# *Métodos de Busca em Inteligência Artificial: Uma Análise Simplificada*

por Ricardo Silva

# *O Que São Métodos de Busca?*

A busca é um processo fundamental em inteligência artificial, usado para encontrar soluções para problemas de maneira sistemática. Segundo Russell e Norvig, “o processo de procurar por tal sequência de ações que alcançam o objetivo é chamado de busca”.

## *Mas o Que Buscamos?*

O objetivo da busca é encontrar uma sequência de ações que leve de um estado inicial a um estado objetivo. Esse processo pode ser visto como uma árvore de busca, onde cada nó representa um estado e os ramos representam as ações possíveis a partir desse estado. O objetivo é encontrar um caminho do nó inicial até um nó que satisfaça a condição de objetivo.Após encontrar uma solução, as ações recomendadas podem ser executadas na chamada fase de execução. O agente segue a sequência de ações determinada pela busca e, ao final, pode formular um novo objetivo e repetir o processo.

# *Como a Busca é Estruturada?*

## *Como a Busca é Estruturada?*

Para resolver um problema, um algoritmo de busca recebe um problema como entrada e devolve uma solução na forma de uma sequência de ações. O processo ocorre em três etapas principais:

- **Formulação do Objetivo** - Definir qual é a meta a ser alcançada.
- **Formulação do Problema** - Definir os estados, as ações possíveis e a condição de sucesso.
- **Busca da Solução** - Aplicação de um algoritmo para encontrar um caminho que leve ao objetivo.Esse processo de busca pode ser cega (sem informação) ou heurística (com informação adicional). A busca cega apenas expande nós sistematicamente sem conhecimento sobre qual caminho é melhor. Já a busca heurística utiliza estimativas para guiar o processo de busca de forma mais eficiente.

## *Busca Heurística (Com Informação)*

A busca heurística utiliza estimativas para guiar o processo de busca de forma mais eficiente, usando uma função heurística h(n) para estimar o custo do estado atual até o objetivo.

- **g(n):** Custo real para mover de um estado a outro.
- **h(n):** Custo estimado para alcançar o objetivo (obtido por cálculo ou conhecimento especializado).

As heurísticas admissíveis nunca superestimam o custo real, garantindo a solução ótima.

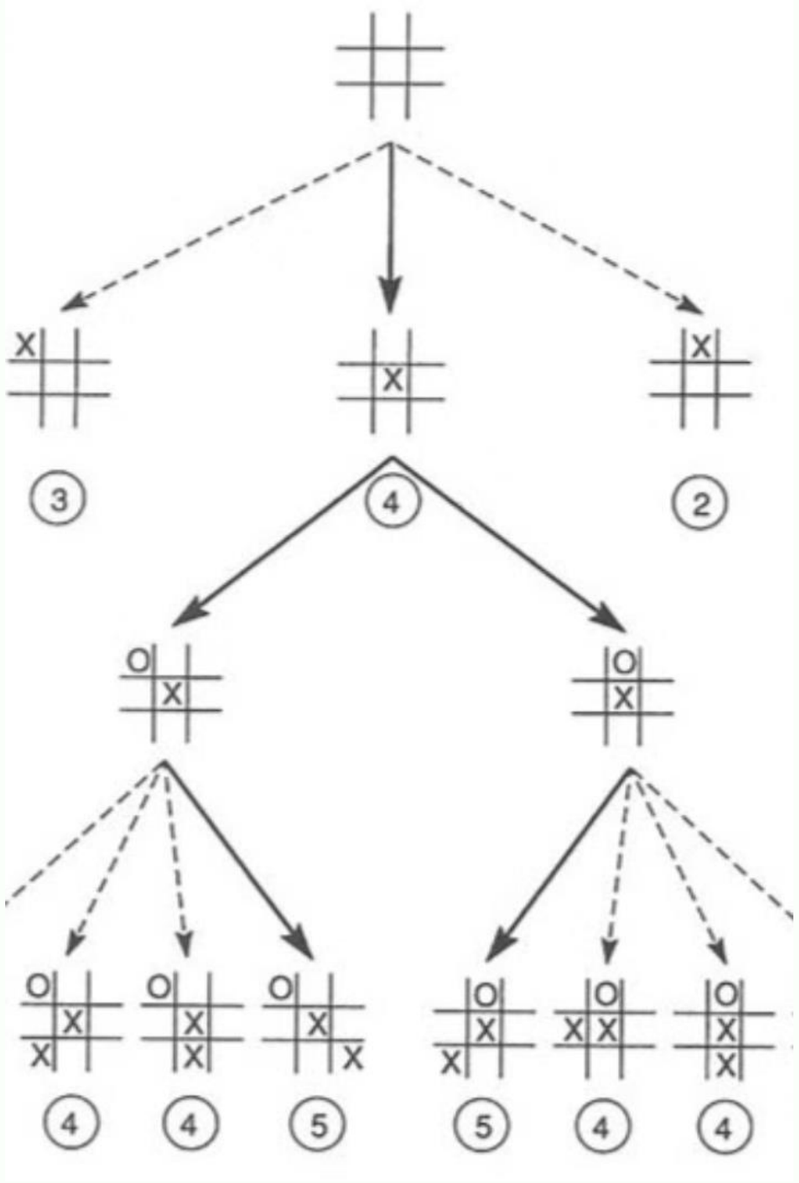
# *Exemplo Prático: Jogo da Velha com Heurística*

## 1 *Análise da Heurística*

Na imagem ao lado temos o método de busca por heurística, para a solução do problema do jogo da velha, onde a heurística(dica) é admissível, pois deve começar pelo pontos chaves da tabela(dicas números 2, 3 e 4).

## 2 *Movimentos Iniciais*

- Para o canto;
- Para o centro de um lado;
- Para o centro da grade;




## 3 *Implementação*

O método de busca heurística guia as jogadas, informando a posição de zero ou “0”.

# *Para que servem os métodos de busca?*

## *1. Busca Sem Informação (ou Cega) – Força Bruta*

Métodos que não utilizam conhecimento adicional sobre o problema além da estrutura do espaço de busca. Apenas geram sucessores e distinguem estados objetivos de estados não objetivos.

### *Busca em Largura (BrFS - Breadth-First Search):*

Expande os nós mais rasos primeiro, garantindo encontrar a solução mais curta para problemas de custo uniforme. Possui complexidade de espaço elevada devido à necessidade de armazenar muitos nós na memória.

### *Busca em Profundidade (DFS - Depth-First Search):*

Explora um caminho até o final antes de retroceder, podendo ser implementada com memória limitada. No entanto, não é completa nem ótima em espaços de busca infinitos.

### *Busca de Custo Uniforme*

Expande o nó com o menor custo acumulado g(n)g(n)g(n), sendo ótima para problemas com custos variáveis de transição. Pode ser muito lenta devido ao grande número de expansões necessárias.

## *2. Busca Com Informação (ou Heurística)*

Utiliza uma função heurística h(n), que estima o custo do estado atual até o objetivo. Isso melhora a eficiência, pois permite que a busca priorize caminhos promissores.

**Heurística ou Dica:**

### *g(n)*

Custo real para mover de um estado a outro (exemplo: distância percorrida em um mapa).

### *h(n)*

Custo estimado para alcançar o objetivo. Pode ser obtido por cálculo ou conhecimento especializado.

### *Heurísticas Admissíveis*

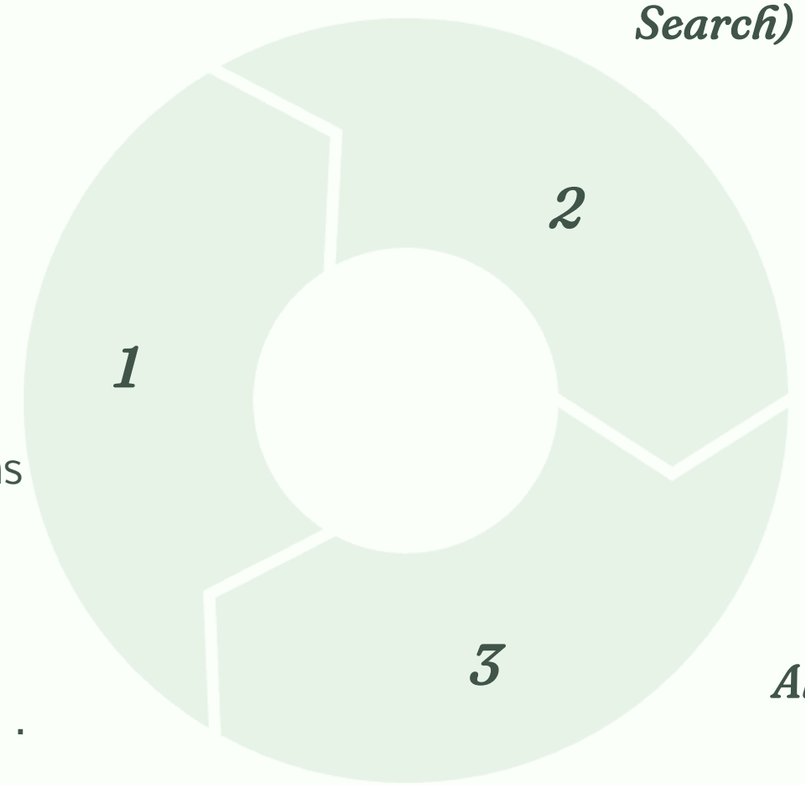
Nunca superestimam o custo real, garantindo que a solução encontrada seja ótima.

Em resumo, a busca heurística utiliza informações adicionais para guiar o processo de busca, tornando-o mais eficiente e direcionado. A escolha da heurística correta é fundamental para o sucesso do algoritmo.

# 3. Métodos Específicos de Busca Heurística

### *Subida de Encosta (Hill Climbing)*

- Algoritmo local que sempre escolhe a melhor opção disponível no momento, sem considerar o caminho futuro.
- Baseia-se na busca em profundidade, mas sem backtracking.
- Pode ficar preso em máximos locais, platôs ou cordilheiras, falhando em encontrar a melhor solução global.




### *Busca Gulosa (Greedy Best-First Search)*

- Expande o nó mais próximo do objetivo com base na heurística h(n) sem considerar o custo acumulado g(n).
- É rápida, mas não necessariamente ótima, pois pode seguir caminhos subótimos.

### *Algoritmo A**

- Combina os custos g(n) e h(n) para escolher o nó mais promissor:
  - f(n)=g(n)+h(n)
- É completo e ótimo, desde que h(n)h(n)h(n) seja admissível (para buscas em árvore) ou consistente (para buscas em grafo).
- Pode exigir grande espaço de memória devido à necessidade de armazenar muitos estados.

# *Categorias de Métodos de Busca e Suas Aplicações*

1

### *Busca em Largura*

Ideal para encontrar o caminho mais curto em grafos não ponderados.

2

### *Busca de Custo Uniforme*

Útil quando os custos das ações variam e a otimização é essencial.

3

### *Busca em Profundidade*

Adequada para espaços de busca com memória limitada.

4

### *Busca em Profundidade Limitada*

Evita loops infinitos, definindo um limite de profundidade.

5

### *Busca de Aprofundamento Iterativo*

Combina as vantagens da busca em largura e profundidade.

Cada categoria de método de busca tem suas próprias vantagens e desvantagens, tornando-as adequadas para diferentes tipos de problemas. A escolha do método correto depende das características específicas do problema em questão.

# *Modelagem de Problema: O Labirinto*

Há um labirinto (tamanho NxN definido pelo usuário), com M obstáculos (definido pelo usuário), com uma SAÍDA (linha e coluna sorteados). Contudo, este labirinto possui 2 ENTRADAS (linha e coluna sorteadas para cada entrada). Este problema está no repositório de solução da disciplina via o pacote do professor Jomi Hübner. O desafio é fazer com que cada entrada utilize um método de busca definido pelo usuário e o programa gere a solução para cada entrada, comparando as soluções.

# *Estados do Problema do Labirinto*

*1*

### *Classe; atributos; tipos*

- **Labirinto:** tamanho (int), matriz (array bidimensional).
- **Posição:** (x, y) (int, int).
- **Entradas:** entrada1, entrada2 (Posição).
- **Saída:** saida (Posição).
- **Obstáculos:** lista_obstaculos (lista de Posições).

*2*

### *Estado Inicial*

Entradas nas posições sorteadas (Xe1, Ye1) e (Xe2, Ye2).
Saída na posição sorteada (Xs, Ys).
Obstáculos posicionados no grid.

*3*

### *Estados Finais*

Um dos caminhos encontrados chega à saída:
posição_atual == saída.

# *Regras de Transição*

Métodos da classe:

1. mover_cima();
2. mover_baixo();
3. mover_esquerda();
4. mover_direita();

# *Como tratar os estados já visitados*

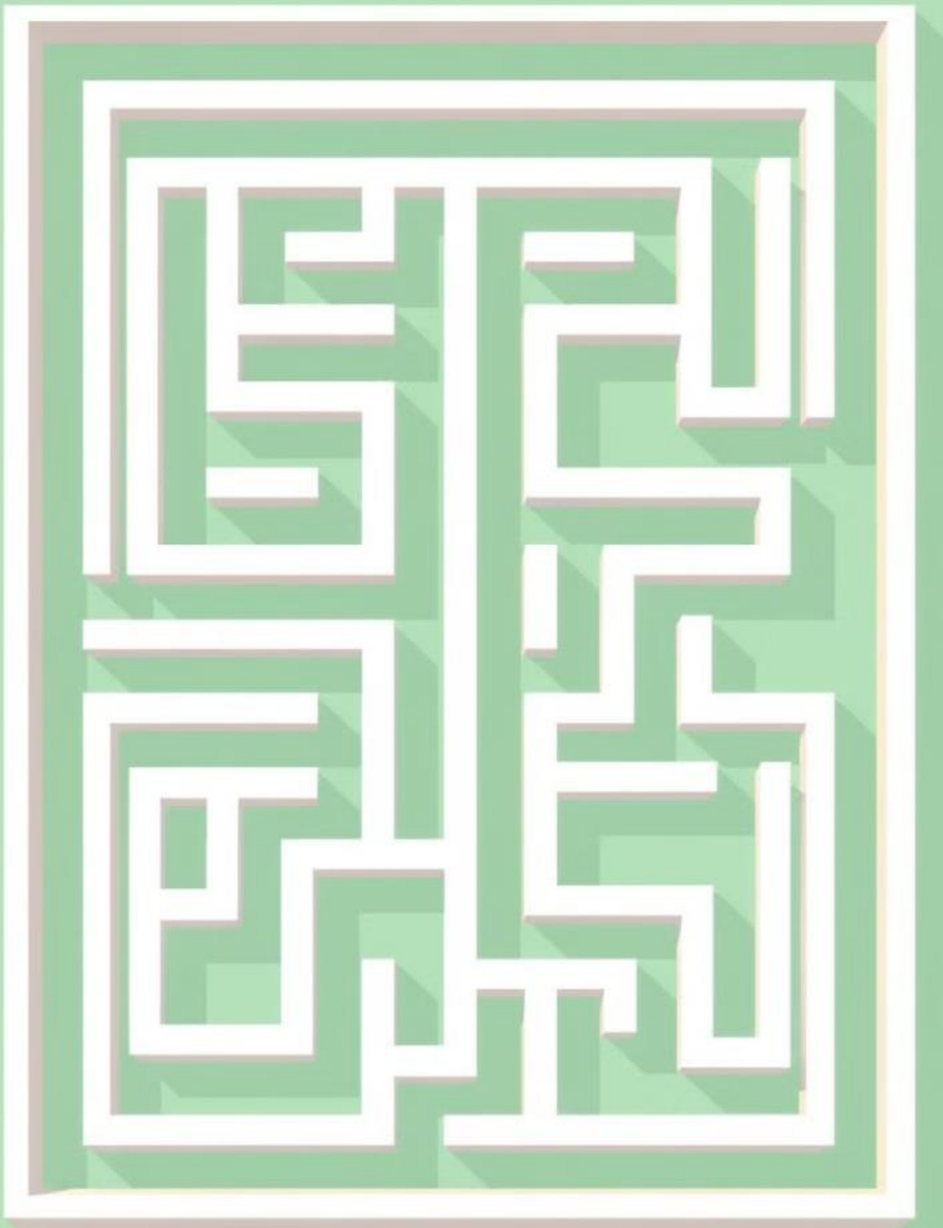
**Lista encadeada** ou **HashSet**, armazenando os estados na forma de string "x,y" para evitar redundâncias.

*Exemplo:*

visitados = {"0,0", "1,0", "2,0"};

# Mapeamento das Restrições

1. *O espaço de busca é finito (N×N).*
2. *Estados inválidos são obstáculos e saídas dos limites do grid.*
3. *As movimentações são apenas nas quatro direções (cima, baixo, esquerda, direita).*
4. *Não há heurística obrigatória, mas pode ser usada para buscas informadas.*



# *Perguntas no Processo de Busca*

Cada algoritmo de busca deve verificar três condições para cada nó antes de expandi-lo:

### *É um estado válido?*

A posição não pode estar fora do grid nem ser um obstáculo.

### *O estado já foi visitado?*

Deve evitar expandir estados redundantes.

### *É a meta?*

Se a posição atual for igual à saída, o algoritmo retorna o caminho encontrado.

# *Solução com Busca em Largura*

- **Estrutura usada:** Fila.

- **Funcionamento:**
  1. Adiciona a entrada na fila.
  2. Expande os movimentos válidos.
  3. Verifica se o nó é o objetivo.
  4. Adiciona novos nós à fila e marca como visitado.
  5. Repete até encontrar a saída.

- **Exemplo de Expansão com Busca em Largura**

  Entrada: (0,0)

  (0,0)

  (0,1) (1,0)

  (0,2) (1,1) (1,1) (2,0)

- **Visitados:** ["0,0", "0,1", "1,0", "0,2", "1,1", "2,0"]

- **Caminho encontrado:** (0,0) → (0,1) → (0,2) → (1,2) → saída.

# *Solução com Busca em Profundidade*

- **Estrutura usada:** Pilha.
- **Funcionamento:**
  - Adiciona a entrada na pilha.
  - Explora um caminho profundamente até encontrar um beco sem saída ou a solução.
  - Se o caminho falhar, retrocede (backtracking).

- **Exemplo de Expansão com Busca em Profundidade:**

  **Entrada:** (0,0)

  (0,0)

  (1,0)

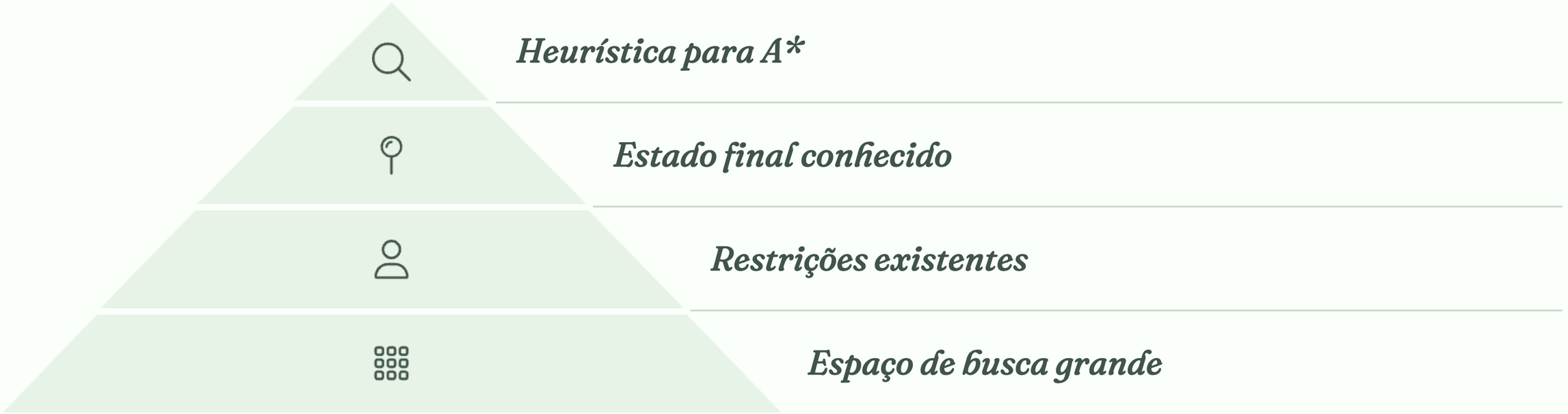
  (2,0)

  (2,1)

  (2,2) → saída
- **Visitados:** ["0,0", "1,0", "2,0", "2,1", "2,2"]

- **Caminho encontrado:** (0,0) → (1,0) → (2,0) → (2,1) → saída.

# *Características do Problema do Labirinto*




1. O espaço de busca pode ser grande para $N$ elevado.
2. O estado final é conhecido, facilitando buscas informadas.
3. Existem restrições (obstáculos, limites do grid).
4. Heurística pode ser aplicada para buscas como A*.