



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. A configuração é a seguinte:

Grupo 1:

Fernando Vicente Grando Monteiro 8992919

Marcos Menon José 8989112

Sverker Fabian Hugert 11462480

Vitor Augusto Martin 8993100

Grupo 4:

Alexandre Zamora Zerbini Denigres - 8583072

Dylan Kim Heleno - 8586072

Henrique Yda Yamamoto - 9349502

Vinicius Augusto Carnevali Miquelin - 8988410

Vinicius Takiuti Miura - 9345874

Grupo 2:

David Calil Spindola Pedro - 8989384

Diego Augusto Vieira Rodrigues - 8989276

Felipe Cominato Nemr - 9345662

Guilherme Sugahara Faustino - 9348971

Grupo 5:

Guilherme Dello Russo - 9345895

Nathan Géraud PERRIN- 10935360

Natália Thoma Ricardo - 9344806

Grupo 7:

Daniel Tsutsumi - 9349005

Gabriel Pinto - 8988017

Juliana Lopes - 8512600

Kaio Takase - 9345690

Matheus Ramalho - 9345710

Grupo 3:

Lucas Hideki Sakurai 8989193

Lucas Pereira Cotrim 8989092

Matheus Torres Guinezi 9345679

Monize Bessa Arabadgi 7961944

Renan Masashi Yamaguchi 8989151

Grupo 6:

Beatriz Santin de Araujo Pinho - 8533851

Bianca Faria Silva - 8991599

Bruna Sayuri de Souza Suzuki - 7987501

Murillo José Almeida Faria de Oliveira - 9436785



Cronograma de entrega do trabalho em grupo

O cronograma de trabalho será dividido em "milestones":

Setembro 2019						
webcid.com.br						
Domingo	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					
7: Independência do Brasil 22: Início da primavera 06 - Quarto Crescente 14 - Lua Cheia 21 - Quarto Minguante 28 - Lua Nova						

Outubro 2019						
webcid.com.br						
Domingo	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		
20: Início do horário de verão 12: Nsa. Sra. Aparecida 15: Dia dos Professores 05 - Quarto Crescente 13 - Lua Cheia 21 - Quarto Minguante 28 - Lua Nova						

- 25/09 - início do processo, definição do domínio
- 02/10 - definição do algoritmo e implementação
- 09/10 - entrega do programa final
- 16/10 - competição



Cronograma de entrega do trabalho em grupo

O segundo trabalho consiste em procurar uma aplicação real de mercado que usa Inteligência Artificial clássica (não machine learning que é objeto de outra disciplina). O trabalho do grupo é avaliar esta aplicação tecnicamente, modelos aplicados, algoritmos de busca, classificação como sistema especialista, etc. uso de computação evolutiva (algoritmo genético, fuzzy logic...), problemas de implementação, uso e manutenção, satisfação do usuário com o sistema, e finalmente, desempenho. O cronograma de milestones é o seguinte

06/11 - texto (pdf)
apresentando a aplicação

13/11 - resumo da análise
completa

27/11 - apresentação do
trabalho final

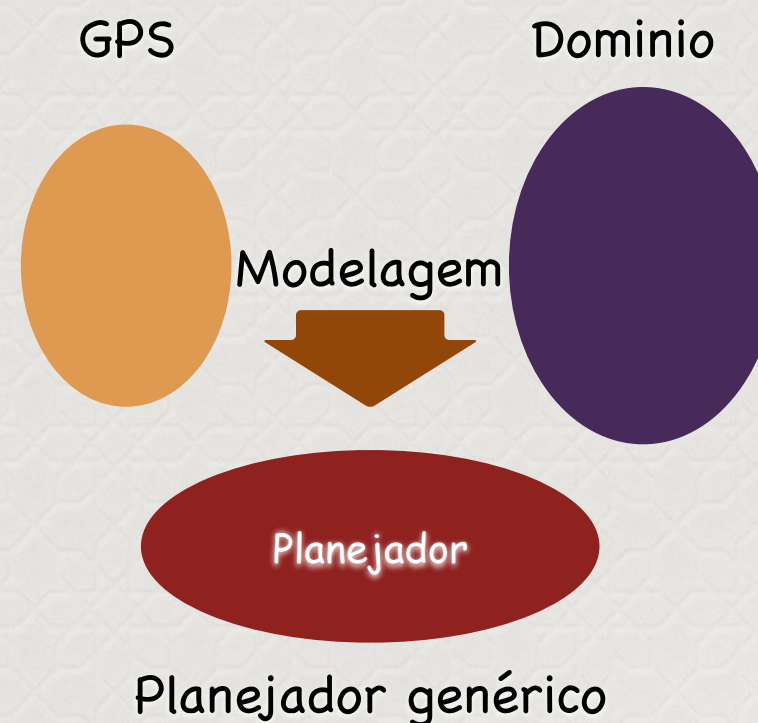
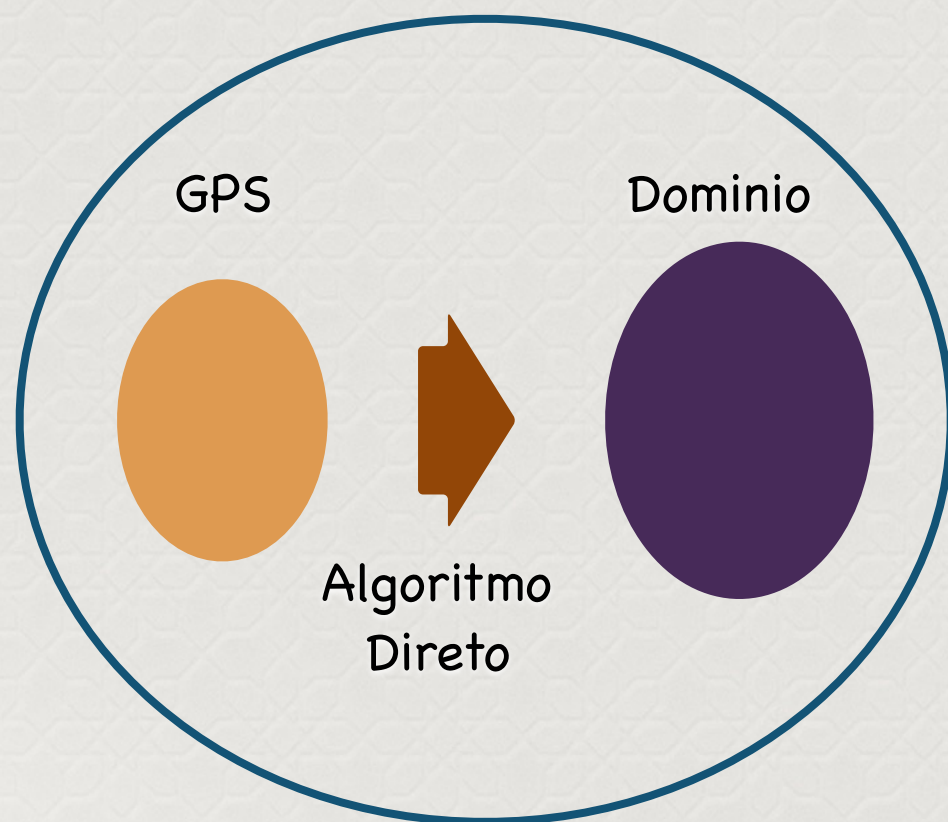




Dezembro 2019						
webcid.com.br						
Domingo	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				
21: Início do verão 25: Natal						
04 - Quarto Crescente 12 - Lua Chela 19 - Quarto Minguante 26 - Lua Nova						



Na aula passada falamos sobre o STRIPS e conseqüentemente sobre "planning" e concluimos que algoritmos de resolução de problemas baseados no STRIPS podem ser de dois tipos:





O método do STRIPS é baseado em estado-transição e consiste em:

- Modelar o domínio onde o plano será aplicado: o que implica em definir claramente estados e restrições que devem nortear a aplicação de ações e operadores.
- Definir o problema de planejamento, isto é, as ações admissíveis, pre e pós-condições para a aplicação destas ações;
- Definir um processo de busca, direta, heurística ou definir um modelo de problema para ser resolvido (este último caso está fora do escopo desta disciplina).



Temos portanto 3 opções:

- Uma busca cega (inteligente?)

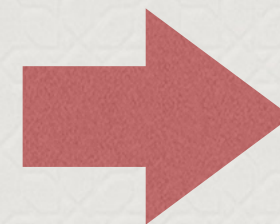


- Uma busca orientada por heurísticas!
- Tratar o problema por análise de propriedades para obter maior eficiência



Entretanto... uma busca cega deve ser feita sobre uma estrutura

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.

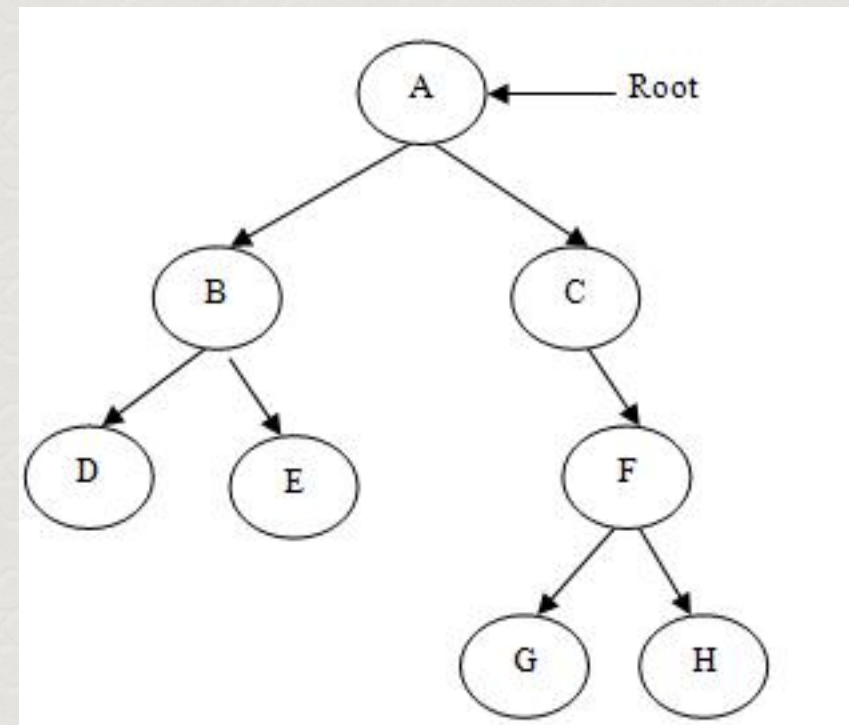


estrutura



Usando árvores como base para a solução

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





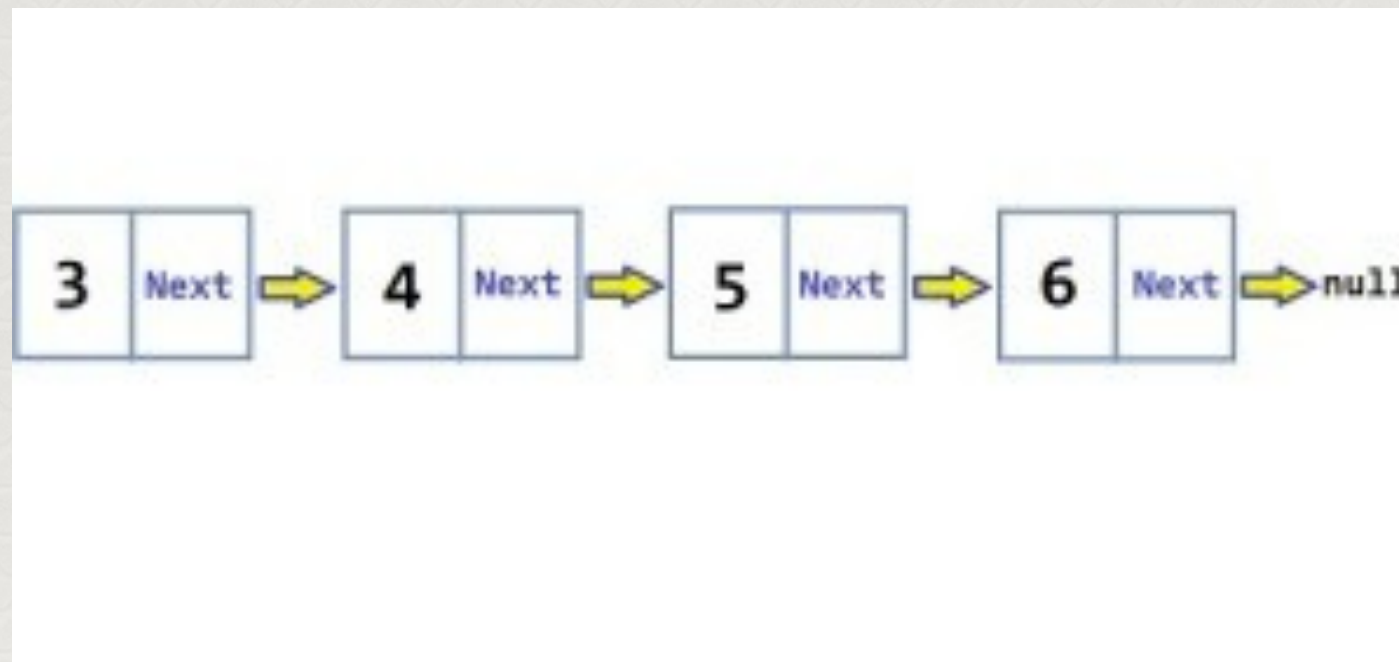
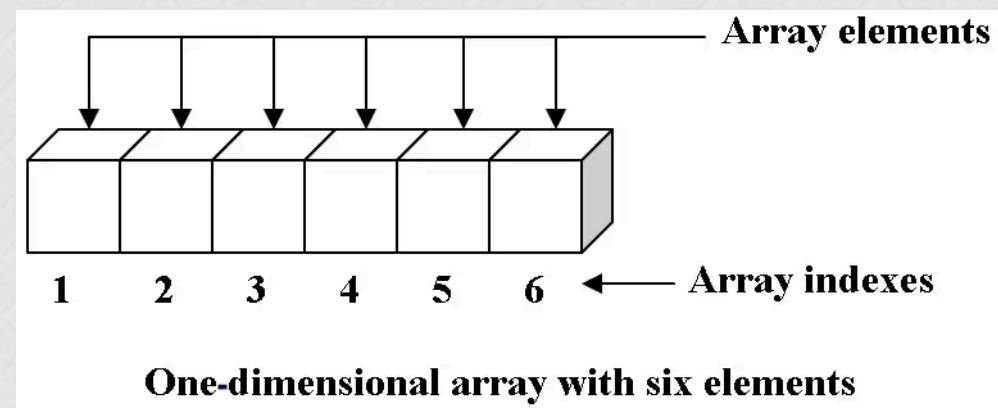
Estruturas e Listas

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, isto é, composta de uma cabeça (head) que é o primeiro elemento da lista, e de um corpo (body) que é a sub-lista restante. Estes dois elementos básicos são suficientes para dar suporte a todas os algoritmos de manipulação de estruturas como árvores, vetores, etc.

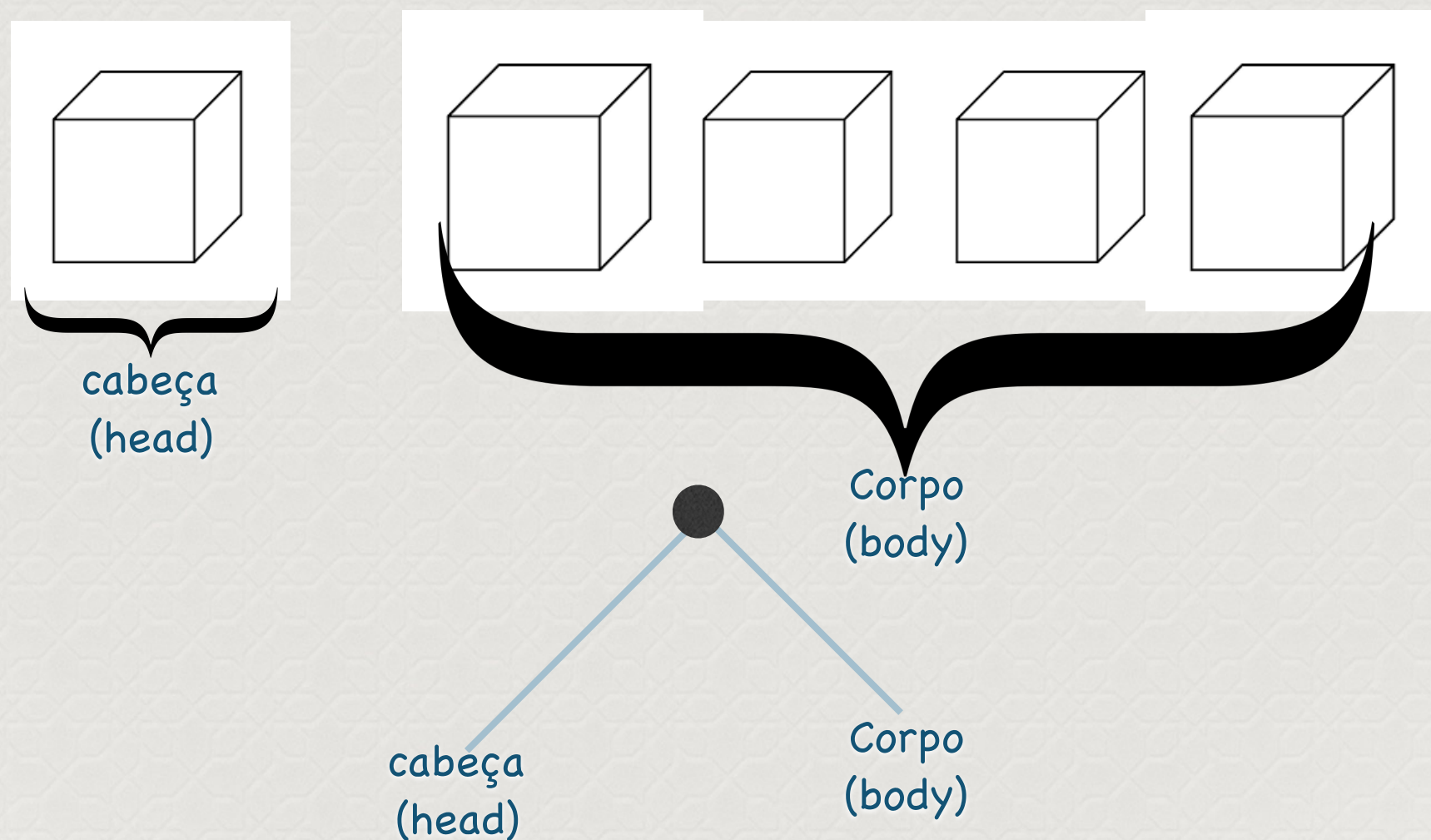


Instancias da estrutura abstrata lista



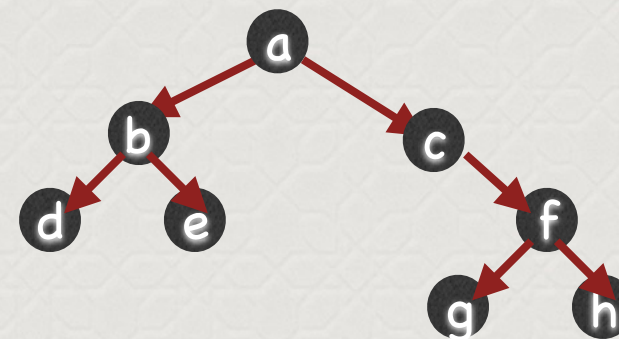
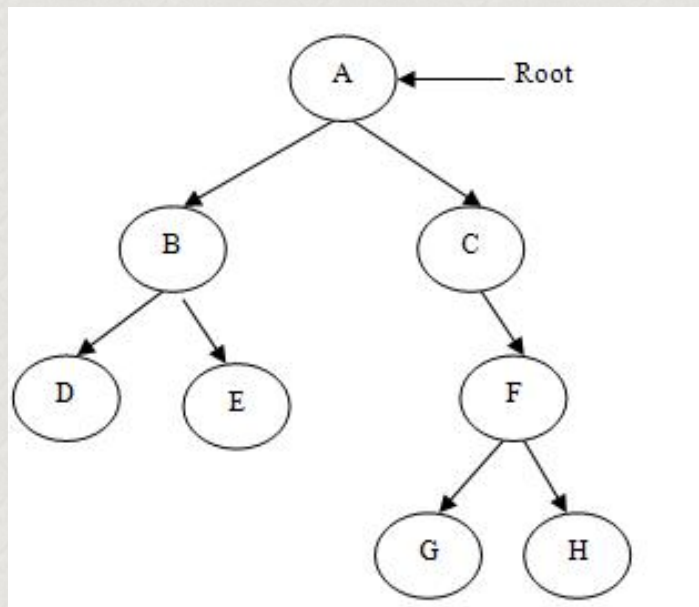


Estrutura abstrata de lista





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]

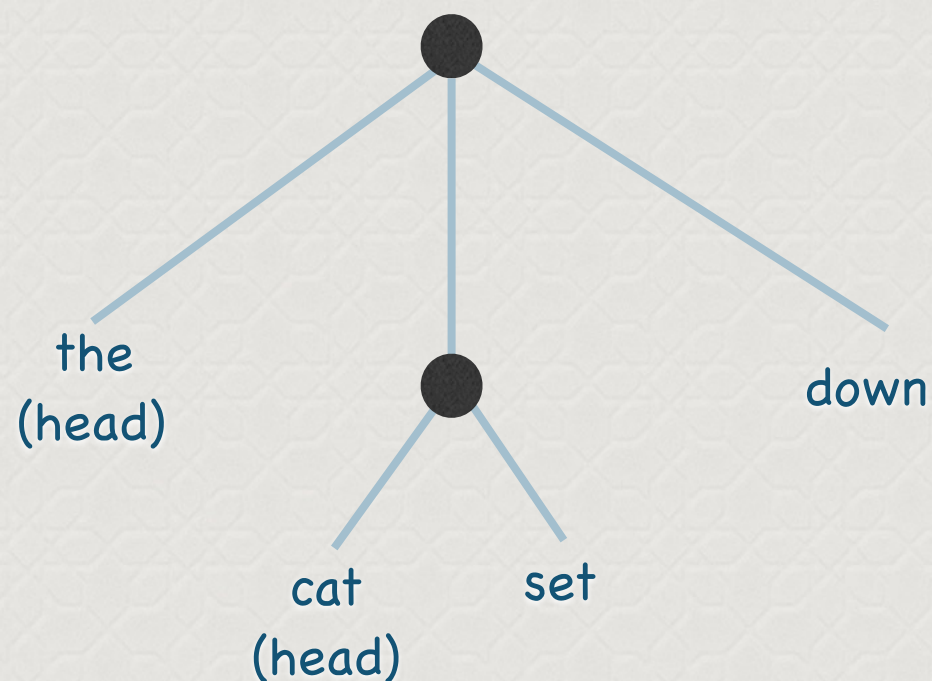
Table 3.1. Some lists with their head and tail

Programming in Prolog, Clocksin & Mellish



Em prolog...

[the, [cat, set], down]






www.swi-prolog.org/pldoc/man

www.swi-prolog.org/pldoc/man

Bookmarks Bar (Chrom... | Bookmarks | Artificial Intelligence: | Notícias | Popular | Save to Mendeley

Did you know? You can add menus to the swipl-win.exe console in windows | Search Documentation:


SWI Prolog

Getting started quickly

Home | DOWNLOAD | DOCUMENTATION | TUTORIALS | COMMUNITY | USERS | WIKI

Documentation
Reference manual
Overview
Getting started quickly
Starting SWI-Prolog
Adding rules from the console
Executing a query
Examining and modifying your p
Stopping Prolog
The user's initialisation file
Initialisation files and goals
Command line options
GNU Emacs Interface
Online Help
Command line history
Reuse of top-level bindings
Overview of the Debugger
Compilation
Environment Control (Prolog flags
An overview of hook predicates
Automatic loading of libraries
Packs: community add-ons
Garbage Collection
The SWI-Prolog syntax
Rational trees (cyclic terms)
Just-in-time clause indexing
Wide character support
System limits
SWI-Prolog and 64-bit machines
Packages

2.1 Getting started quickly

2.1.1 Starting SWI-Prolog

2.1.1.1 Starting SWI-Prolog on Unix

By default, SWI-Prolog is installed as ``swipl'`. The command line arguments of SWI-Prolog itself and its utility programs are documented using standard Unix **man** pages. SWI-Prolog is normally operated as an interactive application simply by starting the program:

```
$ swipl
Welcome to SWI-Prolog ...
...
1 ?-
```

After starting Prolog, one normally loads a program into it using [consult/1](#), which may be abbreviated by putting the name of the program file between square brackets. The following goal loads the file [likes.pl](#) containing clauses for the predicates `likes/2`:

```
?- [likes].
true.
?-
```

Alternatively, the source file may be given as command line arguments:

```
$ swipl likes.pl
Welcome to SWI-Prolog ...
...
1 ?-
```




lpn.swi-prolog.org/lpnpag.php

lpn.swi-prolog.org/lpnpag.php

Bookmarks Bar (Chrom... Bookmarks Artificial Intelligence: Notícias Popular Save to Mendeley

This version of Learn Prolog Now! embeds [SWI SH](#), [SWI-Prolog](#) for SHaring. The current version rewrites the Learn Prolog Now! HTML on the fly, recognising source code and example queries. It is not yet good at recognising the relations between source code fragments and queries. Also Learn Prolog Now! needs some updating to be more compatible with SWI-Prolog. All sources are on GitHub:

LearnPrologNow Fork 44 LPN SWISH Proxy Fork 7 SWISH Fork 62

Learn Prolog Now!

by Patrick Blackburn, Johan Bos, and Kristina Striegnitz

LPN! Home
Free Online Version
Paperback English
Paperback Français
Teaching Prolog
Prolog Implementations
Prolog Manuals
Prolog Links
Thanks!
Contact us

[\[next \]](#) [\[prev \]](#) [\[prev-tail \]](#) [\[tail \]](#) [\[up \]](#)

Chapter 4 Lists

This chapter has three main goals:

1. To introduce lists, an important recursive data structure often used in Prolog programming.
2. To define the member/2 predicate, a fundamental Prolog tool for manipulating lists.
3. To introduce the idea of recursing down lists.

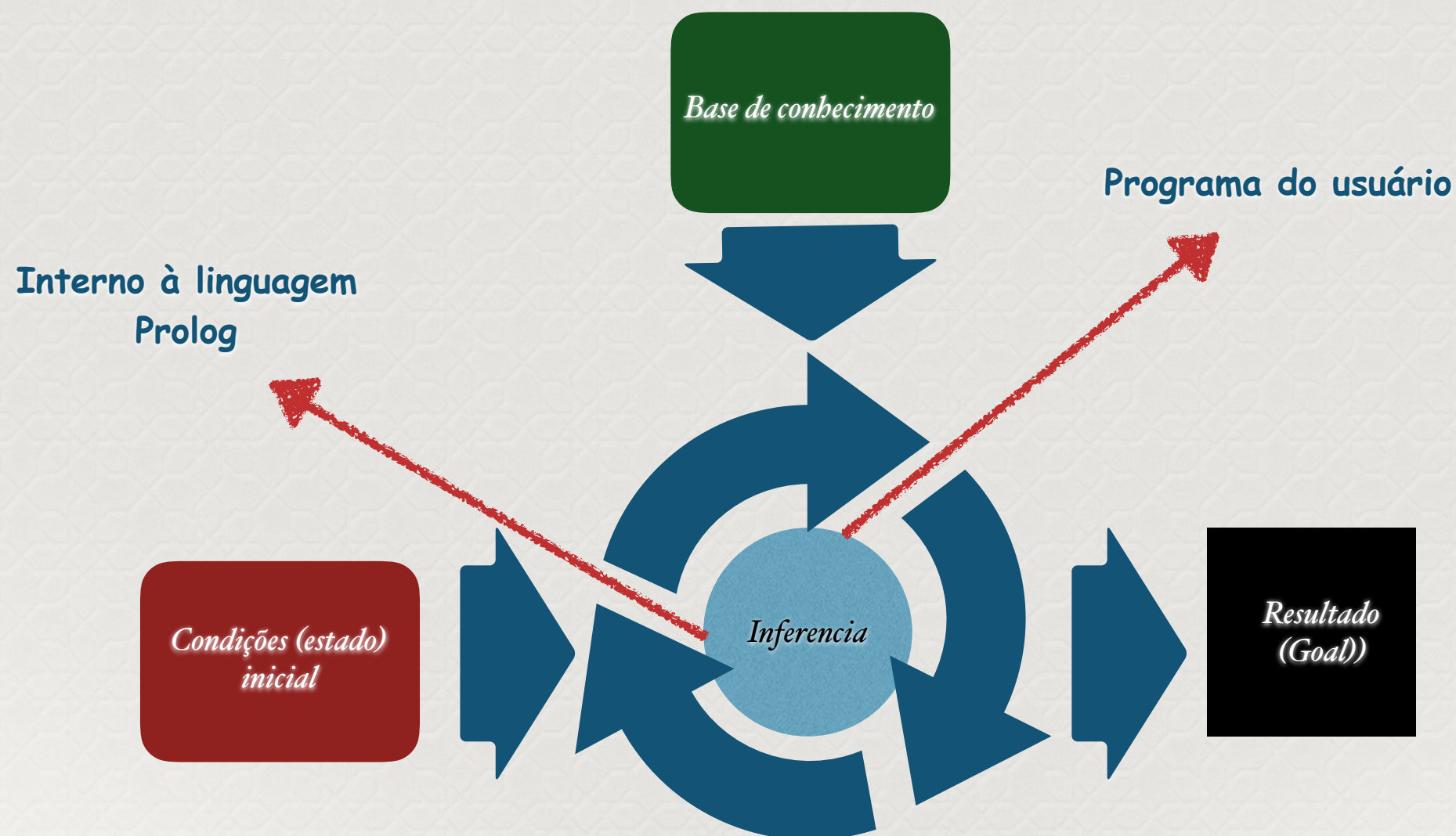
4.1 [Lists](#)
4.2 [Member](#)
4.3 [Recursing down Lists](#)
4.4 [Exercises](#)
4.5 [Practical Session](#)

[\[next \]](#) [\[prev \]](#) [\[prev-tail \]](#) [\[front \]](#) [\[up \]](#)

© 2006-2012 [Patrick Blackburn](#), [Johan Bos](#), [Kristina Striegnitz](#)



Em uma primeira abordagem, gostaríamos de ter “agentes inteligentes” capazes de “resolver problemas”. O que significa isso?





O predicado (interno) "member" checa se um dado elemento pertence à lista...

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




Usando os predicados do sistema...

swish.swi-prolog.org/p/Tic-Tac-Toe.swinb

AliExpress Booking.com Dafiti Americanas Facebook Getting Started

SWISH File Edit Examples Help

111 users online Search

Tic-Tac-Toe Program

1 Your Prolog rules and facts go here ...

trace, member(felipe,[david,diego,felipe,guilherme]).

Call: lists:member(felipe,[david,diego,felipe,guilherme])

Exit: lists:member(felipe,[david,diego,felipe,guilherme])

true

trace, member(jeronimo,[david,diego,felipe,guilherme]).

Call: lists:member(jeronimo,[david,diego,felipe,guilherme])

Fail: lists:member(jeronimo,[david,diego,felipe,guilherme])

false

?- trace, member(jeronimo,[david,diego,felipe,guilherme]).

Examples History Solutions

table results Run!



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

`:- member(felipe, [david, diego, felipe, guilherme])`

`member(felipe, [felipe| T])? False`

`:- member(felipe, [diego, felipe, guilherme])`

`member(felipe, [felipe| T])? False`

`:- member(felipe, [felipe, guilherme])`

`member(felipe, [felipe| T])? True`

`T=[guilherme]`



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

`:- member(jeronimo, [david, diego, felipe, guilherme])`

`member(jeronimo, [jeronimo| T])? False`

`:- member(jeronimo, [diego, felipe, guilherme])`

`member(jeronimo, [jeronimo| T])? False`

`:- member(jeronimo, [felipe, guilherme])`

`member(jeronimo, [jeronimo| T])? False`

`:- member(jeronimo, [guilherme])`

`member(jeronimo, [jeronimo| T])? False`

`:- member(jeronimo, [])`

False



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).
```




Usando a máquina de inf. do Prolog ...

swish.swi-prolog.org/p/Tic-Tac-Toe.swinb

AliExpress Booking.com Dafiti Americanas Facebook Getting Started

SWISH File Edit Examples Help

107 users online Search

Tic-Tac-Toe Program

1 Your Prolog rules and facts go here ...

trace, append([guilherme_russo, natan, natalia],[fernando, marcos, sverker], Y).

Call: lists:append([guilherme_russo, natan, natalia], [fernando, marcos, sverker], _3864)
Exit: lists:append([guilherme_russo, natan, natalia], [fernando, marcos, sverker], [guilherme_russo, natan, natalia, fernando, marcos, sverker])
Y = [guilherme_russo, natan, natalia, fernando, marcos, sverker]

?- trace, append([guilherme_russo, natan, natalia],[fernando, marcos, sverker], Y).

Examples History Solutions

table results Run!



append([], L, L).

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

False

append([guilherme_russo, natan, natalia],[fernando, marcos, sverker], Y).

X=guilherme_russo

L1=[natan, natalia]

L2=[fernando, marcos, sverker]

Y= [guilherme_russo | L3]

append([natan, natalia],[fernando, marcos, sverker], L3).

X=natan

L1=[natalia]

L2=[fernando, marcos, sverker]

Y= [natan | L3]



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

```
append([natalia],[fernando, marcos, sverker], L3).
```

```
    X=natalia  
    L1=[]  
    L2=[fernando, marcos, sverker]  
    Y= [natalia | L3]
```

```
append([], [fernando, marcos, sverker], L3).
```

```
L3=[fernando, marcos, sverker]
```

```
Y=[guilherme_russo, natan, natalia, fernando, marcos, sverker]
```




Estes são predicados básicos para montar uma árvore ou grafo (uma lista) - usando o append/3 - e fazer ua busca em árvore - usando o member/2.



edge(12, 10)

edge(10,12)

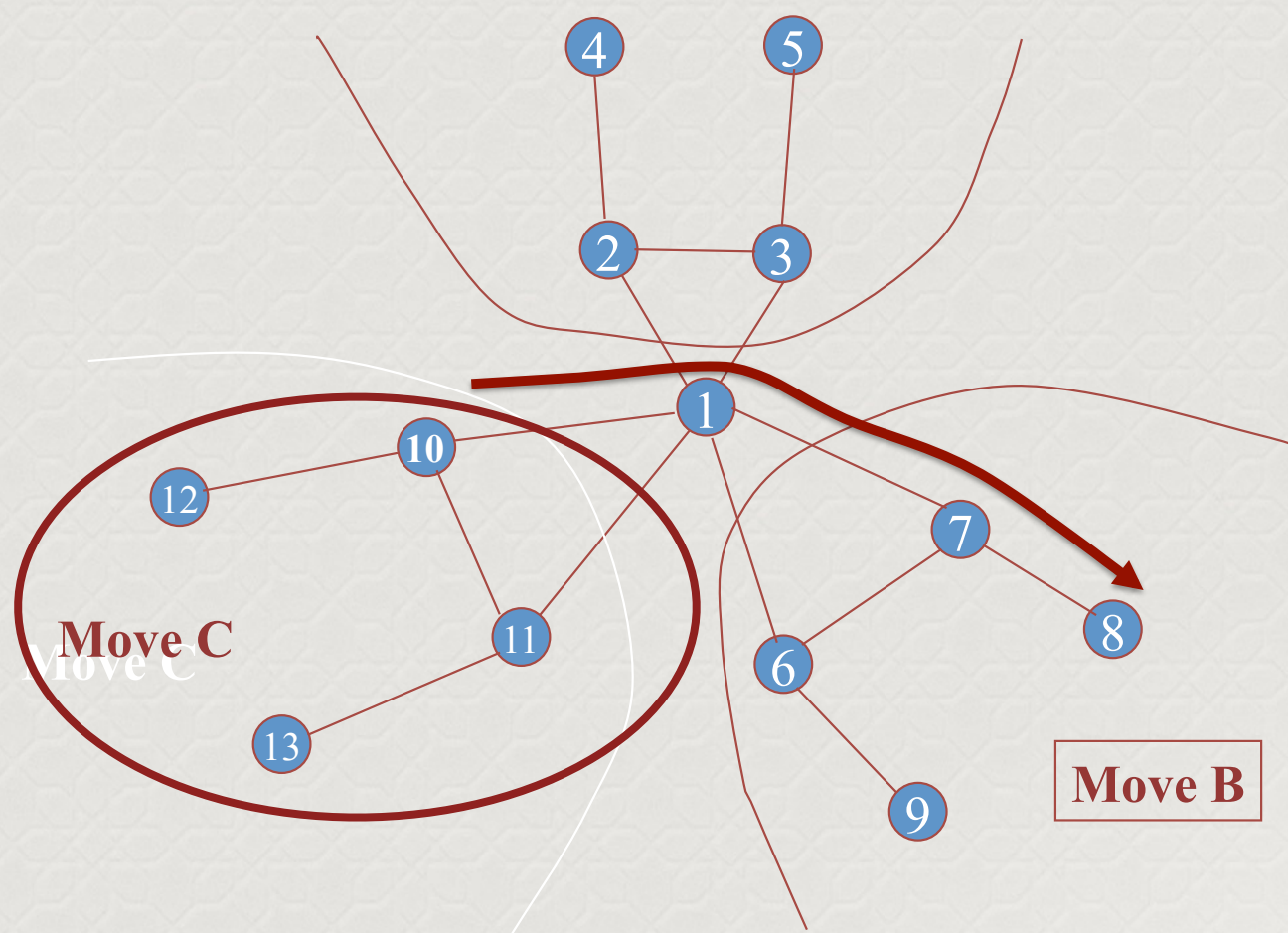
edge(10,11)

edge(11,10)

edge(13,11)

edge(11,13)

	10	11	12	13
10	0	1	1	0
11	1	0	0	1
12	1	0	0	0
13	0	1	0	0





Escola Politécnica da USP - Depto. de Enga. Mecatrônica

SWISH File Edit Examples Help

101 users online Search (new)

Tic-Tac-Toe Program +

```
1 edge(12, 10).
2 edge(10,12).
3 edge(10,11).
4 edge(11,10).
5 edge(13,11).
6 edge(11,13).
7
8 printedge(X,Y) :- edge(X,Y), write("1 ").
9 printedge(X,Y) :- \+ edge(X,Y), write("0 ").
10
11 printmatrix :-
12   Li member/2 (autoload from lists): True if Elem is a member of List.
13   member(Y, List),
14   nl,
15   member(X, List),
16   printedge(X,Y),
17   fail.
18
```

printmatrix.

```
0 1 1 0
1 0 0 1
1 0 0 0
0 1 0 0 false
```

?- printmatrix.

Examples History Solutions table results Run!

	10	11	12	13
10	0	1	1	0
11	1	0	0	1
12	1	0	0	0
13	0	1	0	0



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.



Red arrows indicate the order of search.

Push into the stack the neighbours of the vertex currently being processed and Pop the vertex. Repeat until stack is not empty.

Vertex	Stack
	0
0	1, 2
1	3, 4, 5, 2
3	4, 5, 2
4	5, 2
5	2
2	6
6	

Depth First Search



A busca informada é um algoritmo interessante quando sabemos que a solução é uma das folhas. No exemplo abaixo temos uma árvore e estamos buscando nós que estão nas folhas.

Ainda na hipótese que podemos gerar todo o espaço de estados (o que não será possível na maioria dos problemas práticos) podemos representar o grafo ao lado pelas arestas:

$s(a,b)$.

$s(a,c)$.

$s(b,d)$.

$s(b,e)$.

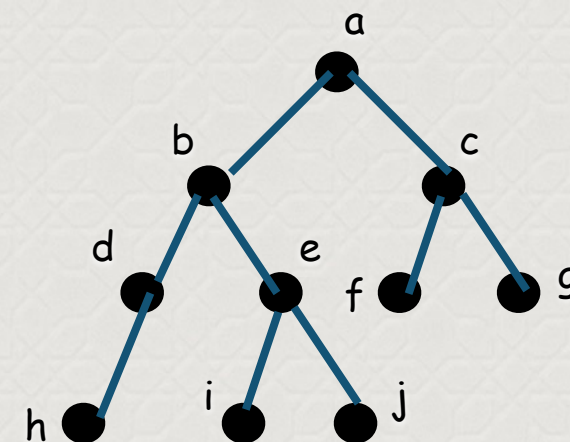
$s(c,f)$.

$s(c,g)$.

$s(d,h)$.

$s(e,i)$.

$s(e,j)$.





Os objetivo de busca serão especificados diretamente no programa (no caso geral uma função de interface pode ser inserida para permitir que um usuário defina os objetivos de busca) :

$goal(f).$
 $goal(j).$

$s(a,b).$

$s(a,c).$

$s(b,d).$

$s(b,e).$

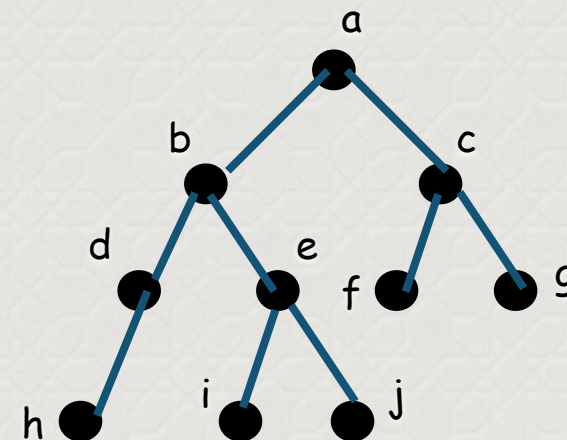
$s(c,f).$

$s(c,g).$


$s(d,h).$

$s(e,i).$

$s(e,j).$





File Edit Examples Help

111 users online

Search


Google


25

(new)

Tic-Tac-Toe Program Program

```
1 % solve( Node, Solution):  
2 %   Solution is an acyclic path (in reverse order) between Node and a goal  
3  
4 solve( Node, Solution) :-  
5   depthfirst( [], Node, Solution).  
6  
7 % depthfirst( Path, Node, Solution):  
8 %   extending the path [Node | Path] to a goal gives Solution  
9  
10 depthfirst( Path, Node, [Node | Path] ) :-  
11   goal( Node).  
12  
13 depthfirst( Path, Node, Sol) :-  
14   s( Node, Node1),  
15   \+ member( Node1, Path),           % Prevent a cycle  
16   depthfirst( [Node | Path], Node1, Sol).  
17  
18 depthfirst2( Node, [Node], _ ) :-  
19   goal( Node).  
20  
21 depthfirst2( Node, [Node | Sol], Maxdepth) :-  
22   Maxdepth > 0,  
23   s( Node, Node1),  
24   Max1 is Maxdepth - 1,  
25   depthfirst2( Node1, Sol, Max1).  
26  
27  
28 goal(f).  
29 goal(j).  
30 s(a,b).  
31 s(a,c).  
32 s(b,d).  
33 s(b,e).  
34 s(c,f).  
35 s(c,g).
```



 solve(a,X).

X = [j, e, b, a]
X = [f, c, a]

Next 10 100 1,000 Stop

?- solve(a,X).

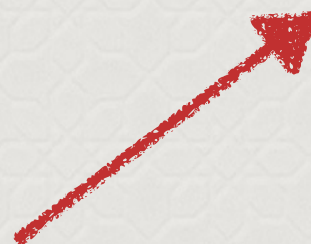
Examples History Solutions

☐ table results Run!



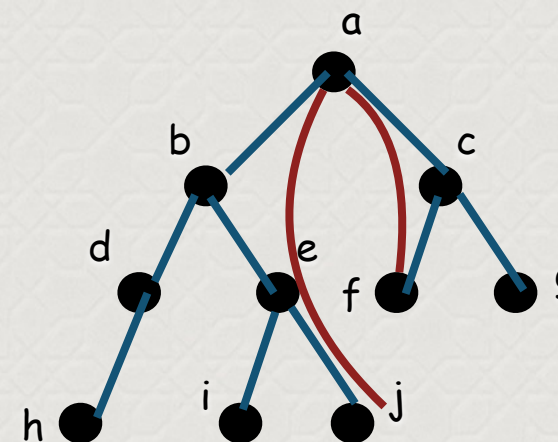
O resultado da busca é positivo, isto é o "goal" foi atingido, e o sistema devolve o caminho - na ordem invertida - desde o elemento alvo da busca até a raiz da árvore.

No caso do elemento "j" o caminho é a lista [j, e, b, a]. No caso do elemento "f" o caminho é a lista [f, c, a].



$goal(f).$
 $goal(j).$

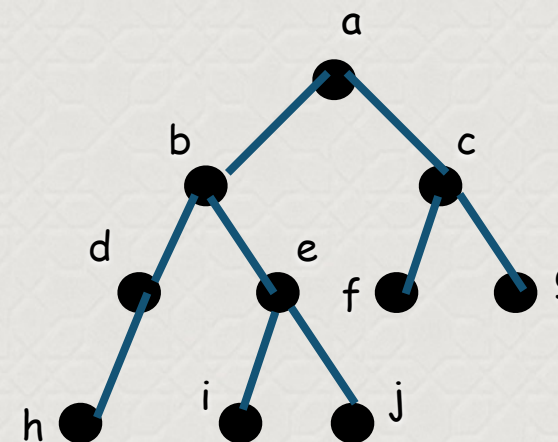
$s(a,b).$
 $s(a,c).$
 $s(b,d).$
 $s(b,e).$
 $s(c,f).$
 $s(c,g).$
 $s(d,h).$
 $s(e,i).$
 $s(e,j).$









Vamos agora inserir um novo goal que é um elemento que não consta da estrutura descrita ao lado. O que se espera é que o programa Prolog falhe, isto é, termine com False.

goal(z).
goal(f).
goal(j).
s(a,b).
s(a,c).
s(b,d).
s(b,e).
s(c,f).
s(c,g).
s(d,h).
s(e,i).
s(e,j).










File Edit Examples Help

105 users online25new

Tic-Tac-Toe Program Program +

```
1
2
3
4 solve( Node, Solution) :-
5   depthfirst( [], Node, Solution).
6
7 % depthfirst( Path, Node, Solution):
8 %   extending the path [Node | Path] to a goal gives Solution
9
10 depthfirst( Path, Node, [Node | Path] ) :-
11   goal( Node).
12
13 depthfirst( Path, Node, Sol) :-
14   s( Node, Node1),
15   \+ member( Node1, Path),           % Prevent a cycle
16   depthfirst( [Node | Path], Node1, Sol).
17
18 depthfirst2( Node, [Node], _) :-
19   goal( Node).
20
21 depthfirst2( Node, [Node | Sol], Maxdepth) :-
22   Maxdepth > 0,
23   s( Node, Node1),
24   Max1 is Maxdepth - 1,
25   depthfirst2( Node1, Sol, Max1).
26
27 goal(z).
28 goal(f).
29 goal(j).
30 s(a,b).
31 s(a,c).
32 s(b,d).
33 s(b,e).
34 s(c,f).
35 s(c,g).
36 s(d,h).
37 s(e,i).
38 s(e,j).
```



solve(a,X).

X = [j, e, b, a]
X = [f, c, a]
false

?- solve(a,X).

Examples History Solutions

☐ table results Run!

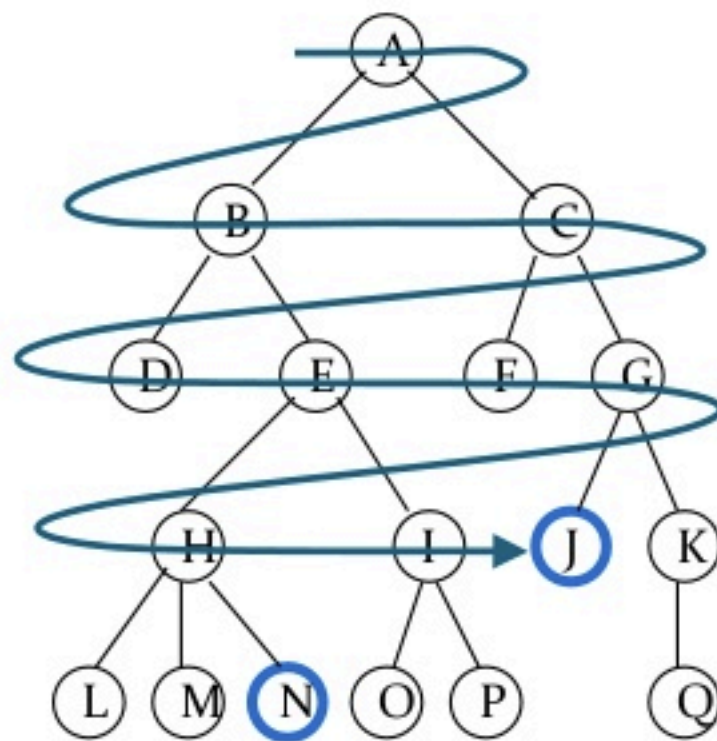


Outro algoritmo clássico de busca é a “busca em largura”. Nesse caso temos uma estrutura organizada de modo que os elementos procurados estão muito provavelmente antes de se atingir as folhas. Portanto se privilegia a busca no mesmo nível da árvore ao invés da busca em profundidade.



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
- Nodes 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) Brevemente a busca em largura ;
- ii) Os algoritmos A e A^* ;
- iii) busca bi-direcional;



Cronograma de entrega do trabalho em grupo

O cronograma de trabalho será dividido em "milestones":

Setembro 2019						
webcid.com.br						
Domingo	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					
7: Independência do Brasil 22: Início da primavera 06 - Quarto Crescente 14 - Lua Cheia 21 - Quarto Minguante 28 - Lua Nova						

Outubro 2019						
webcid.com.br						
Domingo	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		
20: Início do horário de verão 12: Nsa. Sra. Aparecida 15: Dia dos Professores 05 - Quarto Crescente 13 - Lua Cheia 21 - Quarto Minguante 28 - Lua Nova						

- 25/09 - início do processo, definição do domínio
- 02/10 - definição do algoritmo e implementação
- 09/10 - entrega do programa final
- 16/10 - competição



Até a próxima aula!