envolvidas por [e] (por exemplo [H|T])
- Em PROLOG os elementos das listas **não têm de ser do mesmo tipo** (por exemplo, [a,2,abc,[x,1,zzz]])

_listas

- [] é a lista vazia
- [a] é uma lista com 1 elemento (a)
- [X] é uma lista com 1 elemento (a variável X)
- [b,Y] é uma lista com 2 elementos (b e a variável Y)
- [X,Y,Z] é uma lista com 3 elementos (as variáveis X,Y e Z)
- [H|T] é uma lista com cabeça H e cauda T

Veamos algumas questões PROLOG:

```
?- [H|T]=[a,b,c,d] .  
H = a ,  
T = [b,c,d]
```

```
?- [H|T]=[a,b,X] .  
H = a ,  
T = [b,X] ,  
X = _
```

```
?- [H|T]=[a] .  
H = a ,  
T = []
```

```
?- [H|T]=[[a,b],3,[d,e]] .  
H = [a,b] ,  
T = [3,[d,e]]
```

```
?- [H]=[] .  
no
```

```
?- [H|T]=[] .  
no
```


_listas

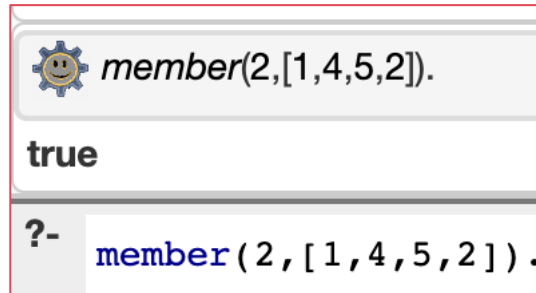
```
percorreRec([]):-write('Chegou ao fim da lista').  
percorreRec([H|T]):-write('A percorrer elemento '),  
                    write(H),  
                    write('. Cauda da lista:'),  
                    write(T),  
                    write('\n'),  
                    percorreRec(T).
```

```
?- percorreRec([1,5,3,5,6]).  
A percorrer elemento 1. Cauda da lista:[5,3,5,6]  
A percorrer elemento 5. Cauda da lista:[3,5,6]  
A percorrer elemento 3. Cauda da lista:[5,6]  
A percorrer elemento 5. Cauda da lista:[6]  
A percorrer elemento 6. Cauda da lista:[]  
Chegou ao fim da lista
```

_alguns predicados úteis

- member/2
- append/3
- reverse/2
- delete/3

- Outros: <https://www.swi-prolog.org/pldoc/man?section=lists>



The screenshot shows a Prolog interpreter window with a light gray background. At the top, there is a gear icon followed by the query `member(2,[1,4,5,2]).`. Below this, the response `true` is displayed. At the bottom, there is a prompt `?-` followed by the query `member(2,[1,4,5,2]).` which is partially highlighted in blue.

_soluções múltiplas

`findall (X, Q, L)`

- L é a lista dos X que atendem a Q, se não houver solução `findall` tem sucesso com L=[]

`setof (X, Q, L)`

- L é a lista dos X que atendem a Q, L vem ordenada, elementos repetidos são eliminados, se não houver solução `setof` falha

`bagof (X, Q, L)`


- L é a lista dos X que atendem a Q, se não houver solução `bagof` falha

```
fica(porto,portugal).
fica(pocinho, portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).
```

```
passa(douro,porto).
passa(douro,pocinho).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).
```

```
rio_portugues(X):-passa(X, C), fica(C, portugal).
```

_soluções múltiplas

 `findall(R, rio_portugues(R), L).`


`L = [douro, douro, tejo, minho]`

?- `findall(R, rio_portugues(R), L).`

 `setof(R, rio_portugues(R), L).`

`L = [douro, minho, tejo]`

?- `setof(R, rio_portugues(R), L).`

 `bagof(R, rio_portugues(R), L).`

`L = [douro, douro, tejo, minho]`

?- `bagof(R, rio_portugues(R), L).`

```
fica(porto,portugal).
fica(pocinho, portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).
```

```
passa(douro,porto).
passa(douro,pocinho).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).
```

```
rio_portugues(X):-passa(X, C), fica(C, portugal).
rio_pais(X, P):-passa(X, C), fica(C, P).
```

_soluções múltiplas



```
findall(R, rio_pais(R, alemanha), L).
```

```
L = []
```

```
?-
```

```
findall(R, rio_pais(R, alemanha), L).
```



```
setof(R, rio_pais(R, alemanha), L).
```

```
false
```

```
?-
```

```
setof(R, rio_pais(R, alemanha), L).
```



```
bagof(R, rio_pais(R, alemanha), L).
```

```
false
```

```
?-
```

```
bagof(R, rio_pais(R, alemanha), L).
```

_soluções múltiplas

```
fica(porto,portugal).
fica(pocinho, portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).
```

```
passa(douro,porto).
passa(douro,pocinho).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).
```

```
rio_portugues(X):-passa(X, C), fica(C, portugal).
rio_pais(X, P):-passa(X, C), fica(C, P).
```

% calcula o tamanho de uma lista (também existe a função length em Prolog)

```
tam_lista([], 0).
tam_lista(_|T, R):-tam_lista(T, R1), R is R1 + 1.
```

% conta quantos rios passam num dado país


```
conta_rios(P, C):-findall(R, rio_pais(R,P),L), tam_lista(L,C).
```

% conta quantos rios passam num dado país

```
conta_rios(P, C):-bagof(R, rio_pais(R,P),L), tam_lista(L,C).
```


% conta quantos rios passam num dado país

```
conta_rios(P, C):-setof(R, rio_pais(R,P),L), tam_lista(L,C).
```

 conta_rios(portugal, C).

C = 4

?- conta_rios(portugal, C).

 conta_rios(portugal, C).

C = 4

?- conta_rios(portugal, C).

 conta_rios(portugal, C).

C = 3

?- conta_rios(portugal, C).

_soluções múltiplas

```
fica(porto,portugal).
fica(pocinho, portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).
```

```
passa(douro,porto).
passa(douro,pocinho).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).
```

```
rio_portugues(X):-passa(X, C), fica(C, portugal).
rio_pais(X, P):-passa(X, C), fica(C, P).
```

% calcula o tamanho de uma lista (também existe a função length em Prolog)

```
tam_lista([], 0).
tam_lista(_|T, R):-tam_lista(T, R1), R is R1 + 1.
```

% conta quantos rios passam num dado país

```
conta_rios(P, C):-findall(R, rio_pais(R,P),L), tam_lista(L,C).
```

% conta quantos rios passam num dado país

```
conta_rios(P, C):-bagof(R, rio_pais(R,P),L), tam_lista(L,C).
```

% conta quantos rios passam num dado país

```
conta_rios(P, C):-setof(R, rio_pais(R,P),L), tam_lista(L,C).
```

 conta_rios(portugal, C).

C = 4

?- conta_rios(portugal, C).

 conta_rios(portugal, C).

C = 4

?- conta_rios(portugal, C).

 conta_rios(portugal, C).

C = 3

?- conta_rios(portugal, C).

_soluções múltiplas

```
fica(porto,portugal).
fica(pocinho, portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).
```

```
passa(douro,porto).
passa(douro,pocinho).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).
```

```
rio_portugues(X):-passa(X, C), fica(C, portugal).
rio_pais(X, P):-passa(X, C), fica(C, P).
```

% calcula o tamanho de uma lista (também existe a função length em Prolog)

```
tam_lista([], 0).
tam_lista(_|T, R):-tam_lista(T, R1), R is R1 + 1.
```

% conta quantos rios passam num dado país

```
conta_rios(P, C):-findall(R, rio_pais(R,P),L), tam_lista(L,C).
```

% conta quantos rios passam num dado país

```
conta_rios(P, C):-bagof(R, rio_pais(R,P),L), tam_lista(L,C).
```

% conta quantos rios passam num dado país

```
conta_rios(P, C):-setof(R, rio_pais(R,P),L), tam_lista(L,C).
```

 conta_rios(alemanha, C).

C = 0

?- conta_rios(alemanha, C).

 conta_rios(alemanha, C).

false

?- conta_rios(alemanha, C).

 conta_rios(alemanha, C).

false

?- conta_rios(alemanha, C).

_soluções múltiplas

```
fica(porto,portugal).
fica(pocinho, portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).
```

```
passa(douro,porto).
passa(douro,pocinho).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).
```

```
rio_portugues(X):-passa(X, C), fica(C, portugal).
rio_pais(X, P):-passa(X, C), fica(C, P).
```

```
% calcula o tamanho de uma lista (também existe a função length em Prolog)
tam_lista([], 0).
tam_lista(_|T, R):-tam_lista(T, R1), R is R1 + 1.
```

```
% conta quantos rios passam num dado país
conta_rios(P, C):-findall(R, rio_pais(R,P),L), tam_lista(L,C).
```

```
% conta quantos rios passam num dado país
conta_rios(P, C):-bagof(R, rio_pais(R,P),L), tam_lista(L,C).
```

```
% conta quantos rios passam num dado país
conta_rios(P, C):-setof(R, rio_pais(R,P),L), tam_lista(L,C).
```

```
conta_rios(alemanha, C).
```

C = 0

```
?- conta_rios(alemanha, C).
```

```
conta_rios(alemanha, C).
```

false

```
?- conta_rios(alemanha, C).
```

```
conta_rios(alemanha, C).
```

false

```
?- conta_rios(alemanha, C).
```

_soluções múltiplas

```
fica(porto,portugal).
fica(pocinho, portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).
```

```
passa(douro,porto).
passa(douro,pocinho).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).
```

```
rio_portugues(X):-passa(X, C), fica(C, portugal).
rio_pais(X, P):-passa(X, C), fica(C, P).
```

% calcula o tamanho de uma lista (também existe a função length em Prolog)

```
tam_lista([], 0).
tam_lista(_|T, R):-tam_lista(T, R1), R is R1 + 1.
```

% conta quantos rios passam num dado país

```
conta_rios(P, C):-setof(R, rio_pais(R,P),L), tam_lista(L,C), !.
conta_rios(_, 0).
```



```
conta_rios(portugal, C).
```

C = 3

?-

```
conta_rios(portugal, C).
```



```
conta_rios(alemanha, C).
```

C = 0

?-

```
conta_rios(alemanha, C).
```

_soluções múltiplas

- Considere Ex2. da FP 2
 - `matou/2` - `matou(assassino, vítima).`
 - `irmao/2`
 - `casados/2` - `casados(homem, mulher)`
- O que se pretende com cada uma das seguintes queries?
 - `findall(X, (homem(X), matou(_,X)), L).`
 - `findall(X, (homem(X), matou(X,_)), L).`
 - `setof(X, (irmao("Tyrion",X);irmao(X, "Tyrion")), L).`
 - `findall(_, casados(_,_), L), length(L,R).`

Inteligência Artificial

Programação em Lógica III

Licenciatura em Engenharia Informática
2024/2025