Escola Politécnica da USP - Depto. de Enga. Mecatrônica

PMR-3510 Inteligência Artificial

Aula 6- Resolução de problemas por máquinas usando estruturas

Prof. José Reinaldo Silva

reinaldo@usp.br

Trabalho em grupo

Os grupos estão já definidos e registrados no e-disciplinas. A configuração é a seguinte:

Grupo 1:
Alyson Akio Haro
Alexandre Inoue
Eduardo Kose
Vitor Fukuda

Grupo 2:
Alex Majima
Gabriel Ferreira
Lucas de Angelis
Thiago Ferraz

Grupo 3:
Gabriel Tutia
Lucas Palopoli
Pedro dos Santos Melo
Samuel Monção

Grupo 4:
Gustavo Novello
Luiz Guilherme Sabino
Alessandro Ezequiel Junior
Gabriel Negre

Grupo 5:
Gabriel Yida
Vinicius Santiago
Danilo Polidoro

Grupo 6:
Guilherme Aires de França
Henrique Peterlevitz
Wagner Geraldo Ferreira

Estrutura de um "resolvedor automático de problemas"

Para dotar uma máquina da capacidade de resolver problemas (ou uma classe de problemas) é preciso ter uma estrutura com os seguintes atributos:

1. uma descrição clara do "estado inicial" ou seja das condições iniciais do problema a ser resolvido; ✓
2. uma descrição clara do objetivo ou "estado final", de modo que seja possível saber quando (e se) o problema foi resolvido; ✓
3. em cada estágio do processo de solução saber quais os próximos estados que podem ser atingidos; ✓
4. poder escolher um (ou o melhor) caminho entre os estados acima;
5. saber que operadores (ou passos) aplicar para fazer a "transição" para um próximo estado;
6. discernir se estamos convergindo para a solução.

Cronograma de entrega do trabalho em grupo

O cronograma de trabalho será divido em "milestones":

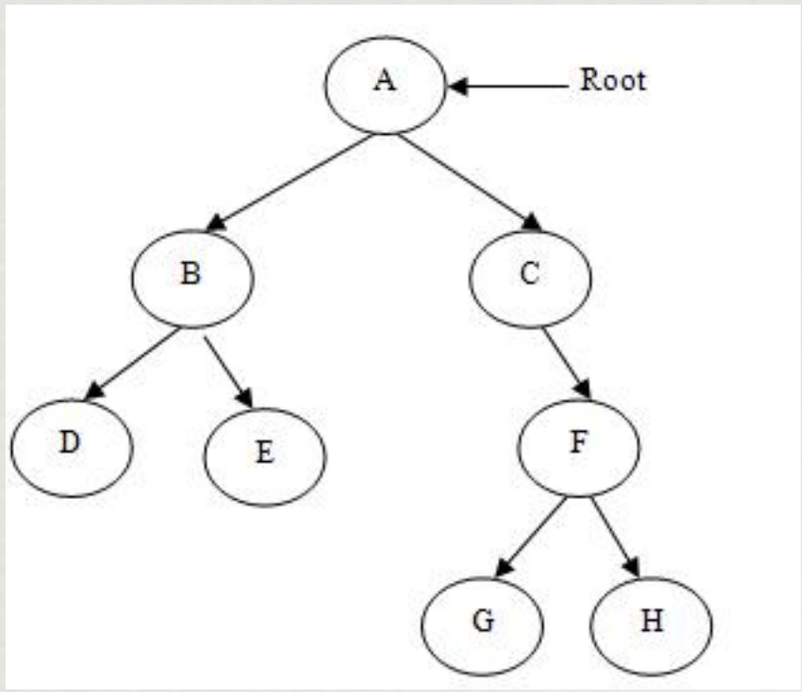
Uma outra forma seria gerar o espaço de estados enquanto se busca a solução

1. uma descrição clara do "estado inicial" ou seja das condições iniciais do problema a ser resolvido;
2. uma descrição clara do objetivo ou "estado final", de modo que seja possível saber quando (e se) o problema foi resolvido;
3. em cada estágio do processo de solução saber quais os próximos estados que podem ser atingidos;
4. poder escolher um (ou o melhor) caminho entre os estados acima;
5. saber que operadores (ou passos) aplicar para fazer a "transição" para um próximo estado;
6. discernir se estamos convergindo para a solução.

→ *estrutura*

Usando árvores como base para a solução

Uma opção muito interessante é modelar o espaço de estados na forma de uma árvore

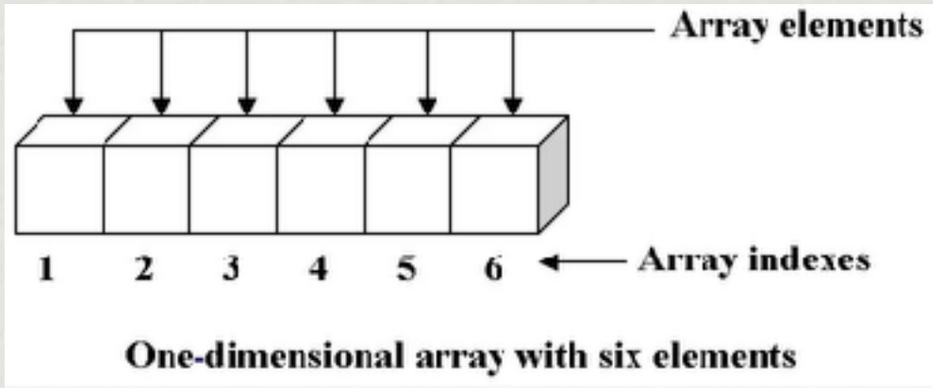


Estrutura de Listas

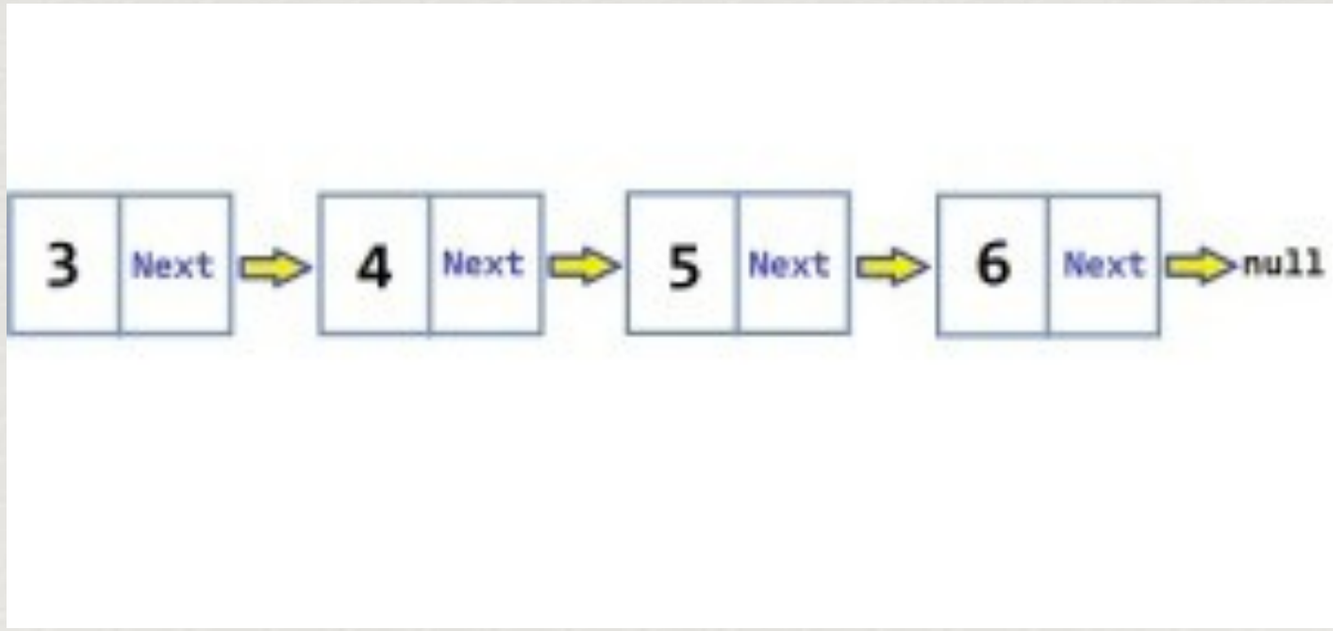
Uma estrutura básica em programação (lógica) é a lista. Conceitualmente uma lista é uma sequencia de registros homogênea. Normalmente estas listas são indexadas (arrays) ou direcionadas por ponteiros (listas ligadas).

No caso da programação em Prolog a lista segue seu conceito mais básico e é composta de uma cabeça (head) ou o primeiro elemento da lista, e o corpo (body) que é o restante da lista. Assim, o primeiro elemento é referenciado com um componente da lista e o restante como uma lista

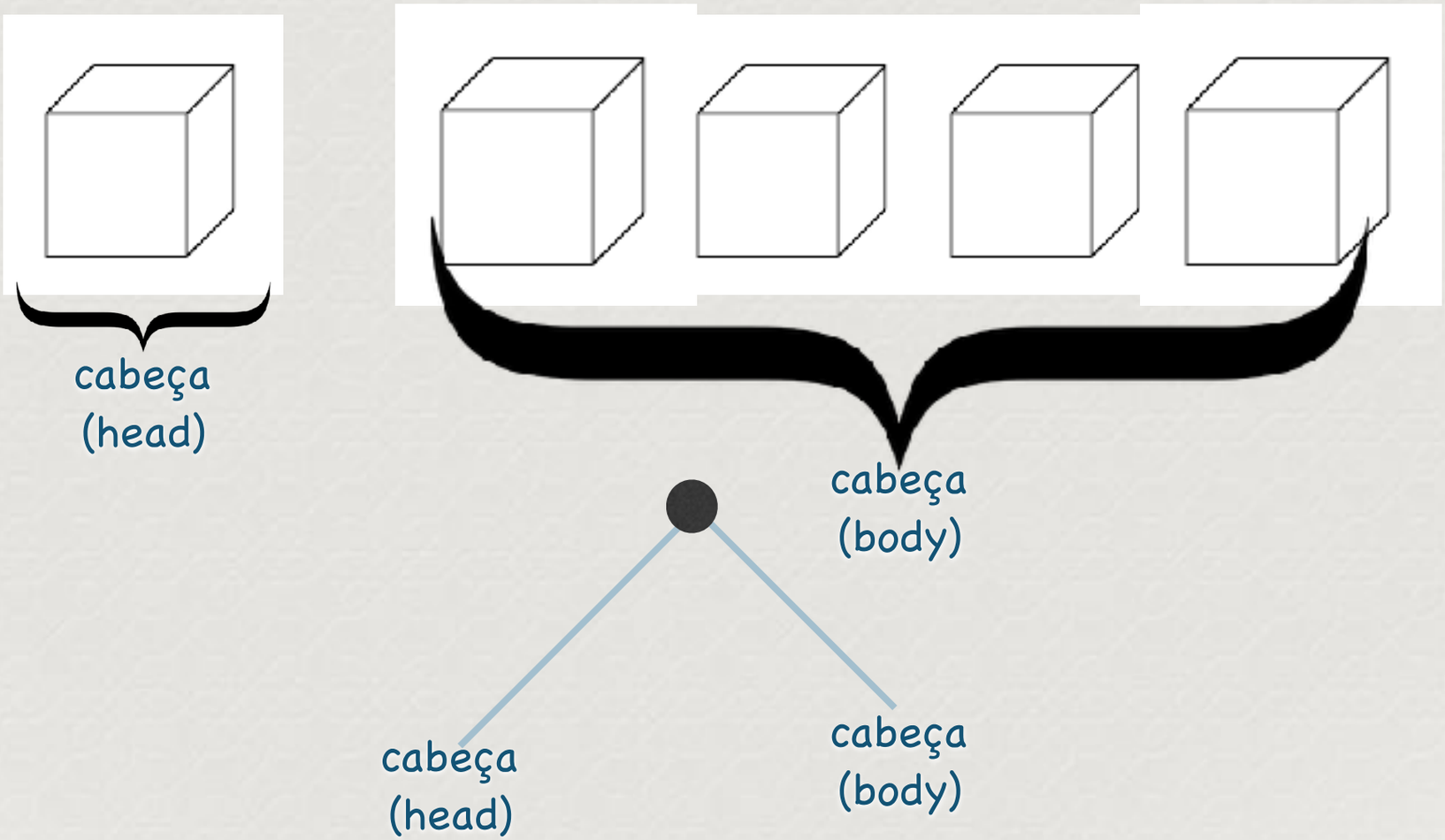
Instancias da estrutura abstrata de lista



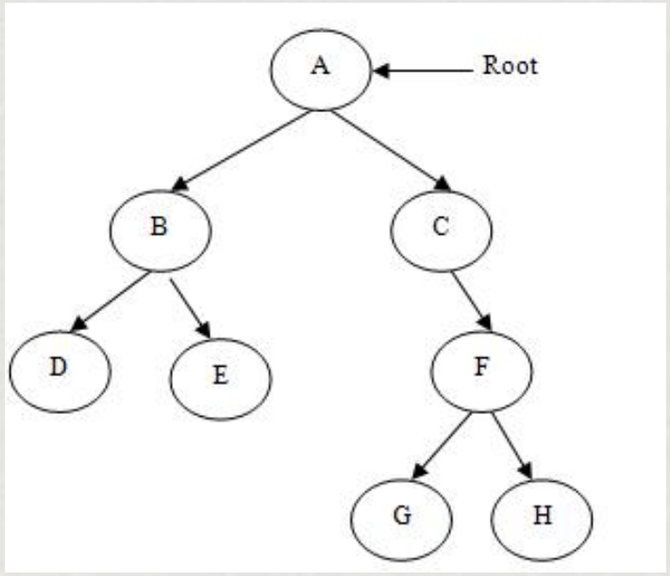
One-dimensional array with six elements

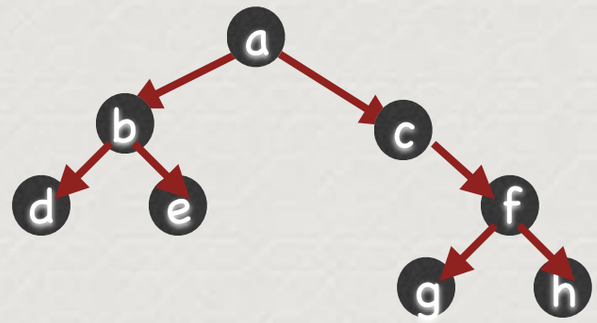


Estrutura abstrata de lista



Portanto é possível utilizar uma lista para implementar uma estrutura de árvore





[a, [b, [d, e]], [c, [f, [g, h]]]]

Em prolog...

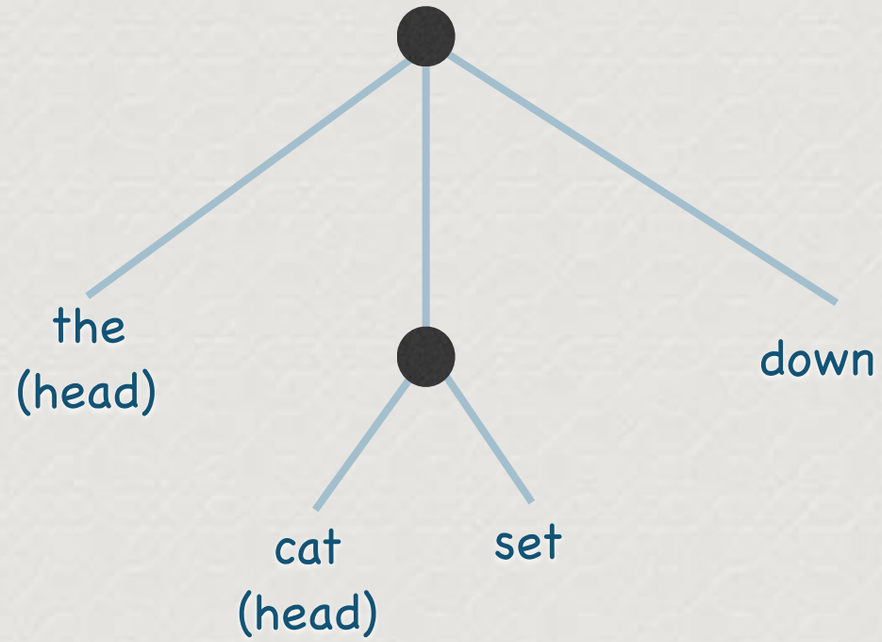
List	Head	Tail
[a, b, c]	a	[b, c]
[]	(none)	(none)
[[the, cat], sat]	[the, cat]	[sat]
[the, [cat, sat]]	the	[[cat, sat]]
[the, [cat, sat], down]	the	[[cat, sat], down]
[X+Y, x+y]	X+Y	[x+y]

Table 3.1. Some lists with their head and tail

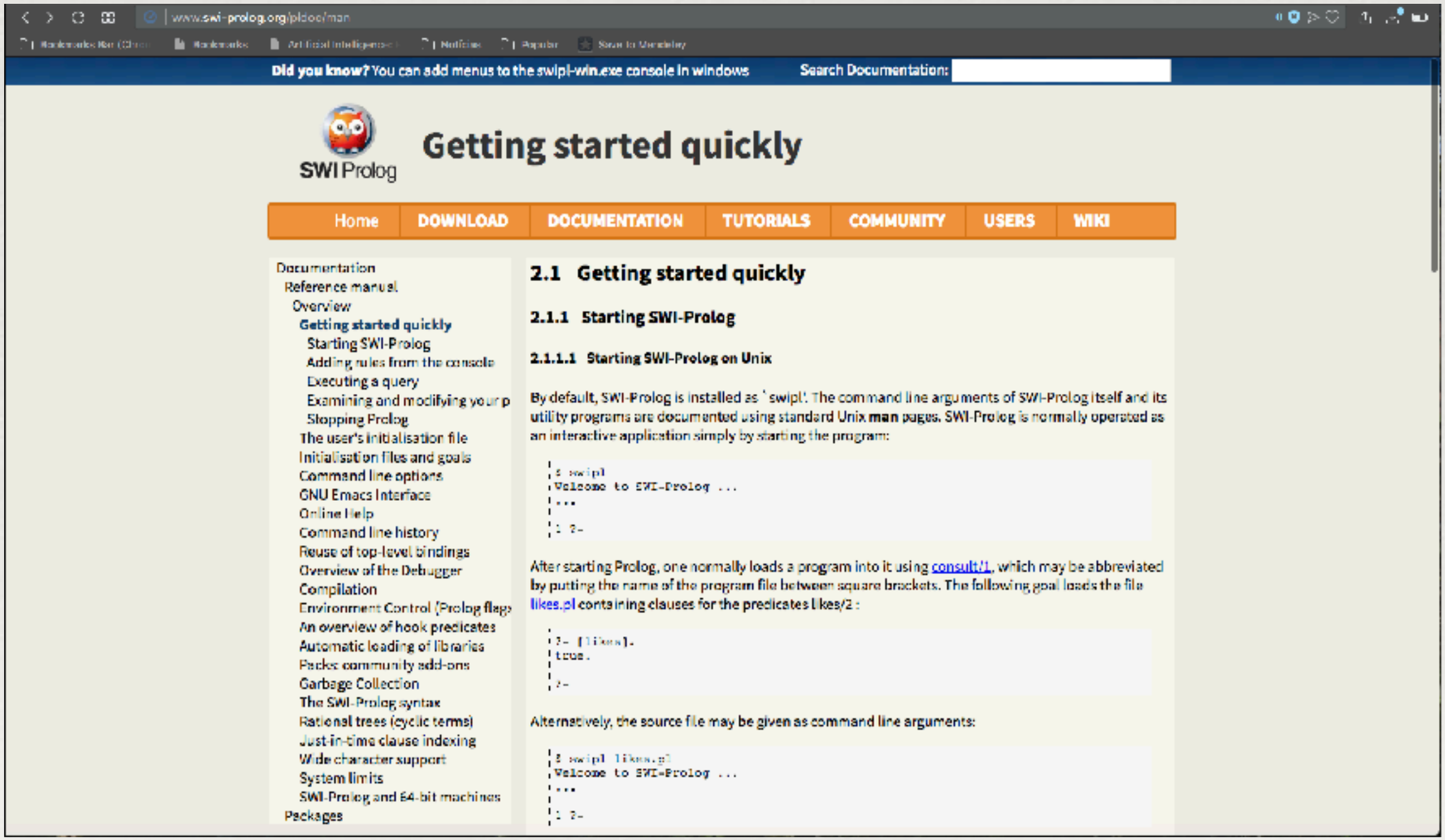
Programming in Prolog, Clocksin & Mellish

Em prolog...

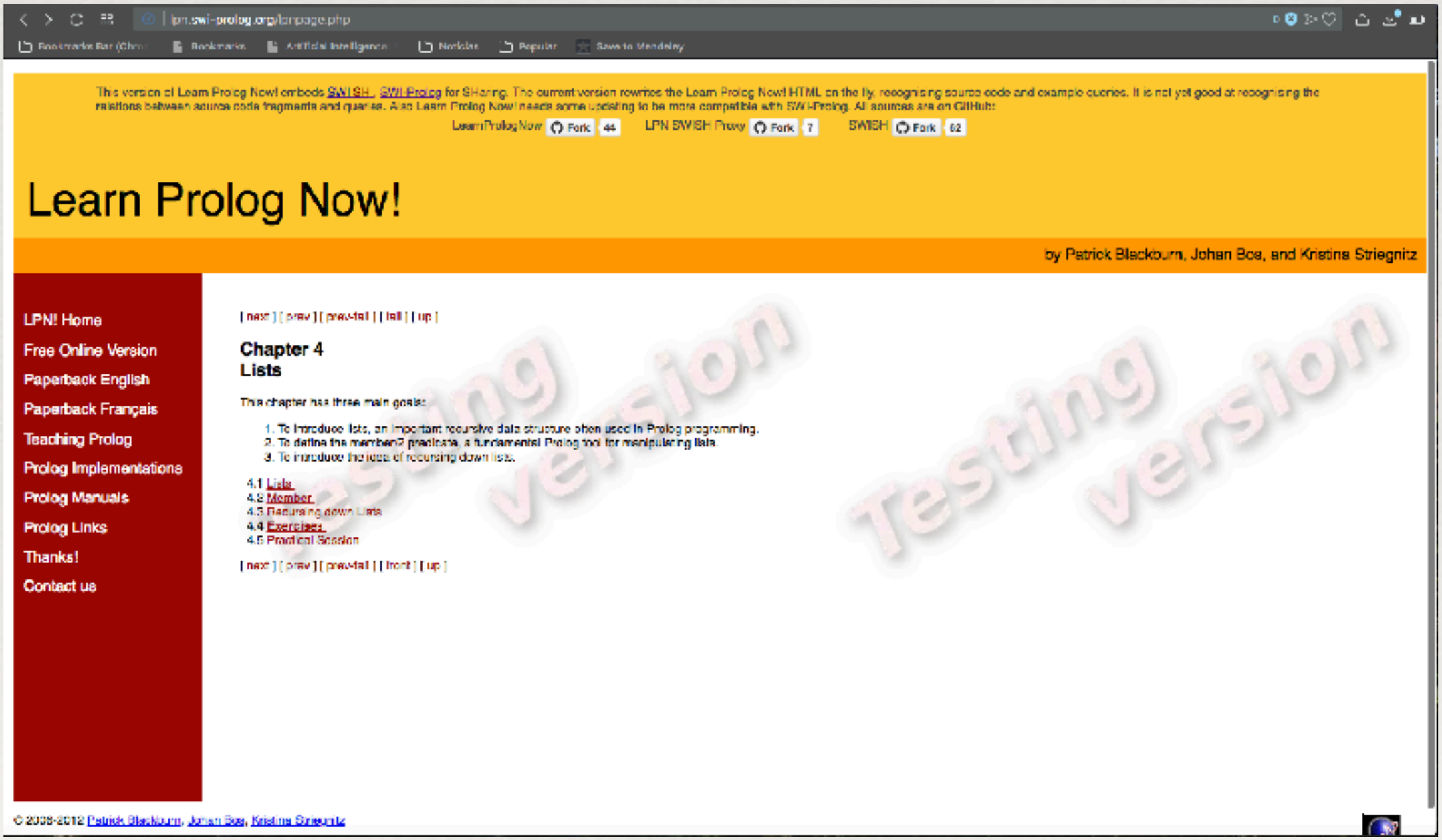
[the, [cat, set], down]



www.swi-prolog.org/pldoc/man



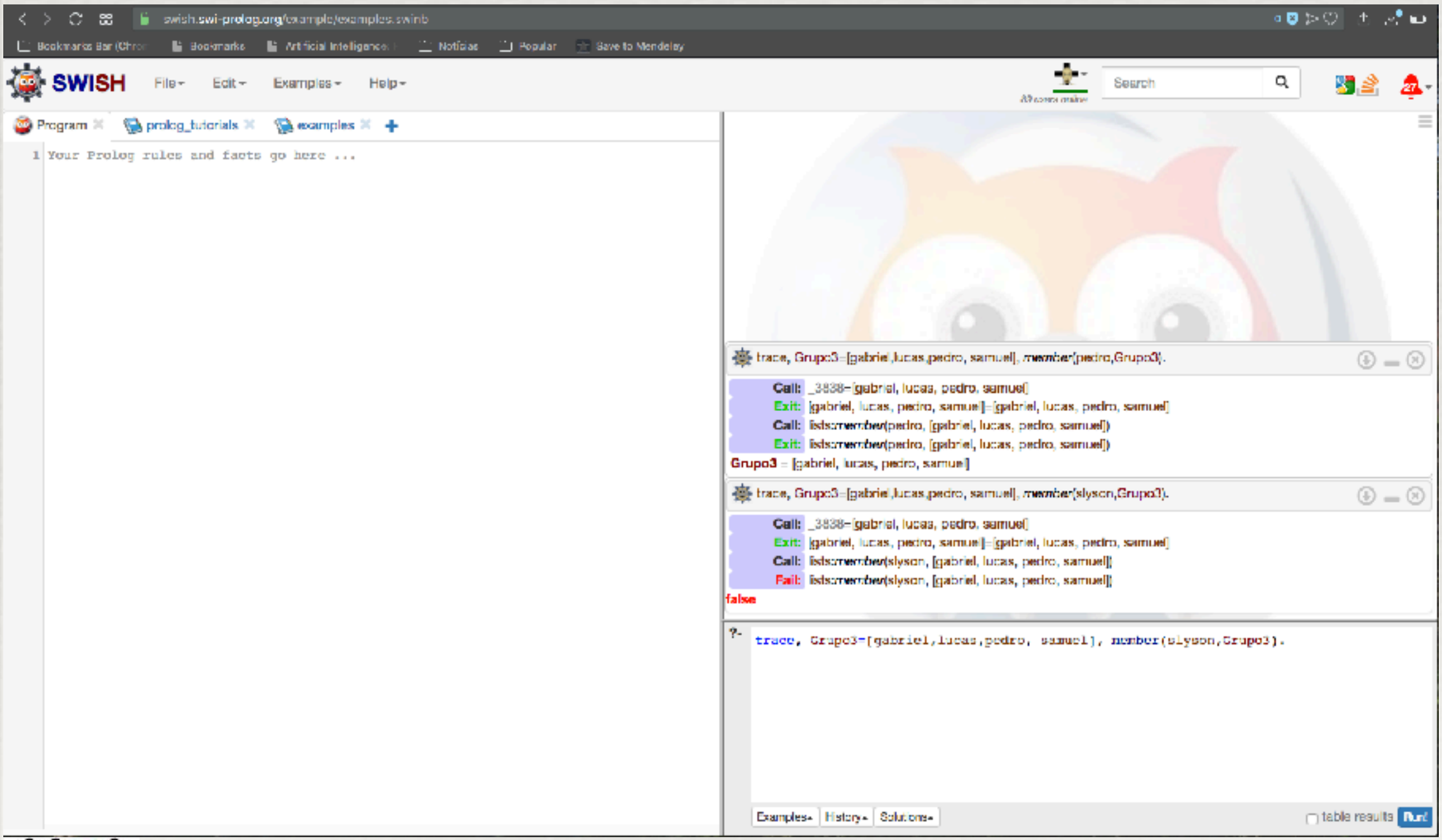
lpn.swi-prolog.org/lpnpage.php



Se desejamos ter um procedimento que checa se um dado elemento pertence à lista...

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
```

Exemplo...

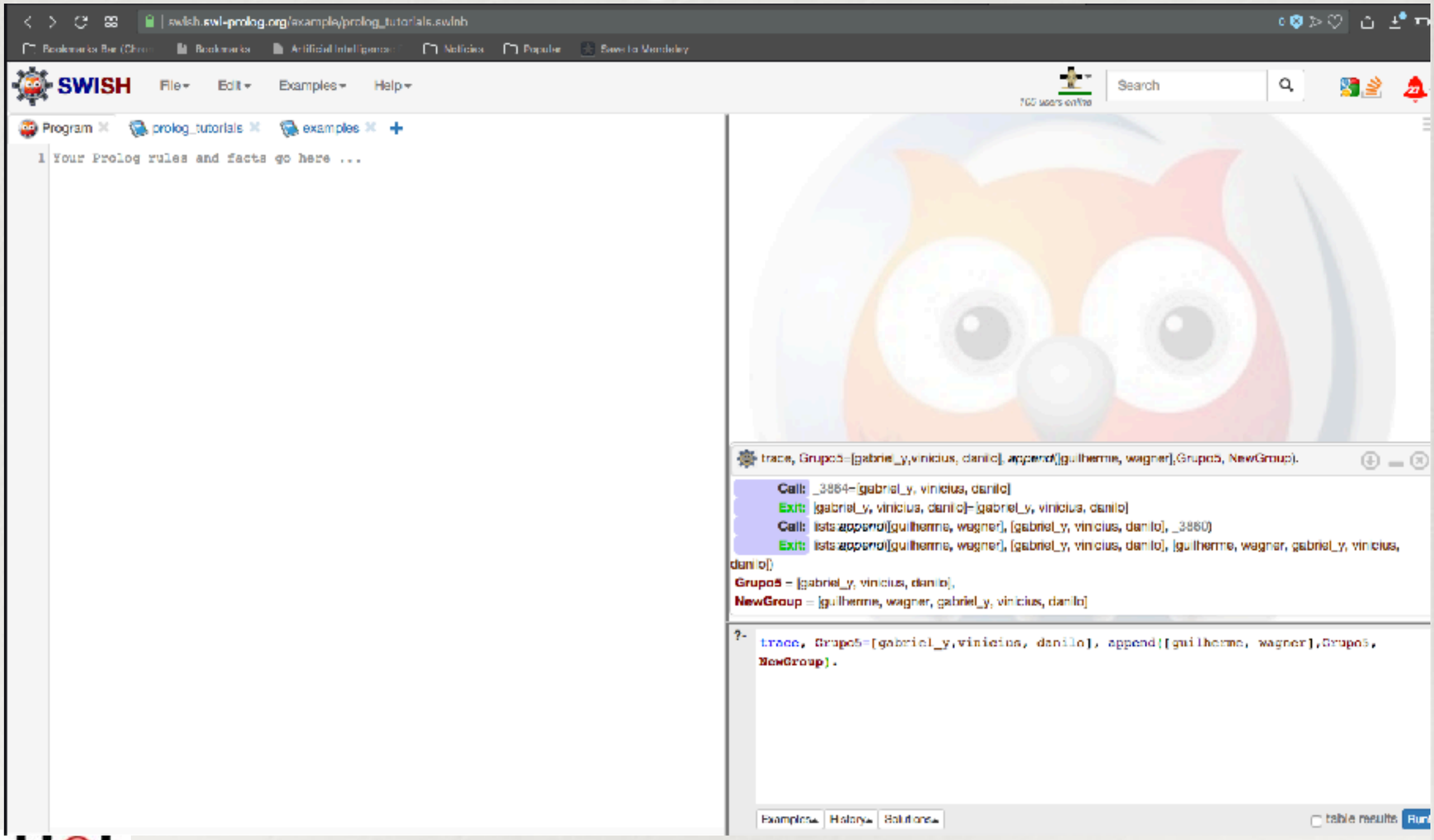

swish.swi-prolog.org/example/examples.swinb
SWISH
File
Edit
Examples
Help
Search
Program
prolog_tutorials
examples
1 Your Prolog rules and facts go here ...
trace, Grupo3=[gabriel,lucas,pedro, samuel], member(pedro,Grupo3).
Call: _3838=[gabriel, lucas, pedro, samuel]
Exit: [gabriel, lucas, pedro, samuel]=[gabriel, lucas, pedro, samuel]
Call: lists:member(pedro, [gabriel, lucas, pedro, samuel])
Exit: lists:member(pedro, [gabriel, lucas, pedro, samuel])
Grupo3 = [gabriel, lucas, pedro, samuel]
trace, Grupo3=[gabriel,lucas,pedro, samuel], member(slyson,Grupo3).
Call: _3838=[gabriel, lucas, pedro, samuel]
Exit: [gabriel, lucas, pedro, samuel]=[gabriel, lucas, pedro, samuel]
Call: lists:member(slyson, [gabriel, lucas, pedro, samuel])
Fail: lists:member(slyson, [gabriel, lucas, pedro, samuel])
false
?- trace, Grupo3=[gabriel,lucas,pedro, samuel], member(slyson,Grupo3).
Examples
History
Solutions
table results
Run!

operador interno append/3

Uma operação básica com listas é juntar duas listas em uma terceira lista.
O operador append/3 pode fazer isso facilmente...

```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

Exemplo...


swish.swi-prolog.org/example/prolog_tutorials.swinb
SWISH
File
Edit
Examples
Help
Search
Program
prolog_tutorials
examples
1 Your Prolog rules and facts go here ...
trace, Grupo5=[gabriel_y,vinicius, danilo], append([guilherme, wagner],Grupo5, NewGroup).
Call: _3884=[gabriel_y, vinicius, danilo]
Exit: [gabriel_y, vinicius, danilo]=[gabriel_y, vinicius, danilo]
Call: lists:append([guilherme, wagner], [gabriel_y, vinicius, danilo], _3890)
Exit: lists:append([guilherme, wagner], [gabriel_y, vinicius, danilo], [guilherme, wagner, gabriel_y, vinicius, danilo])
Grupo5 = [gabriel_y, vinicius, danilo],
NewGroup = [guilherme, wagner, gabriel_y, vinicius, danilo]
?- trace, Grupo5=[gabriel_y,vinicius, danilo], append([guilherme, wagner],Grupo5, NewGroup).
Examples
History
Solutions
table results
Run

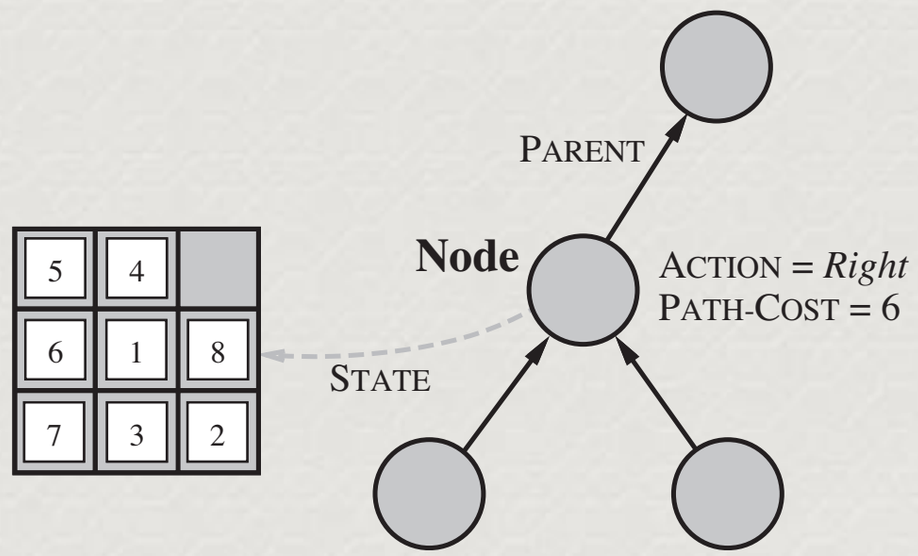
Voltando ao projeto dos grupos...

Não devemos de forma alguma antecipar a programação do algoritmo para resolver o jogo de tiles antes de seguir o processo (aqui colocado como uma "boa prática" , e mostrado na transparência de no. 3 esta aula. Entretanto, para que não pareça a "missão impossível", bem acima do "nivel do curso", vamos já "baixar à terra", e colocar alguns procedimentos que deverão ser utilizados e que não constituem o núcleo do projeto.

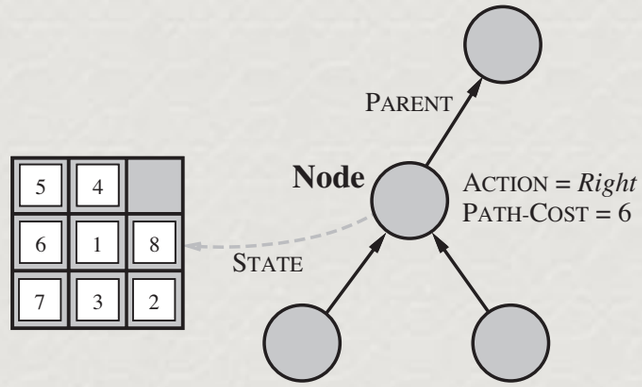
Um destes procedimentos é justamente a entrada de dados: como poderemos fazer um sistema Prolog (especificamente o Swich) ler uma tabela de tiles. Utilizaremos um procedimento simples, baseado em listas como vimos anteriormente.

A tabela de tiles

A tabela de tiles tem NOVE posições e o conteúdo de cada uma destas posições é um inteiro de zero a oito, onde o zero significa o tile para onde se pode mover os demais vizinhos.



O programa em prolog a seguir insere uma tabela de tiles de entrada. Não é exatamente uma "interface", não tem correção de entradas erradas, somente preserva o número de dados de entrada.




```
add_tile(X,Y,K) :- X < 10, read(Z), append(Y, [(X,Z)], L),  N is X + 1,
     add_tile(N, L, K).
add_tile(10, X, X).
in_data(Index, Board) :- Index=1, Tile_board=[], add_tile(Index, Tile_board, Board).
```

swish.swi-prolog.org/example/iris.swinb

SWISH File Edit Examples Help

Program

```
1
2 add_tile(X,Y,X):- X < 10, read(S), append(Y, [[X,S]],L),  N is X + 1,
3       add_tile(N, L, X).
4 add_tile(10, X, X).
5 in_data(Index, Board):- Index=1, Tile_board=[], add_tile(Index, Tile_board, Board).
6
```

```
Exit: read(2)
Call: lists:append([[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3]], [[9,2]], _8574)
Exit: lists:append([[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3]], [[9,2]], [[1,5], [2,4], [3,0], [4,6], [5,1],
[6,8], [7,7], [8,3], [9,2]])
Call: _8624 is 9+1
Exit: 10 is 9+1
Call: add_tile(10, [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3], [9,2]], _8416)
Call: 10<10
Fail: 10<10
Redo: add_tile(10, [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3], [9,2]], _8416)
Exit: add_tile(10, [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3], [9,2]], [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8],
[7,7], [8,3], [9,2]])
Exit: add_tile(9, [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3]], [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7],
[8,3], [9,2]])
Exit: add_tile(8, [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7]], [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3],
[9,2]])
Exit: add_tile(7, [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8]], [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3], [9,2]])
Exit: add_tile(6, [[1,5], [2,4], [3,0], [4,6], [5,1]], [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3], [9,2]])
Exit: add_tile(5, [[1,5], [2,4], [3,0], [4,6]], [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3], [9,2]])
Exit: add_tile(4, [[1,5], [2,4], [3,0]], [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3], [9,2]])
Exit: add_tile(3, [[1,5], [2,4]], [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3], [9,2]])
Exit: add_tile(2, [[1,5]], [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3], [9,2]])
Exit: add_tile(1, [], [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3], [9,2]])
Exit: in_data(1, [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3], [9,2]])
X = 1,
Y = [[1,5], [2,4], [3,0], [4,6], [5,1], [6,8], [7,7], [8,3], [9,2]]
```

0.870 seconds cpu time

```
?- trace,in_data(X,Y).
```

Examples History Solutions table results Run!

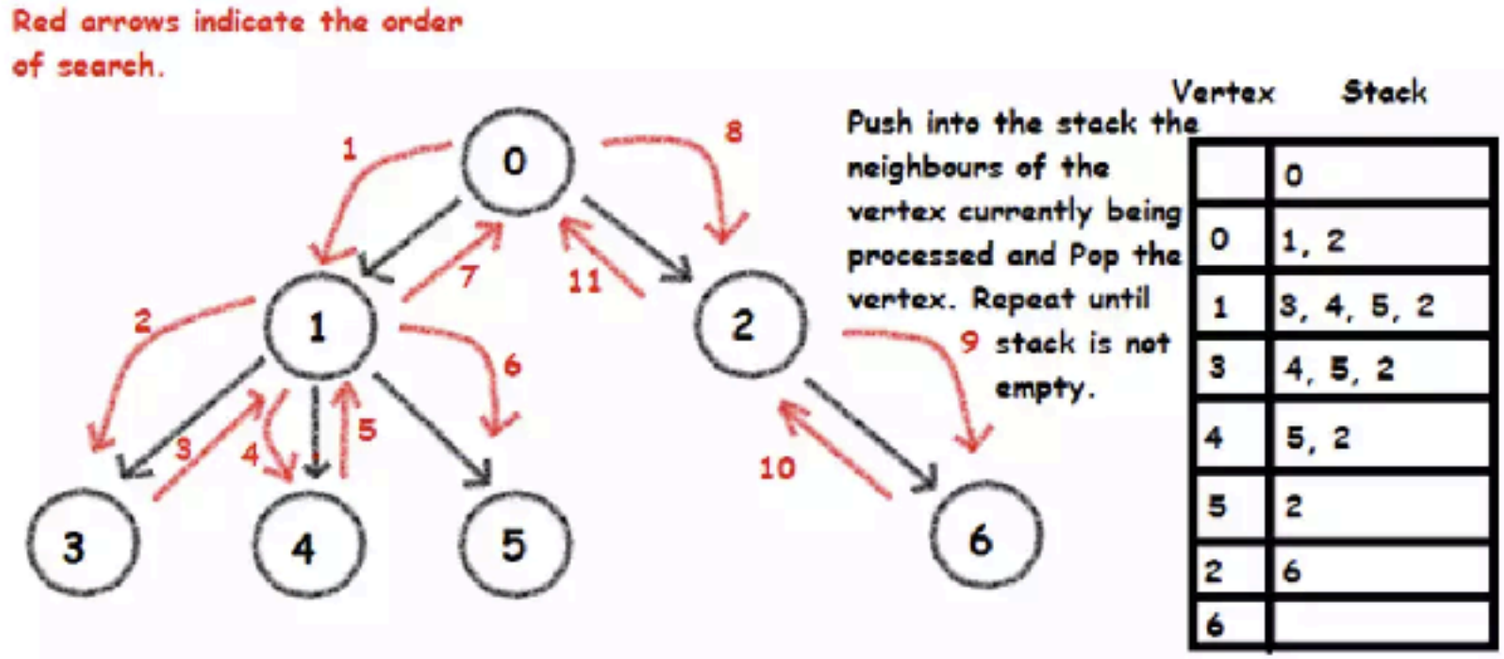
5	4	
6	1	8
7	3	2

Algoritmos de busca

Busca não informada - quando todos os nós gerados são igualmente promissores, ou não se tem informação sobre o seu potencial: busca em profundidade, busca em largura, busca de custo uniforme

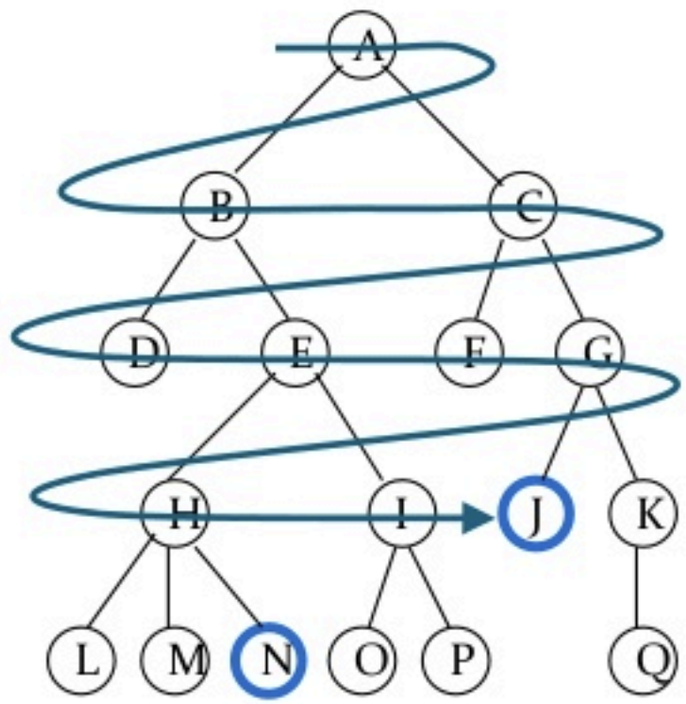
Busca informada - quando conhecimento heurístico pode ser levantado que distingue entre os nós gerados em um mesmo nível da árvore.

Busca em profundidade



Busca em largura

Breadth-first searching[1]



- A breadth-first search (BFS) explores nodes nearest the root before exploring nodes further away
- For example, after searching A, then B, then C, the search proceeds with D, E, F, G
- Node are explored in the order A B C D E F G H I J K L M N O P Q
- J will be found before N

Nas próximas aulas veremos

i) busca em profundidade limitada;
ii) busca de aprofundamento interativo;
iii) busca bi-direcional;

Até a próxima aula!