GSI010 - Programação Lógica
Listas

Recursão de cauda

Definição

uma técnica de recursão que faz menos uso de memória durante o processo de empilhamento, o que a torna mais rápida que a recursão comum.

Recursão de cauda

Definição

uma técnica de recursão que faz menos uso de memória durante o processo de empilhamento, o que a torna mais rápida que a recursão comum.

Exemplo: Qual é o comprimento de uma lista?

- Base: [] tem tamanho zero.
- Lista não vazia: tem o tamanho da sublista sem a cabeça, i.e., cauda, mais 1 (um)

Recursão comum

```
tam([], 0).
```

Recursão comum

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

Recursão comum

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

```
?- tam([a,b,c,d,e,[a,x],t],X).
X=7;
true.
```

Recursão comum

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

```
?- tam([a,b,c,d,e,[a,x],t],X).
X=7;
true.
```

Problema: é obrigado a ir até o caso base e depois retornar na recursão para fazer a soma.

Árvore recursão comum: tam

```
?- tam([a,b,c], N).
```

Árvore recursão comum: tam

```
?- tam([a,b,c], N).
```

```
false.
```

```
?- tam([b,c], N1), N1 is N+1.
```

Árvore recursão comum: tam

```
?- tam([a,b,c], N).

false.

?- tam([b,c], N1), N1 is N+1.

false.

?- tam([c],N2), N2 is N1+1, N1 is N+1
```

Árvore recursão comum: tam

```
?- tam([a,b,c], N).

false.

?- tam([b,c], N1), N1 is N+1.

false.

?- tam([c],N2), N2 is N1+1, N1 is N+1

false.

?- tam([],N3), N3 is N2+1, N2 is N1+1, N1 is N+1.
```

Árvore recursão comum: tam

```
?- tam([a,b,c], N).

false.

?- tam([b,c], N1), N1 is N+1.

false.

?- tam([c],N2), N2 is N1+1, N1 is N+1

false.

?- tam([],N3), N3 is N2+1, N2 is N1+1, N1 is N+1.

N3 = 0, N2 = 1, N1 = 2, N = 3
```

Recursão de Cauda

Características

- uso de acumulador para fazer a conta enquanto faz a recursão
- uso de predicado "fachada", sem acumulador

```prolog
tamAcum([], Acum, Tam) :- Tam = Acum.
tamAcum([_|L], Acum, Tam) :- Acum1 is Acum + 1,
    tamAcum(L, Acum1, Tam).
tam(L, Tam) :- tamAcum(L, 0, Tam).
```

Recursão de Cauda

Características

- uso de acumulador para fazer a conta enquanto faz a recursão
- uso de predicado "fachada", sem acumulador

```prolog
tamAcum([], Acum, Tam) :- Tam = Acum.
tamAcum([_|L], Acum, Tam) :- Acum1 is Acum + 1,
    tamAcum(L, Acum1, Tam).
tam(L, Tam) :- tamAcum(L, 0, Tam).
```

Vantagens

- não usa pilha de recursão e economiza memória
- Prolog substitui automaticamente recursão de cauda por laço de iteração "for" ou "while"
- O custo $O(n)$ de espaço da pilha de recursão torna-se $O(1)$

Comparação

Recursão comum

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

Recursão de Cauda

```
tamAcum([], Acum, Tam) :- Tam = Acum.
tamAcum([_|L], Acum, Tam) :- Acum1 is Acum + 1, tamAcum(L, Acum1, Tam).
tam(L, Tam) :- tamAcum(L, 0, Tam).
```

Árvore de recursão com cauda

```
?- tamAcum([a,b,c],0,Tam).
```

```
1 tamAcum([], Acum, Tam) :- Tam = Acum.
2 tamAcum([_|L], Acum, Tam) :- Acum1 is Acum + 1, tamAcum(L, Acum1, Tam).
3 tam(L, Tam) :- tamAcum(L, 0, Tam).
```

Árvore de recursão com cauda

`?- tamAcum([a,b,c],0,Tam).`

`false.`

`?- tamAcum([b,c],1,Tam).`

```
tamAcum([], Acum, Tam) :- Tam = Acum.
tamAcum([_|L], Acum, Tam) :- Acum1 is Acum + 1, tamAcum(L, Acum1, Tam).
tam(L, Tam) :- tamAcum(L, 0, Tam).
```

Árvore de recursão com cauda

```
?- tamAcum([a,b,c],0,Tam).
```

```
false.
```

```
?- tamAcum([b,c],1,Tam).
```

```
false.
```

```
?- tamAcum([c],2,Tam).
```

```
1 tamAcum([], Acum, Tam)
      :- Tam = Acum.
2 tamAcum([_|L], Acum,
      Tam) :- Acum1 is
      Acum + 1, tamAcum(
      L, Acum1, Tam).
3 tam(L, Tam) :- tamAcum
      (L, 0, Tam).
```

Árvore de recursão com cauda

```
?- tamAcum([a,b,c],0,Tam).
```

```
false.
```

```
?- tamAcum([b,c],1,Tam).
```

```
false.
```

```
?- tamAcum([c],2,Tam).
```

```
false.
```

```
?- tamAcum([],3,Tam).
```

```
tamAcum([], Acum, Tam)
    :- Tam = Acum.
tamAcum([_|L], Acum,
    Tam) :- Acum1 is
    Acum + 1, tamAcum(
    L, Acum1, Tam).
tam(L, Tam) :- tamAcum
    (L, 0, Tam).
```

Árvore de recursão com cauda

```
?- tamAcum([a,b,c],0,Tam).
```

`false.`

```
?- tamAcum([b,c],1,Tam).
```

`false.`

```
?- tamAcum([c],2,Tam).
```

`false.`

```
?- tamAcum([],3,Tam).
```

`false.` `Tam = 3`

```
tamAcum([], Acum, Tam)
    :- Tam = Acum.
tamAcum([_|L], Acum,
    Tam) :- Acum1 is
    Acum + 1, tamAcum(
    L, Acum1, Tam).
tam(L, Tam) :- tamAcum
    (L, 0, Tam).
```

Outro exemplo: soma de lista de números

Recursão comum

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

Outro exemplo: soma de lista de números

Recursão comum

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

Recursão de cauda

```
soma(L, S) :- soma(L, 0, S).
```

Outro exemplo: soma de lista de números

Recursão comum

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

Recursão de cauda

```
soma(L, S) :- soma(L, 0, S).
soma([X], Acum, S) :- S is Acum + X.
```

Outro exemplo: soma de lista de números

Recursão comum

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

Recursão de cauda

```
soma(L, S) :- soma(L, 0, S).
soma([X], Acum, S) :- S is Acum + X.
soma([X|C], Acum, S) :- Acum2 is Acum + X,
                        soma(C, Acum2, S).
```

Exercício: produto interno

Produto interno

$(2,3,4) * (4,2,1) = 2 * 4 + 3 * 2 + 4 * 1$

Recursão comum

```
prodInt([],[],0).
prodInt([X1|L1], [Y1|L2], R) :- prodInt(L1, L2, R1),
                                R is R1 + X1*Y1.
```

Exercício: produto interno

Produto interno

$(2, 3, 4) * (4, 2, 1) = 2 * 4 + 3 * 2 + 4 * 1$

Recursão comum

```
prodInt([],[],0).
prodInt([X1|L1], [Y1|L2], R) :- prodInt(L1, L2, R1),
                                R is R1 + X1*Y1.
```

Fazer

versão com recursão de cauda

Referências

- Luis, A. M. Palazzo, Introdução à programação prolog, Educat, 1997
- Slides profs. Elaine Faria, Hiran Nonato e Gabriel Coutinho - UFU
- Slides da Profa. Solange - ICMC - USP