Escola Politécnica da USP - Depto. de Enga. Mecatrônica

PMR-3510 Inteligência Artificial
Aula 6 - Programação Lógica I

Prof. José Reinaldo Silva

reinaldo@usp.br

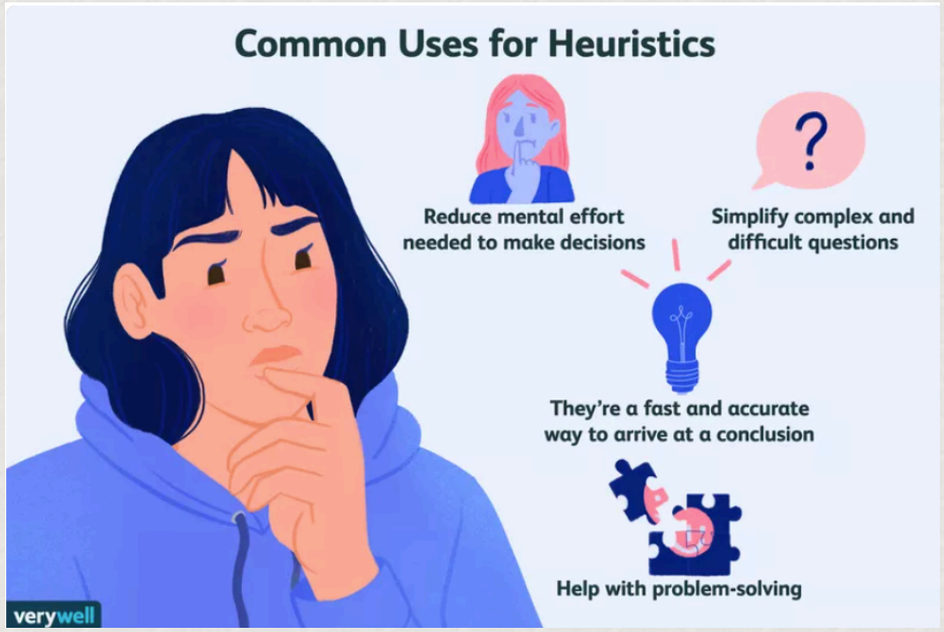
O algoritmo A*

O algoritmo A* é baseado no algoritmo de Dijkstra: uma varredura em um grafo é feita onde uma função de avaliação é usada para "escolher o melhor caminho". Esta função de avaliação é composta por uma função custo g(n) e uma heurística h(n),

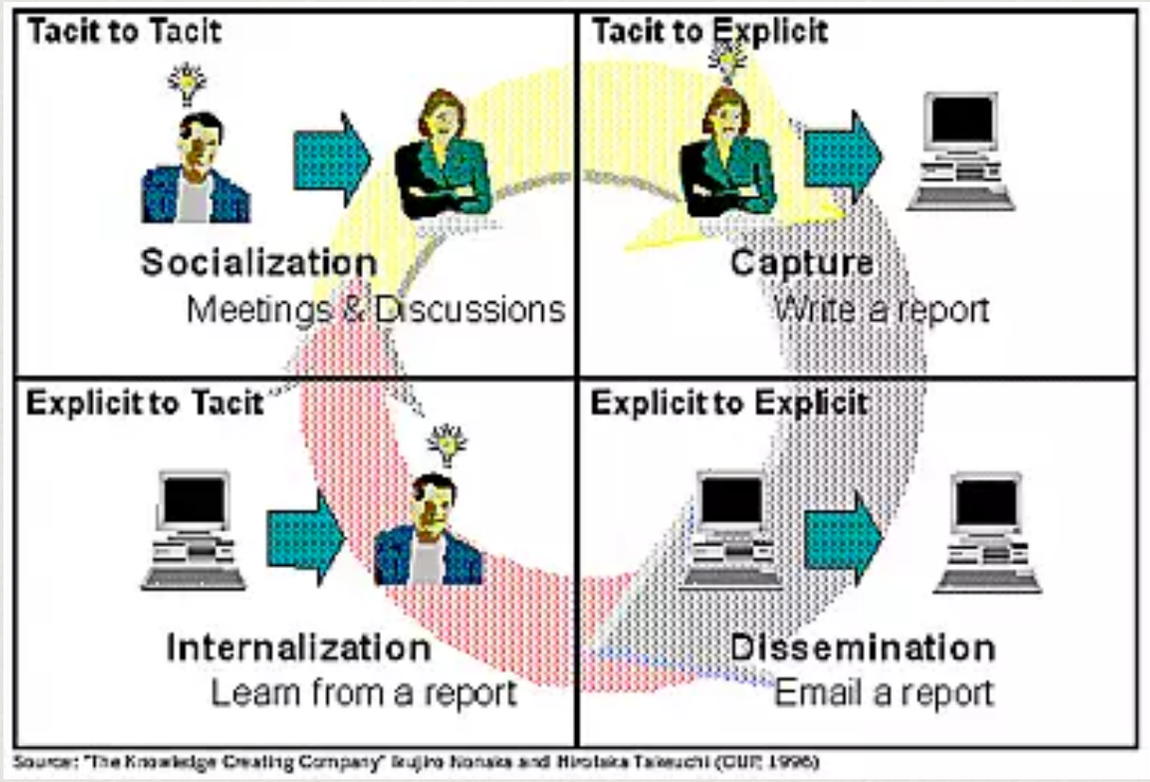
$$f(n) = g(n) + h(n)$$

No exemplo do jogo de tiles g(n)=p, onde p era o nível de profundidade do elemento visitado na árvore de busca.

Heuristics

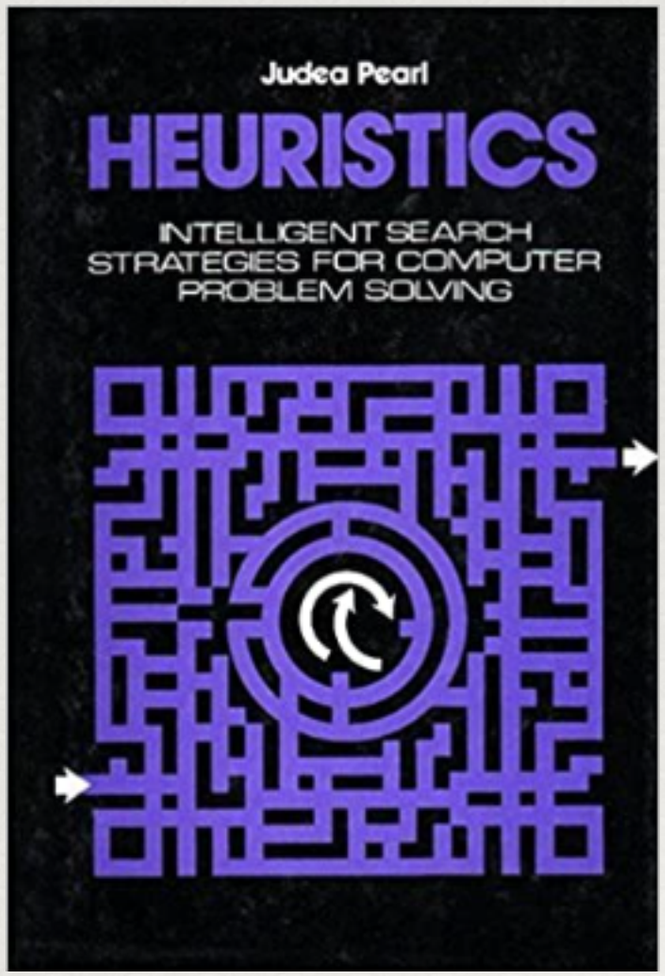


For Engineering purposes, a heuristic is a pattern extracted from the analysis of the "problem", that is, of this state space (if connected to problem-solving by search).



Como é usada na Engenharia (e em IA) a heurística ficaria entre o conhecimento tácito e o explícito (formal).

Judea Pearl se notabilizou pelo estudo teórico e prático das heurísticas, forma de aquisição e desenvolvimento. O interesse se justificava pela aplicação na resolução de problemas e apoio a decisão usando IA, usando métodos clássicos e estatística Bayesiana.



Judea Pearl

Human Heuristics for AI-Generated Language Are Flawed

Maurice Jakesch[1,2,*], Jeffrey T Hancock[3], Mor Naaman[1,2]

[1]Cornell University, [2]Cornell Tech, [3]Stanford University

* Corresponding author: mpj32@cornell.edu

Human communication is increasingly intermixed with language generated by AI. Across chat, email, and social media, AI systems produce smart replies, autocompletes, and translations. AI-generated language is often not identified as such but poses as human language, raising concerns about novel forms of deception and manipulation. Here, we study how humans discern whether one of the most personal and consequential forms of language – a self-presentation – was generated by AI. In six experiments, participants (N = 4,600) tried to detect self-presentations generated by state-of-the-art language models. Across professional, hospitality, and dating settings, we find that humans are unable to detect AI-generated self-presentations. Our findings show that human judgments of AI-generated language are handicapped by intuitive but flawed heuristics such as associating first-person pronouns, spontaneous wording, or family topics with humanity. We demonstrate that these heuristics make human judgment of generated language predictable *and* manipulable, allowing AI systems to produce language perceived as more human than human. We discuss solutions, such as AI accents, to reduce the deceptive potential of generated language, limiting the subversion of human intuition.

Keywords: human-AI interaction, language generation, cognitive heuristics, risks of AI

Author Contributions: M.J. conducted the study, analyzed the results, and wrote the initial draft of the manuscript. M.J. and M.N. developed the study design and research framework. All authors contributed to the final manuscript.

Competing Interest Statement: The authors declare that they have no conflict of interest.

Funding information: This material is based upon work supported by the National Science Foundation under Grant No. CHS 1901151/1901329 and the German National Academic Foundation.

A dificuldade (e também o "processo inteligente") na resolução de problemas é a identificação e desenvolvimento de heurísticas. Em alguns casos o processo é intuitivo (jogos, processos administrativos, processos sequenciais, etc.), em outros o problema pode ser comparável à busca não-informada ou pior.

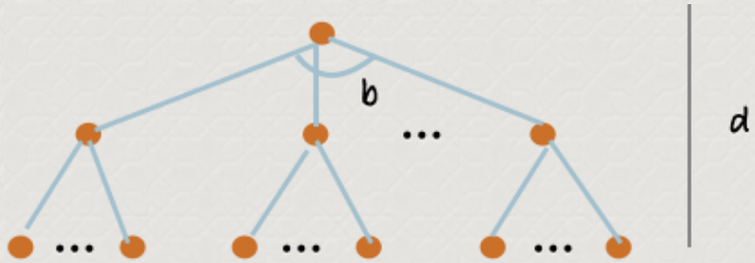
General idea

(Admissible) heuristic functions obtained as
(optimal) cost functions of relaxed problems

Examples

- Euclidian distance in Path Finding
- Manhattan distance in N-puzzle
- Spanning Tree in Traveling Salesman Problem
- Shortest Path in Job Shop Scheduling

O maior impacto provocado pela heurística é afetar o expoente d (a profundidade) no processo de busca de um fator k.



$$S = (b^d - 1)/(b - 1) \sim O(b^d)$$

	Search Cost (nodes generated)			Effective Branching Factor		
d	BFS	A*(h_1)	A*(h_2)	BFS	A*(h_1)	A*(h_2)
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

Figure 3.26 Comparison of the search costs and effective branching factors for 8-puzzle problems using breadth-first search, A* with h_1 (misplaced tiles), and A* with h_2 (Manhattan distance). Data are averaged over 100 puzzles for each solution length d from 6 to 28.

Platform Cloud Use cases Resources Docs Pricing FREE DOWNLOAD

https://memgraph.com/blog/graph-algorithms-applications

Categories

Graph Streaming

Applications of the 20 most popular graph algorithms

Benchmarking

Graphs for stream devs

Real-time Analytics

Real-time Data Sources

Streams for graph devs

Product

Tutorials

Python

Company

Graph Streaming

Applications of the 20 most popular graph algorithms

by Memgraph March 11, 2022

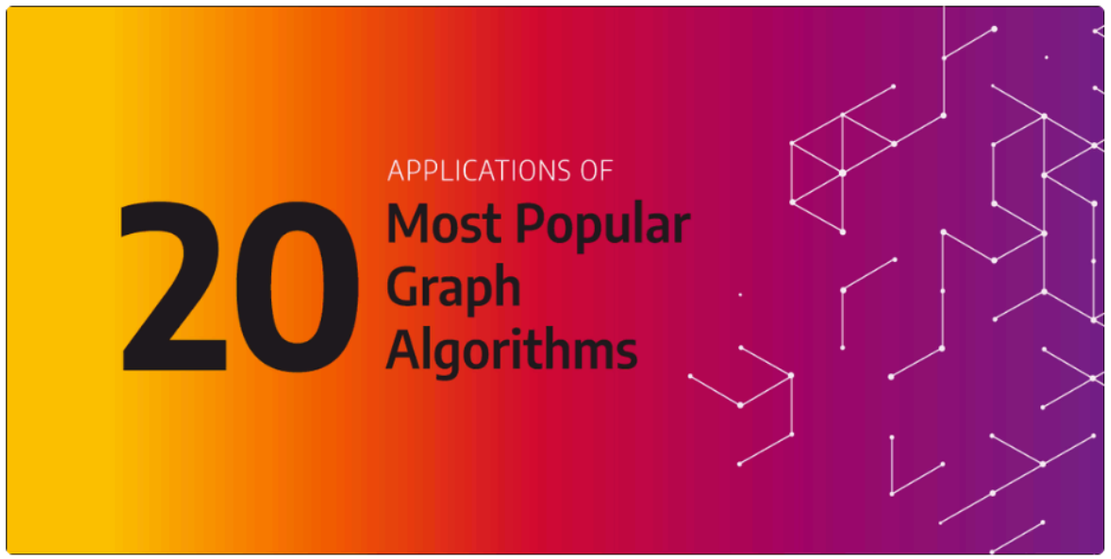


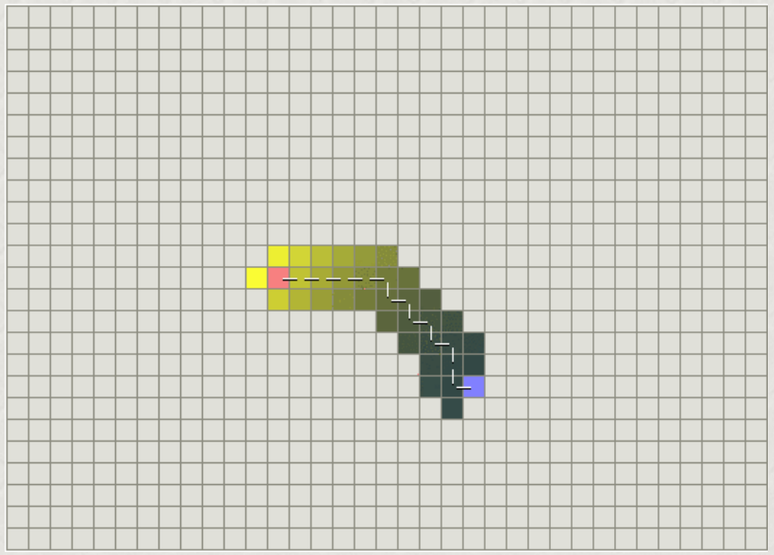
Table of Contents

Applications of A* algorithm

- It is commonly used in web-based maps and games to find the shortest path at the highest possible efficiency.
- A* is used in many artificial intelligence applications, such as search engines.
- It is used in other algorithms such as the Bellman-Ford algorithm to solve the shortest path problem.
- The A* algorithm is used in network routing protocols, such as RIP, OSPF, and BGP, to calculate the best route between two nodes.

... de olho no mercado!

Trajectory greed search with A*



SWISH File Edit Examples Help

204 users online

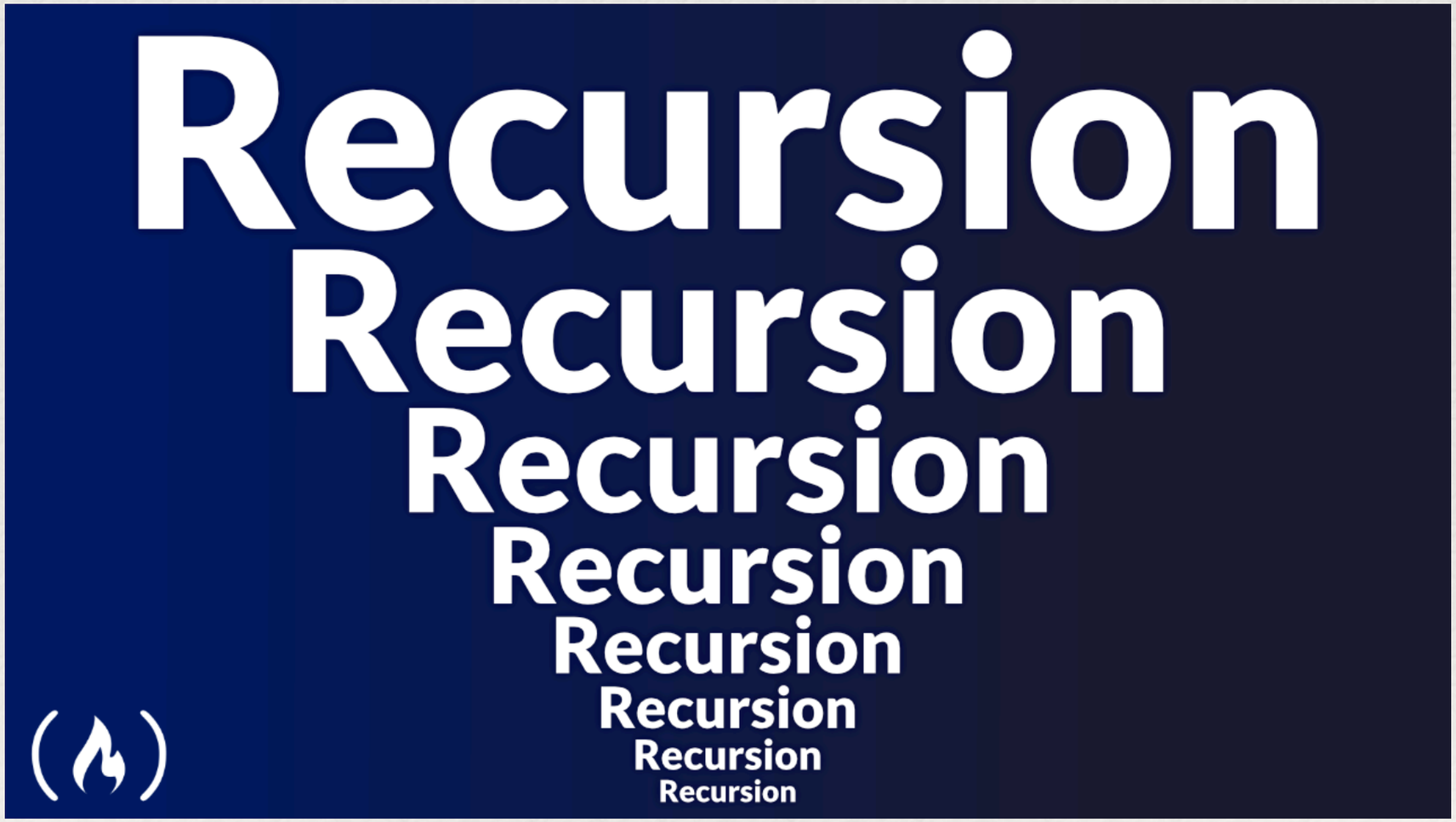
BuscaA.pl

consultorpc BuscaA

```
1  inv([],[]).
2  inv([E|R],S):-inv(R,L),append(L,[E],S).
3  % inicio a - fin f
4  arista(a,b,8).
5  arista(a,c,14).
6  arista(b,d,38).
7  arista(c,b,9).
8  arista(c,d,24).
9  arista(c,e,7).
10 arista(d,g,9).
11 arista(e,d,13).
12 arista(e,g,29).
13 arista(e,f,9).
14 h(a,g,30).
15 h(b,g,26).
16 h(c,g,21).
17 h(d,g,7).
18 h(e,g,22).
19 h(f,g,36).
20 h(g,g,0).
21
22 % [([N|V],D2,F)]: [N|V]= caminho , D2= costo real para alcanzar el nodo Objetivo
23 %               , F= estimativa de costo total
24 % V = Nodos visitados / N = Nodo filho extendido / O= Nodo objetivo
25 % DA = distancia recorrida actual / D = distancia recorrida actualizada
26 % S = caminos posibles / C = caminos encontrados / H = heuristica
27 % F= estimativa de costo total
28 gerarCamino([],_,_,S,S).
29 gerarCamino([(_,N,Z)|R],O,(V,DA),C,S):-D is DA+Z,h(N,O,H),F is D+H,
30                                        append(C,[([N|V],D,F)],L),gerarCamino(R,O,(V,DA),L,S).
31
32 % NE = Nodo filhos Extendidos / N = Nodo actual / O = Nodo objetivo
33 % V = Nodos visitados / D= distancia recorrida / S = caminos extendidos
34 extends filhos(N,O,(V,D, ),S):-findall((N,Y,Z),(arista(N,Y,Z),not(member(Y,V))),NE),
```

?- Your query goes here ...

Um recurso muito importante na programação declarativa é a recursão.



Uso da programação recursiva

Um algoritmo recursivo consiste em, problema P(n), onde n é o número de dados de entrada, basear a solução para "n" na solução do problema P(n-1), com uma condição de parada para P(1) (ou P(0)).

Uso da programação recursiva

A recursão pode ser usada em qualquer linguagem de programação mas é uma característica das linguagens funcionais e declarativas (Prolog, Lisp, etc.). As linguagens declarativas mais usadas hoje são o HTML, SQL e XML.

Linguagens funcionais

- Clojure
- Elixir
- Elm
- Erlang
- F#
- Haskell
- Kotlin
- PureScript
- Racket
- ReasonML
- Rust
- Scala
- Swift

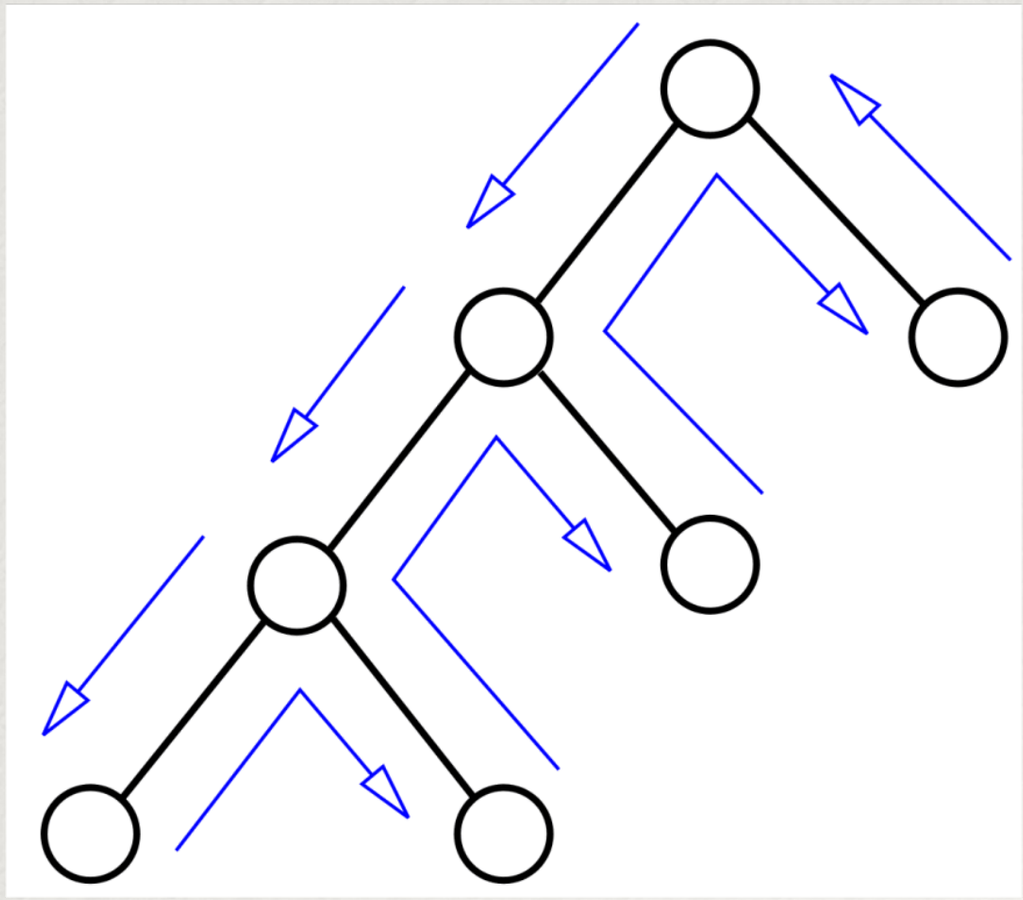
Programação declarativa

Na programação declarativa, as linguagens são mais apropriadas para lidar com objetivos gerais (além do estado inicial e das condições de contorno) buscando sempre uma estratégia geral. Os "passos" do programa não são determinados de forma sistêmica, e assume-se que o programa tem todos os elementos para achar uma solução (é fechado) se ela existir.

Principais linguagens declarativas:

- Prolog.
- Lisp.
- Haskell.
- Miranda.
- Erlang.
- SQL (in the broadest sense)

O uso do “stack" é importante para o backtracking.



Pilha

Qualquer que seja o algoritmo utilizado é preciso guardar a memória dos estados visitados, seja como resposta do problema, seja para conferir que não se visita o mesmo estado, configurando um loop.

Backtracking Example & Explanation :

Prolog Program

```
/* Facts */
likes(mary, food). // mary likes food.
likes(mary, wine). // mary likes wine

/* Rule */
likes(john, X):-likes(mary,X).
// john likes everything whatever mary like.

/* Goal */
?- likes(john, What).
// What does john like.
```

Query Prompt

Used 'trace' in SWI-Prolog to explain Backtracking Process.

```
?- trace.
true.

?- likes(john, What).
   Call: (6) likes(john, _G7400) ? creep
   Call: (7) likes(mary, _G7400) ? creep
   Exit: (7) likes(mary, food) ? creep
   Exit: (6) likes(john, food) ? creep
What = food ;
   Redo: (7) likes(mary, _G7400) ? creep
   Exit: (7) likes(mary, wine) ? creep
   Exit: (6) likes(john, wine) ? creep
What = wine ;

?- nodebug.
```

Resolver este problema usando IA, ou, programar um agente inteligente para resolver este problema (qualquer que seja o estado inicial) significa:

i) desenvolver a função de avaliação, função custo e heurística(s);
ii) verificar se a heurística é admissível e consistente e se a função custo é monotônica.
iii) Testar a função de avaliação em um caso simples;
iv) Fazer o programa em SWISH Prolog.

Sobre recursão

O exemplo mais clássico sobre recursão é o cálculo do fatorial de um inteiro n. Nesse caso temos que:

fat(1) = 1; (condição de parada)

fat(n) = n.fat(n-1);

...

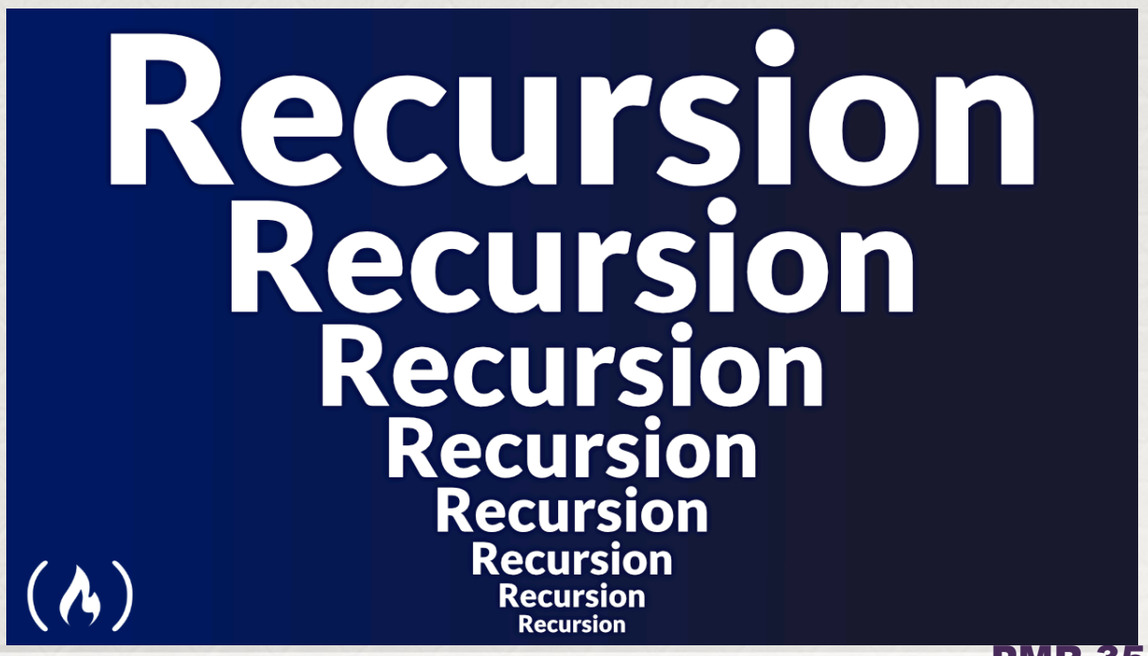
Processamento;

Chamada recursiva;

Processamento;

...

A cada chamada recursiva o argumento deve ser reduzido (ou decrementado) de modo a convergir para a condição de parada



SWISH File Edit Examples Help

Program Program +

```
1  /* ============================================
2   * Exemplo de programação recursiva: cáldulo do fatorial
3   *
4   */
5
6
7
8  fatorial(1,1).
9  fatorial(X,Y):- K is X -1, fatorial(K, Z), Y is (X * Z).
10
11 input:- write("Ente com um inteiro positivo"), read(N), fatorial(N,X), write(X).
```

```
8  fatorial(1,1).
9  fatorial(X,Y):- K is X -1, fatorial(K, Z), Y is (X * Z).
10
11 input:- write("Ente com um inteiro positivo"), read(N), fatorial(N,X), write(X).
```

input.

Ente com um inteiro positivo

25

15511210043330985984000000

true 1

Next | 10 | 100 | 1,000 | Stop

```
?- input.
```

Examples▲ | History▲ | Solutions▲ | table results | Run!

Sobre algoritmos e programação em IA:

i) Uma característica das aplicações práticas da IA é a flexibilidade e captura do processo cognitivo por trás dos frameworks, agentes e sistemas;
ii) Isso implica que vai ser preciso alguma programação seja para desenvolver o sistema de base, seja para prover as adaptações;
iii) Nesse momento pode fazer uma diferença razoável ter a possibilidade de usar programação declarativa, exceto quando se tratar de análise de dados.

Linguagens de programação mais usadas em IA:

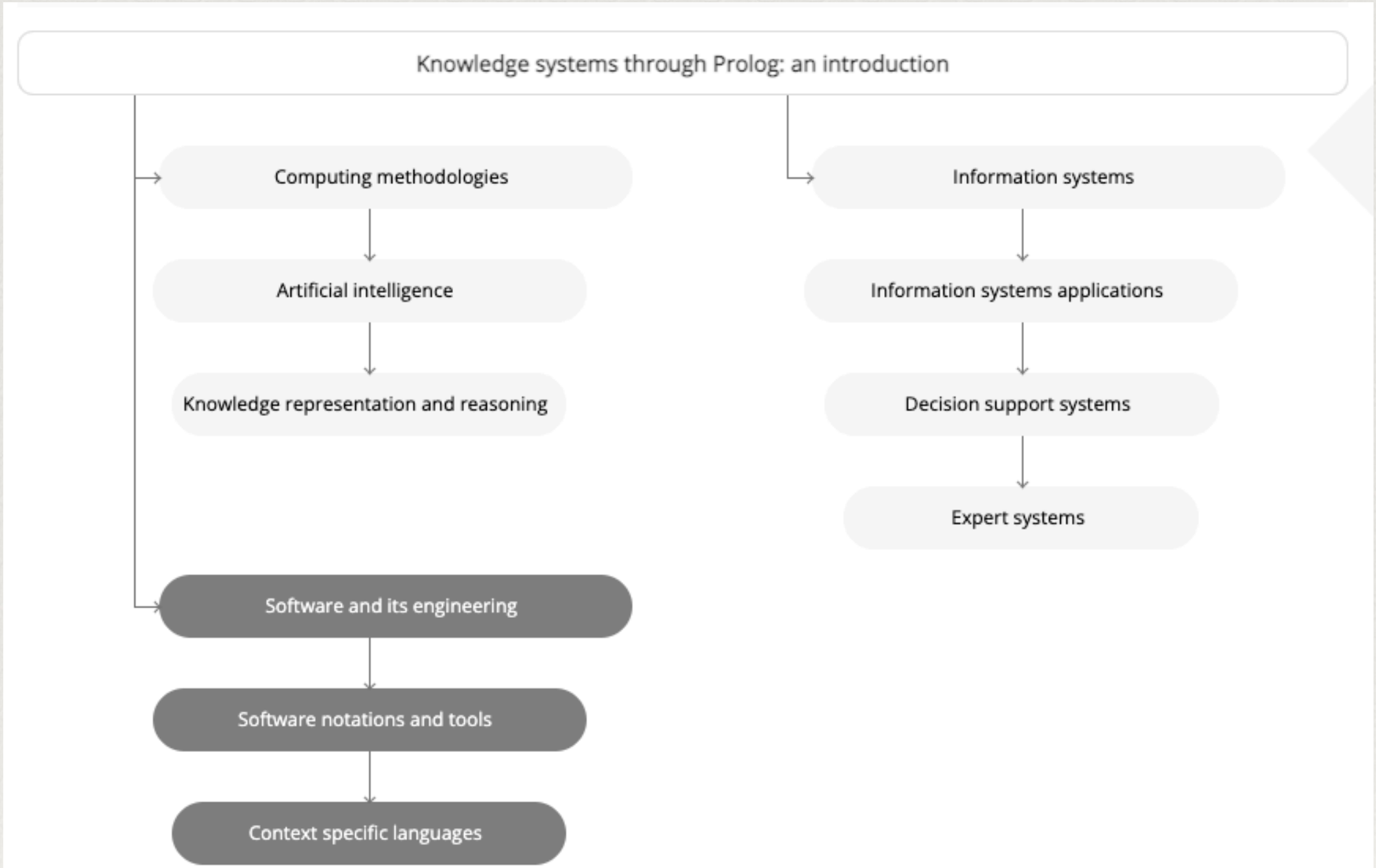
- Python
- C++
- Java
- LISP
- **Prolog**

- Few developers are well acquainted with Lisp programming.
- Being a vintage programming language artificial intelligence, LISP requires configuration of new software and hardware to accommodate it use.

PROLOG

Prolog is also one of the oldest programming languages thus also suitable for the development of programming AI. Like Lisp, it is also a primary computer language for artificial intelligence. It has mechanisms that facilitate flexible frameworks developers enjoy working with. It is a rule-based and declarative language as it contains facts and rules that dictate its artificial intelligence coding language.

Prolog supports basic mechanisms such as pattern matching, tree-based data structuring, and automatic backtracking essential for AI programming. Other than its extensive use in AI projects, Prolog is also used for creation of medical systems.


Knowledge systems through Prolog: an introduction
Computing methodologies
Artificial intelligence
Knowledge representation and reasoning
Information systems
Information systems applications
Decision support systems
Expert systems
Software and its engineering
Software notations and tools
Context specific languages

Search projects

Help Sponsors Log in Register

pylog 1.1

```
pip install pylog
```

Latest version

Released: Sep 7, 2019

Python implementation of Prolog features.

Navigation

- Project description
- Release history
- Download files

Project links

- Homepage

Project description

Abstract (outline from The Programming Journal)

Context: What is the broad context of the work? What is the importance of the general research area?

Pylog inhabits three programming contexts.

a) Pylog explores the integration of two distinct programming language paradigms: (i) the modern general purpose programming paradigm, which often includes features of procedural programming, object-oriented programming, functional programming, and meta-programming, here represented by Python, and (ii) logic programming, whose primary features are logic variables (and unification) and built-in depth-first search, here represented by Prolog. These logic programming feature are generally missing from modern general purpose languages. Pylog illustrates how these two features can be implemented in and integated into Python.

A partir da próxima aula discutiremos como elaborar programas e sistemas de uso prático com um conteúdo cognitivo e de resolução de problemas mais profundo, portanto mais a programação "declarativa".

Não deixe o exercício-programa para a última hora.

```prolog
% Simple Prolog Planner for the 8 Puzzle Problem

% This predicate initialises the problem states. The first argument
% of solve/3 is the initial state, the 2nd the goal state, and the
% third the plan that will be produced.

test(Plan):-
    write('Initial state:'),nl,
    Init= [at(tile4,1), at(tile3,2), at(tile8,3), at(empty,4), at(tile2,5), at(tile6,6), at(tile5,7), at(tile1,8), at(tile7,9)],
    write_sol(Init),
    Goal= [at(tile1,1), at(tile2,2), at(tile3,3), at(tile4,4), at(empty,5), at(tile5,6), at(tile6,7), at(tile7,8), at(tile8,9)],
    nl,write('Goal state:'),nl,
    write(Goal),nl,nl,
    solve(Init,Goal,Plan).

solve(State, Goal, Plan):-
    solve(State, Goal, [], Plan).

%Determines whether Current and Destination tiles are a valid move.
is_movable(X1,Y1) :- (1 is X1 - Y1) ; (-1 is X1 - Y1) ; (3 is X1 - Y1) ; (-3 is X1 - Y1).


% This predicate produces the plan. Once the Goal list is a subset
% of the current State the plan is complete and it is written to
% the screen using write_sol/1.

solve(State, Goal, Plan, Plan):-
    is_subset(Goal, State), nl,
    write_sol(Plan).

solve(State, Goal, Sofar, Plan):-
    act(Action, Preconditions, Delete, Add),
    is_subset(Preconditions, State),
    \+ member(Action, Sofar),
    delete_list(Delete, State, Remainder),
    append(Add, Remainder, NewState),
    solve(NewState, Goal, [Action|Sofar], Plan).
```

Equipe 1

Bernardo Moredo Rocco
Victor Benito Pereira
Rahul Casanovas

Equipe 2

João Victor Lins Fregnan
Victor Kendy Kamia
Vinicius Araújo da Costa

Equipe 3

Lucas Nascimento Tulha
Marcelo Alonso Cordova
Vidal Gonzalo Rojas

Equipe 4

Daniel Willian Domingues
Gabriel Antonio Ken Chang
Mauricio Hiroki Tomida
Maykon Souza Cruz

Equipe 5

Giulia Duo Cardella
Guilherme Rodrigues Monteiro
Luis Fernando Ferreira da Silva
Yae Do Choi

Equipe 6

Gabriela Fonseca Fanucchi
Guilherme Henrique de Oliveira
Yuri Lopes Pamplona

Equipe 7

Lucas Oliveira Reis
Pablo Nabil Villar Bou Assi
Samuel Flávio de Paiva Araújo
Vinicius Rocha Paiva

Estas equipes serão cadastradas hoje no e-disciplinas. Assim, não será mais possível abrir novas equipes ou inserir novos membros.

Perguntas?