Inteligência Artificial 04/05 — Aula 7

1. Nos exercícios que se seguem, sobre árvores binárias, iremos usar a seguinte representação em termos prolog:

- `[]` representa uma árvore vazia
- `no(X,LeftTree,RightTree)` representa um nó de uma árvore, onde `X` é o valor do nó, e `LeftTree` e `RightTree` são as representações respectivas da sub-árvore direita e da sub-árvore esquerda.
- Uma árvore é representada pelo seu nó raíz.

Como exemplo, considere a seguinte árvore e a sua respectiva representação em Prolog:

```
        5
       / \
      /   \
     3     8
    / \   / \
   1   4 7   9
```

```
no(5,no(3,no(1,[],[]),no(4,[],[])),no(8,no(7,[],[]),no(9,[],[])))
```

(a) Crie um predicado `treeSum(+Tree,?Sum)` que sucede se `Tree` for uma árvore binária cujos valores no nós são números e `Sum` for a soma de todos esses valores. Exemplos:

```
?- treeSum(no(5,no(3,[],[]),no(8,[],[])),S).
S = 16 ? ;
no

?- treeSum(no(5,no(a,[],[]),no(8,[],[])),S).
no
```

(b) Crie um predicado `depth(+Tree,?D)` que sucede se `Tree` for uma árvore binária e `D` for a sua profundidade. Por profundidade de uma árvore entende-se o máximo número de nós que temos percorrer para chegar da raíz a uma folha da árvore. Por definição, a profundidade de uma árvore vazia é zero. Exemplos (próxima página):

```
?- depth(no(5,no(3,[],[]),no(8,[],[])),D).
D = 2 ? ;
no

?- depth(no(5,no(3,no(1,[],[]),[]),[]),D).
D = 3 ? ;
no
```

(c) Crie um predicado `balanced(+Tree)` que sucede se `Tree` for uma árvore binária bem balançada. Uma árvore binária é bem balançada se em cada nó a diferença entre a profundidade da sub-árvore esquerda e da sub-árvore direita não exceder um. Exemplos:

```
?- balanced(no(5,no(a,[],[]),no(8,[],[]))).
yes

?- balanced(no(5,no(3,no(1,[],[]),[]),no(8,[],[]))).
yes

?- balanced(no(5,no(3,no(1,[],[]),[]),[])).
no
```

(d) Crie um predicado `inorder(+Tree,?List)` que sucede se `Tree` for uma árvore binária e `List` for a lista resultante de fazer uma travessia *inorder* da árvore. Uma travessia *inorder* começa por percorrer toda a sub-árvore esquerda, depois percorre o valor do nó actual e por fim percorre toda a sub-árvore direita. Exemplos:

```
?- inorder(no(5,no(3,no(1,[],[]),no(4,[],[])),no(8,[],[])),L).
L = [1,3,4,5,8] ? ;
no

?- inorder(no(5,no(3,no(1,[],[]),[]),[]),L).
L = [1,3,5] ? ;
no
```

(e) Crie um predicado `levelorder(+Tree,?List)` que sucede se `Tree` for uma árvore binária e `List` for a lista resultante de fazer uma travessia em largura da árvore. Uma travessia em largura passa primeiro por todos os nós de nivel zero, depois pelos nós de nível um, depois do nível dois, etc. Dentro do mesmo nível, os nós são percorridos da esquerda para a direita. Exemplos (próxima página):

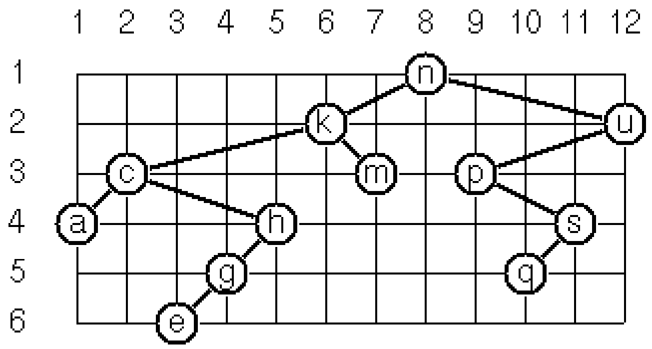
```
?- levelorder(no(5,no(3,no(1,[],[]),no(4,[],[])),no(8,[],[])),L).
L = [5,3,8,1,4] ? ;
no

?- levelorder(no(5,no(3,no(1,[],[]),[]),[]),L).
L = [5,3,1] ? ;
no

?- levelorder(no(5,[],no(3,[],no(1,[],[]))),L).
L = [5,3,1] ? ;
no
```

(f) Imagine que quer desenhar uma árvore binária no ecrã. Uma possível metodologia seria distribuir os nós por uma grelha rectangular do seguinte modo (a figura dá um exemplo):

- A posição `X` de um nó é igual à sua posição numa pesquisa *inorder* da árvore
- A posição `Y` de um nó é igual à profundidade do nó na árvore



Uma maneira de preparar uma árvore é modificar cada um dos seus nós `no(N,Left,Right)` para `no(N,X,Y,Left,Right)` onde `X` e `Y` são a posição do nó na grelha como atrás definido. Crie um predicado `layout(+Tree,?NewTree)` que sucede se `Tree` for uma árvore binária e `List` for a árvore resultante, com informação da sua posição, como foi atrás descrito.

```
?- layout(no(5,[],no(3,[],no(1,[],[]))),L).
L = no(5,1,1,[],no(3,2,2,[],no(1,3,3,[],[]))) ? ;
no

?- layout(no(5,no(3,no(1,[],[]),[]),[]),L).
L = no(5,3,1,no(3,2,2,no(1,1,3,[],[]),[]),[]) ? ;
no
```

```
?- layout(no(5,no(3,[],no(4,[],[])),no(8,[],[])),L).
L = no(5,3,1,no(3,1,2,[],no(4,2,3,[],[])),no(8,4,2,[],[])) ? ;
no
```

2. DESAFIO:

(Este problema saiu no CENPL'98)

"À entrada da casa do Miguel há uma escada com 10 degraus. Cada vez que entra em casa, o Miguel avança pelas escadas subindo um ou dois degraus em cada passada. De quantas maneiras diferentes pode o Miguel subir as escadas?"

in "Pierre Berloquin, 100 Jogos Lógicos, O Prazer da Matemática, no 4, Gradiva"

Crie um predicado para resolver este jogo para um qualquer número de degraus.

O predicado deve chamar-se `degraus(+Degraus,-NumManeiras,-Lista)` onde `Degraus` é o número de degraus a considerar, `NumManeiras` é o número de maneiras diferentes de subir as escadas e `Lista` é a lista das possibilidades de subida (cada possibilidade é, por sua vez, também representada por uma lista). Vê os exemplos para perceberes melhor. Como ajuda, sugerimos o uso do predicado `findall/3` exemplos:

```
?- degraus(3,N,L).
L = [[1,1,1],[1,2],[2,1]],
N = 3 ? ;
no

?- degraus(4,N,L).
L = [[1,1,1,1],[1,1,2],[1,2,1],[2,1,1],[2,2]],
N = 5 ? ;
```

DESAFIO EXTRA:

Consegue detectar alguma lógica no `NumManeiras`, ou seja, no número de possibilidades diferentes? Consegue explicá-la? É alguma sequência conhecida? Consegue fazer um novo predicado `degrausNum/2` que devolva apenas o número de possibilidades diferentes, mas funcione para números muitos maiores?