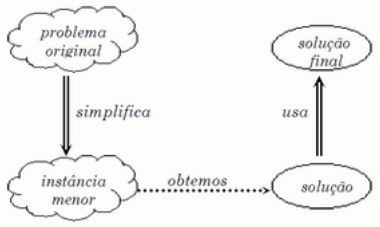
PROLOG 4

Baseado no material Prolog, de Silvio Lago.

Programação Recursiva Recursividade é fundamental em Prolog; graças ao seu uso, programas realmente práticos podem ser implementados.

4.1 Recursividade

A recursividade é um princípio que nos permite obter a solução de um problema a partir da solução de uma instância menor dele mesmo. Para aplicar esse princípio, devemos assumir como hipótese que a solução da instância menor é conhecida. Por exemplo, suponha que desejamos calcular 2^{11}. Uma instância menor desse problema é 2^{10} e, para essa instância, "sabemos"que a solução é 1024. Então, como $2 \times 2^{10} = 2^{11}$, concluímos que $2^{11} = 2 \times 1024 = 2048$.



A figura acima ilustra o princípio de recursividade. De modo geral, procedemos da seguinte maneira: simplificamos o problema original transformando-o numa instância menor; então, obtemos a solução para essa instância e a usamos para construir a solução final, correspondente ao problema original.

O que é difícil de entender, a priori, é como a solução para a instância menor é obtida. Porém, não precisamos nos preocupar com essa parte. A solução da instância menor é gerada pelo próprio mecanismo da recursividade. Sendo assim, tudo o que precisamos fazer é encontrar uma simplificação adequada para o problema em questão e descobrir como a solução obtida recursivamente pode ser usada para construir a solução final.

4.2 Predicados recursivos

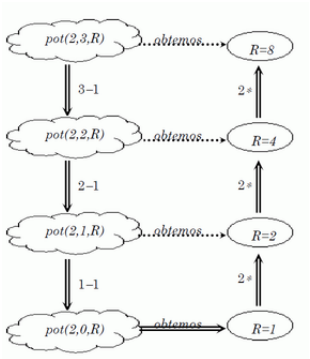
A definição de um predicado recursivo é composta por duas partes:

1. base: resolve diretamente a instância mais simples do problema.

2. passo: resolve instâncias maiores, usando o princípio de recursividade.

```
Programa 4.1: Cálculo de potência.
% pot(Base,Expoente,Potência)
pot(_,0,1).      % base
pot(B,N,P) :-    % passo
   N>0,          % condição do passo
   M is N-1,     % simplifica o problema
   pot(B,M,R),   % obtém sol. da instância menor
   P is B*R.     % constrói solução final
```

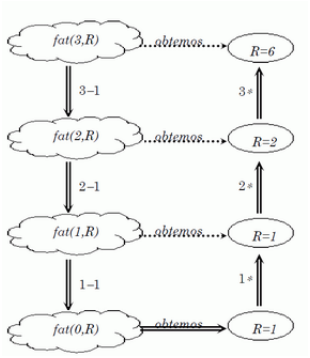
O Programa 4.1 mostra a definição de um predicado para calcular potências. A base para esse problema ocorre quando o expoente é 0, já que qualquer número elevado a 0 é igual a 1. Por outro lado, se o expoente é maior que 0, então o problema deve ser simplificado, ou seja, temos que chegar um pouco mais perto da base. A chamada recursiva com M igual a N-1 garante justamente isso. Portanto, após um número finito de passos, a base do problema é atingida e o resultado esperado é obtido. Um modo de entender o funcionamento dos predicados recursivos é desenhar o fluxo de execução.



A cada expansão, deixamos a oval em branco e rotulamos a seta que sobre com a operação que fica pendente. Quando a base é atingida, começamos a preencher as ovais, propagando os resultados de baixo para cima e efetuando as operações pendentes. O caminho seguido é aquele indicado pelas setas duplas. As setas pontilhadas representam a hipótese de recursividade.

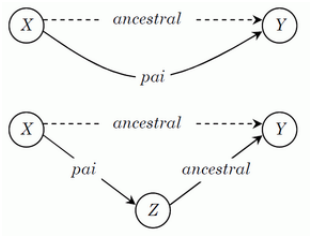
```
Programa 4.2: Cálculo de fatorial.
% fat(Número,Fatorial)
fat(0,1).     % base
fat(N,F) :-   % passo
   N>0,       % condição do passo
   M is N-1, % simplif. problema
   fat(M,R), % solução da inst. menor
   F is N*R. % constrói solução final
```

O Programa 4.2 mostra mais um exemplo de predicado recursivo e a figura a seguir mostra o fluxo de execução para a consulta ?- fat(3,R).



4.3 Relações transitivas

Se uma relação r é transitiva, então r(x,y) e r(y,z) implicam r(x,z). Um exemplo desse tipo de relação é a relação ancestral: se Adão é ancestral de Seth e Seth é ancestral de Enos, então Adão também é ancestral de Enos. Uma relação transitiva é sempre definida em termos de uma outra relação, denominada relação base. No caso da relação ancestral, a relação base é a relação pai. Assim, podemos dizer que se um indivíduo x é pai de um indivíduo y, então x é ancestral de y. Além disso, se x é pai de z e z é ancestral de y, então x também é ancestral de y. Essas regras podem ser visualizadas nos grafos de relacionamentos a seguir:

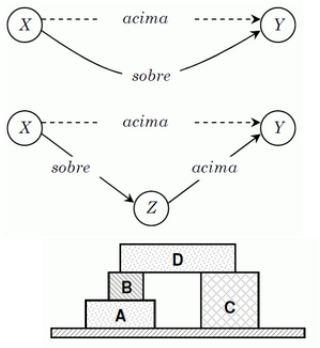



```
Programa 4.3: A relação transitiva ancestral.
pai(adão,cain).
pai(adão,abel).
pai(adão,seth).
pai(seth,enos).
ancestral(X,Y) :- pai(X,Y).
ancestral(X,Y) :- pai(X,Z), ancestral(Z,Y).
```

Veja uma consulta que poderia ser feita com o Programa 4.3:

```
?- ancestral(X,enos).
X = seth ;
X = adão
```

Outro exemplo interessante de relação transitiva é a relação acima, cuja relação base é a relação sobre. Os grafos a seguir descrevem essa relação:




```
Programa 4.4: A relação transitiva acima.
sobre(b,a).
sobre(d,b).
sobre(d,c).
acima(X,Y) :- sobre(X,Y).
acima(X,Y) :- sobre(X,Z), acima(Z,Y).
```

Veja uma consulta que poderia ser feita com o Programa 4.4:

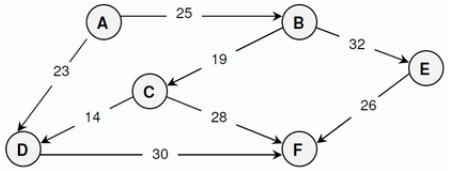
```
?- acima(X,a).
X = b ;
X = d
```

☞ Para você fazer

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

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

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