# Aula 06:
# Computação Evolutiva e Conexionista – Programação Genética

Prof. Hugo Puertas de Araújo
hugo.puertas@ufabc.edu.br
Sala: 509.2 (5º andar / Torre 2)

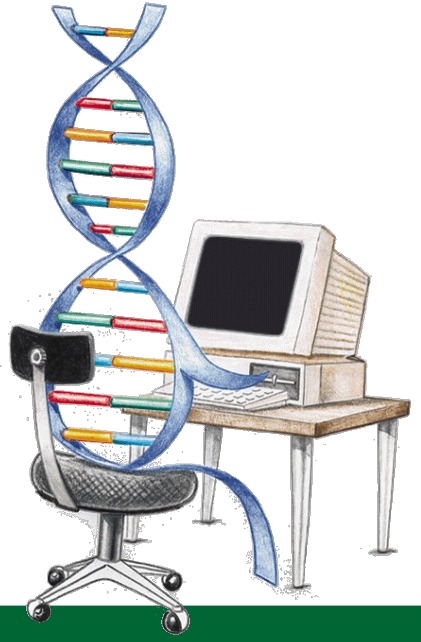
# Agenda

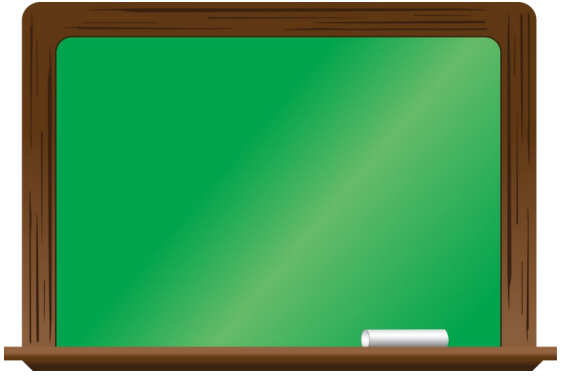
- Programação Genética
  - Visão Geral
  - Definições dos parâmetros da Programação Genética (PG)

# Computação Evolutiva e Conexionista

# Desafio

- Como os computadores podem aprender a resolver problemas sem ser explicitamente programados??





# Programação Automática

- Indução de programas, Síntese de programas
- Descoberta no espaço de possíveis programas, de um programa de computador que produza a saída que satisfaça o objetivo do problema.
- Se estamos interessados em que computadores resolvam problemas sem ter sido explicitamente programados para tal, a estrutura deve ser um programa (Koza)

# Visão geral

- Desenvolvida: EUA nos anos 1990
- Pioneiro: J. Koza
- Aplicação típica:
  - Aprendizagem de máquina (Predição, classificação,...)
- Características
  - Competem com as Redes Neurais Artificiais
  - Necessitam de uma população muito grande (milhares)
  - Processo lento de convergência
- Características especiais
  - Cromossomos não-lineares: Árvores, grafos.

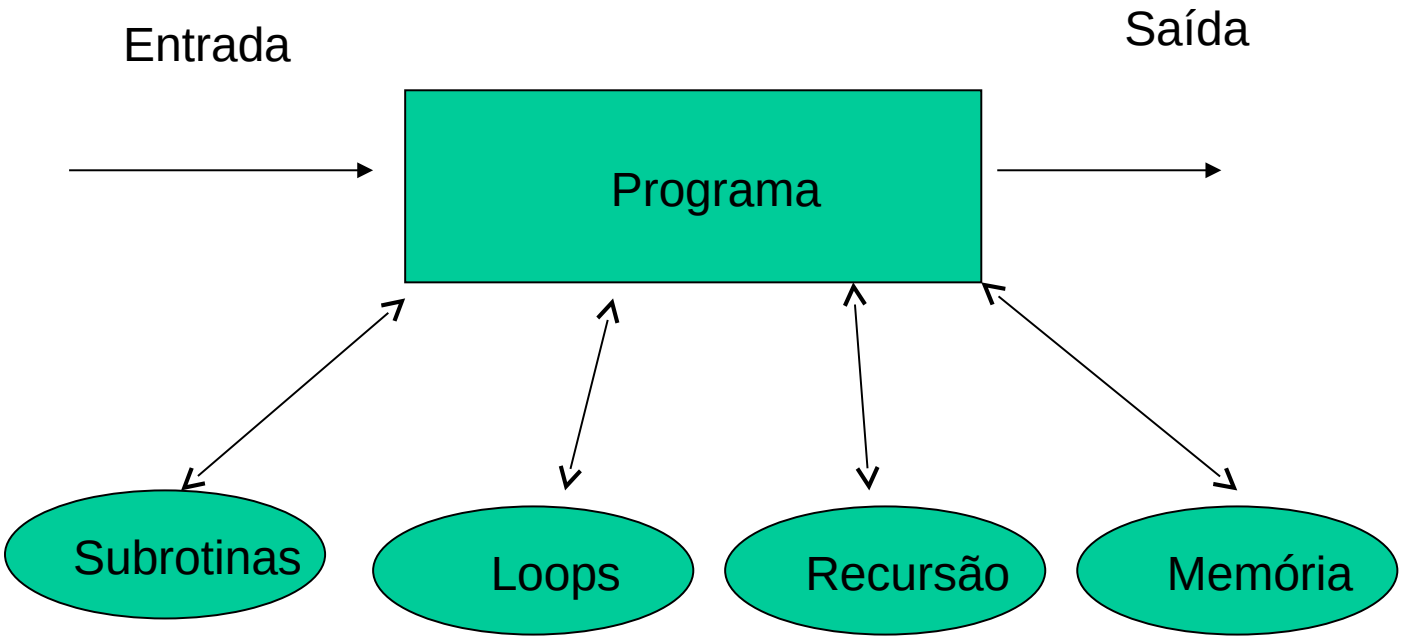
# Visão geral

| | |
|---|---|
| Representação | Árvores, Grafos |
| Recombinação | Troca de sub-Árvores |
| Mutação | Alteracões aleatórias na árvore |
| Seleção de Pais | Proporcional ao Fitness |
| Seleção por sobrevivência | Recolocação |

# Programa

- Parse tree, Program tree
- Representação natural, a maioria dos compiladores traduzem o programa para o parse tree.
- Funções e terminais: alfabeto do programa a ser induzido.
- Conjunto de terminais: variáveis e constantes do programa.
- Funções: soma, substração, divisão, ...

# Programa

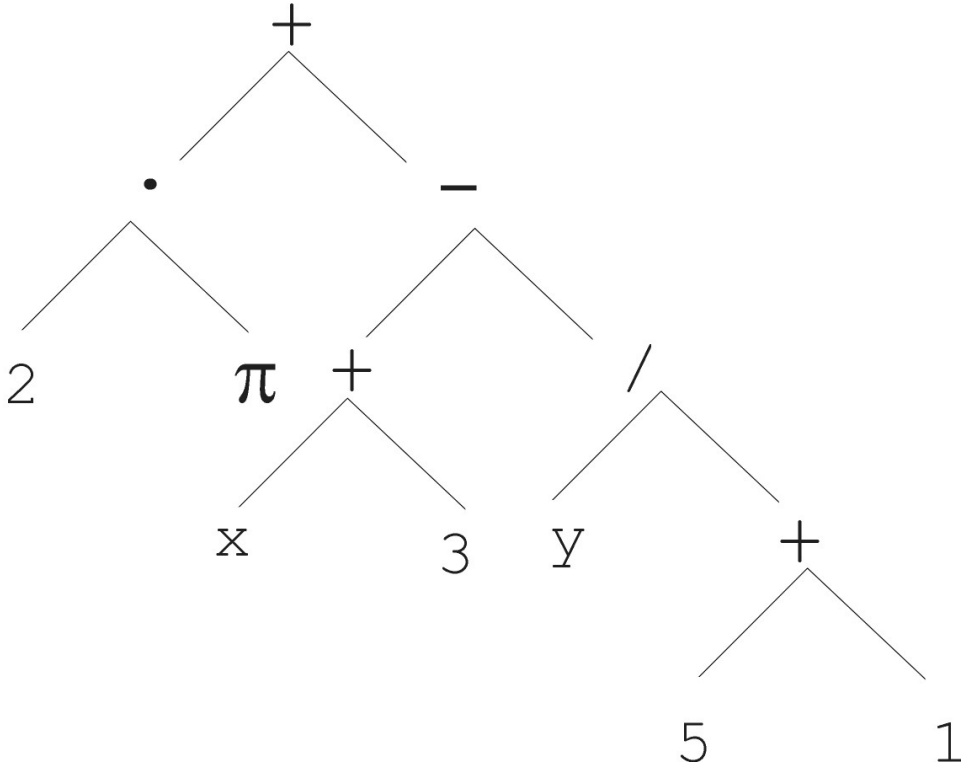

Entrada
Saída
Programa
Subrotinas
Loops
Recursão
Memória

# Representação baseada em árvores

- As Árvores são uma forma  universal de representação
  - Fórmula aritmética: $2 \cdot \pi + \left( (x+3) - \frac{y}{5+1} \right)$
  - Fórmula lógica: $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$
  - Programa:

```
i =1;
while (i < 20)
{
     i = i +1
}
```

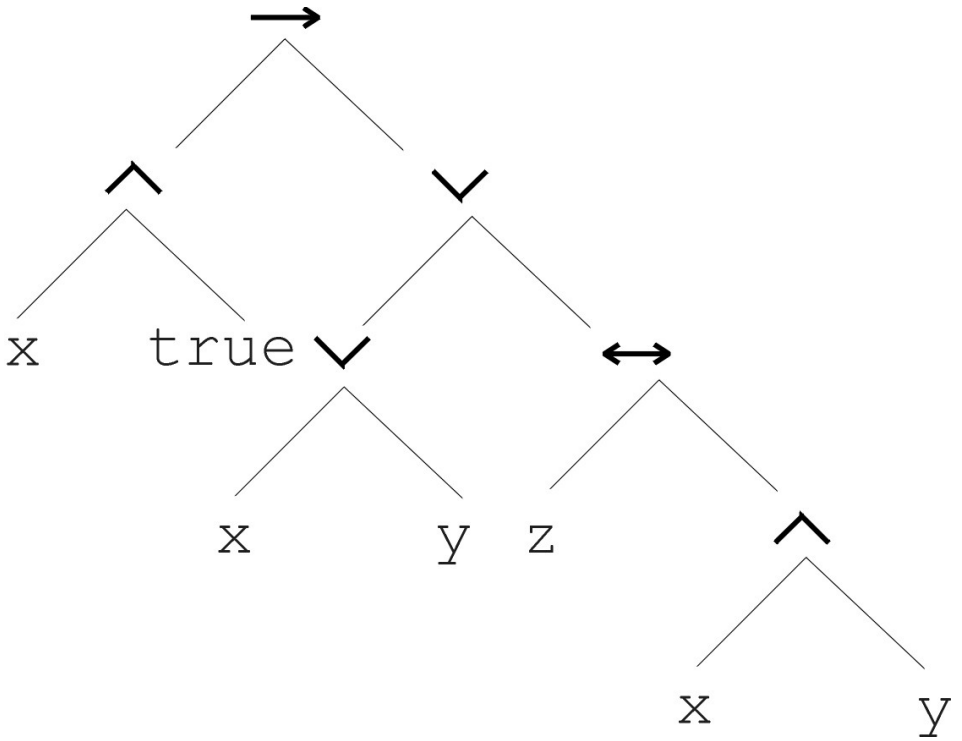
# Representação baseada em árvores



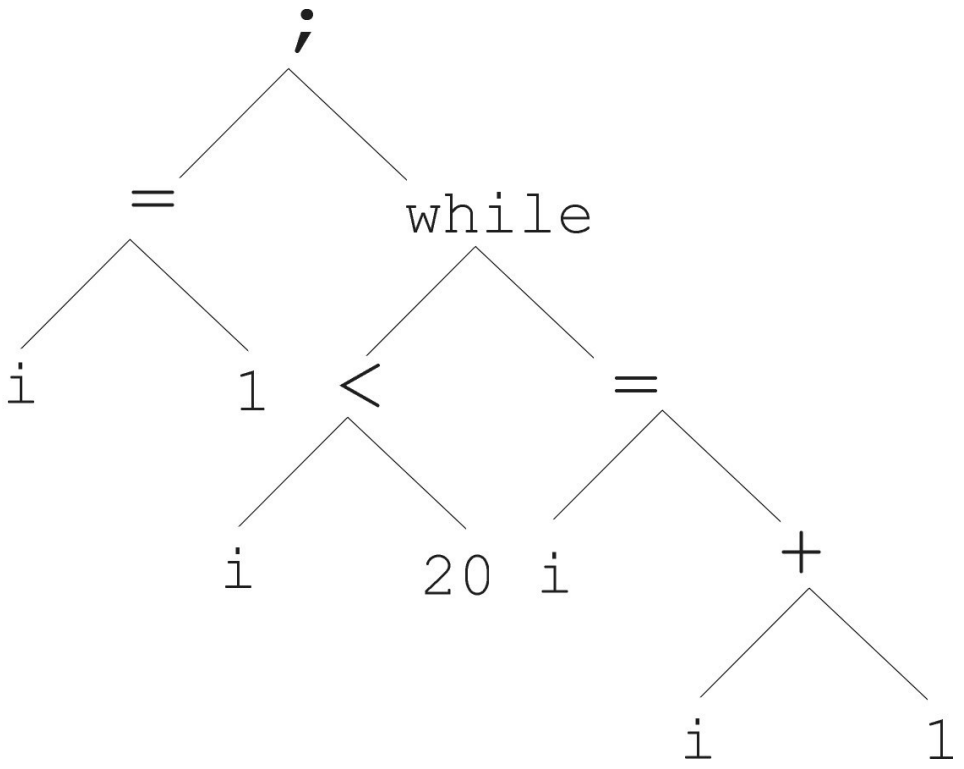

$$2 \cdot \pi + \left( (x+3) - \frac{y}{5+1} \right)$$

# Representação baseada em árvores

$$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$



# Representação baseada em árvores



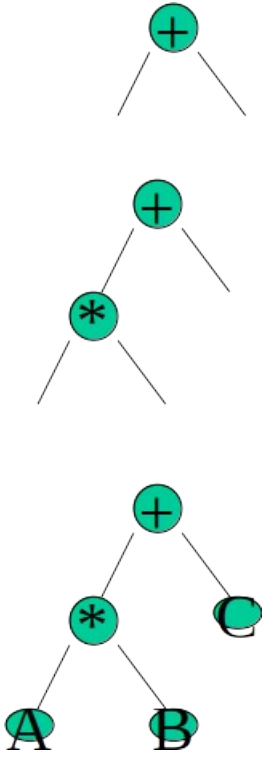

```
i =1;
while (i < 20)
{
        i = i +1
}
```

# Representação baseada em árvores

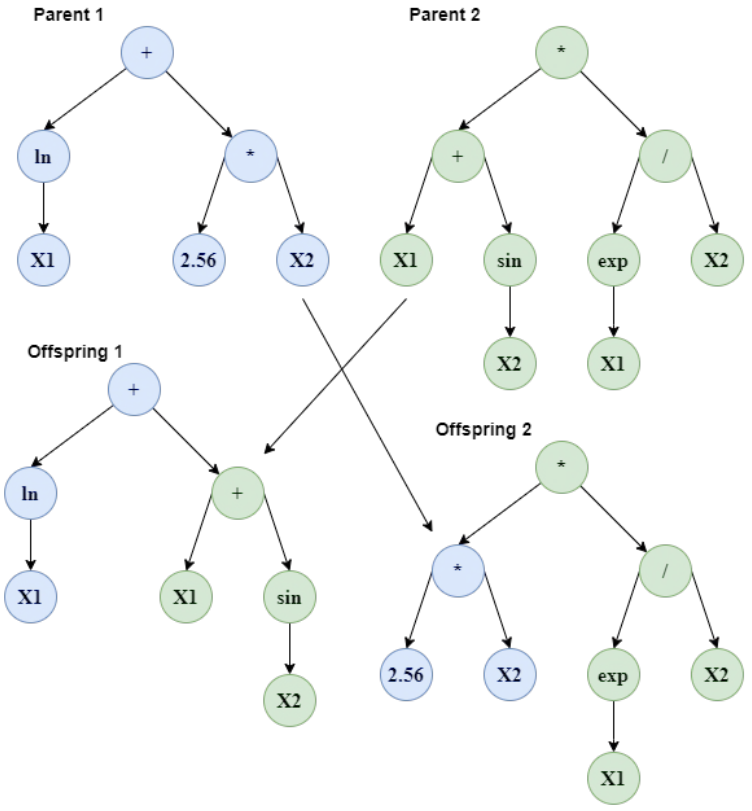
- Expressões simbólicas podem ser definidas por
  - Um conjunto terminal T (Folhas)
  - Conjunto de Funções F (com a aridade da função de símbolos)
- Adoção da definição recursiva geral:
  - Todo t $\in$ T é uma expressão correta
  - f(e1,...,em) é uma expressão correta se f $\in$ F, onde a aridade(f) = n e e1,...,em são expressões corretas
- Em PG, a árvore pode variar em largura e profundidade

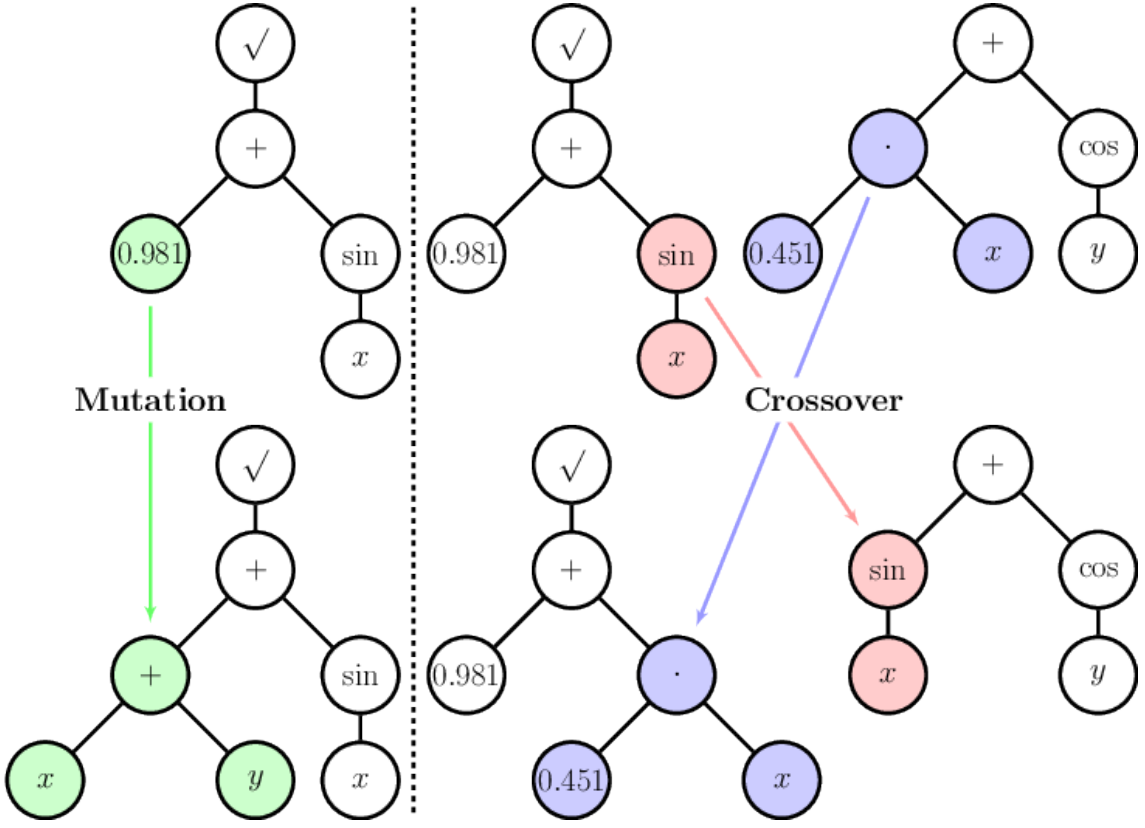
# Exemplo de criação de  programa

- Terminais: T={A,B,C}
- Funções: F={+,*,%,If,Lte}
- Comece com + e 2 arg
- Continue com * e 2 arg
- Complete com terminais A,B,C
- O resultado é um progr. executável



# Operadores


Parent 1
Parent 2
Offspring 1
Offspring 2


Mutation
Crossover

# Mutação

- Dois tipos de mutuações são possíveis:
  - Uma função pode substituir uma função ou um terminal pode substituir outro;
  - Uma subtree pode ser substituída por um novo subtree, gerado aleatoriamente: envolve selecionar um nó não-terminal e aplicar uma produção selecionada randomicamente até a profundidade máxima da árvore seja alcançada.

# Seleção

Exercício

**E.: Como é feita a seleção em GP?**

# Seleção

- Executa-se o programa e avalia o quão bom ele é em resolver o problema (fitness).
- Seleciona-se os melhores para a próxima geração

# Memória

- Estruturas de dados:
  - Stacks
  - Filas
  - Listas
  - Anéis

Estruturas simples, mas limitadas

# Bloating

- Bloating é o aumento excessivo do tamanho dos programas (indivíduos) ao longo das gerações. Isso pode acontecer por diversos motivos, como:
  - Mutação: A mutação pode inserir novos genes nos programas, aumentando seu tamanho.
  - Crossover: O crossover pode combinar genes de pais com tamanhos diferentes, gerando filhos maiores.
  - Seleção: Se a seleção favorecer programas maiores, mesmo que não sejam necessariamente mais aptos, o bloating pode ocorrer.

# Bloating – Efeitos negativos

- Diminuição da eficiência: Programas maiores podem ser mais lentos para serem executados e avaliados.
- Dificuldade de interpretação: Programas maiores podem ser mais difíceis de serem interpretados e debugados.
- Superajuste: Programas maiores podem ter maior probabilidade de se ajustar ao conjunto de dados de treinamento, mas apresentar menor desempenho em dados novos.

# Bloating – Técnicas p/ evitá-lo

- Limite de tamanho: Definir um limite máximo para o tamanho dos programas.
- Penalização por tamanho: Penalizar programas maiores durante a seleção.
- Parcimônia: Favorecer programas menores que apresentam a mesma performance que programas maiores.
- Crossover: ajustar o operador para evitar a criação de filhos muito grandes.

# Exemplo: Biblioteca genepro

https://colab.research.google.com/drive/1753VfN1Yv_nguW-Tov1aJ1WhH530J98N?usp=sharing

https://colab.research.google.com/drive/17IJGf94B_JtbOrGj8GaHgytQzmEbtqbP?usp=sharing