Exemplos de Programação em Prolog

Gerar e Testar é uma técnica muito usada na progamação em Prolog.

Na procura de soluções para um dado problema, um predicado **gera** uma possível solução e outro predicado **testa** se a solução cadidata verifica os requesitos impostos pelo problema (ou seja, é efectivamente uma solução).

```
procura(X) :- gera(X), testa(X).
```

Se o teste falhar novas soluções são geradas pelo mecanismo de backtracking.

Exemplo: Testar se duas listas se intersectam (i.e. se têm um elemento comum).

```
intersecta(Xs,YS) :- member(X,Xs), member(x,Ys).
```

Aqui o 1º `member` gera um elemento da lista `Xs` e o 2ª `member` testa se esse elemento está em `Ys`.

O problema das N rainhas

Colocar N rainhas num tabuleiro de xadrez NxN de nodo a que nenhuma rainha possa ser capturada.

Podemos modelar este problema de várias maneiras.

Solução 1: Representar cada rainha por um par com as suas cpordenadas (X,Y). A solução final é dada por uma lista de N rainhas que não se atacam.

Dado que, para não se atacarem, as rainhas terão de estar colocadas em colunas distintas, podemos fixar as coordenadas X. A solução terá a seguinte forma

```
[(1,Y1), (2,Y2), ..., (N,YN)]
```

O problema reduz-se agora a em encontrar instâncias deste padrão que não se ataquem.

```
rainhas1(N,L) :- template(1,N,L), solucao1(N,L).

template(N,N,[(N,_)]).
template(M,N,[(M,_)|L]) :- M<N, M1 is M+1, template(M1,N,L).
```



```
rainhas1(N,L) :- template(1,N,L), solucao1(N,L).

template(N,N,[(N,_)]).
template(M,N,[(M,_)|L]) :- M<N, M1 is M+1, template(M1,N,L).

solucao1(_,[]).
solucao1(N,[(X,Y)|Resto]) :- solucao1(N,Resto),
                             entre(1,N,L),
                             member(Y,L),            % gera
                             naoataca((X,Y),Resto).  % testa

entre(M,N,[M|L]) :- M<N, M1 is M+1, entre(M1,N,L).
entre(N,N,[N]).

naoataca(_,[]).
naoataca((X,Y),[(X1,Y1)|Resto]) :- X =\= X1, Y =\= Y1,
                                   X-X1 =\= Y-Y1, X-X1 =\= Y1-Y,
                                   naoataca((X,Y),Resto).
```

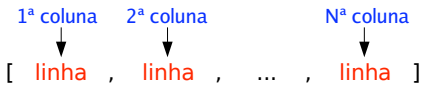
Exemplo:

```
| ?- rainhas1(4,S).
S = [(1,3),(2,1),(3,4),(4,2)] ? ;
S = [(1,2),(2,4),(3,1),(4,3)] ? ;
no
```


Solução 2: Coordenada X de cada rainha dada pela posição na lista.
A solução final é uma permutação da lista [1,2,...,N] sem "ataques".

Dado que uma solução para este problema tem necessariamente que colocar cada rainha numa coluna diferente (e isso era dado à partida na resolução 1), podemos omitir a informação sobre as coordenadas X: ela será dada pela posição na lista.

Uma representação mais económica é representar o tabuleiro como a lista das coordenadas Y das rainhas: [Y1,Y2,...,YN]



Para evitar ataques horizontais, as rainhas não podem estar numa mesma linhas. Isto impõem restrições às coordenadas Y: as N rainhas têm que ocupar N linhas diferentes.

O problema reduz-se então ao problema de *encontrar uma permutação da lista [1,2,...,N] em que não haja ataques diagonais*.

```
rainhas2(N,S) :- entre(1,N,L),
                 permutation(L,S),      % gera
                 segura(S).             % testa

segura([Y|Ys]) :- segura(Ys), \+ ataca(Y,Ys).
segura([]).

% uma rainha ataca outra a uma distância de N colunas, se esta segunda tiver
% uma coordenada-Y que é maior ou menor N unidades (casas) do que a primeira
% rainha

ataca(R,L) :- ataca(R,1,L).

ataca(R,N,[Y|_]) :- R is Y+N ; R is Y-N.
ataca(R,N,[_|Ys]) :- N1 is N+1, ataca(R,N1,Ys).
```

Exemplo:

```
| ?- rainhas2(8,S).
S = [1,5,8,6,3,7,2,4] ? ;
S = [1,6,8,3,7,4,2,5] ? ;
S = [1,7,4,6,8,2,5,3] ? ;
S = [1,7,5,8,2,4,6,3] ?
...
```


Exercícios:

1. Construa uma 3ª solução para o problema das N rainhas, que em vez de testar a permutação completa (i.e. colocar todas as rainhas e depois testar) teste cada rainha à medida que a coloca no tabuleiro. Note que a solução final é cronstruida utilizando um acumulador.
   A solução a apresentada continua a usar a técnica de gerar e testar ?

2. Defina um predicado `gnat(+N,?X)` para gerar, por backtracking, números naturais sucessivos até `N`.

3. Usando o predicado anterior, escreva predicado `raiz(+N,?I)` para calcular a raiz quadrada inteira de um número natural `N`, definido como sendo o número `I` tal que $I^2 \leq N$ e $(I+1)^2 > N$.

O problema de colorir um mapa

Colorir uma mapa de modo a que regiões vizinhas não tenham a mesma cor (sabe-se que 4 cores são suficientes para colorir qualquer mapa).

Instanciar uma *estrutura de dados desenhada especialmente para um problema* é um meio eficaz de implementar soluções de gerar e testar. O controlo da construção da solução final é feito pela unificação.

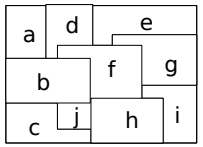
Solução: Implementar o seguinte algoritmo (suportado por uma estrutura de dados adquada)

Para cada região do mapa:
- escolher uma cor;
- escolher (ou verificar) as cores para as regiões vizinhas, das cores que sobram.

Estrutura de dados: `[ regiao(nome,cor,lista de cores dos vizinhos), ...]`

Exemplo:

```
[ regiao(a,A,[D,B]), regiao(b,B,[A,D,F,H,J,C]),
  regiao(c,C,[B,J,H]), regiao(d,D,[A,E,F,B]),
  regiao(e,E,[D,F,G]), ... ]
```



O seguinte programa usa a técnica de geração e teste.

```
color_map([],_).
color_map([R|Rs],Cores) :- color_region(R,Cores),
                           color_map(Rs,Cores).

color_region(regiao(Nome,Cor,Vizinhos),Cores) :- select(Cor,Cores,Cores1),
                                                 members(Vizinhos,Cores1).

members([X|Xs],Ys) :- member(X,Ys), members(Xs,Ys).
members([],_).
```

A partilha de variáveis, na estrutura de dados de suporte, assegura que a mesma região não possa ser colorida com cores diferentes, nas diversas iterações do algoritmo.

Exercício: Pretende-se construir um programa que permita testar esta solução de coloração de mapas.

1. Defina na base de conhecimento exemplos de mapas (associe a cada mapa um nome).
2. Defina o predicado `colorir(?Nome,+Cores,?Pintura)` que dado o nome do mapa e a lista de cores a usar, produz em `Pintura` a associação entre dada região do mapa e a respectiva cor.
3. Analise o funcionamento do programa fazendo a traçagem da sua execução.