Métodos de Busca

Técnicas IA

Temas

Tópicos

1. O que são?
2. Para que servem?
3. Categorias e Aplicações
4. Modelagem de um problema

O que são métodos de busca?

Técnicas de IA para encontrar um caminho até um estado final (conhecido ou não).

Baseiam-se em uma modelagem do problema:

- Estados
- Regras de transição
- Restrições
- Conjunto de visitados
- Função meta

Os métodos de busca são as estratégias usadas para aplicar as regras de transição entre os estados.

Para que servem os métodos de busca?

Resolver problemas de forma automatizada e inteligente.

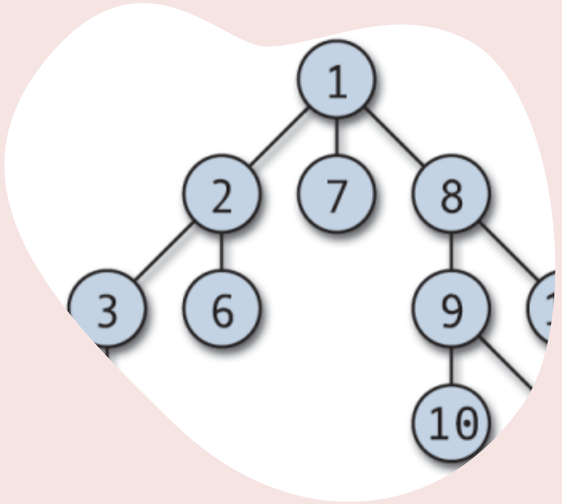
Explorar espaços de estados para alcançar objetivos específicos.

Utilizados quando não se conhece de antemão o caminho até a solução.

Aplicações:

- ✓ Planejamento de rotas
- ✓ Jogos e tomada de decisão
- ✓ Robôs autônomos
- ✓ Sistemas especialistas e IA em geral

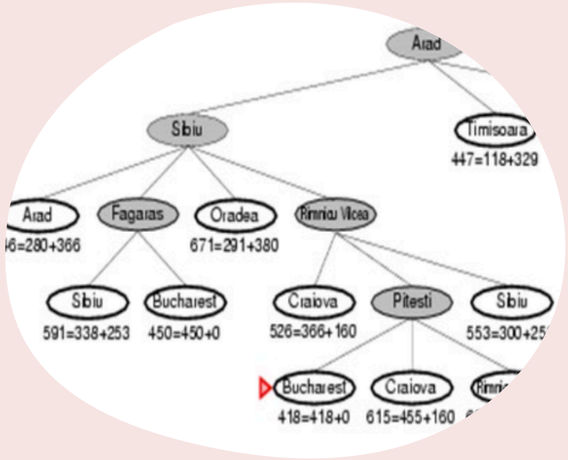
Categorias de métodos de busca



Cegos (força bruta)

Não utilizam nenhuma informação adicional.

Aplicáveis quando:
- Não se tem heurística
- Poucas restrições
- Há hardware sobrando



Informados (heurísticos)

Usam dicas ou estimativas (heurísticas) para guiar a busca.

Ideais quando:
- Há muitas restrições
- O hardware é limitado
- Existe alguma “intuição” sobre o caminho

Busca Cega (Não Informada)

Busca em Largura (**BFS**)
Estratégia tipo “amplitude”
Usa uma fila (FIFO)
✓ Garante o menor número de passos
✗ Usa muita memória

Busca em Profundidade (**DFS**)
Estratégia tipo “profundidade”
Usa uma pilha (ou recursão)
✓ Pouco consumo de memória
✗ Pode cair em ciclos ou caminhos ruins

Busca de Custo Uniforme (**UCS**)
Escolhe sempre o nó com menor custo acumulado
✓ Boa para caminhos com custos variados
✗ Pode ser lenta

Busca Heurística (Informada)

Usa informação privilegiada (heurística) para aplicar regras de transição.
A heurística pode vir de um especialista ou ser gerada pelo computador.
Tipos de custo usados:

- Custo real: g(n)
- Custo estimado: h(n)

Aplicável quando:

✓ Há muitas restrições
✓ Se conhece parcialmente o caminho
✓ Recursos computacionais são limitados

Busca A*

Combina g(n) + h(n)
Tenta corrigir o impulso da gulosa com o histórico do caminho
✓ Completa e ótima (se h(n) for admissível)
✗ Pode ser pesada em memória

Hill Climbing (Subida da Encosta)

Explora sempre o melhor vizinho (menor g(n))
Tipo profundidade
✓ Simples e rápida
✗ Pode parar em máximos locais

Busca Gulosa (Greedy)

Baseada no menor h(n)
Tipo amplitude
✓ Vai direto ao alvo
✗ Pode ignorar melhores caminhos

Comparativo entre algoritmos

Método	Tipo	Usa Heurística?	Caminho Ótimo?	Estrutura
Largura (BFS)	Cega	✗	✓	Fila
Profundidade	Cega	✗	✗	Pilha
UCS	Cega	✗	✓ (por custo)	Fila por custo
Gulosa	Heurística	✓ (h(n))	✗	Fila por heurística
A*	Heurística	✓ (g(n)+h(n))	✓	Fila por f(n)
Hill Climb	Heurística	✓ (g(n))	✗	Vizinho melhor

Modelagem do Labirinto com duas entradas

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). 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

Classe: LabirintoObstaculos

Atributos:

- matriz: Matriz de caracteres (char[N][N]) representando o labirinto.
- linhaEntrada1, colunaEntrada1: Posição da primeira entrada (int, int).
- linhaEntrada2, colunaEntrada2: Posição da segunda entrada (int, int).
- linhaSaida, colunaSaida: Posição da saída (int, int).
- op: String que descreve a operação que gerou o estado.

Estado inicial:

Matriz inicializada com:

- Duas entradas ('E') em posições aleatórias.
- Uma saída ('S') em uma posição aleatória.
- Obstáculos ('@') distribuídos aleatoriamente com base na porcentagem definida.
- Células livres ('O') no restante do labirinto.

Estados finais:

Qualquer uma das entradas ('E') alcança a saída ('S'), ou seja:

- linhaEntrada1 == linhaSaida && colunaEntrada1 == colunaSaida OU
- linhaEntrada2 == linhaSaida && colunaEntrada2 == colunaSaida.

Regras de Transição do Problema do Labirinto

Métodos da Classe

- Mover entrada 1 para cima
- Mover entrada 1 para baixo
- Mover entrada 1 para esquerda
- Mover entrada 1 para direita
- Mover entrada 2 para cima
- Mover entrada 2 para baixo
- Mover entrada 2 para esquerda
- Mover entrada 2 para direita

Cada método

- Verifica se o movimento é válido (não sair do labirinto e não colidir com obstáculos).
- Atualiza a posição da entrada 1 na matriz.

Mapear as Restrições do Problema

Movimentos válidos:

- Não sair dos limites do labirinto.
- Não colidir com obstáculos ('@').

Objetivo:

- Uma das entradas ('E') deve alcançar a saída ('S').

Duas entradas:

- As entradas não podem ocupar a mesma posição.
- As entradas não podem ser colocadas em cima de obstáculos ou da saída.

Como Tratar os Visitados

Lista encadeada ou HashSet:

- Converter o estado atual em uma string única (por exemplo, concatenando todas as posições da matriz e das entradas/saída).

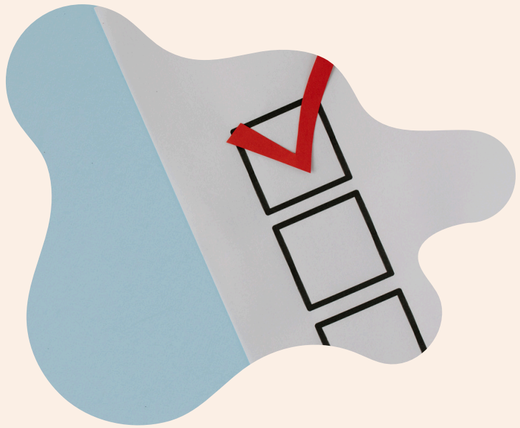
Verificação de visitados:

- Antes de adicionar um novo estado à lista de sucessores, verificar se ele já foi visitado usando a string única.

No Processo de Resolução do Problema por Métodos de Busca, São Feitas 3 Perguntas/Métodos:

É válido?

- Verifica se o movimento não viola as restrições (limites do labirinto e obstáculos).
- Implementado nos métodos de movimento.



Foi visitado?

- Verifica se o estado já foi explorado anteriormente.
- Implementado usando a lista de visitados (HashSet ou LinkedList).

É a meta/objetivo?

- Verifica se uma das entradas alcançou a saída.
- Implementado no método ehMeta().

Características do Problema

Espaço de busca:

- Depende do tamanho do labirinto (NxN).
- O espaço cresce exponencialmente com o aumento de N.

Estado final:

- Conhecido (uma das entradas alcança a saída).

Restrições:

- Movimentos limitados (cima, baixo, esquerda, direita).
- Obstáculos bloqueiam caminhos.

Heurística:

- Pode ser utilizada (por exemplo, distância de Manhattan até a saída) para otimizar a busca (A*).

.

Obrigada!

Att. Luana Hilbert