

Redes Neurais (Inteligência Artificial)

Aula 03 – Resolução de Problemas por
Meio de Busca

Edirlei Soares de Lima
<edirlei@iprj.uerj.br>



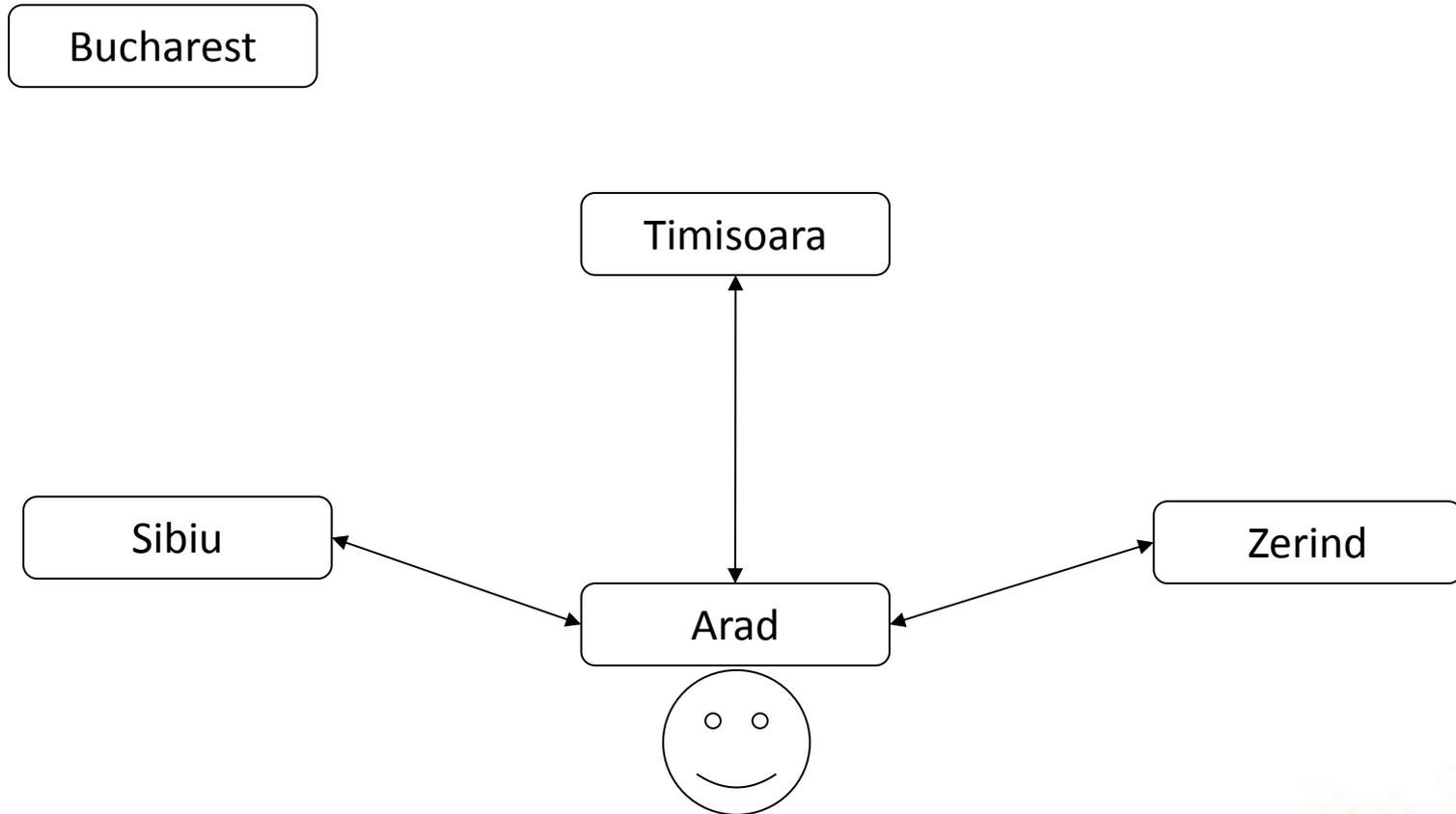
Introdução

- **Agentes Autônomos:**
 - Entidades capazes de observar o ambiente e agir de forma de forma autônoma, buscando atingir um determinado objetivo.
- **Tipos de Agentes:**
 - Agentes reativos simples;
 - Agentes reativos baseado em modelo;
 - Agentes baseados em objetivos;
 - Agentes baseados na utilidade;
 - Agentes baseados em aprendizado;

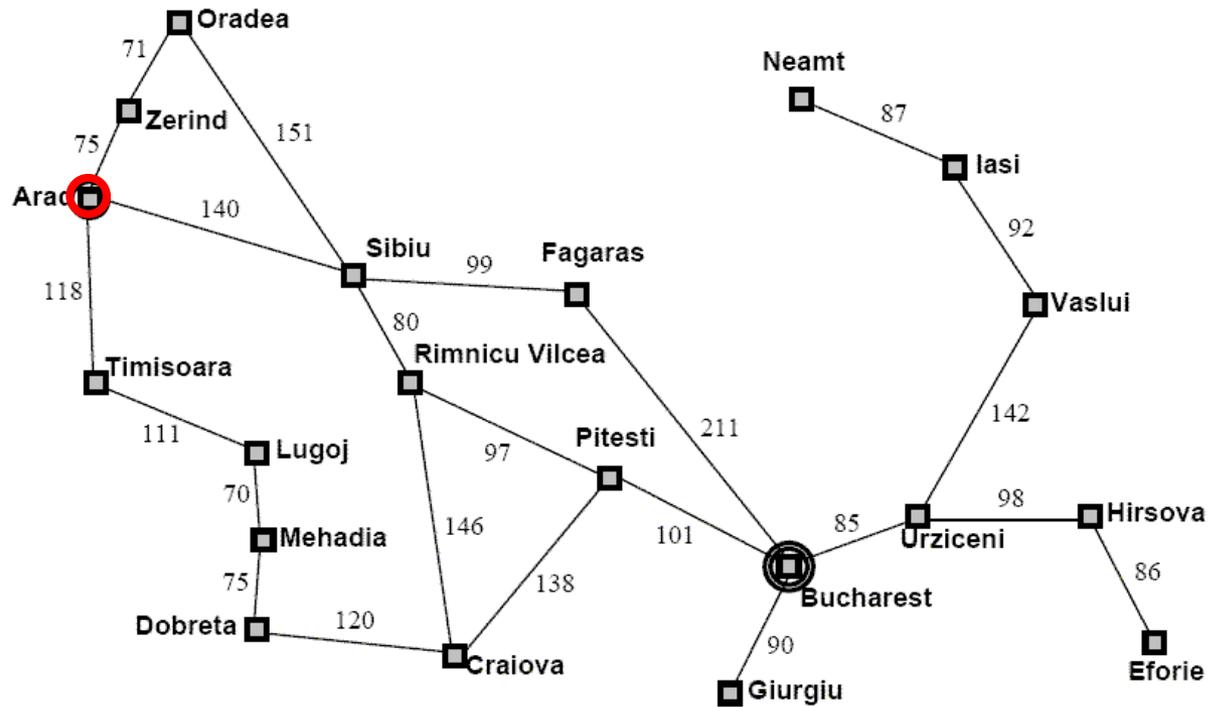
Problema de Busca

- **Objetivo:** Conjunto de estados que satisfazem o objetivo.
 - **Tarefa de Busca:** Encontrar a sequencia de ações que leva do estado atual até um estado objetivo.
 - Quais são os estados?
 - Quais são as ações?
 - Nível de abstração?
- 

Problema de Busca



Problema de Busca



Problema de Busca

- O processo de tentar encontrar uma sequencia de ações que leva de um estado até um estado objetivo é chamado de **busca**.
- Uma vez encontrada a solução, o agente pode **executar** a sequencia de ações para chegar no objetivo.
- Fases:
 - Formular objetivo
 - Buscar objetivo
 - Executar sequencia de ações

Definição do Problema

- A **definição do problema** é a primeira e mais importante etapa do processo de resolução de problemas de inteligência artificial por meio de buscas.
 - Consiste em analisar o **espaço de possibilidades** de resolução do problema, encontrar sequências de ações que levem a um objetivo desejado.
- 

Definição de um Problema

- **Estado Inicial:** Estado inicial do agente.
 - Ex: Em(Arad)
- **Estado Final:** Estado buscado pelo agente.
 - Ex: Em(Bucharest)
- **Ações Possíveis:** Conjunto de ações que o agente pode executar.
 - Ex: Ir(Cidade, PróximaCidade)
- **Espaço de Estados:** Conjunto de estados que podem ser atingidos a partir do estado inicial.
 - Ex: Mapa da Romênia.
- **Custo:** Custo numérico de cada caminho.
 - Ex: Distância em KM entre as cidades.

Considerações em Relação ao Ambiente

- **Estático:**

- O Ambiente não pode mudar enquanto o agente está realizando a resolução do problema.

- **Observável:**

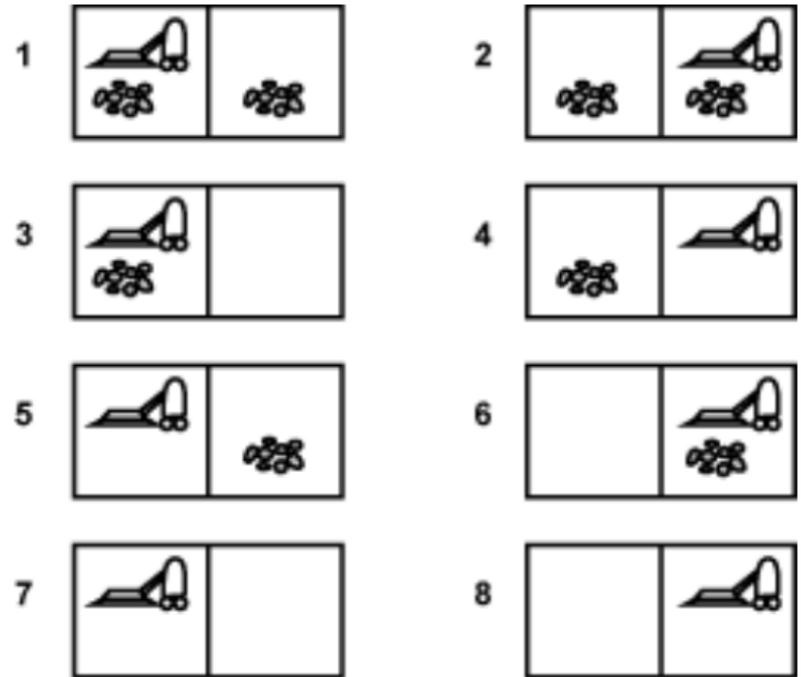
- O estado inicial do ambiente precisa ser conhecido previamente.

- **Determinístico:**

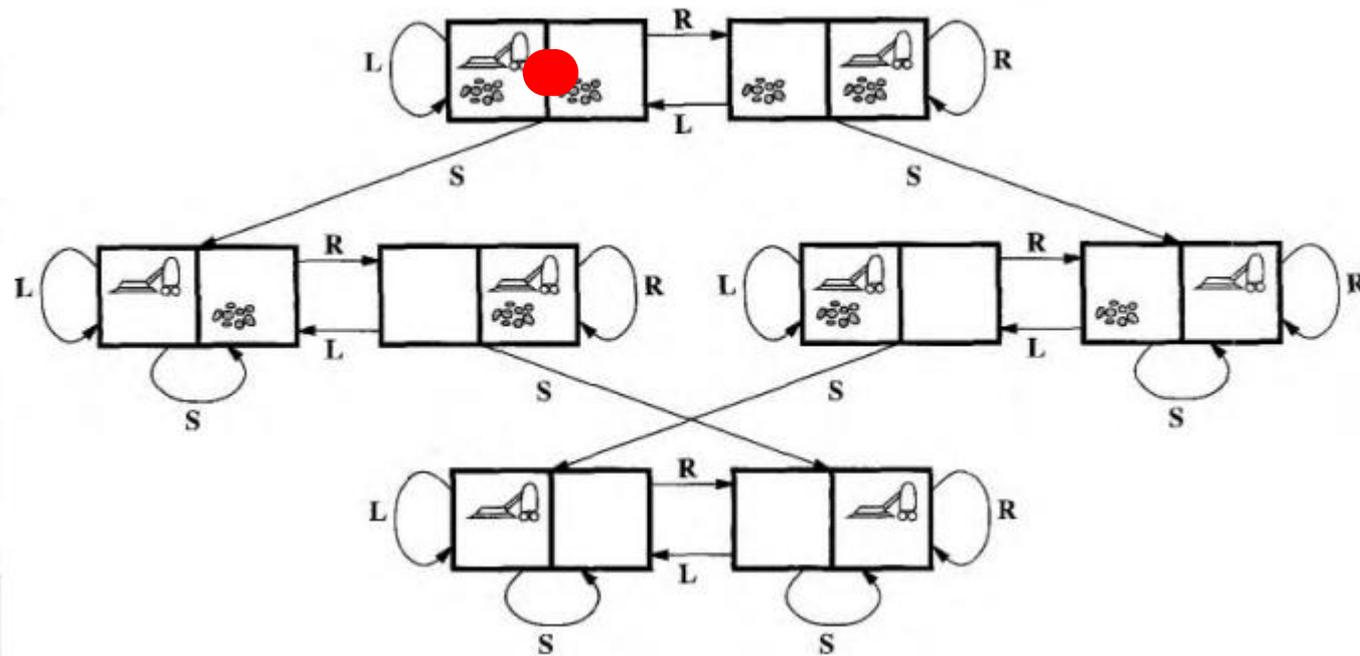
- O próximo estado do agente deve ser determinado pelo estado atual + ação. A execução da ação não pode falhar.

Exemplo: Aspirador de Pó

- **Espaço de Estados:** 8 estados possíveis (figura ao lado);
- **Estado Inicial:** Qualquer estado;
- **Estado Final:** Estado 7 ou 8 (ambos quadrados limpos);
- **Ações Possíveis:** Mover para direita, mover para esquerda e limpar;
- **Custo:** Cada passo tem o custo 1, assim o custo do caminho é definido pelo número de passos;



Exemplo: Aspirador de Pó



Exemplo: 8-Puzzle

- **Espaço de Estados:** 181.440 possíveis estados;
- **Estado Inicial:** Qualquer estado;
- **Estado Final:** Figura ao lado – Goal State;
- **Ações Possíveis:** Mover o quadrado vazio para direita, para esquerda, para cima ou para baixo;
- **Custo:** Cada passo tem o custo 1, assim o custo do caminho é definido pelo número de passos;
- **15-puzzle (4x4)** – 1.3 trilhões estados possíveis.
- **24-puzzle (5x5)** – 10^{25} estados possíveis.

| | | |
|---|---|---|
| 7 | 2 | 4 |
| 5 | | 6 |
| 8 | 3 | 1 |

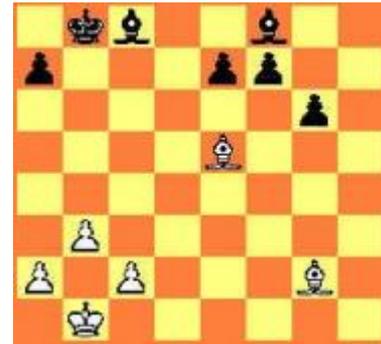
Start State

| | | |
|---|---|---|
| | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Goal State

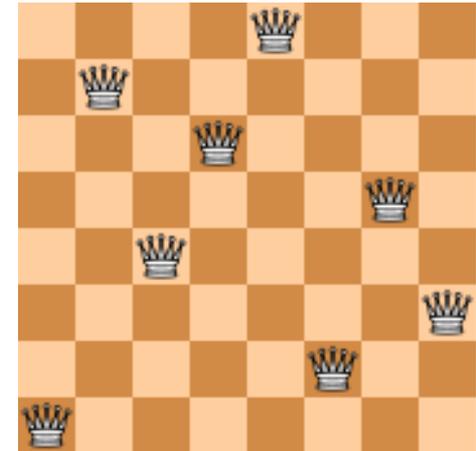
Exemplo: Xadrez

- **Espaço de Estados:** Aproximadamente 10^{40} possíveis estados (Claude Shannon, 1950);
- **Estado Inicial:** Posição inicial de um jogo de xadrez;
- **Estado Final:** Qualquer estado onde o rei adversário está sendo atacado e o adversário não possui movimentos válidos;
- **Ações Possíveis:** Regras de movimentação de cada peça do xadrez;
- **Custo:** Quantidade de posições examinadas;



Exemplo: 8 Rainhas (Incremental)

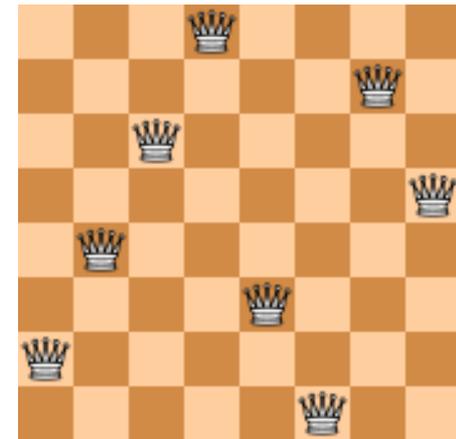
- **Espaço de Estados:** Qualquer disposição de 0 a 8 rainhas no tabuleiro (1.8×10^{14} possíveis estados);
- **Estado Inicial:** Nenhuma rainha no tabuleiro;
- **Estado Final:** Qualquer estado onde as 8 rainhas estão no tabuleiro e nenhuma está sendo atacada;
- **Ações Possíveis:** Colocar uma rainha em um espaço vazio do tabuleiro;
- **Custo:** Não importa nesse caso;



* O jogo possui apenas 92 possíveis soluções (considerando diferentes rotações e reflexões). E apenas 12 soluções únicas.

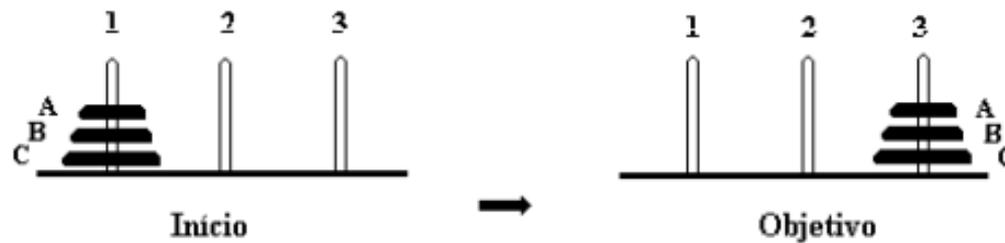
Exemplo: 8 Rainhas (Estados Completos)

- **Espaço de Estados:** Tabuleiro com n rainhas, uma por coluna, nas n colunas mais a esquerda sem que nenhuma rainha ataque outra (2057 possíveis estados);
- **Estado Inicial:** Nenhuma rainha no tabuleiro;
- **Estado Final:** Qualquer estado onde as 8 rainhas estão no tabuleiro e nenhuma está sendo atacada;
- **Ações Possíveis:** Adicionar uma rainha em qualquer casa na coluna vazia mais à esquerda de forma que não possa ser atacada;
- **Custo:** Não importa nesse caso;

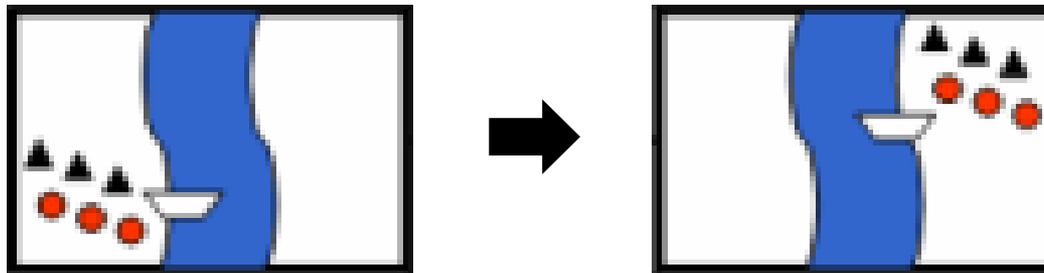


Exercícios

- Torre de Hanói?



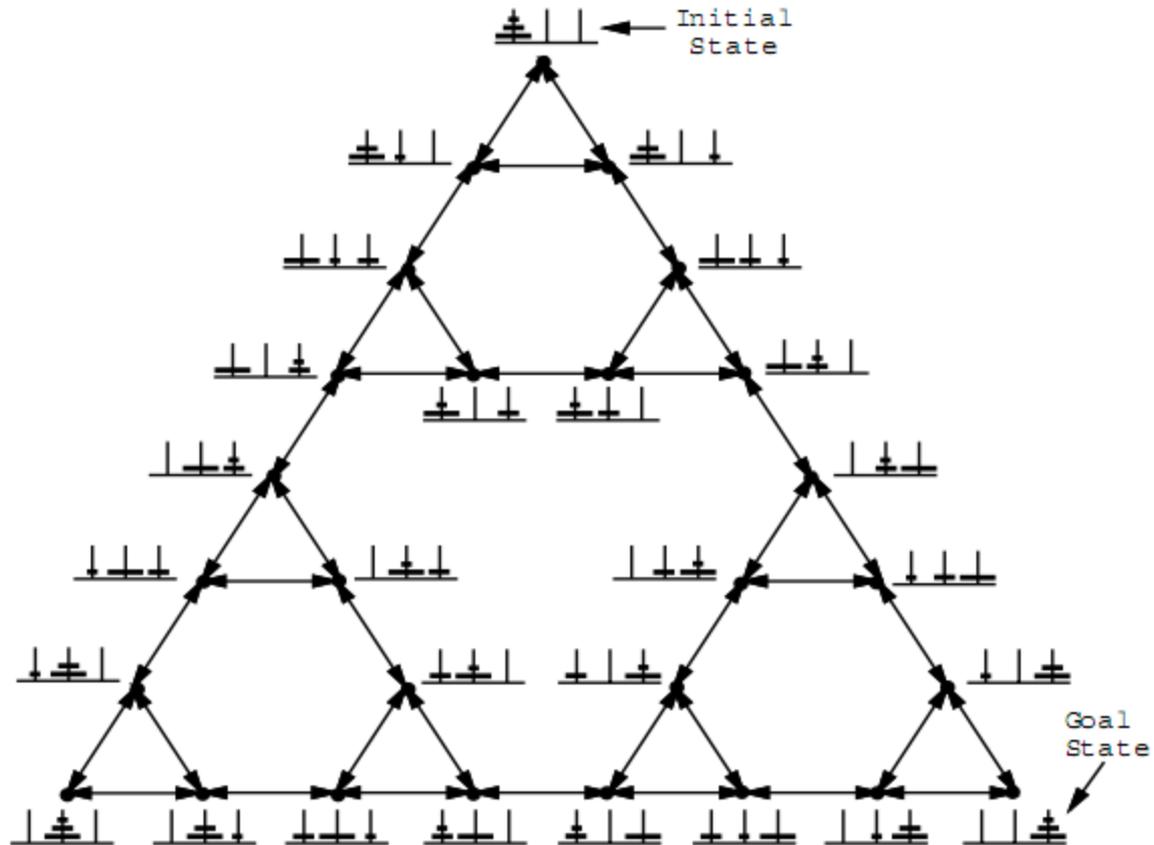
- Canibais e Missionários?



Exercícios

- Torre de Hanói:
 - **Espaço de Estados:** Todas as possíveis configurações de argolas em todos os pinos (27 possíveis estados).
 - **Ações Possíveis:** Mover a primeira argola de qualquer pino para o pino da direita ou da esquerda.
 - **Custo:** Cada movimento tem 1 de custo.

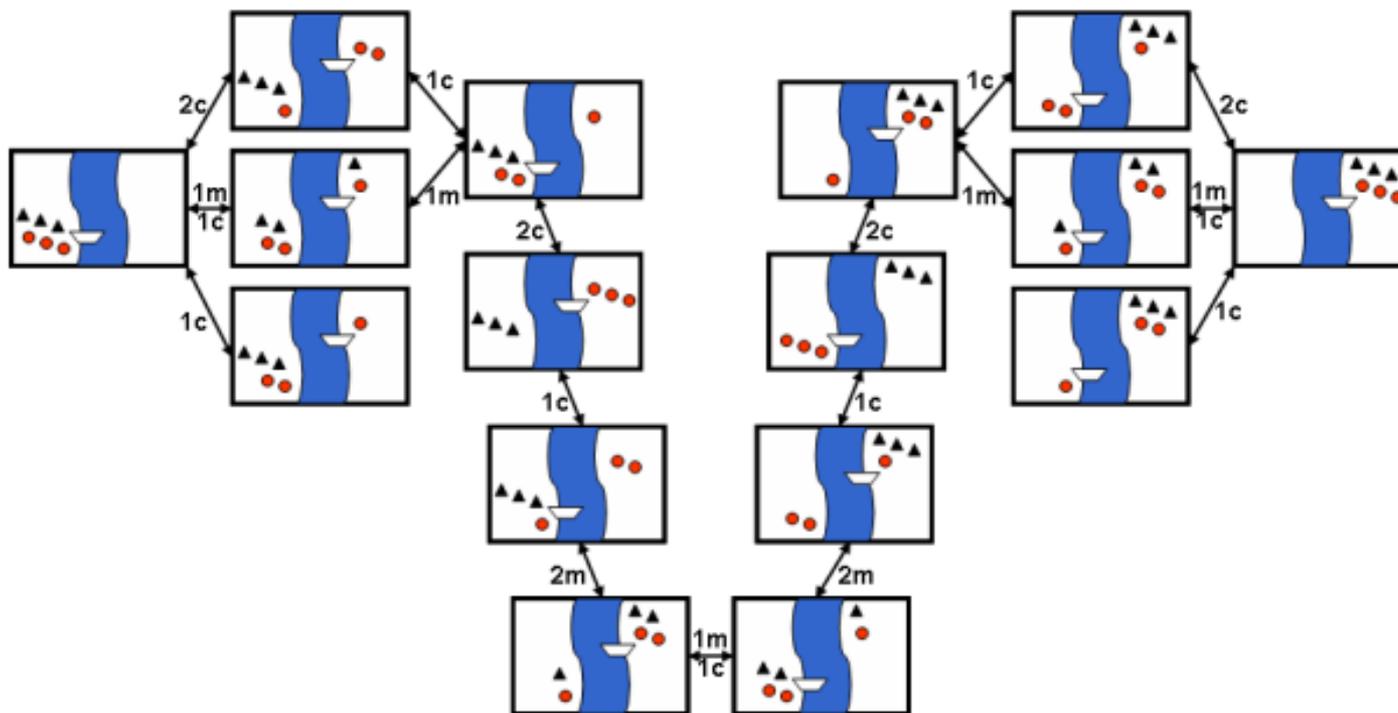
Exercícios



Exercícios

- Canibais e Missionários:
 - **Espaço de Estados:** Todas as possíveis configurações validas de canibais e missionários em cada lado do rio (16 possíveis estados).
 - **Ações Possíveis:** Mover 1 ou 2 personagens (canibais ou missionários) para o outro lado do rio. O número de canibais em um determinado lado do rio não pode ser maior do que o número de missionários.
 - **Custo:** Cada movimento tem 1 de custo.

Exercícios



Aplicações em Problemas Reais

- **Cálculo de Rotas:**

- Planejamento de rotas de aviões;
- Sistemas de planejamento de viagens;
- Caixeiro viajante;
- Rotas em redes de computadores;
- Jogos de computadores (rotas dos personagens);

- **Alocação**

- Salas de aula;
- Máquinas industriais;

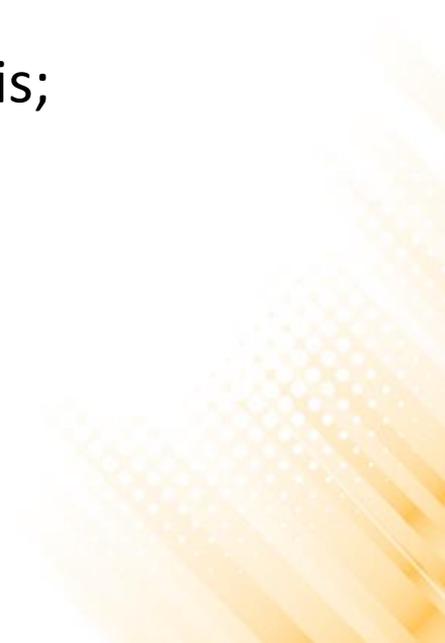
Aplicações em Problemas Reais

- **Circuitos Eletrônicos:**

- Posicionamento de componentes;
- Rotas de circuitos;

- **Robótica:**

- Navegação e busca de rotas em ambientes reais;
- Montagem de objetos por robôs;

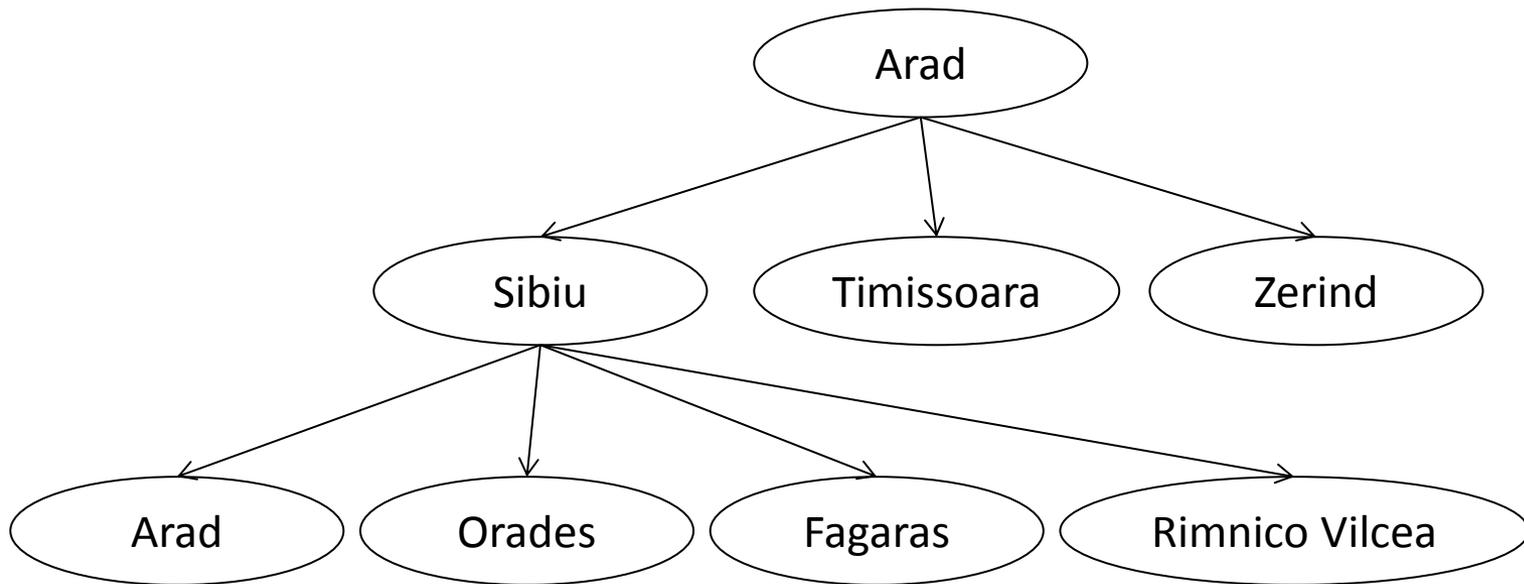


Como Encontrar a Solução?

- Com o problema bem formulado, o estado final (objetivo) deve ser “**buscado**” no espaço de estados.
- A busca é representada em uma **árvore de busca**:
 - Raiz: corresponde ao estado inicial;
 - Expande-se o estado corrente, gerando um novo conjunto de sucessores;
 - Escolhe-se o próximo estado a expandir seguindo uma **estratégia de busca**;
 - Prossegue-se até chegar ao estado final (solução) ou falhar na busca pela solução;

Buscando Soluções

- **Exemplo: Ir de Arad para Bucharest**

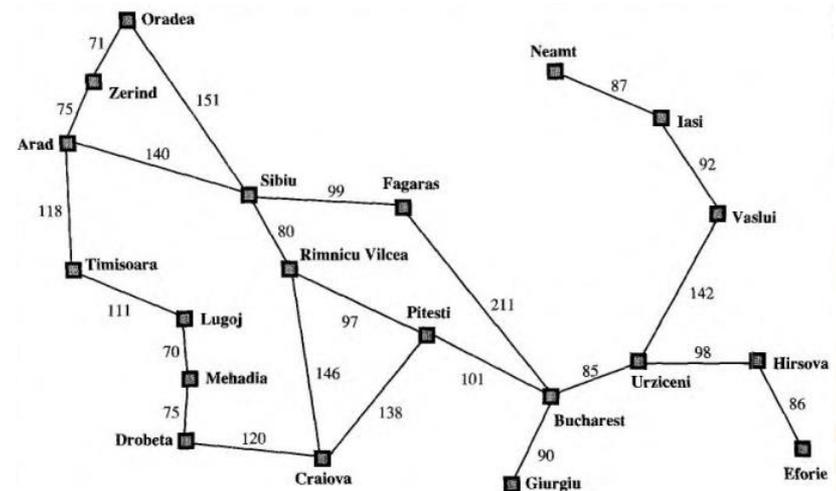


Buscando Soluções

- O espaço de estados é **diferente** da árvore de buscas.

- **Exemplo:**

- 20 estados no espaço de estados;
- Número de caminhos infinito;
- Árvore com infinitos nós;



Código Descritivo – Busca em Árvore

Função BuscaEmArvore(*Problema*, *Estratégia*) **retorna** solução ou falha

Início

Inicializa a arvore usando o estado inicial do *Problema*

loop do

se não existem candidatos para serem expandidos **então**

retorna falha

Escolhe um nó folha para ser expandido de acordo com a *Estratégia*

se Se o nó possuir o estado final **então**

retorna solução correspondente

se não

expande o nó e adiciona os nós resultantes a arvore de busca

Fim

Pseudocódigo – Busca em Árvore

Função BuscaEmArvore(*Problema*, *fronteira*) **retorna** solução ou falha

Início

fronteira ← InserirNaFila(FazNó(*Problema*[EstadoInicial]), *fronteira*)

loop do

se FilaVazia(*fronteira*) **então**

retorna falha

nó ← RemovePrimeiro(*fronteira*)

se nó[Estado] for igual a *Problema*[EstadoFinal] **então**

retorna Solução(nó)

fronteira ← InserirNaFila(Expandefronteira(nó, *Problema*), *fronteira*)

Fim

- A função **Solução** retorna a sequência de nós necessários para retornar a raiz da árvore.
- Considera-se *fronteira* uma estrutura do tipo fila.

Medida de Desempenho

- **Desempenho do Algoritmo:**
 - (1) O algoritmo encontrou alguma solução?
 - (2) É uma boa solução?
 - Custo de caminho (qualidade da solução).
 - (3) É uma solução computacionalmente barata?
 - Custo da busca (tempo e memória).
- **Custo Total**
 - Custo do Caminho + Custo de Busca.

Métodos de Busca

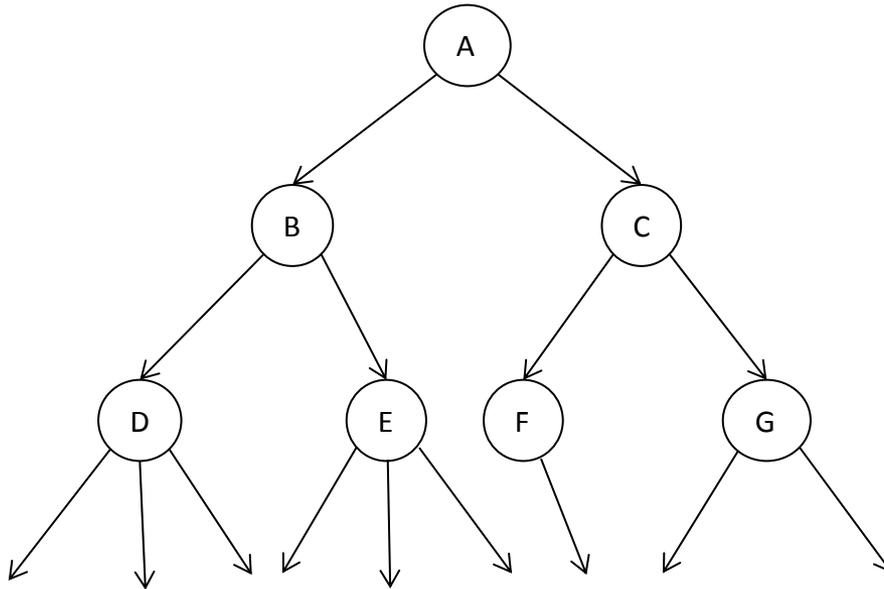
- **Busca Cega ou Exaustiva:**
 - Não sabe qual o melhor nó da fronteira a ser expandido. Apenas distingue o estado objetivo dos não objetivos.
- **Busca Heurística:**
 - Estima qual o melhor nó da fronteira a ser expandido com base em funções heurísticas.
- **Busca Local:**
 - Operam em um único estado e movem-se para a vizinhança deste estado.

Busca Cega

- **Algoritmos de Busca Cega:**
 - Busca em largura;
 - Busca em profundidade;
 - Busca de custo uniforme;
 - Busca com aprofundamento iterativo;

Busca em Largura

- **Estratégia:**
 - O nó raiz é expandido, em seguida todos os nós sucessores são expandidos, então todos próximos nós sucessores são expandidos, e assim em diante.



Busca em Largura

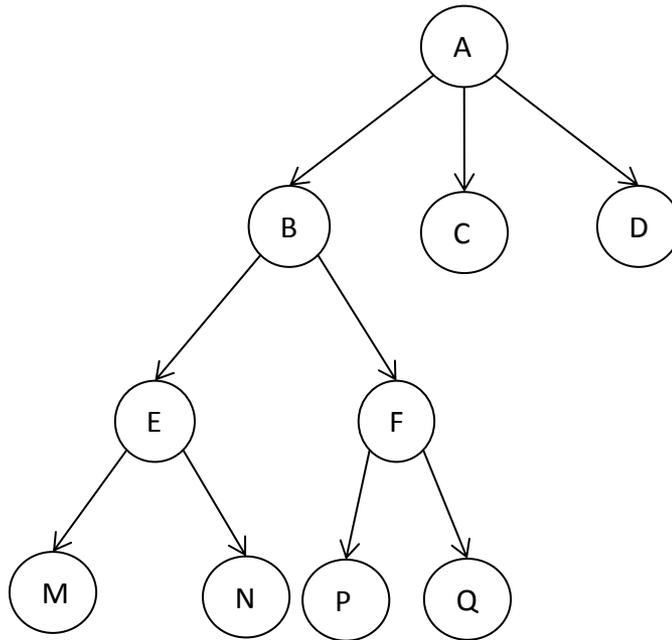
- Pode ser implementado com base no pseudocódigo da função “BuscaEmArvore” apresentado anteriormente. Utiliza-se uma estrutura de fila (first-in-first-out) para armazenar os nós da fronteira.
- **Complexidade:** $O(b^{d+1})$

| Profundidade (d) | Nós | Tempo | Memória |
|----------------------|-----------|----------|---------|
| 2 | 1100 | 0.11 ms | 107 KB |
| 4 | 111,100 | 11 ms | 10.6 MB |
| 6 | 10^7 | 1.1 seg | 1 GB |
| 8 | 10^9 | 2 min | 103 GB |
| 10 | 10^{11} | 3 horas | 10 TB |
| 12 | 10^{13} | 13 dias | 1 PB |
| 14 | 10^{15} | 3.5 anos | 99 PB |

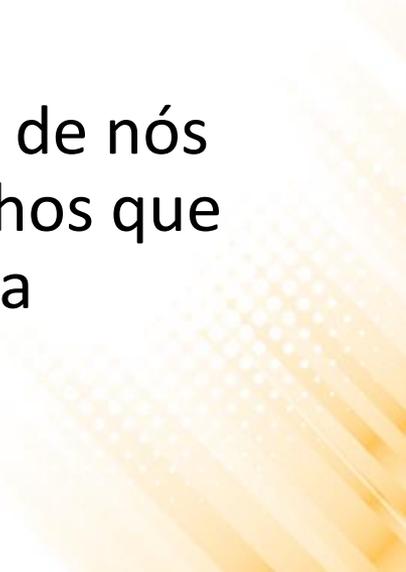
* Considerando o número de folhas $b = 10$ e cada nó ocupando 1KB de memória.

Busca em Profundidade

- **Estratégia:**
 - Expande os nós da vizinhança até o nó mais profundo.



Busca em Profundidade

- Pode ser implementado com base no pseudocódigo da função “BuscaEmArvore” apresentado anteriormente. Utiliza-se uma estrutura de pilha (last-in-first-out) para armazenar os nós das fronteiras.
 - Pode também ser implementado de forma recursiva.
 - **Consome pouca memória**, apenas o caminho de nós sendo analisados precisa armazenado. Caminhos que já foram explorados podem ser descartados da memória.
- 

Busca em Profundidade

- Uso de memória pela **busca em largura** em uma árvore com 12 de profundidade: 1000 TB.
- Uso de memória pela **busca em profundidade** em uma árvore com 12 de profundidade: 118 KB.
- **Problema:** O algoritmo pode fazer uma busca muito longa mesmo quando a resposta do problema esta localizado a poucos nós da raiz da árvore.

Exercício 01

- Considerando o seguinte labirinto e dispondo os estados sucessores na seguinte ordem: norte, leste, oeste, sul.

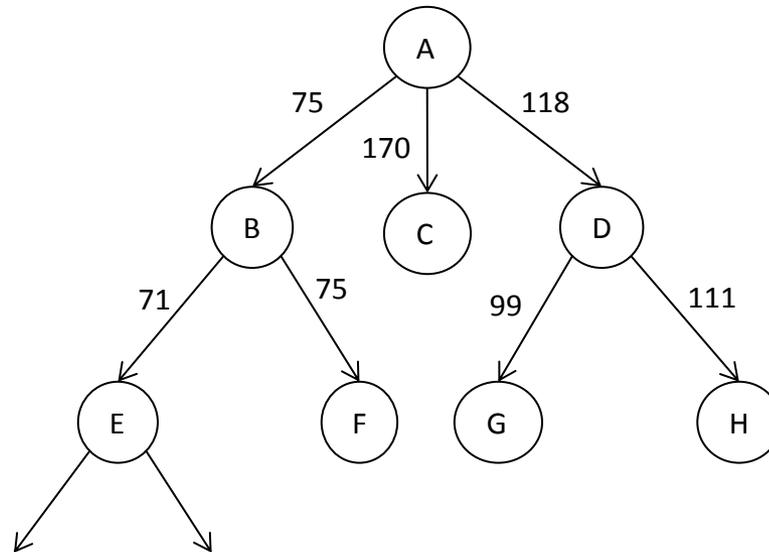
- a) Em qual ordem uma busca em profundidade visita as salas do labirinto?
- b) Em qual ordem uma busca em largura visita as salas do labirinto?

| | | | | |
|------------|---|---|---|-----------|
| A | B | C | D | E Goal |
| F | G | H | I | J |
| K | L | M | N | O |
| P | Q | E | S | T |
| U Start | V | X | Y | Z |

Busca de Custo Uniforme

- **Estratégia:**

- Expande sempre o nó de menor custo de caminho. Se o custo de todos os passos for o mesmo, o algoritmo acaba sendo o mesmo que a busca em largura.

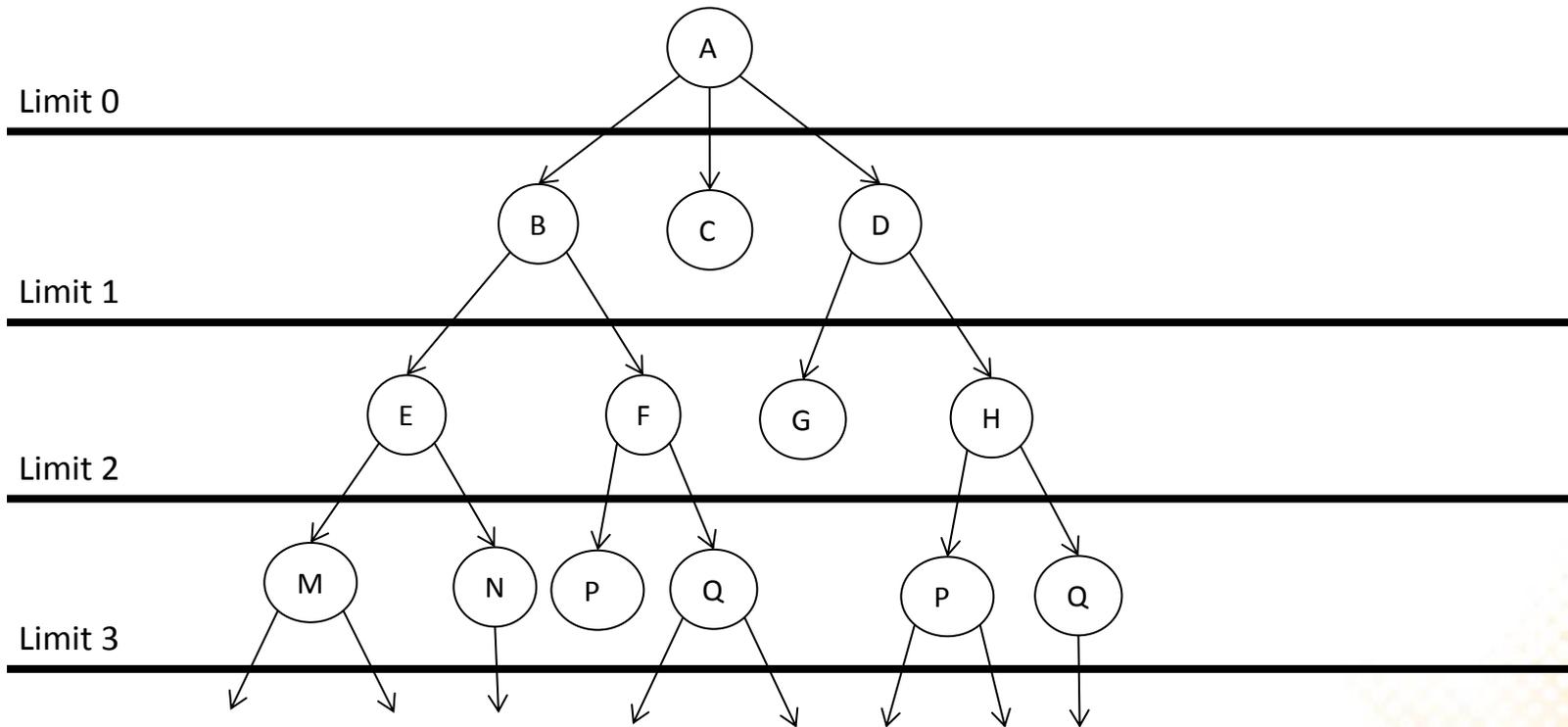


Busca de Custo Uniforme

- A primeira solução encontrada é a **solução ótima** se custo do caminho sempre aumentar ao longo do caminho, ou seja, não existirem operadores com custo negativo.
- Implementação semelhante a busca em largura. Adiciona-se uma **condição de seleção** dos nós a serem expandidos.
- **Complexidade:** $O(b^{1+(C/\alpha)})$
 - Onde:
 - $C = \text{custo da solução ótima};$
 - $\alpha = \text{custo mínimo de uma ação};$

Busca com Aprofundamento Iterativo

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.

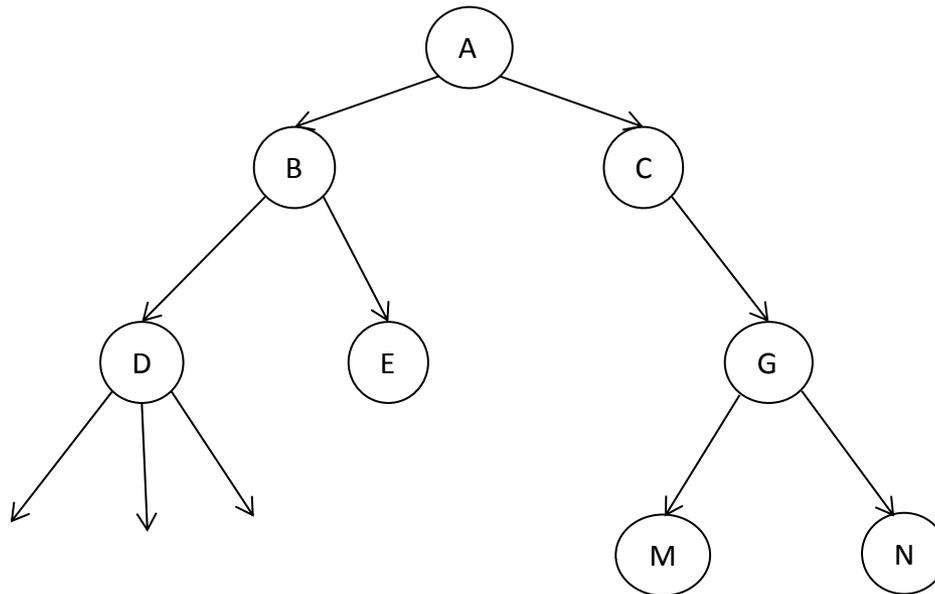


Busca com Aprofundamento Iterativo

- Combina os benefícios da busca em largura com os benefícios da busca em profundidade.
 - Evita o problema de caminhos muito longos ou infinitos.
 - A repetição da expansão de estados não é tão ruim, pois a maior parte dos estados está nos níveis mais baixos.
 - Cria menos estados que a busca em largura e consome menos memória.
- 

Busca Bidirecional

- **Estratégia:**
 - A busca se inicia ao mesmo tempo a partir do estado inicial e do estado final.



Comparação dos Métodos de Busca Cega

| Critério | Largura | Uniforme | Profundidade | Aprofundamento Iterativo | Bidirecional |
|------------------|------------------|-----------------------|---------------------|---------------------------------|---------------------|
| Completo? | Sim ¹ | Sim ^{1,2} | Não | Sim ¹ | Sim ^{1, 4} |
| Ótimo? | Sim ³ | Sim | Não | Sim ³ | Sim ^{3, 4} |
| Tempo | $O(b^{d+1})$ | $O(b^{1+(C/\alpha)})$ | $O(b^m)$ | $O(b^d)$ | $O(b^{d/2})$ |
| Espaço | $O(b^{d+1})$ | $O(b^{1+(C/\alpha)})$ | $O(bm)$ | $O(bd)$ | $O(b^{d/2})$ |

b = fator de folhas por nó.

d = profundidade da solução mais profunda.

m = profundidade máxima da árvore.

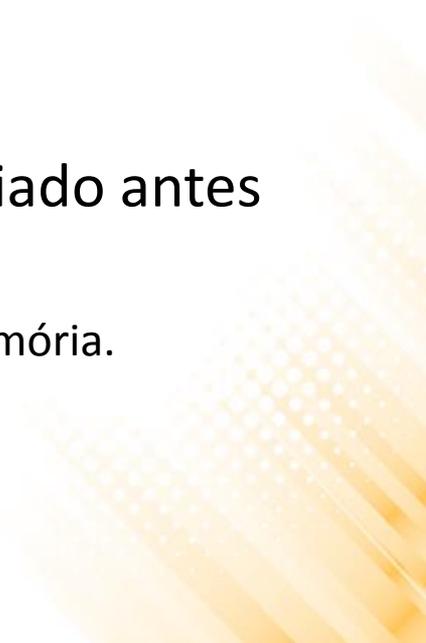
¹ completo se b for finito.

² completo se o custo de todos os passos for positivo.

³ ótimo se o custo de todos os passos for idêntico.

⁴ se ambas as direções usarem busca em largura.

Como evitar estados repetidos?

- Estados repetidos sempre vão ocorrer em problemas onde os estados são reversíveis.
 - Como evitar?
 - Não retornar ao estado “pai”.
 - Não retorna a um ancestral.
 - Não gerar qualquer estado que já tenha sido criado antes (em qualquer ramo).
 - Requer que todos os estados gerados permaneçam na memória.
- 

Leitura Complementar

- Russell, S. and Norvig, P. **Artificial Intelligence: a Modern Approach**, 3rd Edition, Prentice-Hall, 2009.
- **Capítulo 3: Solving Problems by Searching**

