



INTELIGÊNCIA ARTIFICIAL

Prof. Msc. Marcos Alexandruk

alexandruk@uni9.pro.br

<https://github.com/alexandruk/ia>



Aula 01 (parte 1)

Introdução

O que é Inteligência Artificial?

A Inteligência Artificial é uma ciência relativamente recente.

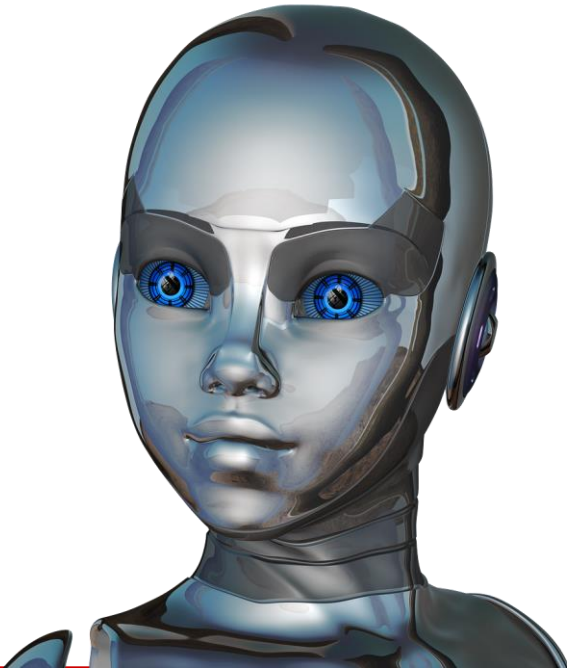
O termo I.A. (Inteligência Artificial) foi usado pela primeira vez em 1956.

A Inteligência Artificial sistematiza e automatiza as tarefas através da percepção, compreensão, prevenção e manipulação de situações voltadas para a resolução de problemas reais.

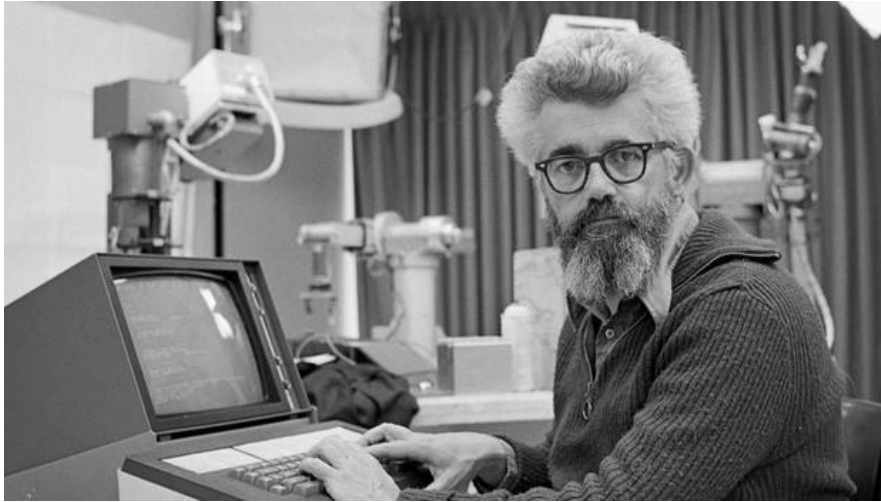
Atualmente a Inteligência Artificial envolve vários subcampos:

- aprendizado de máquina
- percepção
- tarefas específicas para resolução de problemas
- demonstração de teoremas matemáticos
- criação e arte
- diagnóstico de doenças, etc.

Inteligência Artificial é uma ciência relevante para todas as esferas da atividade intelectual humana.



John McCarthy



(Foto: Reprodução/Stanford)

John McCarthy, professor de matemática do Dartmouth College, em New Hampshire, é apontado como autor do termo Inteligência Artificial (I.A.).

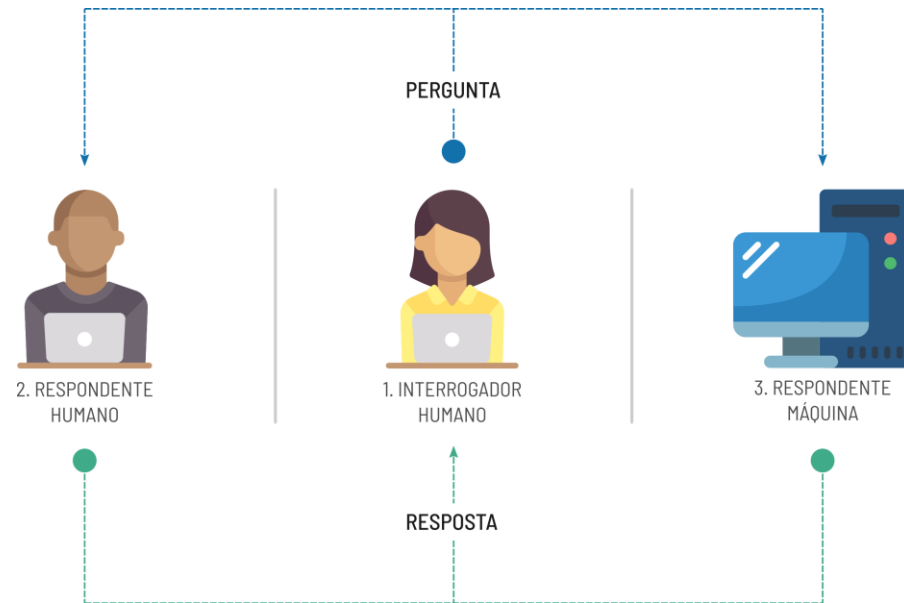
Criador da linguagem de programação **Lisp** (uma das primeiras linguagens aplicadas à Inteligência Artificial).

A linguagem Lisp é interpretada.

O usuário digita expressões em uma linguagem formal definida e recebe de volta a avaliação de sua expressão.

A PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE
Disponível em: <http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html>

Teste de Turing



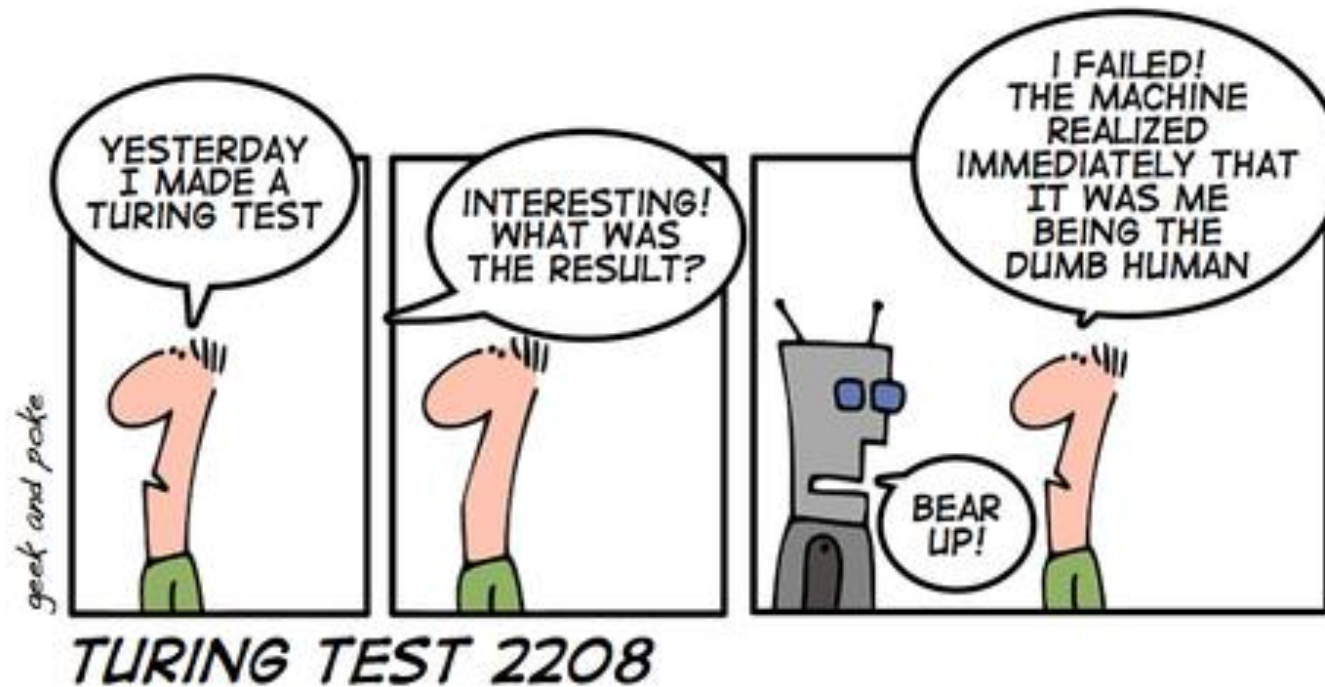
Alan Turing publicou em 1950 um artigo chamado "Computing Machinery and Intelligence" na revista filosófica Mind e propôs um teste com a pretensão de provar que um computador é inteligente.

O computador passaria no teste se um interrogador humano, depois de submeter algumas questões por escrito, não conseguisse identificar se as respostas foram escritas por uma pessoa ou por um computador. (O teste evita totalmente a interação física direta entre o interrogador e o computador.)

Para que um computador passe no teste, é necessário o desenvolvimento de um sistema que tenha minimamente os seguintes recursos:

- Processamento de linguagem natural;
- Representação do conhecimento;
- Raciocínio automatizado;
- Aprendizado de máquina.

Teste de Turing



Allan Turing: O Jogo da Imitação - Trailer (2015)



Teste de Turing

Aplicativos atuais estão muito próximo de serem utilizados com uma interface **homem-máquina** que simularia, quase que totalmente, a interação **homem-homem**.

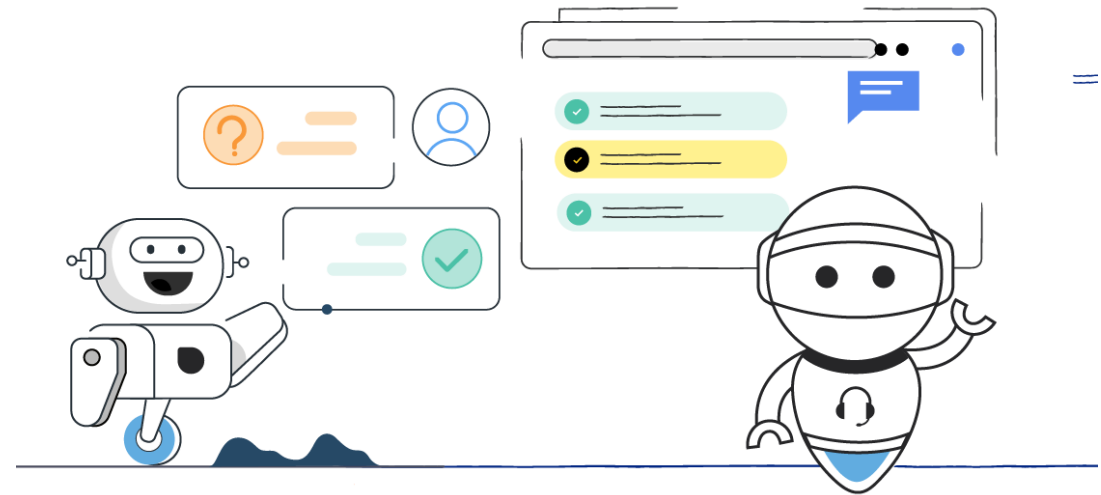
Chatbots

Um chatbot é um software usado para simular interações com humanos por meio de texto ou voz. Para isso, esses chatbots são treinados com palavras específicas predefinidas, com o objetivo de responderem à alguma pergunta que contenha uma dessas palavras já conhecida pelo software.

<https://www.ibm.com/br-pt/cloud/watson-assistant>

Exemplo:

<https://www.smatbot.com/>



Definição de IA conforme frame de quatro categorias

Sistemas que pensam como seres humanos

"O novo e interessante esforço para fazer os computadores pensarem...máquinas com mentes, no sentido total e literal." (Haugeland, 1985)

"[Automatização de] atividades que associamos ao pensamento humano, atividades como a tomada de decisões, a resolução de problemas, o aprendizado..." (Bellman, 1978)

Sistemas que pensam racionalmente

"O estudo das faculdades mentais pelo uso de modelos computacionais."
(Charniak e McDermott, 1985)

Sistemas que atuam como seres humanos

"A arte de criar máquinas que executam funções que exigem inteligência quando executadas por pessoas".
(Kurzweil, 1990)

Sistemas que atuam racionalmente

"A Inteligência Computacional é o estudo do projeto de agentes inteligentes."
(Poole et al., 1998)

Fundamentos da Inteligência Artificial

A I.A. é multidisciplinar. As principais áreas do conhecimento que influenciam essa ciência são:

Filosofia

- Dualismo
- Materialismo
- Indução
- Teoria da confirmação

Matemática

- Regras formais e lógica
- Algoritmo
- Teorema da incerteza
- Probabilidade

Economia

- Teoria da decisão
- Teoria dos jogos
- Pesquisa Operacional

Neurociência

- Neurônios

Psicologia

- Behaviorismo
- Psicologia Cognitiva
- Ciência Cognitiva

Engenharia da computação

- Máquina programável
- Máquina Analítica
- Sistemas Operacionais
- Linguagens de programação

Teoria de controle e cibernética

- Teoria de controle
- Cibernética
- Função Objetivo

Linguística

- Linguística computacional

Unimate: O primeiro robô industrial (1961)



Eliza: O primeiro Chatbot (1966)

```
Welcome to
EEEEEE LL      IIII  ZZZZZZ  AAAAA
EE      LL      II    ZZ      AA  AA
EEEEEE LL      II    ZZ      AAAAAA
EE      LL      II    ZZ      AA  AA
EEEEEE LLLLLL  IIII  ZZZZZZ  AA  AA

Eliza is a mock Rogerian psychotherapist.
The original program was described by Joseph Weizenbaum in 1966.
This implementation by Norbert Landsteiner 2005.

ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

O software Eliza, criado por Joseph Weizenbaum no laboratório de Inteligência Artificial do MIT, foi o primeiro programa para **processamento de linguagem natural** da história.

A ideia básica é simular a conversação entre homem e máquina.

A principal implementação do programa mostra a simulação da conversa entre um paciente e seu psicólogo, na qual o usuário é o paciente e o software o psicólogo. Na época, até mesmo alguns acadêmicos acreditaram que o sistema poderia influenciar positivamente a vida de pessoas que sofrem com problemas psicológicos, sendo capaz de complementar o tratamento dos pacientes.

ALICE: Chatbot (1995)



O A.L.I.C.E. (Artificial Linguistic Internet Computer Entity), criado na Lehigh University por Richard S. Wallace, em 1995, é um dos chatbots mais populares da atualidade.

Faz parte do Projeto Pandora que envolve a criação de bots de todos os tipos, especialmente para bate-papo.

Desde página ALICE, o usuário pode iniciar uma conversa e dificilmente percebe que está falando com um robô.

Para "conversar" com o ALLICE acesse:

<https://www.pandorabots.com/pandora/talk?botid=b8d616e35e36e881>

ALVINN: Veículo autônomo (1989)



O ALVINN (Autonomous Land Vehicle In a Neural Network) obtém imagens de uma câmera e um telêmetro a laser como entrada e produz como saída a direção que o veículo deve viajar para seguir a estrada.

O treinamento foi conduzido usando imagens de estradas simuladas.

Honda Asimo: Humanoide (2000)

The image shows the logo for 'Auto Express'. The word 'Auto' is in a large, bold, italicized sans-serif font. Below it, the word 'EXPRESS' is in a smaller, bold, italicized sans-serif font. Both words are white with a slight drop shadow, set against a solid red rectangular background.

Google AutoDraw (2017)



AutoDraw

Fast drawing for everyone.

Ferramenta composta por um sistema robusto de inteligência artificial, e à medida que é usada, fica mais inteligente. Aumenta aos poucos a capacidade de reconhecer novos desenhos e formatos corretamente.

Para usar basta acessar www.autodraw.com e clicar em 'Start Drawing'. Depois, selecione a ferramenta de desenho e crie uma forma à mão livre.

Feito isso, na parte superior da tela, há uma barra com os desenhos sugeridos pela inteligência artificial do Google com base no rascunho inicial.

Selecionar uma das opções e o desenho é automaticamente alterado conforme a opção selecionada. O resultado pode ser compartilhado nas redes sociais.

Principais áreas da Inteligência Artificial

Robótica

Reconhecimento de Padrões

Visão Computacional

Bases de Dados Inteligentes

Processamento de Liguagem
Natural

Prova de Teoremas

Sistemas Especialistas

Jogos

Resumo da aula

A Inteligência Artificial é uma área relativamente recente na ciência e tem como subcampos o aprendizado de máquina, percepção, tarefas específicas de resolução de problemas, demonstração de teoremas matemáticos, criação e arte, diagnóstico de doenças, etc.

Alan Turing propôs um teste com a pretensão de provar que um computador é inteligente (Teste de Turing).

Definições de Inteligência Artificial conforme o framework de quatro categorias:

- Sistemas que pensam como seres humanos;
- Sistemas que pensam racionalmente;
- Sistemas que atuam como seres humanos;
- Sistemas que atuam racionalmente.

A Inteligência Artificial é multidisciplinar e abrange campos da Filosofia, Matemática, Economia, Neurociência, Psicologia, Engenharia da Computação, Teoria de Controle e Cibernética e Linguística.

Referências

Braga, A. P.; Ludermir, T. B.; Carvalho, A. P. **Redes Neurais Artificiais - Teoria e aplicações**. Rio de Janeiro: LTC, 2000.

Kovács, Z. L. **Redes Neurais Artificiais - Fundamentos e Aplicações**. São Paulo: Edição Acadêmica São Paulo, 1996.

Michael, N. **Artificial Intelligence - A guide to Intelligent Systems**. Edinburgh: Pearson, 2002.

Minsky, M. **The Emotional Machine**. New York: Simon & Schuster, 2006.

Rich, E.; Kevin, K. **Inteligência Artificial**. São Paulo: Makron Books, 1993.

Suart, R.; Peter, N. **Inteligência Artificial**. Rio de Janeiro: Elsevier, 2004.

Vídeos:

O Jogo da Imitação Trailer Oficial (2015). Disponível em: https://www.youtube.com/watch?v=YIkKbMcJL_4&t=68s

Unimate Robot. <https://www.youtube.com/watch?v=hxsWeVtb-JQ>

Honda's Asimo: the penalty-taking, bar-tending robot. Disponível em: <https://www.youtube.com/watch?v=QdQL11uWWcl>



Aula 01 (parte 2)

Introdução

Linguagens evoluem para suportar o desenvolvimento da IA

Artigo escrito por M. Tim Jones para Mouser Electronics: Languages Evolve to Support AI Development

Disponível em: <https://br.mouser.com/blog/languages-evolve-to-support-ai-development> (inglês)

Disponível em: <https://www.embarcados.com.br/linguagens-de-programacao-inteligencia-artificial/> (português)

"No início, a Inteligência Artificial era criada predominantemente utilizando a linguagem de programação Lisp (LISt Processor) em hardware dedicado que executava operações primitivas. Lisp foi uma das primeiras linguagens e era eficiente no processamento de listas de itens. Os computadores de uso geral então se tornaram mais populares e os modelos de programação seguiram o exemplo. Porém, com o ressurgimento do aprendizado de máquina e, em particular, do aprendizado profundo, novas abordagens e toolkits de software otimizam esses fluxos de dados. Aqui, vamos explorar a confluência de aprendizagem de máquina e plataformas de software."

M. Tim Jones. **Languages Evolve to Support AI Development.**

Disponível em: <https://br.mouser.com/blog/languages-evolve-to-support-ai-development>. Acesso em: 07 ago 2021

Breve História da Inteligência Artificial

"Inteligência artificial e Lisp estavam inexoravelmente entrelaçados porque o conceito e a linguagem se originaram da mesma pessoa, John McCarthy (1927-2011). Em sua forma mais antiga, a Inteligência Artificial concentrava-se mais na pesquisa e no processamento simbólico do que nas abordagens numéricas que dominam a área atualmente. Lisp, com sua capacidade de representar dados complexos de forma simples e natural e seu uso de recursão (que é usado para iteração e pesquisa) o tornou ideal para muitos problemas da época. E com seu interpretador interativo – chamado REPL ou Read Evaluate Print Loop – Lisp tornou a programação exploratória mais fácil, o que era ideal para resolver problemas que não eram totalmente compreendidos.

Mas o poder da linguagem de programação Lisp também foi seu maior detrator; seu estilo funcional de programação era difícil e abriu a porta para novos paradigmas de linguagem de programação. E embora a programação funcional continue a ser usada hoje, as linguagens imperativas, orientadas a objetos e multiparadigma são mais comuns atualmente."

M. Tim Jones. **Languages Evolve to Support AI Development.**

Disponível em: <https://br.mouser.com/blog/languages-evolve-to-support-ai-development>. Acesso em: 07 ago 2021

Linguagens Modernas de Inteligência Artificial

"Embora seja possível desenvolver aplicativos para Inteligência Artificial em qualquer linguagem de programação, algumas são melhores do que outras. Seja a linguagem em si ou o suporte em torno da linguagem, certas linguagens simplificam muito o desenvolvimento de aplicações com Inteligência Artificial."

M. Tim Jones. **Languages Evolve to Support AI Development.**

Disponível em: <https://br.mouser.com/blog/languages-evolve-to-support-ai-development>. Acesso em: 07 ago 2021

Lógica de Programação

"A linguagem Prolog foi introduzida em 1972 e tem suas raízes na lógica de primeira ordem, onde os programas são definidos por fatos e regras. O programa pode ser consultado para aplicar as regras sobre os fatos e produzir um resultado. O Prolog continua sendo amplamente utilizado hoje para aplicações como sistemas especialistas e sistemas de planejamento automatizados. O Prolog foi originalmente projetado para processamento de linguagem natural e continua a encontrar aplicações nesta área."

M. Tim Jones. **Languages Evolve to Support AI Development.**

Disponível em: <https://br.mouser.com/blog/languages-evolve-to-support-ai-development>. Acesso em: 07 ago 2021

Linguagens de programação de uso geral

"Vinte anos após a introdução do **Prolog**, surgiu uma linguagem de propósito geral chamada Python, que foi projetada em torno da legibilidade do código. Embora Python tenha ganhado interesse desde o início como uma linguagem educacional para ensinar programação, ele expandiu em uma linguagem amplamente usada em vários domínios, incluindo inteligência artificial e aprendizado de máquina. Uma das principais vantagens do Python é seu enorme conjunto de bibliotecas e toolkits que tornam a construção de aplicativos mais simples. Por exemplo, Python pode ser usado com o toolkit de código aberto TensorFlow para criar aplicações com técnicas de aprendizado profundo. Isso é benéfico quando você deseja implantar o aprendizado profundo sem desenvolver as estruturas de rede neural profunda detalhadas, que antes seriam necessárias."

M. Tim Jones. **Languages Evolve to Support AI Development.**

Disponível em: <https://br.mouser.com/blog/languages-evolve-to-support-ai-development>. Acesso em: 07 ago 2021

Computação Estatística

"Um modelo semelhante foi usado na **linguagem R**, que é uma linguagem e um ambiente para computação estatística com representação gráfica. R é uma linguagem altamente extensível que é expandida por meio da integração de pacotes. Os pacotes coletam funções e dados juntos para alguma aplicação específica que pode então ser usado em programas R, como funções estatísticas ou toolkits completos de aprendizado profundo. Em 2020, mais de 15.000 pacotes estão disponíveis para a linguagem R."

M. Tim Jones. **Languages Evolve to Support AI Development.**

Disponível em: <https://br.mouser.com/blog/languages-evolve-to-support-ai-development>. Acesso em: 07 ago 2021

Novas abordagens

"Embora Lisp seja predominantemente uma nota de rodapé no aprendizado de máquina hoje, suas raízes funcionais geraram novas linguagens que seguem esse paradigma. A linguagem **Haskell** é uma linguagem puramente funcional com um sistema de tipos forte que resulta em um código mais seguro; uma característica útil quando se considera o aprendizado de máquina é a explosão dos dispositivos da Internet das Coisas. Embora não tenha o amplo conjunto de bibliotecas disponíveis para **Python** e **R**, a linguagem Haskell inclui vinculação para kits de ferramentas de aprendizado de máquina tornando simples construir aplicações de aprendizado de máquina com essa linguagem."

M. Tim Jones. **Languages Evolve to Support AI Development.**

Disponível em: <https://br.mouser.com/blog/languages-evolve-to-support-ai-development>. Acesso em: 07 ago 2021

Toolkits

"Junto com as linguagens, os toolkits e as bibliotecas também evoluíram na busca por aplicações de aprendizado de máquina. Esses toolkits, como o **TensorFlow**, fornecem recursos para linguagens para criar aplicações complexas utilizando aprendizado de máquina sem construir esses recursos do zero. O TensorFlow fornece interfaces para várias linguagens, como **Python**, **Haskell** e **R** e simplifica a construção e implantação de aplicativos de aprendizado profundo."

M. Tim Jones. **Languages Evolve to Support AI Development.**

Disponível em: <https://br.mouser.com/blog/languages-evolve-to-support-ai-development>. Acesso em: 07 ago 2021

Conclusão

"O conceito de Inteligência Artificial e a formação de sua progênie numérica de aprendizado de máquina criaram uma coevolução de linguagens e toolkits. As linguagens fornecem os recursos de propósito geral para a construção de diversas aplicações, enquanto os toolkits expandem essas linguagens com recursos específicos de aprendizado de máquina."

M. Tim Jones. **Languages Evolve to Support AI Development.**

Disponível em: <https://br.mouser.com/blog/languages-evolve-to-support-ai-development>. Acesso em: 07 ago 2021



Aula 01 (parte 3)

Linguagem Lisp

Lisp

"Há dois dialetos principais de Lisp: Common Lisp e Scheme. Eles são baseados em muitas das mesmas idéias, mas ainda assim é diferente o suficiente para levantar debates intensos sobre qual é a melhor opção.

Common Lisp é um padrão ANSI que foi finalizado em 1991, consolidando idéia de muitas Lisps anteriores. É um grande ambiente desenhado para muitos tipos de desenvolvimento de aplicações, mais famosos para inteligência artificial. Scheme por outro lado vem de um mundo mais acadêmico, é deliberadamente mais minimalista e se tornou uma boa linguagem para ensinar ciência da computação e para scripts embutidos. Algumas outras Lisps bem conhecidas você pode rodar em pequenas **DSLs** de aplicações específicas, como Emacs Lisp ou AutoCAD AutoLISP."

***DSL** (Domain-Specific Language) é uma linguagem especializada em um determinado domínio de aplicativo. Por outro lado, uma linguagem GPL (General-Purpose Language), é amplamente aplicável em múltiplos domínios.*

Explorando LISP na JVM.

Disponível em: <https://www.infoq.com/br/articles/lisp-for-jvm/>. Acesso em: 10 ago 2021

Lisp online

O seguinte website disponibiliza um interpretador online para a linguagem Lisp:

https://www.tutorialspoint.com/execute_lisp_online.php

Operações aritméticas básicas

Adição

```
(print (+ 2 3)) ; retorna 5
```

```
(print (+ 2 3 5)) ; retorna 10
```

Subtração

```
(print (- 2 3))
```

Multiplicação

```
(print (* 2 3))
```

Divisão

```
(print (/ 2 3)) ; retorna 2/3
```

```
(print (/ 2.0 3)) ; retorna 0.6666667
```

Operações aritméticas básicas

Potenciação

```
(print (expt 2 3)) ; retorna 8
```

Radiciação

```
(print (sqrt 9)) ; retorna 3
```

Outras operações

Verifica se uma sequência de valores é descendente

`(print (> 5 3)) ; retorna T (True) porque 5 é maior que 3`

Verifica se uma sequência de valores é acendente

`(print (< 5 3)) ; retorna NIL porque 5 não é maior que 3`

`NIL a rigor indica uma lista vazia`

Máximo e mínimo

Apresenta o valor máximo de uma sequência de valores

```
(print (max 5 3 2)) ; retorna 5
```

Apresenta o valor mínimo de uma sequência de valores

```
(print (min 5 3 2)) ; retorna 2
```

Variáveis

```
(setf x 8) ; atribui 8 a variável x
```

```
(print x) ; imprime 8
```

```
(setf z "texto") ; atribui "texto" à variável z
```

```
(print z) ; imprime "texto"
```

```
(princ z) ; imprime texto
```

```
(setf x 3) ; atribui 3 a variável x
```

```
(setf y 4) ; atribui 4 a variável y
```

```
(print (+ x y)) ; imprime 7
```

Listas

```
(setf minhaLista '(java php python))
```

```
(print minhaLista)
```

```
(print (first minhaLista)) ; imprime java (primeiro elemento da lista)
```

```
(print (last minhaLista)) ; imprime python (último elemento da lista)
```

```
(setf minhaLista (cons 'lisp minhaLista)) ; adiciona lisp no início da lista
```

```
(print minhaLista)
```

Estruturas de decisão

```
(setq a 10) ; atribuindo valor à variavel a
```

```
(setq b 5) ; atribuindo valor à variável b
```

```
(if (equal a b) (print "igual") (print "diferente") ) ; verificando se a é igual a b
```

Estruturas de repetição

```
(let ((n 0))  
  (loop  
    (print n)  
    (setq n (1+ n))  
    (when (> n 100)  
      (return 'fim))))
```


Estruturas de repetição

A forma a seguir permite estabelecer variáveis, inicializá-las e incrementá-las automaticamente, testar condições de paragem com indicação do valor a retornar e repetir a execução de código.

```
(do ((n 0 (1+ n)))  
    ((> n 100) 'fim)  
  (print n))
```



Aula 02 (parte 1)

Lógica Proposicional e Lógica de Primeira Ordem

Lógica Proposicional

Uma **proposição** corresponde a uma **sentença bem definida**, isto é, que pode ser **classificada como verdadeira ou falsa**, excluindo-se qualquer outro julgamento.

Lógica Proposicional - Sintaxe

A sintaxe da lógica proposicional define as sentenças permitidas.

Uma proposição que pode ser verdadeira ou falsa.

Os nomes são aleatórios e normalmente são escolhidos através de um mnemônico que representa algum valor para o leitor.

Utilizamos frequentemente letras latinas maiúsculas para definir os símbolos proposicionais (P, Q, R, e assim por diante).

Sentenças atômicas consistem em um único símbolo proposicional.

Sentenças complexas são construídas utilizando-se conectivos lógicos.

Existem cinco conectivos lógicos de uso comum: \sim ou \neg , \wedge , \vee , \rightarrow e \leftrightarrow

Lógica Proposicional - Sintaxe

Conectivos Lógicos

símbolo	significado	descrição
\sim ou \neg	not (não)	Representa uma sentença atômica negada.
\wedge	and (e)	Representa uma conjunção.
\vee	or (ou)	Representa uma disjunção.
\rightarrow	implica	Representa uma implicação. Conhecidas também como declarações Se-Então.
\leftrightarrow	se e somente se	Representa uma sentença bicondicional.

Ordem de Precedência

\sim ou \neg	not (não)
\wedge	and (e)
\vee	or (ou)
\rightarrow e \leftrightarrow	implica e somente se

Lógica Proposicional - Semântica

Na lógica proposicional um modelo fixa o valor (**verdadeiro** ou **falso**) para cada símbolo proposicional.

Como o princípio verdadeiro ou falso é **binário**, para se calcular os modelos possíveis é necessário fazer a potência com base 2 (binária) elevado ao X (número de variáveis).

Assim, se tivermos 3 símbolos proposicionais possíveis, teremos $2^3 = 8$ modelos possíveis.

O cálculo que qualquer sentença é realizado conforme a tabela verdade apresentada a seguir.

Lógica Proposicional - Semântica

Tabela Verdade

P	Q	P or Q $P \vee Q$	P and Q $P \wedge Q$	$P \rightarrow Q$	$P \leftrightarrow Q$	$\sim P$	$\neg Q$
V	V	V	V	V	V	F	F
V	F	V	F	F	F	F	V
F	V	V	F	V	F	V	F
F	F	F	F	V	V	V	V

Lógica Proposicional - Semântica

Na Lógica Proposicional denomina-se:

Tautologia: Proposição logicamente verdadeira.

Contradição: Proposição logicamente falsa.

Contingência: A proposição não é logicamente verdadeira nem falsa.

Lógica Proposicional - Expressões

Resolver a Expressão:

$$A \vee [(\sim B) \wedge C]$$

1º passo

A	B	C			
V	V	V			
V	V	F			
V	F	V			
V	F	F			
F	V	V			
F	V	F			
F	F	V			
F	F	F			

Lógica Proposicional - Expressões

Resolver a Expressão:

$$A \vee [(\sim B) \wedge C]$$

2º passo

A	B	C	$\sim B$		
V	V	V	F		
V	V	F	F		
V	F	V	V		
V	F	F	V		
F	V	V	F		
F	V	F	F		
F	F	V	V		
F	F	F	V		

Lógica Proposicional - Expressões

Resolver a Expressão:

$$A \vee [(\sim B) \wedge C]$$

3º passo

A	B	C	$\sim B$	$\sim B \wedge C$	
V	V	V	F	F	
V	V	F	F	F	
V	F	V	V	V	
V	F	F	V	F	
F	V	V	F	F	
F	V	F	F	F	
F	F	V	V	V	
F	F	F	V	F	

Lógica Proposicional - Expressões

Resolver a Expressão:

$$A \vee [(\sim B) \wedge C]$$

4º passo

A	B	C	$\sim B$	$\sim B \wedge C$	$A \vee [(\sim B) \wedge C]$
V	V	V	F	F	V
V	V	F	F	F	V
V	F	V	V	V	V
V	F	F	V	F	V
F	V	V	F	F	F
F	V	F	F	F	F
F	F	V	V	V	V
F	F	F	V	F	F

Lógica Proposicional - Modus Ponens

Modus Ponens (modo de afirmar).

Se uma declaração ou proposição implica uma segunda, e a primeira declaração ou proposição é verdadeira, então a segunda também é verdadeira.

Se P implica Q e P é verdadeira, então Q é verdadeira.

$P \rightarrow Q$ (P implica em Q)

P (P é verdadeiro)

$\therefore Q$ (então Q é verdadeiro)

\therefore significa logo, portanto, em conclusão ...

Exemplo:

Se está chovendo, nós assistiremos TV.

Está chovendo.

Então, nós assistiremos TV.

Lógica Proposicional - Modus Tollens

Modus tollens (modo de negar) ou negação do consequente, é o nome formal para a prova indireta, também chamado de modo apagógico.

Se P implica Q e P é verdadeira, então Q é verdadeira.

$P \rightarrow Q$ (P implica em Q)

$\neg P$ (P é falso)

$\therefore \neg Q$ (então Q é falso)

Exemplo:

Se existe fogo aqui, então aqui há oxigênio.

Não há oxigênio aqui.

Então aqui não há fogo.

Lógica Proposicional - Leis de Morgan

Definem as regras usadas para converter operações lógicas \vee (ou) em \wedge e vice versa:

$$\sim(p \vee q) = \sim p \wedge \sim q$$

$$\sim(p \wedge q) = \sim p \vee \sim q$$

Exercício:

Use as tabelas verdade para demonstrar as Leis de Morgan.

Lógica Proposicional - Leis de Morgan

$$\sim(p \vee q) = \sim p \wedge \sim q$$

p	q	(p ∨ q)	~(p ∨ q)	~p	~q	~p ∧ ~q
V	V	V	F	F	F	F
V	F	V	F	F	V	F
F	V	V	F	V	F	F
F	F	F	V	V	V	V

Lógica Proposicional - Leis de Morgan

$$\sim(p \wedge q) = \sim p \vee \sim q$$

p	q	(p ∧ q)	~(p ∧ q)	~p	~q	~p ∨ ~q
V	V	V	F	F	F	F
V	F	F	V	F	V	V
F	V	F	V	V	F	V
F	F	F	V	V	V	V

Lógica Proposicional - Calculadora Lógica Online

<https://calculode.com.br/calculadora-tabela-verdade-online-calculadora-logica/>

Lógica de Primeira Ordem

A Lógica de Primeira Ordem, conhecida também como Cálculo de Predicados de Primeira Ordem, é um sistema lógico que estende a Lógica Proposicional e que é estendida pela Lógica de Segunda Ordem.

Os elementos sintáticos básicos da lógica de primeira ordem são os símbolos que representam **objetos**, **relações** e **funções**.

Elas são representadas por três tipos: **símbolos de constantes**, **símbolos de predicados**, e **símbolos de funções**.

A primeira representa os **objetos**, a segunda as **relações** e a terceira as **funções**.

Lógica de Primeira Ordem - Sentenças

As **sentenças atômicas enunciam fatos**. Ela é formada por um símbolo de predicado, seguido por uma lista de termos entre parêntese.

Mãe (Vivian, Théo)

A sentença enuncia: **Vivian é mãe de Théo**

As sentenças podem ter termos mais complexos como argumentos, como por exemplo:

Casado(Pai(João), Mãe(José))

A sentença enuncia: **O pai de João é casado com a mãe de José**

Lógica de Primeira Ordem - Quantificadores

Como na lógica de Primeira Ordem podemos expressar objetos, por vezes é necessário expressar **coleções** inteiras de objetos. Com os **quantificadores** é possível realizar essa alocação.

Em lógica de Primeira Ordem, existem **dois quantificadores**, eles são chamados de **universal** e **existencial**.

Lógica de Primeira Ordem - Quantificador Universal

O **Quantificador Universal** é expresso pelo símbolo \forall , que significa "para todo" ou "para todos".

A sentença $\forall xP$, onde P é qualquer expressão lógica, afirma que **P é verdadeira para todo objeto x** .

Ele é utilizado para declarar o significado intuitivo da quantificação universal.

Lógica de Primeira Ordem - Quantificador Universal

Exemplo:

"Todo paulista é brasileiro."

Qualquer elemento que seja paulista será brasileiro.

Equivalente a: "Se é paulista então é brasileiro."

Simbologia:

Considerando **A** o conjunto dos paulistas, **B** o conjunto dos brasileiros e **x** elementos desses conjuntos, temos:

$$\forall x(A(x) \rightarrow (B(x)))$$

Lê-se: "Para todo (qualquer) elemento, se pertence ao conjunto A então pertence ao conjunto B."

Lógica de Primeira Ordem - Quantificador Existencial

O Quantificador Existencial faz a declaração sobre "algum objeto".

É expresso pelo símbolo \exists , que significa "Existe um x tal que" ou "Para algum x".

Dessa forma, a sentença $\exists xP$ afirma que P é verdadeira para pelo menos um objeto de x .

Conectivos:

\rightarrow é o conectivo natural para se utilizar com o \forall

\wedge é o conectivo natural para se utilizar com \exists

Lógica de Primeira Ordem - Quantificador Existencial

Exemplo:

"Existem brasileiros que gostam de funk."

Simbologia:

Considerando A o conjunto dos brasileiros, B o conjunto de quem gosta de funk e x elementos desses conjuntos, temos:

$$\exists x(A(x) \wedge (B(x)))$$

Lê-se: "Existe elemento que pertence ao conjunto A e pertence ao conjunto B."

Lógica de Primeira Ordem - Quantificador Existencial

Negação

Nenhum A é B \therefore Todo A não é B.

Exemplo:

"Nenhum palmeirense torce para o Corinthians. \therefore Todo palmeirense não torce para o Corinthians."

Simbologia:

Considerando **A** o conjunto dos palmeirenses, **B** o conjunto de quem torce para o Corinthians e **x** elementos desses conjuntos, temos:

$$\neg \exists x (A(x) \wedge B(x))$$

Lê-se: "Não existe elemento que pertence ao conjunto A e pertence ao conjunto B."



Aula 02 (parte 2)

Linguagem Prolog

Prolog

Linguagem Prolog foi criada em 1972, na Universidade de Marseille, França, por Alain Colmerauer e Philippe Roussel.

O nome Prolog para a linguagem concreta foi escolhido por Philippe Roussel como uma abreviação de "PROgrammation en LOGique".

Desde então tem sido utilizada para aplicações de computação simbólica, como banco de dados relacionais, compreensão de linguagens naturais, automação de projetos, análise de estruturas bioquímicas e sistemas especialistas.

Prolog se tornou uma referência quando se trata de linguagem de programação voltada para inteligência artificial e lingüística computacional.

Prolog

Prolog é uma linguagem declarativa, ao invés do programa estipular a maneira de chegar à solução passo-a-passo, como acontece nas linguagens procedimentais ou orientadas a objeto, ele fornece uma descrição do problema que se pretende resolver utilizando uma coleção de fatos e regras (lógica) que indicam como deve ser resolvido o problema proposto. Prolog é, portanto, mais direcionada ao conhecimento do que aos próprios algoritmos.

Além de ser uma linguagem declarativa, outra diferença em relação a outras linguagens é o fato de não possuir estruturas de controle (if-else, do-while, for, switch) presentes na maioria das linguagens de programação. Para isso são utilizados métodos lógicos para declarar como o programa deverá atingir o seu objetivo.

Um programa em Prolog pode rodar em um modo interativo, o usuário formula queries utilizando os fatos e as regras para produzir a solução através do mecanismo de unificação.

Prolog

Os tipos de dados, comuns em outras linguagens, não são empregados no Prolog.

Todos os dados são tratados como sendo de um **único tipo**, conhecido como **termo**, que pode ser uma **constante**, uma **variável** ou um **termo composto**.

Programar em Prolog é bem diferente de programar em uma linguagem procedimental.

Em Prolog são fornecidos os **fatos** e as **regras** para uma **base de dados**, que posteriormente serão executadas consultas (queries) em cima da base de dados.

A estrutura de um fato é formada por um predicado, seus argumentos (objetos) e a instrução é finalizada com um ponto(.) equivalente ao ponto-vírgula das linguagens comuns de programação.

Prolog - Prática

Download do SWI-Prolog:

<https://www.swi-prolog.org/download/stable>



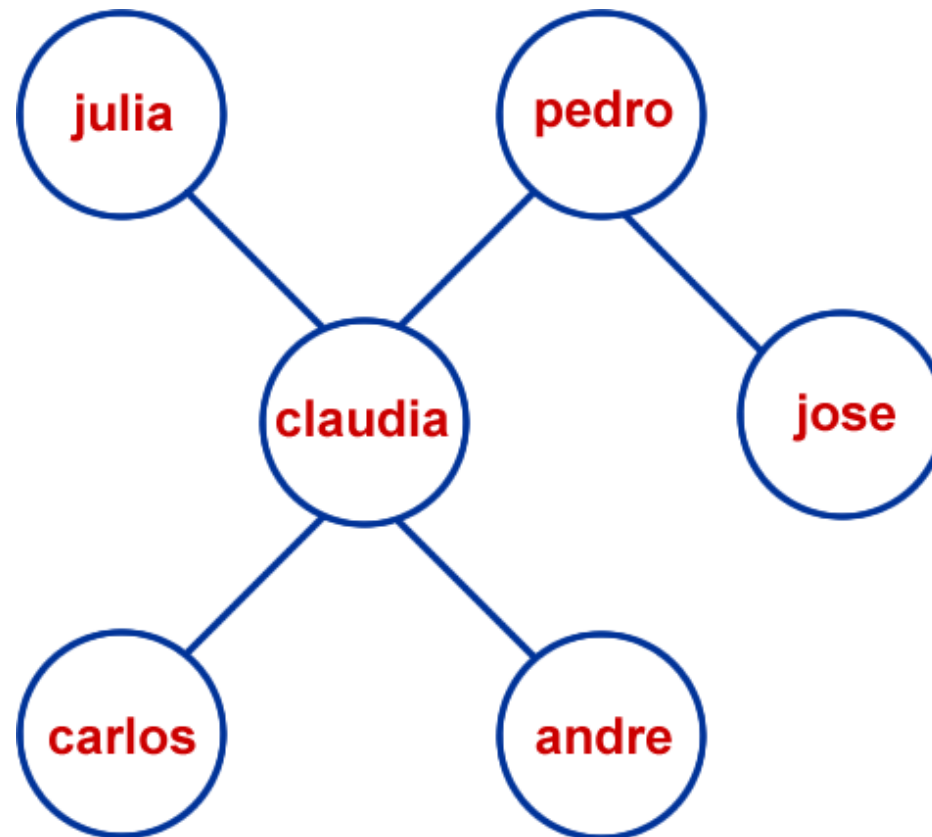
```
SWI-Prolog (AMD64, Multi-threaded, version 8.2.4)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
```

Prolog - Prática

Árvore genealógica simplificada



Prolog - Prática

Para criar um novo arquivo:

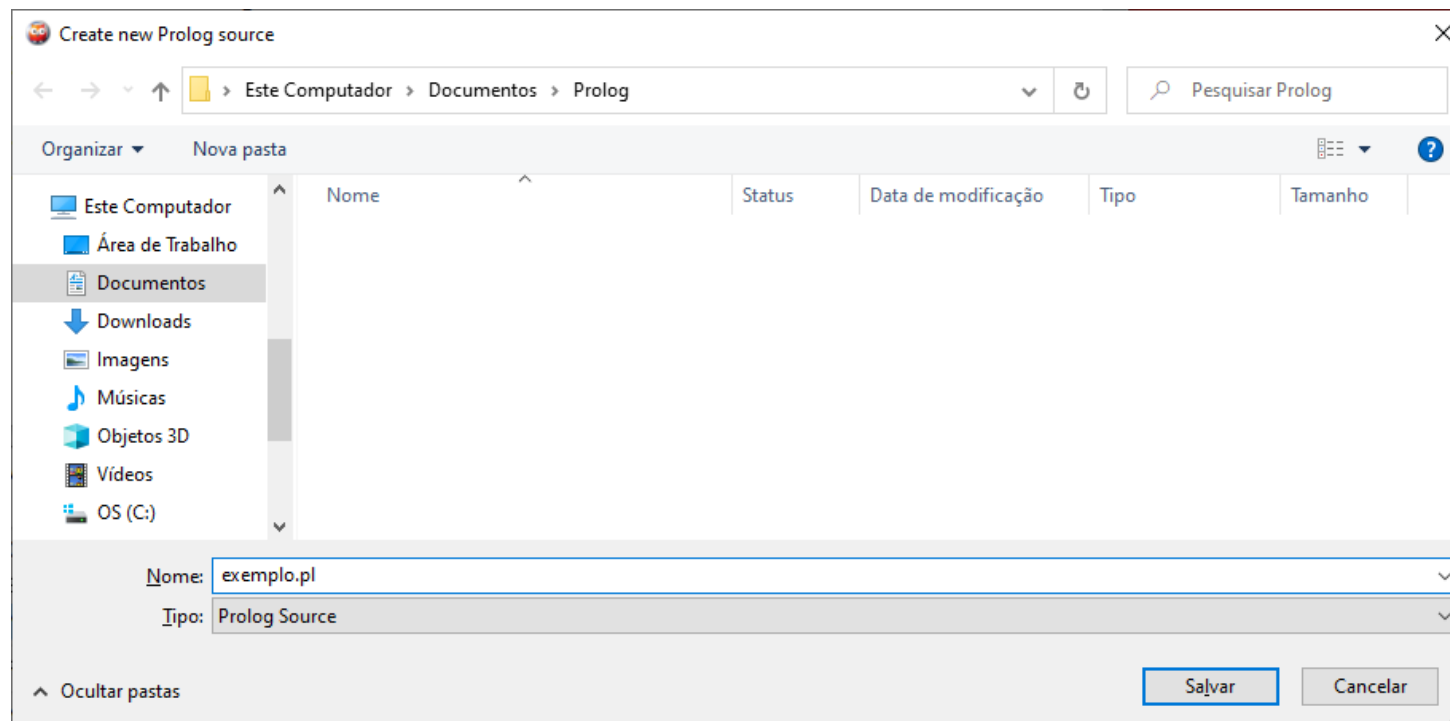
Selecionar: File -> New



Prolog - Prática

Para criar um novo arquivo:

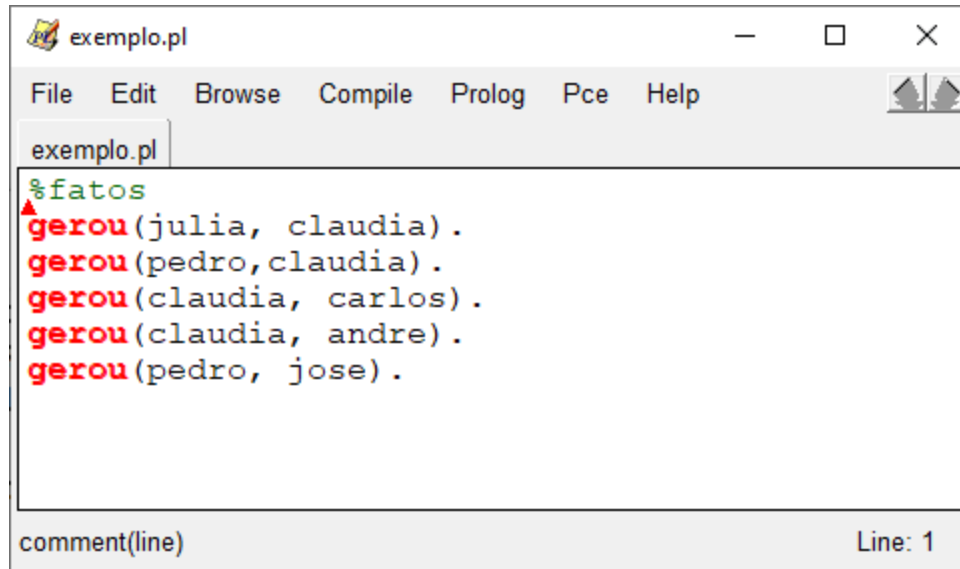
Salvar o arquivo para fonte dos dados (.pl é a extensão padrão)



Prolog - Prática

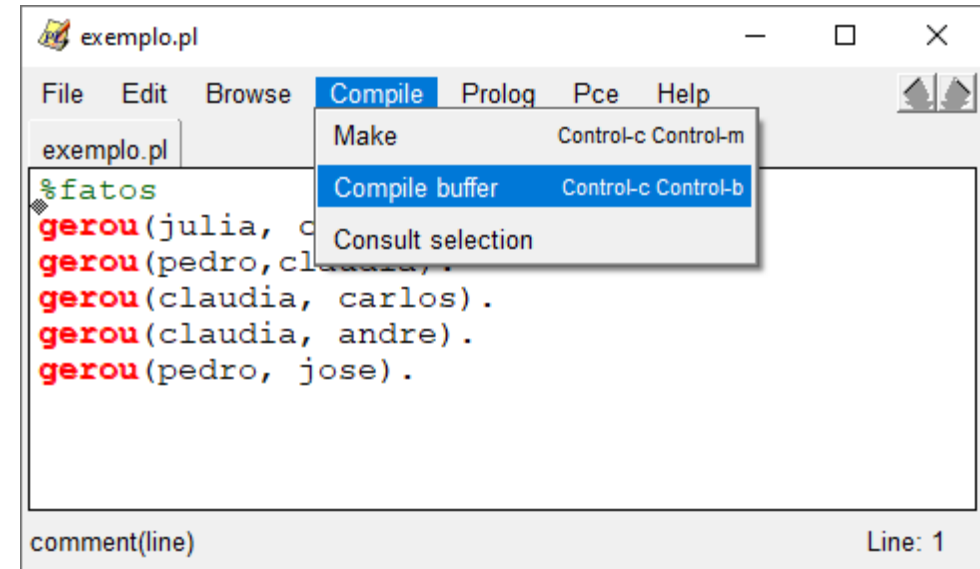
Editar o arquivo fonte.

Após editar o arquivo fonte clicar em: Compile -> Compile buffer



```
%fatos
gerou(julia, claudia).
gerou(pedro, claudia).
gerou(claudia, carlos).
gerou(claudia, andre).
gerou(pedro, jose).
```

comment(line) Line: 1



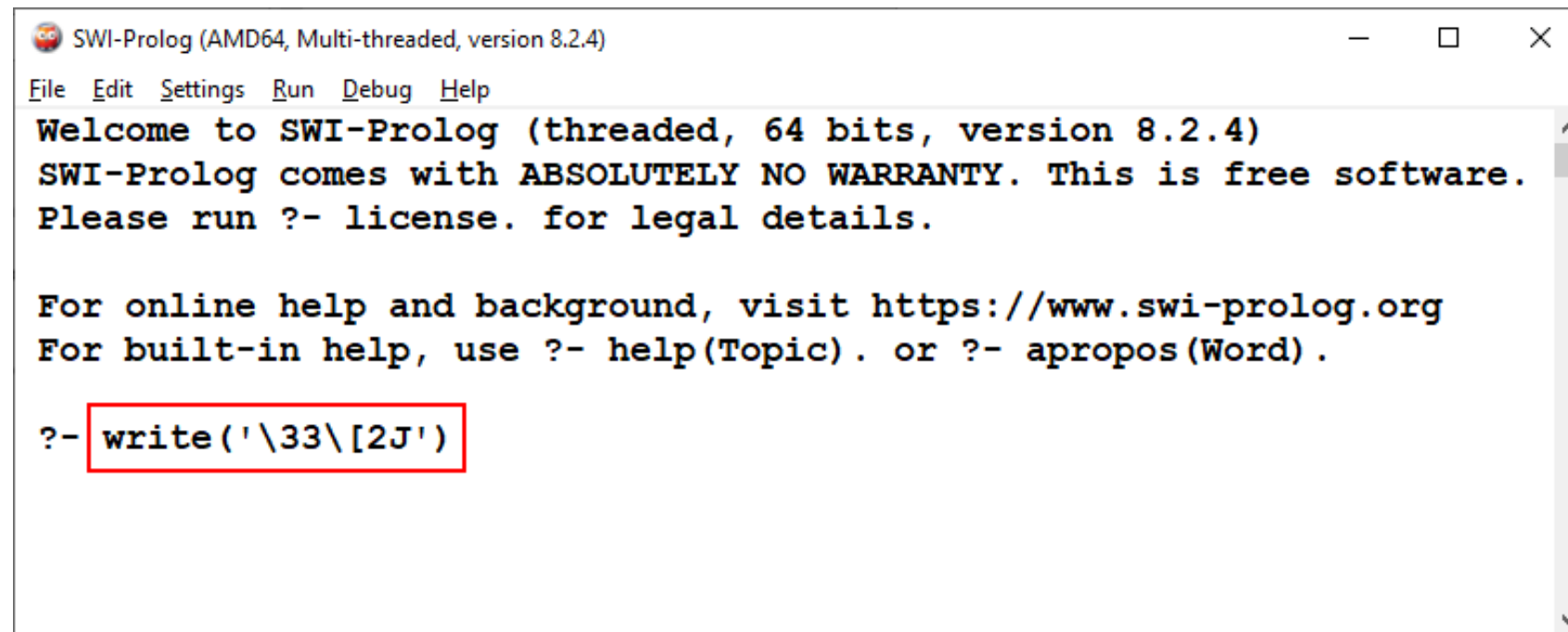
```
%fatos
gerou(julia, claudia).
gerou(pedro, claudia).
gerou(claudia, carlos).
gerou(claudia, andre).
gerou(pedro, jose).
```

comment(line) Line: 1

Prolog - Prática

Para limpar a tela:

Digite: `write('\33\[2J')`



```
SWI-Prolog (AMD64, Multi-threaded, version 8.2.4)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

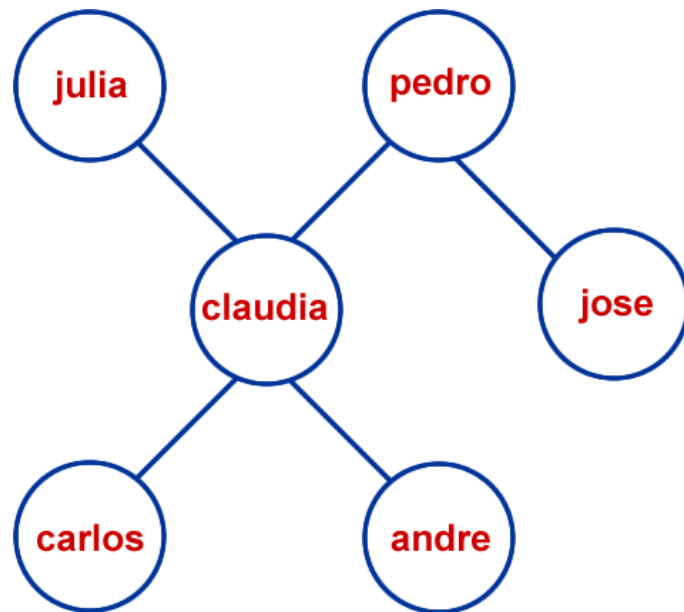
For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- write('\33\[2J')
```

Prolog - Prática

Consultas:

Inserir . (ponto final) no final de cada instrução.



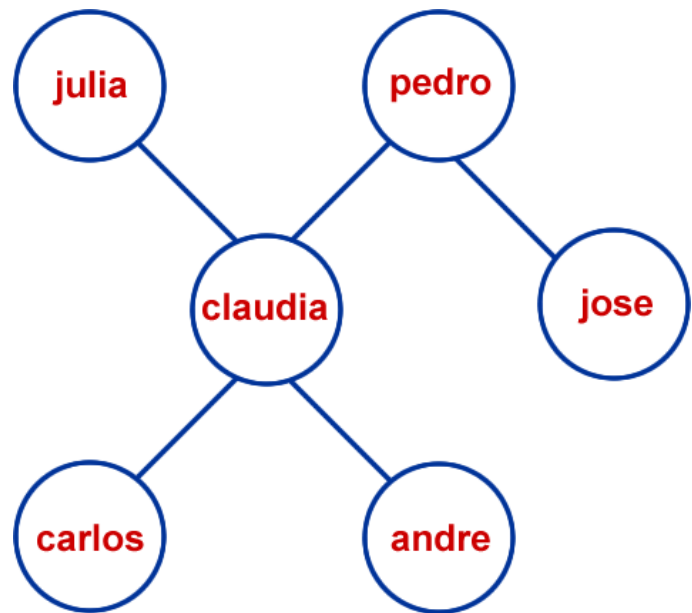
```
julia gerou claudia. (V ou F)  
gerou(julia, claudia).  
true.
```

```
julia gerou jose. (V ou F)  
gerou(julia, jose).  
false.
```

```
SWI-Prolog (AMD64, Multi-threaded, version 8.2.4)  
File Edit Settings Run Debug Help  
  
?- gerou(julia, claudia).  
true.  
  
?- gerou(julia, jose).  
false.  
  
?-
```

Prolog - Prática

Consultas:



Quem Pedro Gerou?
`gerou(pedro, X) .`
`X = claudia ;`
`X = jose .`

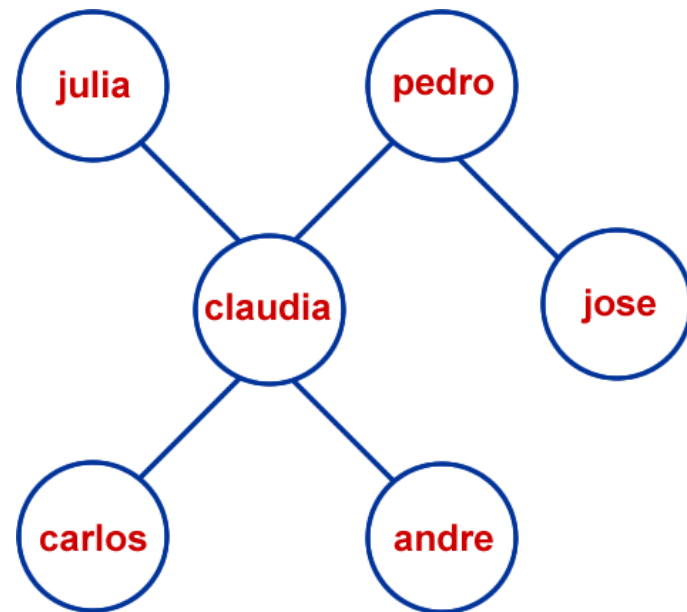
Quem gerou claudia?
`gerou(X, claudia) .`
`X = julia ;`
`X = pedro .`

Quem gerou quem?
`gerou(X, Y) .`
`X = julia,`
`Y = claudia ;`
`X = pedro,`
`Y = claudia ;`
`X = claudia,`
`Y = carlos ;`
`X = claudia,`
`Y = andre ;`
`X = pedro,`
`Y = jose .`

Prolog - Prática

Consultas:

Operadores: AND (,) e OR (;)



Quem gerou carlos E (representado pela ,) andre?

```
gerou(X, carlos), gerou(X, andre).  
X = claudia.
```

Quem gerou carlos E jose?

```
gerou(X, carlos), gerou(X, jose).  
false.
```

Quem gerou carlos OU (representado pelo ;) jose?

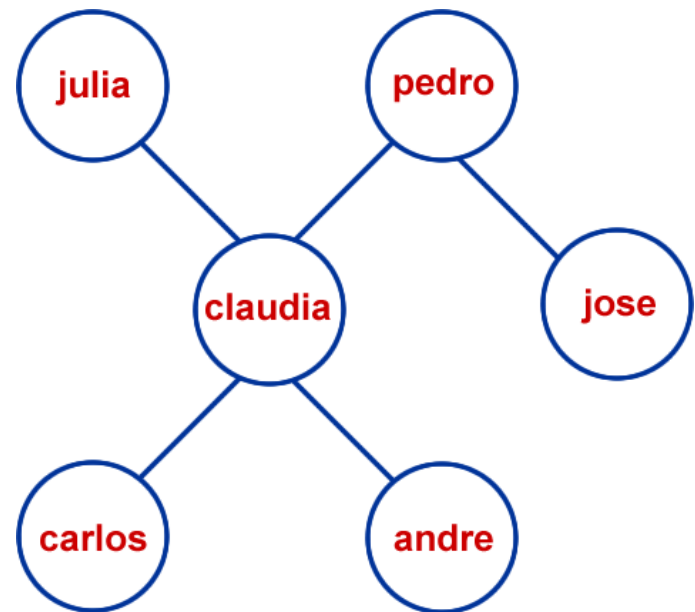
```
gerou(X, carlos); gerou(X, jose).  
X = claudia ;  
X = pedro.
```

Prolog - Prática

Consultas:

Utilizar ; para obter novas linhas.

% é utilizado para comentários



Quem são os avós de carlos? (Os X's na resposta abaixo)

```
gerou(X, Y), gerou(Y, carlos). % X gerou Y e Y gerou carlos
```

```
X = julia,
```

```
Y = claudia ;
```

```
X = pedro,
```

```
Y = claudia
```

Quem são os netos de julia?

```
gerou(julia, Y), gerou(Y, X). % julia gerou Y e Y gerou X (carlos e andre)
```

Quem gerou quem (até os netos)?

```
gerou(X, Y), gerou(Y, Z).
```

```
X = julia,
```

```
Y = claudia,
```

```
Z = carlos ;
```

```
X = julia,
```

```
Y = claudia,
```

```
Z = andre ;
```

```
X = pedro,
```

```
Y = claudia,
```

```
Z = carlos ;
```

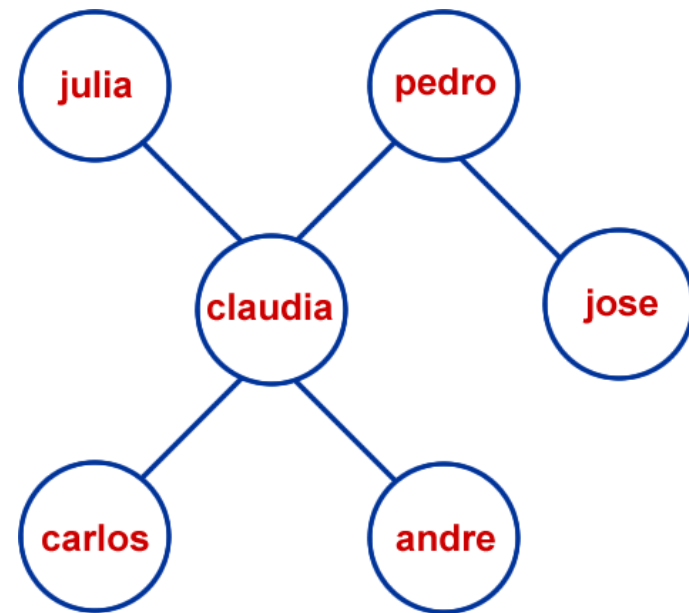
```
X = pedro,
```

```
Y = claudia,
```

```
Z = andre ;
```


Prolog - Prática

Consultas:



Pessoas do sexo masculino:
`masculino(X) .`

Pessoas do sexo feminino:
`feminino(X) .`

```
exemplo.pl
File Edit Browse Compile Prolog Pce Help
exemplo.pl
% fatos
gerou(julia, claudia).
gerou(pedro, claudia).
gerou(claudia, carlos).
gerou(claudia, andre).
gerou(pedro, jose).

masculino(pedro).
masculino(jose).
masculino(carlos).
masculino(andre).
feminino(julia).
feminino(claudia).
```

```
SWI-Prolog (AMD64, Multi-threaded, version 8.2.4)
File Edit Settings Run Debug Help

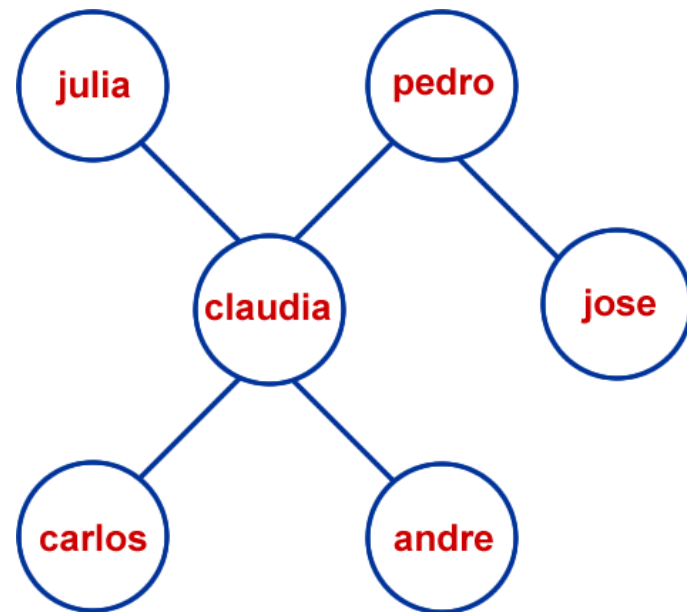
?- masculino(X) .
X = pedro ;
X = jose ;
X = carlos ;
X = andre.

?- feminino(X) .
X = julia ;
X = claudia.

?-
```

Prolog - Prática

Consultas:



Quem são os avós de carlos?

```
gerou(X,Y),gerou(Y,carlos).
```

Quem é a avó de carlos?

```
gerou(X,Y),gerou(Y,carlos),feminino(X).
```

Quem é o avô de carlos?

```
gerou(X,Y),gerou(Y,carlos),masculino(X).
```

```
SWI-Prolog (AMD64, Multi-threaded, version 8.2.4)
File Edit Settings Run Debug Help

?- gerou(X,Y),gerou(Y,carlos).
X = julia,
Y = claudia ;
X = pedro,
Y = claudia .

?-
```

```
SWI-Prolog (AMD64, Multi-threaded, version 8.2.4)
File Edit Settings Run Debug Help

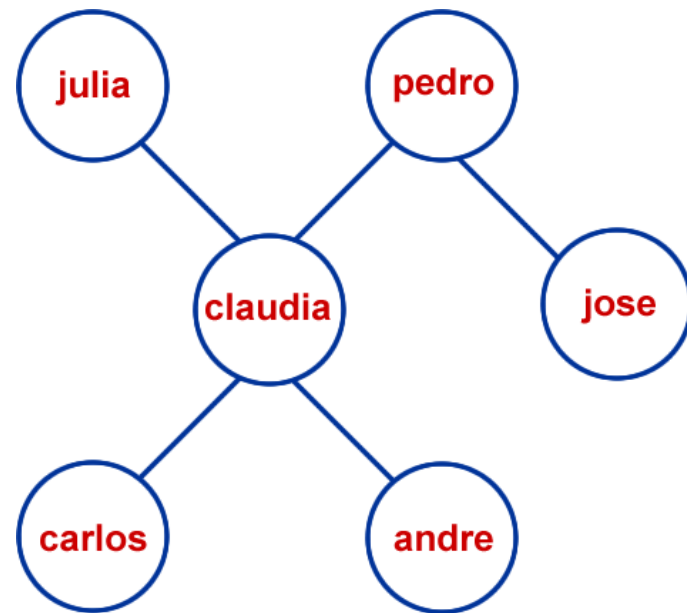
?- gerou(X,Y),gerou(Y,carlos),feminino(X).
X = julia,
Y = claudia .

?- gerou(X,Y),gerou(Y,carlos),masculino(X).
X = pedro,
Y = claudia .

?-
```

Prolog - Prática

Regras:

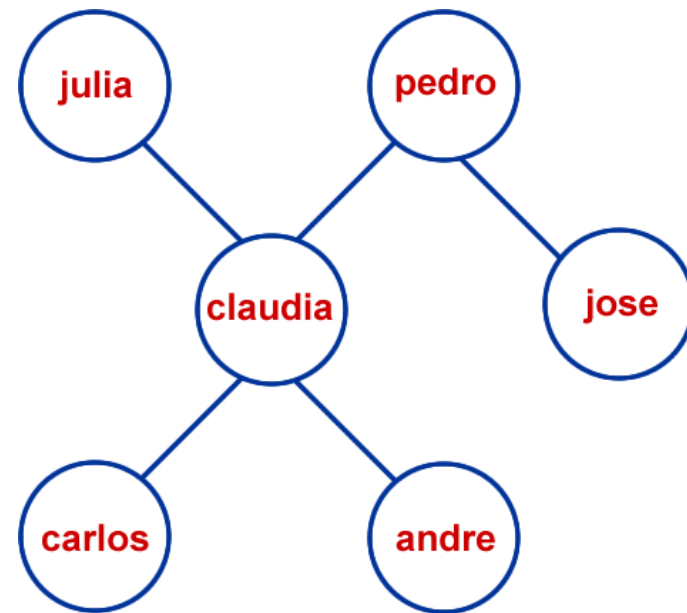


```
exemplo.pl
File Edit Browse Compile Prolog Pce Help
exemplo.pl
masculino(carlos).
masculino(andre).
feminino(julia).
feminino(claudia).
.
%regras
%Y é filho de X se X gerou Y
filho(Y, X) :-
    gerou(X, Y).
%X é pai de Y se X gerou Y e se X é masculino
pai(X, Y) :-
    gerou(X, Y),
    masculino(X).
%X é avo de Z se X gerou Y e Y gerou Z
avos(X, Z) :-
    gerou(X, Y),
    gerou(Y, Z).
```

Line: 26

Prolog - Prática

Regras:



```
exemplo.pl
File Edit Browse Compile Prolog Pce Help
exemplo.pl
masculino(carlos).
masculino(andre).
feminino(julia).
feminino(claudia).
.
%regras
%Y é filho de X se X gerou Y
filho(Y, X) :-
    gerou(X, Y).
%X é pai de Y se X gerou Y e se X é masculino
pai(X, Y) :-
    gerou(X, Y),
    masculino(X).
%X é avo de Z se X gerou Y e Y gerou Z
avos(X, Z) :-
    gerou(X, Y),
    gerou(Y, Z).
```

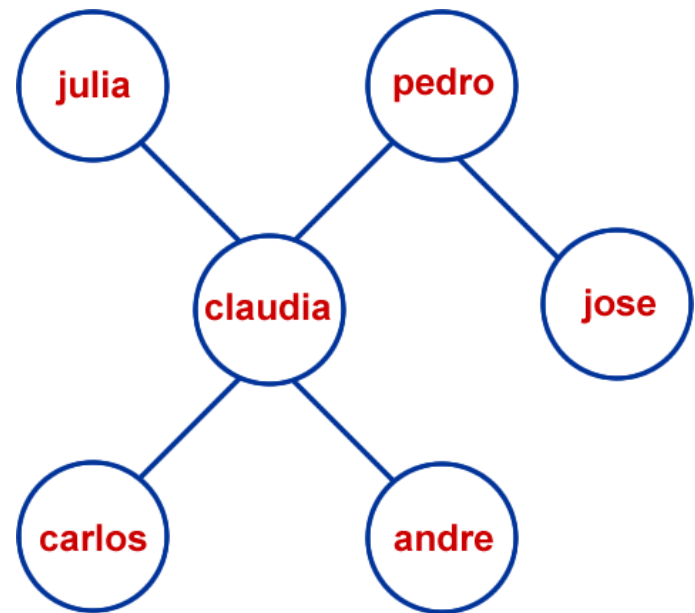
```
SWI-Prolog (AMD64, Multi-threaded,...)
File Edit Settings Run Debug Help

?- filho(X, pedro).
X = claudia ;
X = jose.

?-
```

Prolog - Prática

Regras:



```
exemplo.pl
File Edit Browse Compile Prolog Pce Help
exemplo.pl
masculino(carlos).
masculino(andre).
feminino(julia).
feminino(claudia).

%regras
%Y é filho de X se X gerou Y
filho(Y, X) :-
    gerou(X, Y).
%X é pai de Y se X gerou Y e se X é masculino
pai(X, Y) :-
    gerou(X, Y),
    masculino(X).
%X é avo de Z se X gerou Y e Y gerou Z
avos(X, Z) :-
    gerou(X, Y),
    gerou(Y, Z).
```

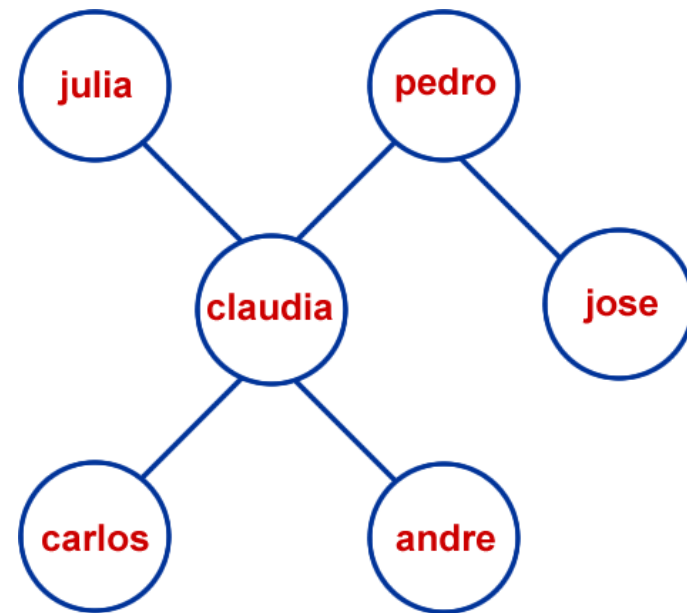
```
SWI-Prolog (AMD64, Multi-threaded,...)
File Edit Settings Run Debug Help

?- pai(X, claudia).
X = pedro.

?-
```

Prolog - Prática

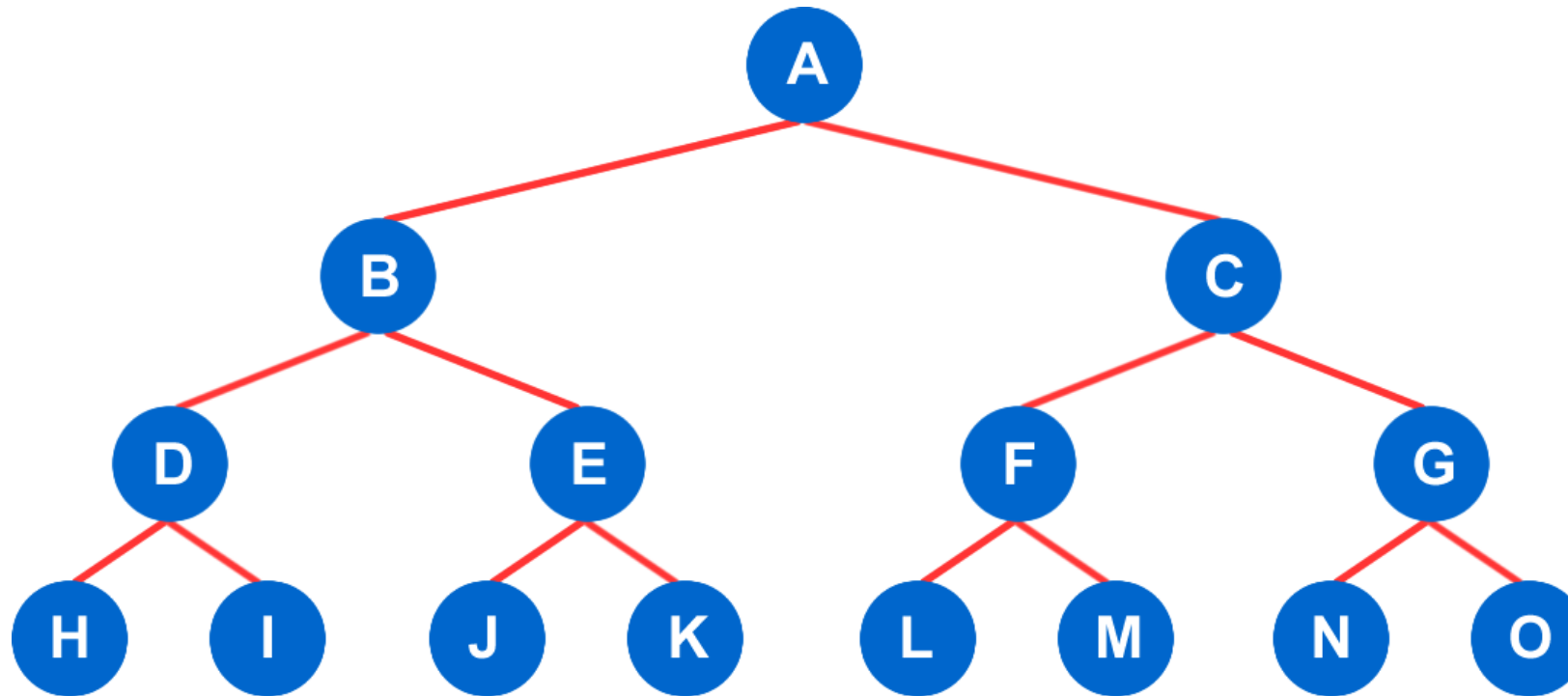
Regras:



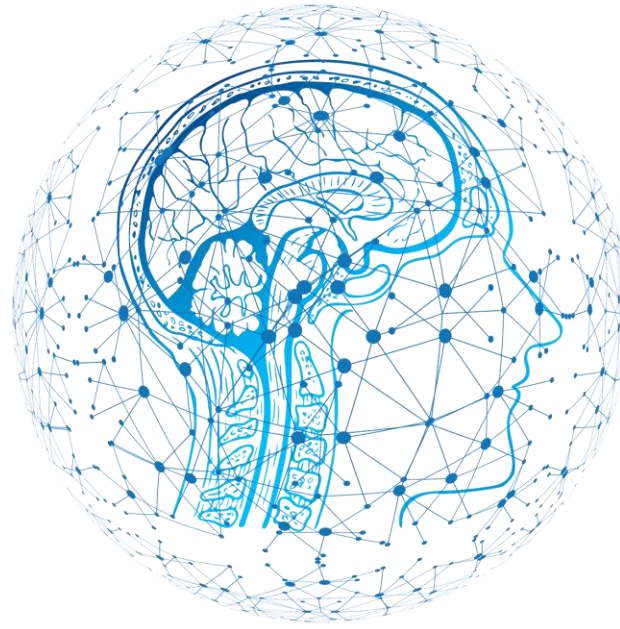
```
exemplo.pl
File Edit Browse Compile Prolog Pce Help
exemplo.pl
masculino(carlos).
masculino(andre).
feminino(julia).
feminino(claudia).
.
%regras
%Y é filho de X se X gerou Y
filho(Y, X) :-
    gerou(X, Y).
%X é pai de Y se X gerou Y e se X é masculino
pai(X, Y) :-
    gerou(X, Y),
    masculino(X).
%X é avo de Z se X gerou Y e Y gerou Z
avos(X, Z) :-
    gerou(X, Y),
    gerou(Y, Z).
```

```
SWI-Prolog (AMD64, Multi-threaded,...)
File Edit Settings Run Debug Help
?- avos(X, andre).
X = julia ;
X = pedro .

?-
```



Solução de problemas por meio de busca



Aula 03

Busca em Extensão (ou Amplitude) e Busca em Profundidade

Definição de problemas e soluções

Componentes de um problema de Inteligência Artificial:

1. Estado inicial
2. Estado final (objetivo)
3. Espaço de estados (todas as opções entre o estado inicial e o estado final)
4. Ações para passar de um estado para outro
5. Solução (caminho que leva do estado inicial ao estado final)

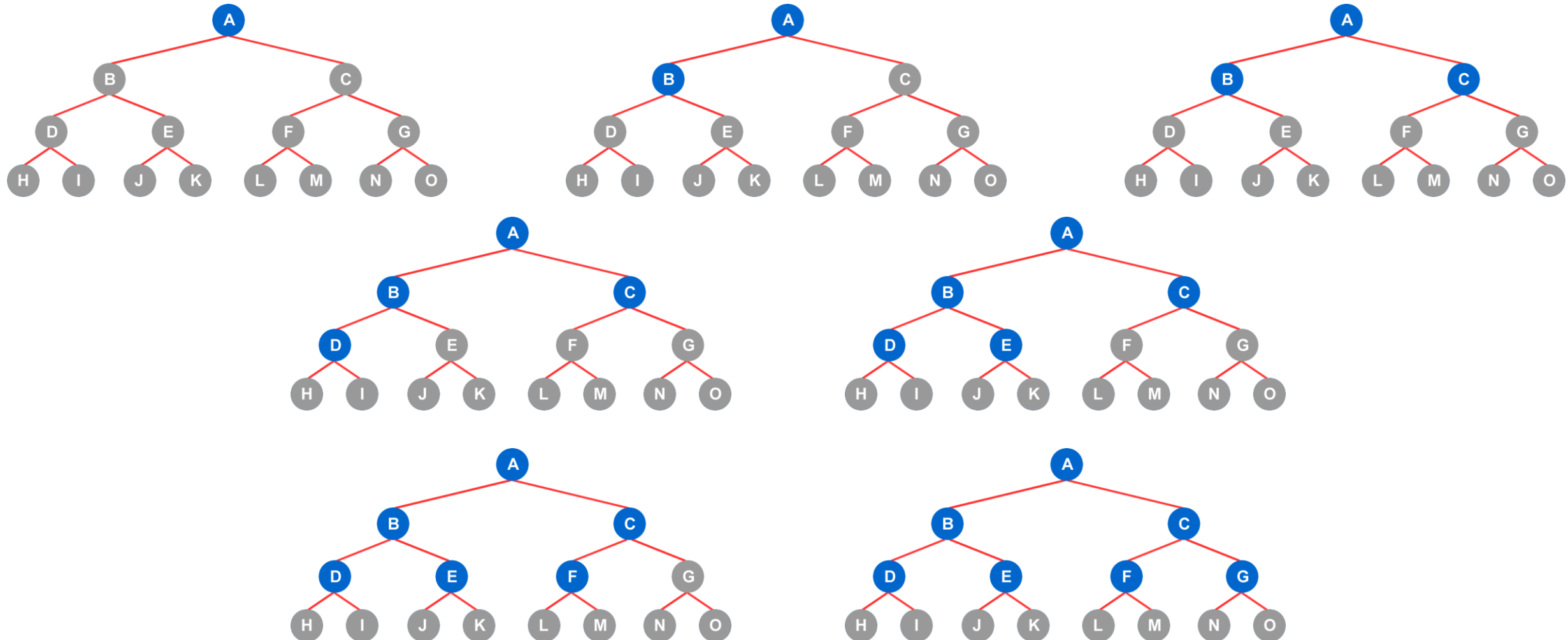
Um algoritmo inteligente apresentará a melhor rota do estado inicial para o estado final.

Busca em Extensão ou Amplitude

A **Busca em Extensão** é uma estratégia simples em que, inicialmente, expande-se o nó raiz; depois expandem-se os sucessores desse nó; logo após, os sucessores de seus sucessores; e assim por diante.

Este tipo de busca pode ser implementado através de uma lista **FIFO (First-In, First-Out)** garantindo que os nós visitados primeiro sejam expandidos primeiro.

Busca em Extensão ou Amplitude



Busca em Extensão ou Amplitude

Na Busca em Extensão todo nó gerado deve permanecer na memória, porque faz parte de um ancestral do nó.

Considerando-se a característica da explosão combinatória que ocorre na preservação dos nós antecedentes, a memória deve ser hiper especificada.

Se a árvore gera b nós no primeiro nível, o segundo nível gera outros b nós, resultará em b^2 nós na árvore; no terceiro nível teremos b^3 nós, e assim sucessivamente. Se a solução estiver no nível d , o algoritmo se expandirá até o nível $d+1$, gerando $b^{d+1} - b$ nós, para o pior caso.

A soma total dos nós será: $1 + b + b^2 + b^3 + \dots + b^d + (b^{d+1} - b)$

Concluindo: problemas de busca de complexidade exponencial, não podem ser resolvidos por métodos sem informação. Exceção vale para pequenos problemas.

Busca em Extensão ou Amplitude

Vamos analisar a busca em extensão a partir de um viés do **problema da jarra de água**.

O problema consiste em: você tem **duas jarras**, uma de **4 litros** e uma de **3 litros**.

Nenhuma delas tem qualquer marcação de medidas.

Há uma bomba que pode ser utilizada para encher as jarras com água.

A **meta** é conseguir colocar exatamente **2 litros de água na jarra de 4 litros**.

O espaço de estados para o problema consiste em um conjunto de pares ordenados (x,y) , de forma que $x=1,2,3,4$ e $y=0,1,2,3$.

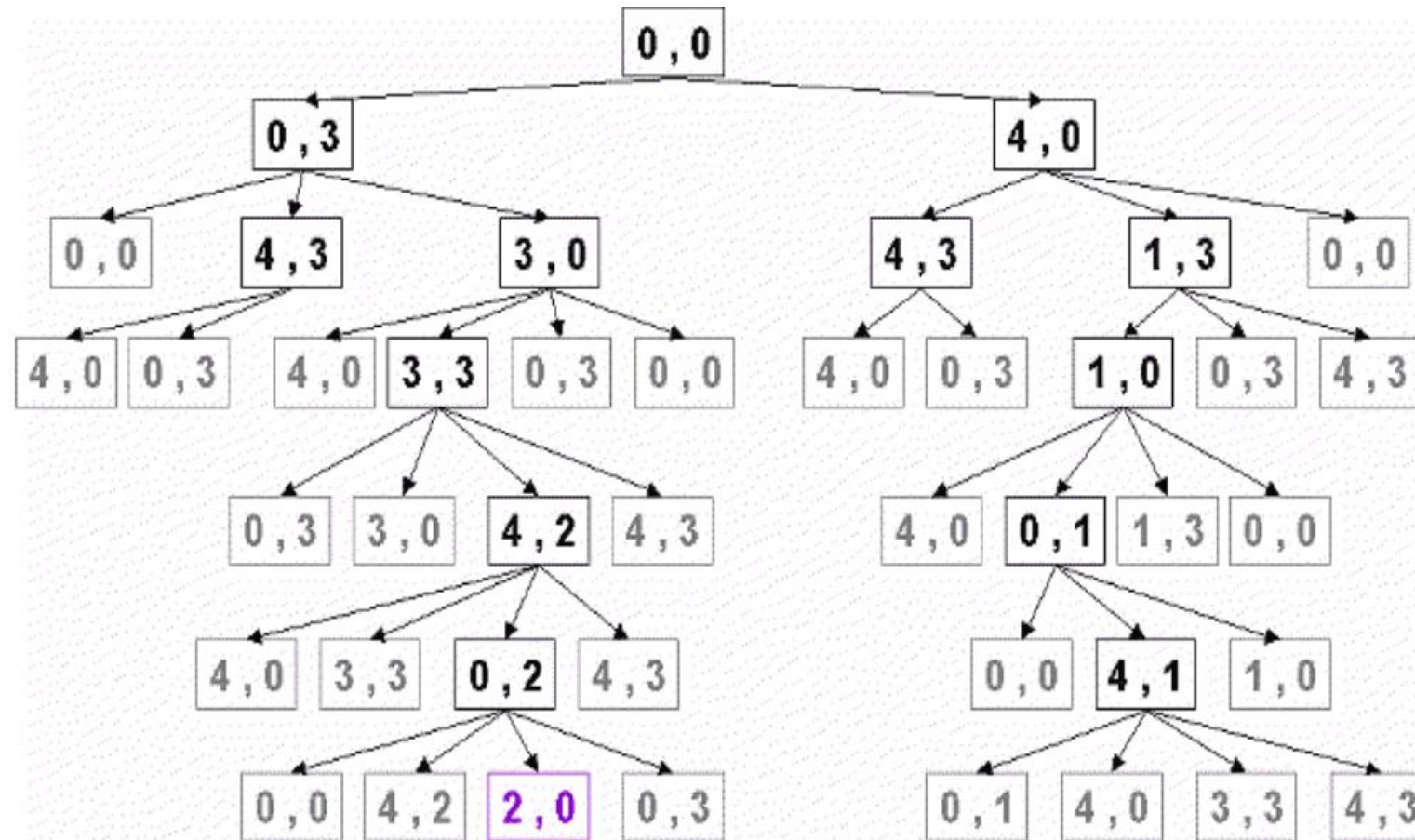
x representa a jarra com capacidade de 4 litros e y representa a jarra de água com 3 litros.

O estado inicial deve ser igual a $(0,0)$.

A meta é encontrar $(2,n)$ sendo n qualquer valor visto que o problema não especifica quantos litros são necessários na jarra de 3 litros.

Inteligência Artificial

Problema da Jarra de Água



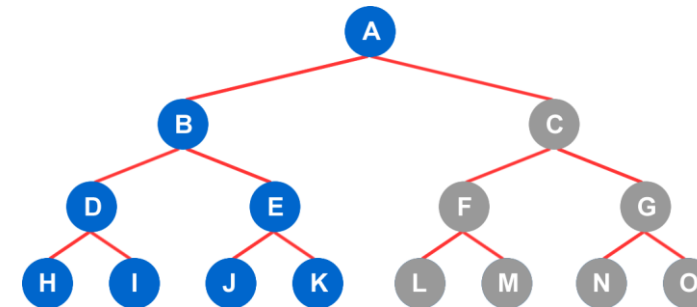
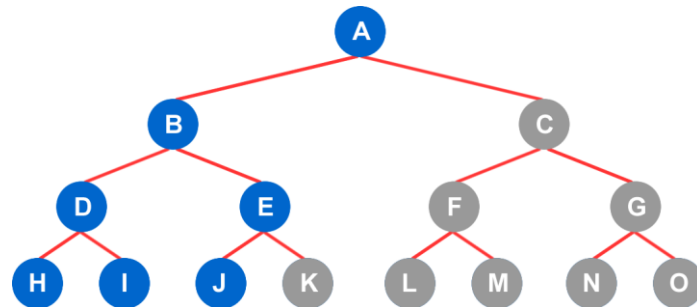
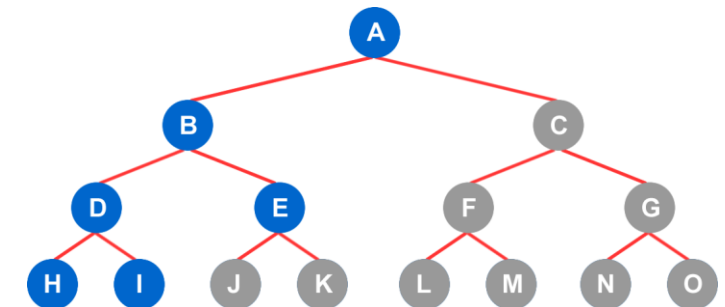
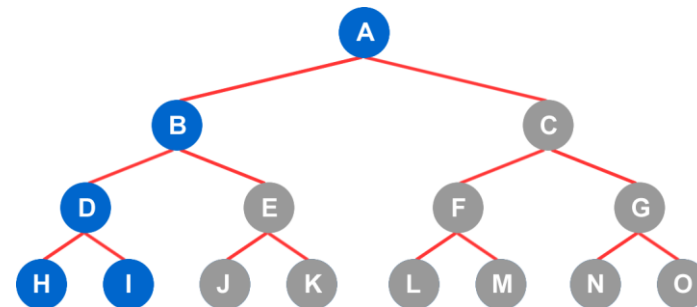
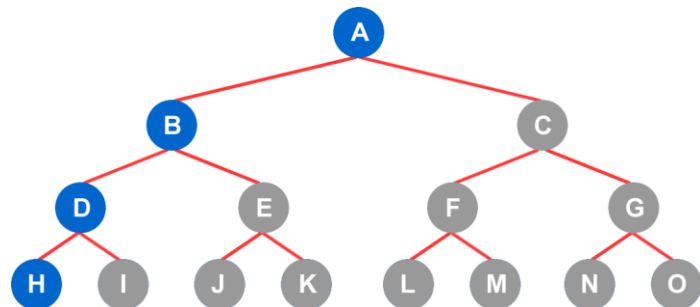
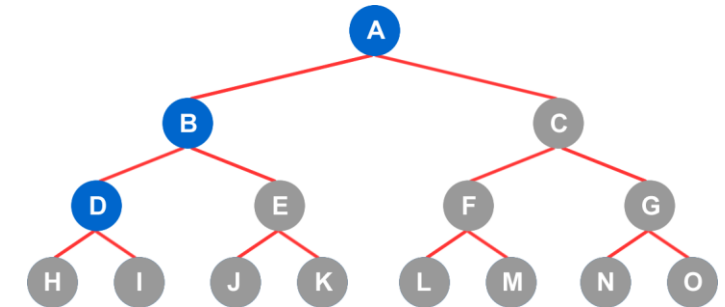
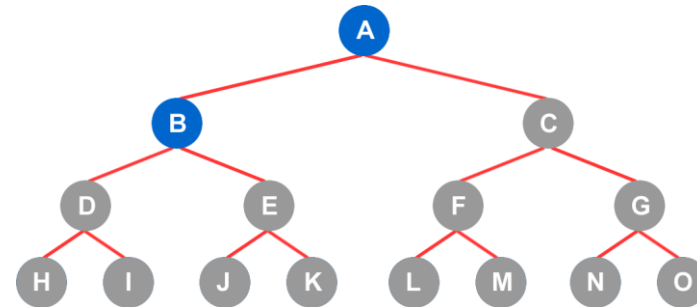
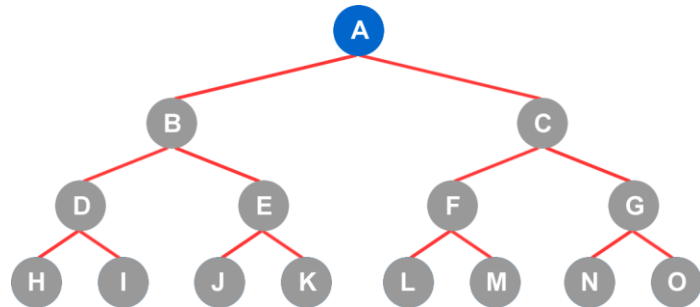
Busca em Profundidade

A **Busca em Profundidade** é uma estratégia que expande sempre o nó mais profundo na borda atual da árvore de busca.

Este tipo de busca pode ser implementado através de uma fila **LIFO (Last-In, First-Out)** também denominada **pilha**.

É comum a implementação por meio de uma função recursiva (que chama a si mesma sucessivamente).

Busca em Profundidade



Nós que foram expandidos e que não possuem descendentes na borda podem ser removidos da memória.

Busca em Profundidade

Uma das vantagens da busca em profundidade é que os requisitos de memória são pequenos se comparados a outras estratégias.

É necessário armazenar um único caminho do nó raiz ao nó visitado. À medida que os nós e seus descendentes são explorados, eles podem ser removidos da memória.

Se considerarmos um fator de ramificação b e uma profundidade máxima m a estratégia de busca em profundidade requer somente $bm + 1$ nós.