



INF 1771 – Inteligência Artificial

Aula 03 – Resolução de Problemas
por Meio de Busca

Introdução

- **Agentes :**

- Entidades capazes de observar o ambiente e agir de forma autônoma com o objetivo de atingir um determinado objetivo.

- **Tipos de Agentes:**

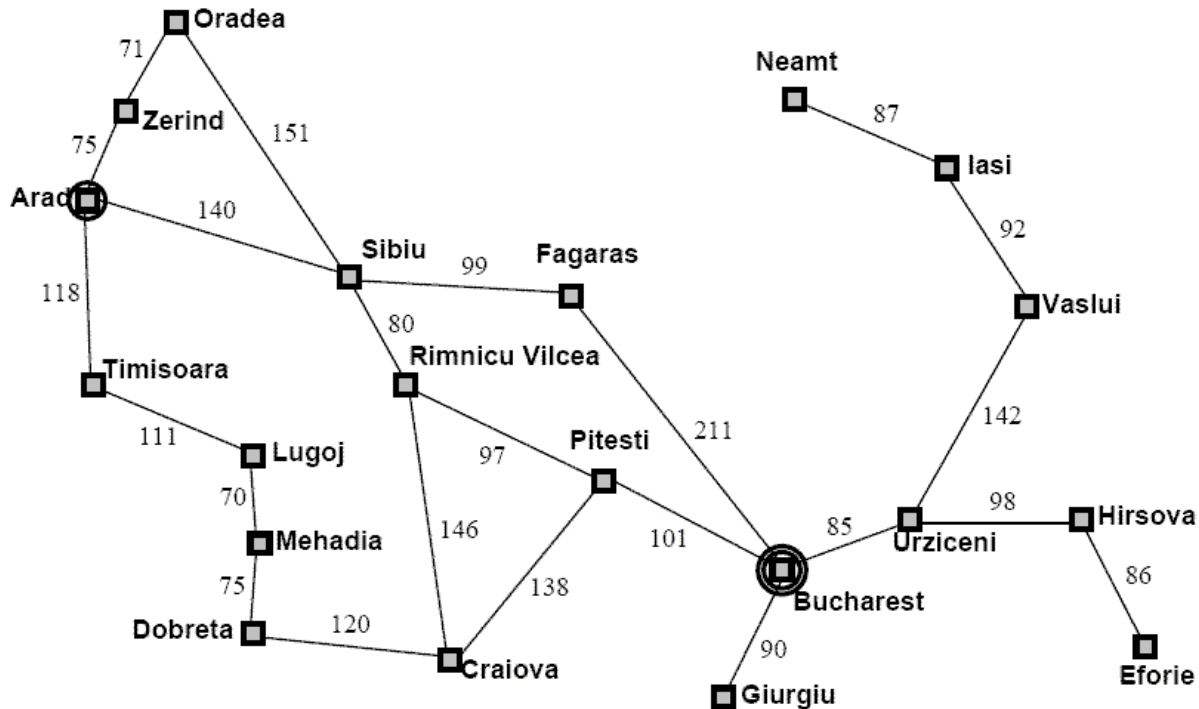
- Agentes reativos simples;
- Agentes reativos baseado em modelo;
- Agentes baseados em objetivos;
- Agentes baseados na utilidade;
- Agentes baseados em aprendizado;

Problema de Busca

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

Problema de Busca

- De férias na Romênia; atualmente em Arad.
- Vôo sai amanhã de Bucareste.



Exemplo: Romênia

- De férias na Romênia; atualmente em Arad.
- Vôo sai amanhã de Bucareste.
- Formular objetivo:
 - Estar em Bucareste
- Buscar Objetivo:
 - Formular problema:
 - estados: cidades
 - ações: dirigir entre as cidades
 - Encontrar solução:
 - sequência de cidades, ex., Arad, Sibiu, Fagaras, Bucareste.

Formulação do Problema

- A **formulação do problema** é a primeira e mais importante etapa do processo de resolução de problemas de inteligência artificial por meio de buscas.
- É o processo de decidir que ações e estados devem ser considerados, dado um objetivo.

Formulação do Problema

- A **formulação do problema** é a primeira e mais importante etapa do processo de resolução de problemas de inteligência artificial por meio de buscas.
- Jogue Xadrez!
- É suficiente?



Formulação do Problema

- como representar o tabuleiro
- configuração inicial do tabuleiro
- regras que definem as jogadas
- explicitar que não basta jogar: é preciso vencer!
- configuração que representa vitória



Componentes de um Problema

- Um problema é definido por mais alguns itens:
 - **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.

Componentes de um Problema

- Um problema é definido por mais alguns itens:

- **Estado Inicial:** Estado inicial do problema.
 - Ex: Em(Arad)

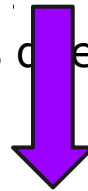
- **Estado Final:** Estado objetivo do problema.
 - Ex: Em(Bucharest)

- **Ações Possíveis:** Conjunto de ações que podem ser tomadas a partir de um estado.
 - Ex: Ir(Cidade, Próxima Cidade)

- **Espaço de Estados:** Conjunto de estados que podem ser atingidos a partir do estado inicial.
 - Ex: Mapa da Romênia.

- **Custo:** Custo numérico de uma ação.
 - Ex: Distância em KM entre duas cidades.

Espaço de estados do problema:
conjunto de todos os estados alcançáveis a partir do estado inicial por qualquer sequência de ações



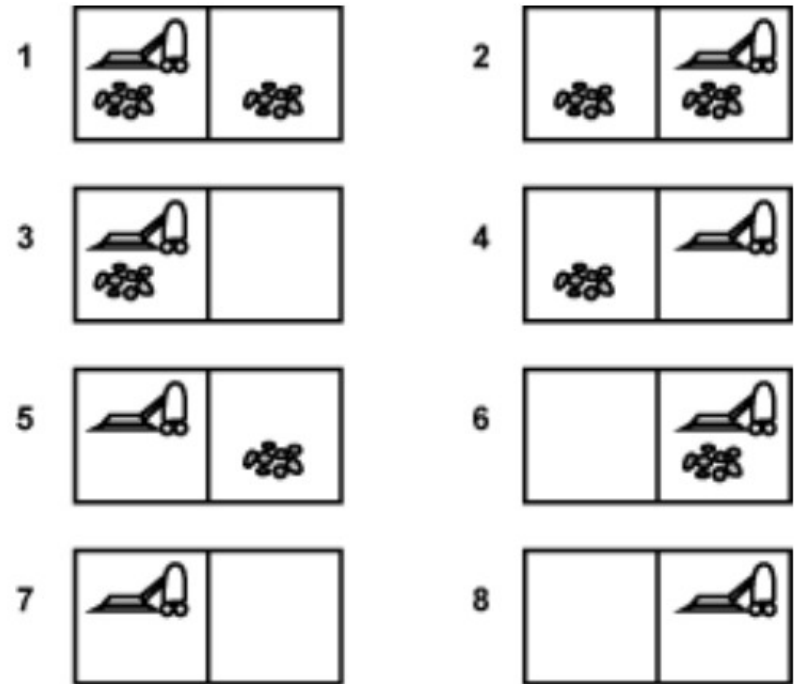
Grafo com os nós sendo os estados e as arestas são as ações

Componentes de um Problema

- Resumindo:
 - Um problema é um objetivo e um conjunto de meios para atingir o objetivo.
- Para ser resolvido por alguma técnica de IA, o problema é modelado considerando:
 - Estado inicial: por onde começar?
 - Objetivo: onde queremos chegar?
 - Ações: para onde ir a partir de cada estado? para onde as ações me levam?
 - Custo: qual o custo ao mudar de estado?

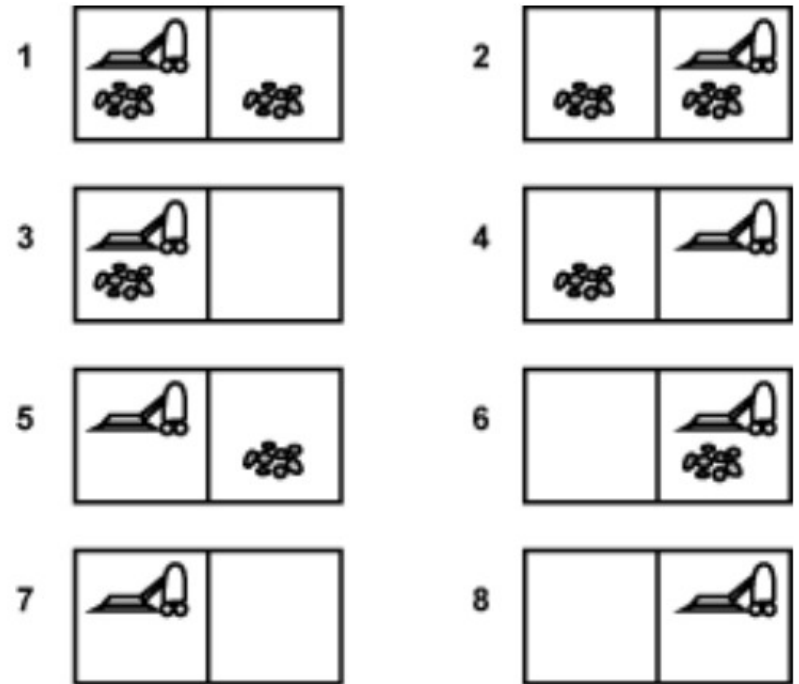
Exemplo: Aspirador de Pó

- Espaço de Estados:
- Estado Inicial:
- Estado Final:
- Ações Possíveis:
- Custo:



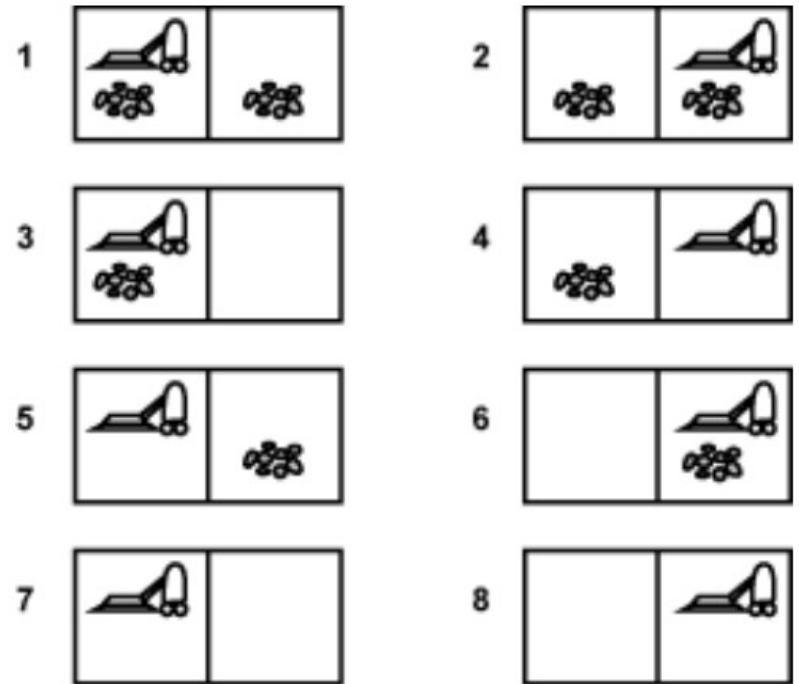
Exemplo: Aspirador de Pó

- **Espaço de Estados:** 8 estados possíveis (figura ao lado);
- **Estado Inicial:**
- **Estado Final:**
- **Ações Possíveis:**
- **Custo:**



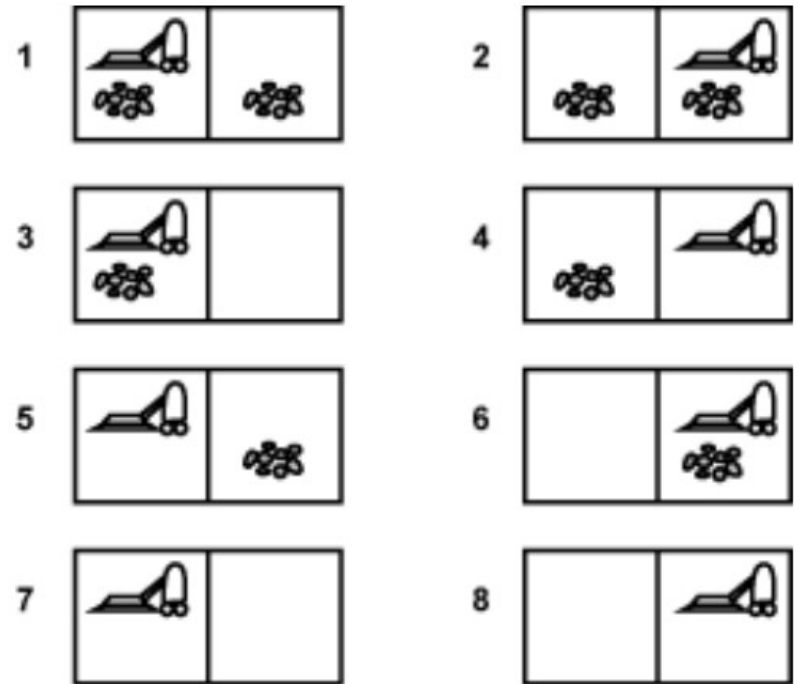
Exemplo: Aspirador de Pó

- **Espaço de Estados:** 8 estados possíveis (figura ao lado);
- **Estado Inicial:** Qualquer estado;
- **Estado Final:**
- **Ações Possíveis:**
- **Custo:**



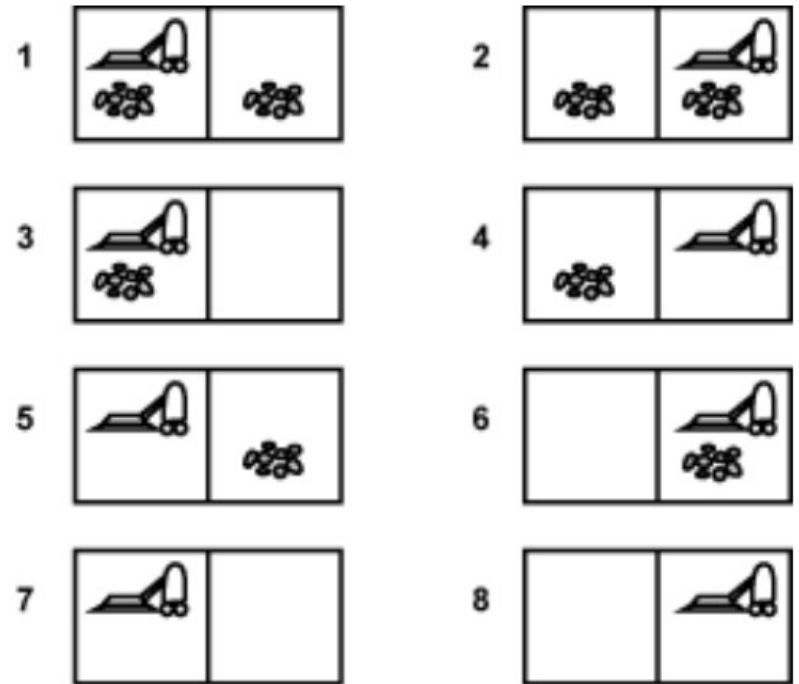
Exemplo: Aspirador de Pó

- **Espaço de Estados:** 8 estados possíveis (figura ao lado);
- **Estado Inicial:** Qualquer estado;
- **Estado Final:** Estado 7 ou 8 (ambos quadrados limpos);
- **Ações Possíveis:**
- **Custo:**



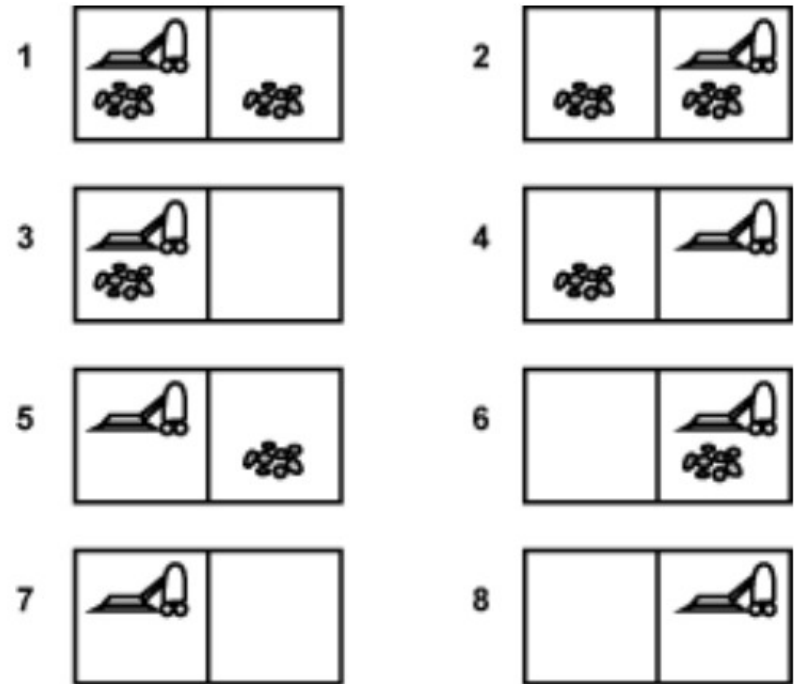
Exemplo: Aspirador de Pó

- **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 aspirar;
- **Custo:**



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 aspirar;
- **Custo:** Cada passo tem o custo 1, assim o custo do caminho é definido pelo número de passos;



Exemplo: 8-Puzzle

- **Espaço de Estados:**
- **Estado Inicial:**
- **Estado Final:**
- **Ações Possíveis:**
- **Custo:**

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

Start State

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

Goal State

Exemplo: 8-Puzzle

- **Espaço de Estados:** 181.440 possíveis estados;
- **Estado Inicial:**
- **Estado Final:**
- **Ações Possíveis:**
- **Custo:**

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

Start State

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

Goal State

Exemplo: 8-Puzzle

- **Espaço de Estados:** 181.440 possíveis estados;
- **Estado Inicial:** Qualquer estado;
- **Estado Final:**
- **Ações Possíveis:**
- **Custo:**

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

Start State

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

Goal State

Exemplo: 8-Puzzle

- **Espaço de Estados:** 181.440 possíveis estados;
- **Estado Inicial:** Qualquer estado;
- **Estado Final:** Figura ao lado – Goal State;
- **Ações Possíveis:**
- **Custo:**

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

Start State

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

Goal State

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:**

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

Start State

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

Goal State

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 |

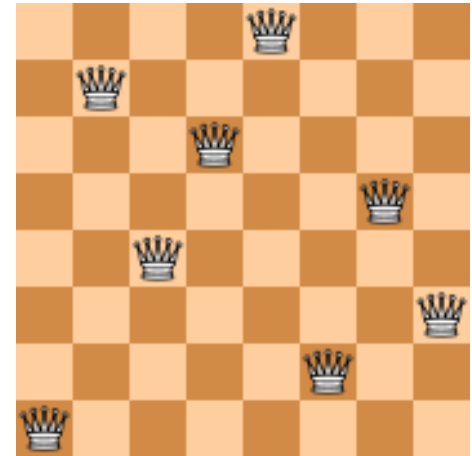
Start State

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

Goal State

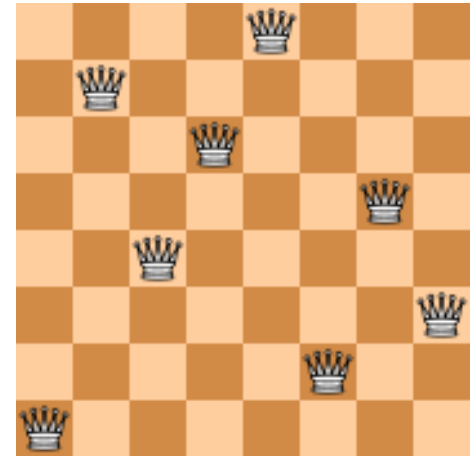
Exemplo: 8 Rainhas

- **Espaço de Estados:**
- **Estado Inicial:**
- **Estado Final:**
- **Ações Possíveis:**
- **Custo:**



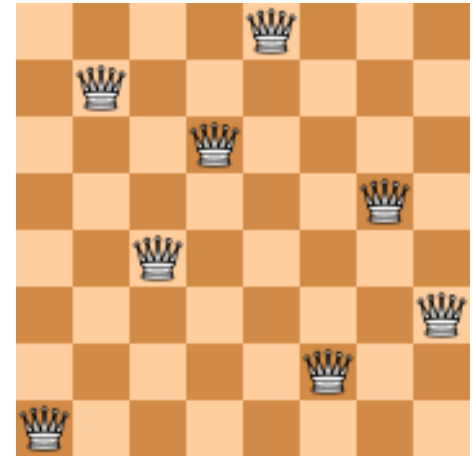
Exemplo: 8 Rainhas

- **Espaço de Estados:** Qualquer disposição de 0 a 8 rainhas no tabuleiro (1.8×10^{14} possíveis estados);
- **Estado Inicial:**
- **Estado Final:**
- **Ações Possíveis:**
- **Custo:**



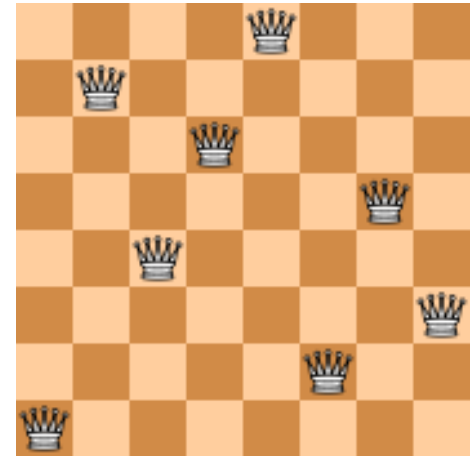
Exemplo: 8 Rainhas

- **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:**
- **Ações Possíveis:**
- **Custo:**



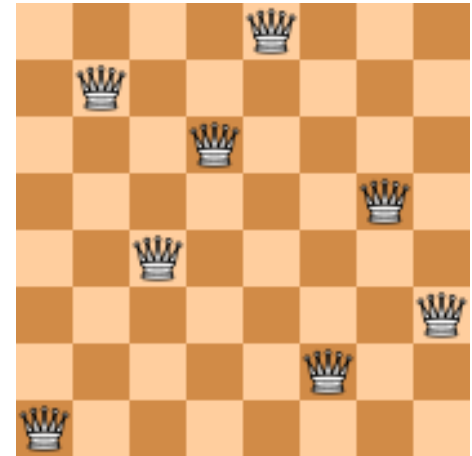
Exemplo: 8 Rainhas

- **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:**
- **Custo:**



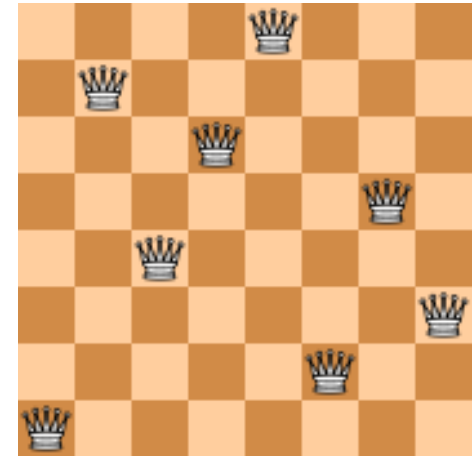
Exemplo: 8 Rainhas

- **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:**



Exemplo: 8 Rainhas

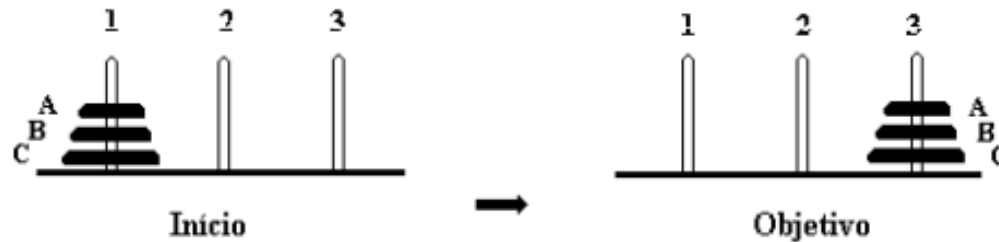
- **Espaço de Estados:** Qualquer disposição de 0 a 8 rainhas no tabuleiro (1.8×10^{14} possíveis estados);
- **Estado Inicial:** Nenhuma rainha no tabuleiro;
- **Estado Final:** Qualquer estado onde as 8 rainhas estão no tabuleiro e nenhuma está sendo atacada;
- **Ações Possíveis:** Colocar uma rainha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



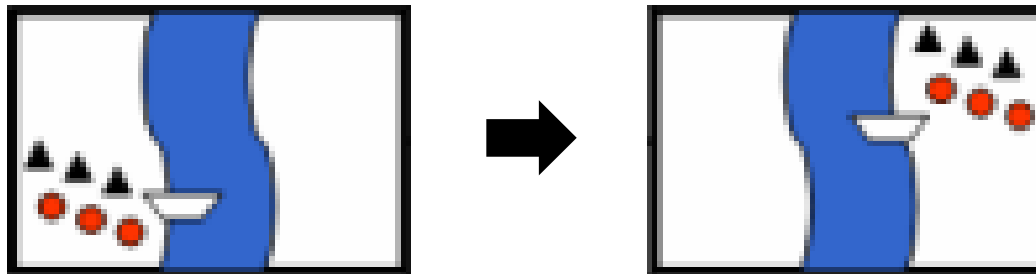
* O jogo possui apenas 92 possíveis soluções (considerando diferentes rotações e reflexões). E apenas 12 soluções únicas.

Exercícios

- Torre de Hanói?



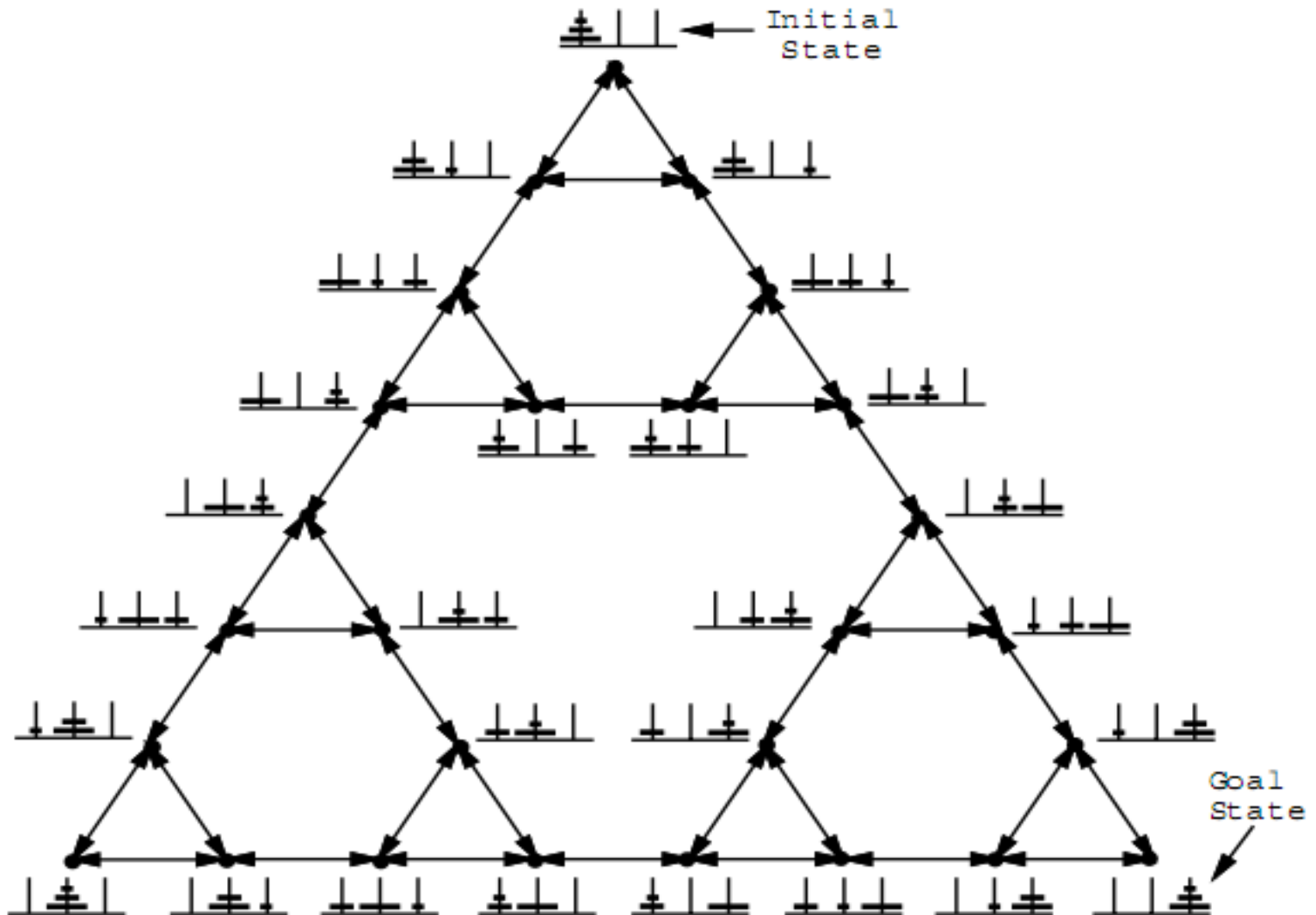
- Canibais e Missionários?



Exercícios

- Torre de Hanói (3 discos):
 - **Espaço de Estados:** Todas as possíveis configurações de argolas em todos os pinos (27 possíveis estados).
 - **Ações Possíveis:** Mover uma argola mais no topo de qualquer pino para o pino da direita ou da esquerda sem que fique em cima de um disco menor.
 - **Custo:** Cada movimento tem 1 de custo.

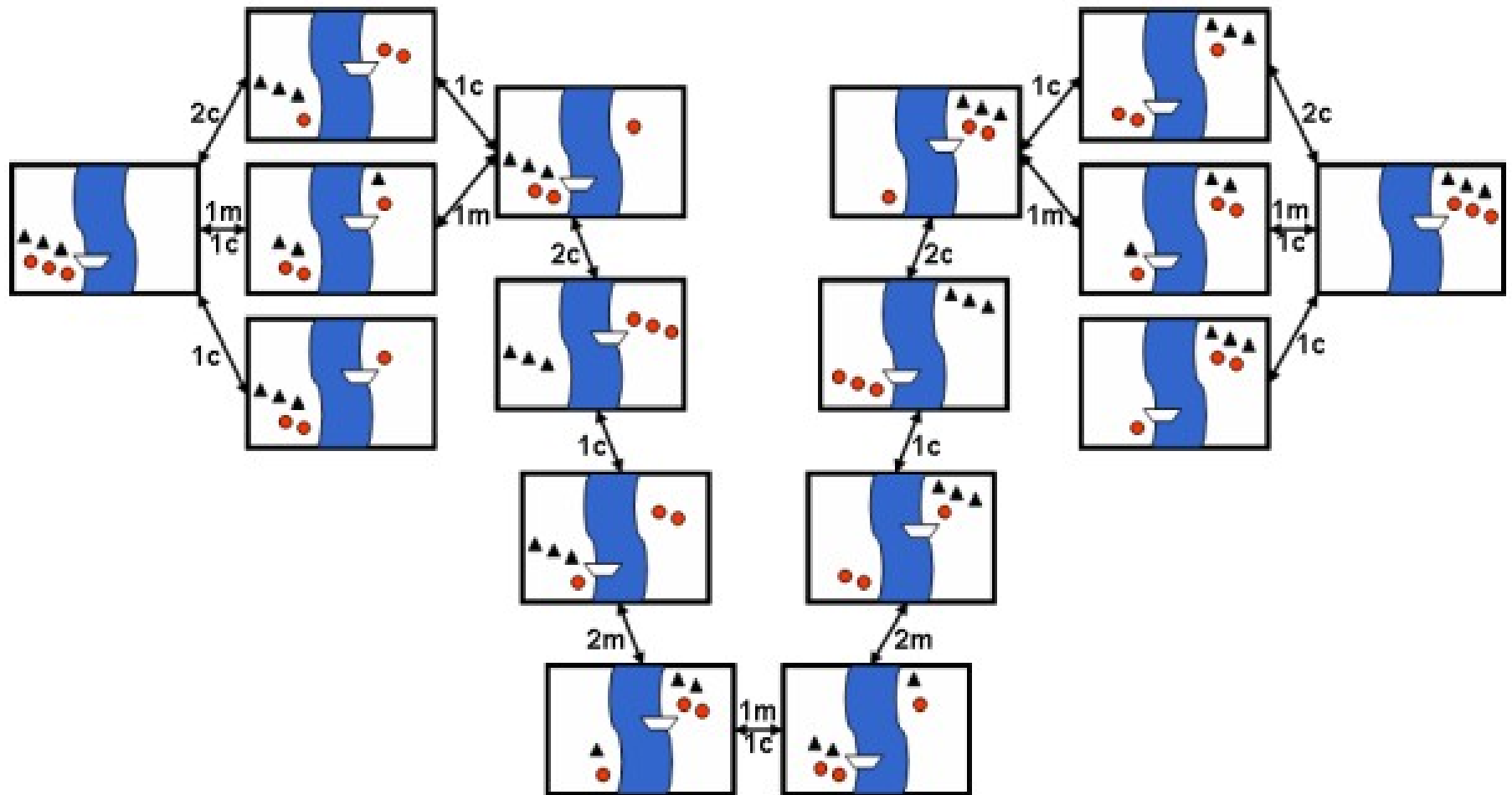
Exercícios



Exercícios

- Canibais e Missionários:
 - **Espaço de Estados:** Todas as possíveis configurações válidas 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



Exemplo: Romênia

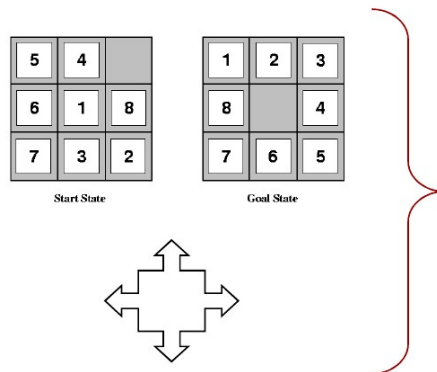
- De férias na Romênia; atualmente em Arad.
- Vôo sai amanhã de Bucareste.
- Formular objetivo:
 - Estar em Bucareste
- Buscar Objetivo:
 - Formular problema:
 - estados: cidades
 - ações: dirigir entre as cidades
 - Encontrar solução:
 - sequência de cidades, ex., Arad, Sibiu, Fagaras, Bucareste.

Como Encontrar a Solução?

- Uma vez o problema bem formulado, o estado final (objetivo) deve ser “**buscado**” no espaço de estados.

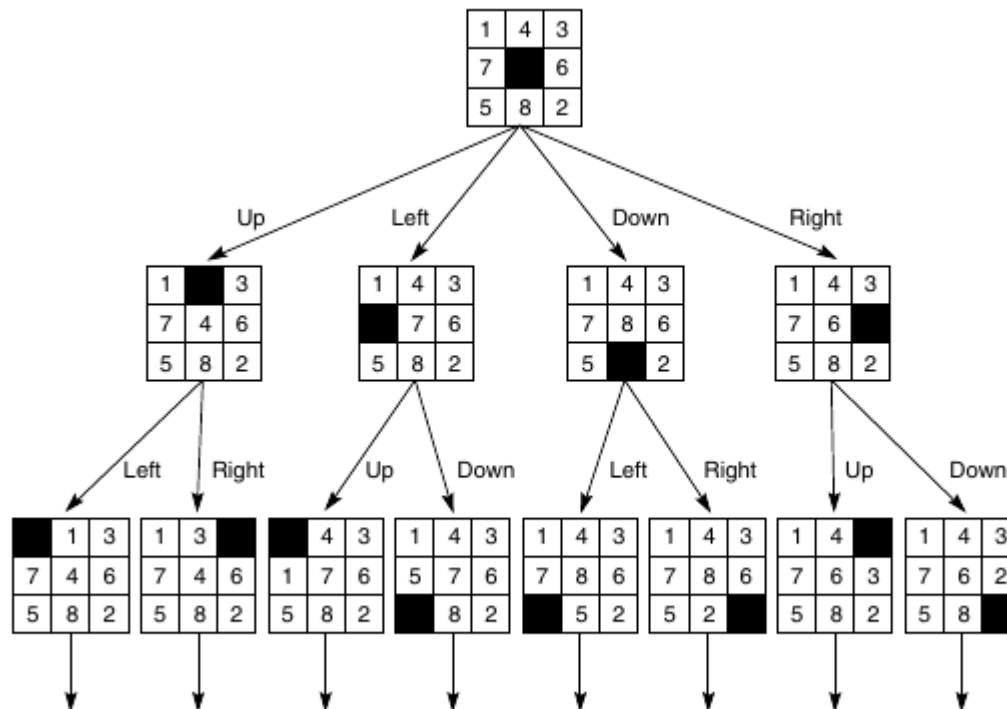
Como Encontrar a Solução?

- Uma vez o problema bem formulado, o estado final (objetivo) deve ser “**buscado**” no espaço de estados.
- Exemplo: 8-Puzzle



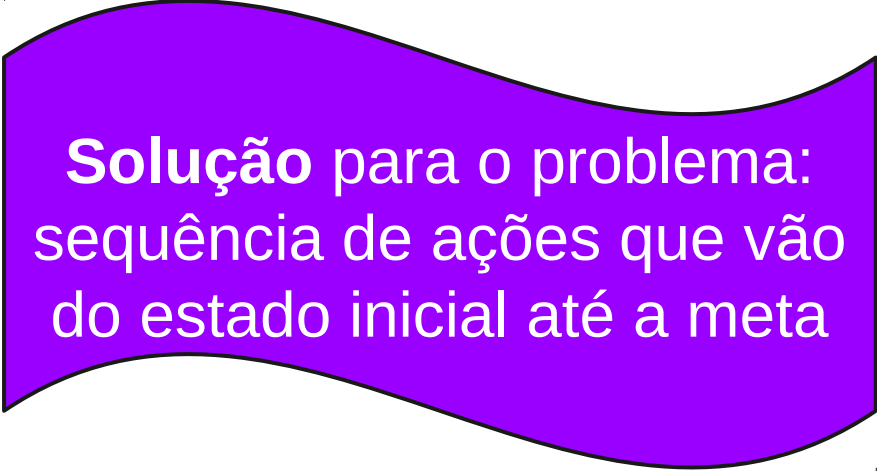
Como Encontrar a Solução?

- Uma vez o problema bem formulado, o estado final (objetivo) deve ser “**buscado**” no espaço de estados.
- Exemplo: 8-Puzzle



Como Encontrar a Solução?

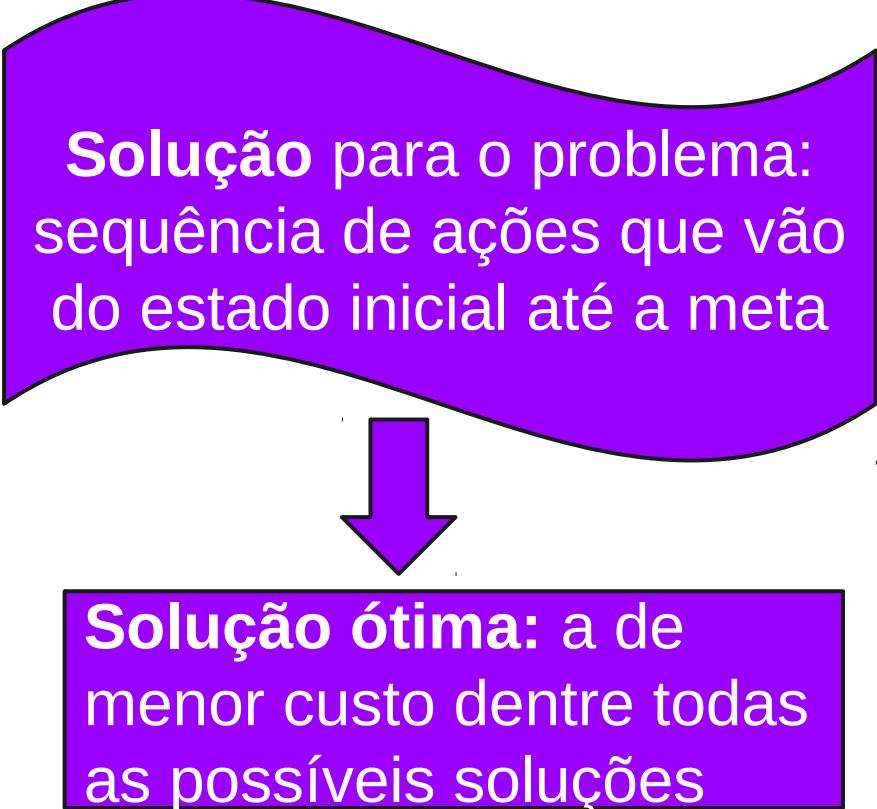
- Uma vez o problema bem formulado, o estado final (objetivo) deve ser “**buscado**” no espaço de estados.



Solução para o problema:
sequência de ações que vão
do estado inicial até a meta

Como Encontrar a Solução?

- Uma vez o problema bem formulado, o estado final (objetivo) deve ser “**buscado**” no espaço de estados.



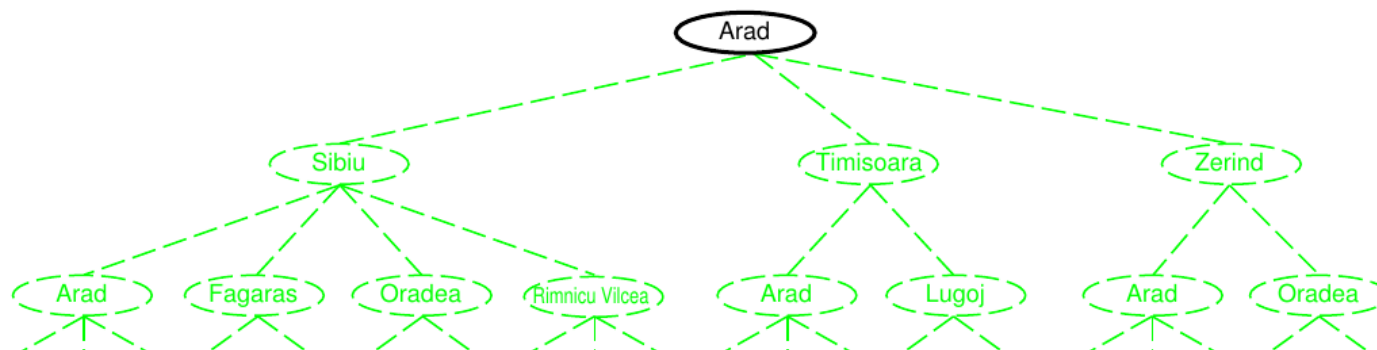
Solução para o problema:
sequência de ações que vão
do estado inicial até a meta

A diagrama ilustra a busca por uma solução ótima. No topo, um retângulo amarelo com bordas onduladas contém a definição de uma solução. Uma seta verde apontando para baixo conecta este retângulo a um retângulo verde sólido na base, que define a solução ótima. O retângulo verde na base também possui uma borda inferior ondulada.

Solução ótima: a de
menor custo dentre todas
as possíveis soluções

Como Encontrar a Solução?

- Uma vez o problema bem formulado, o estado final (objetivo) deve ser “**buscado**” no espaço de estados.
- A busca é representada em uma **árvore de busca**:
 - Nós correspondem a estados
 - Raiz: corresponde ao estado inicial;

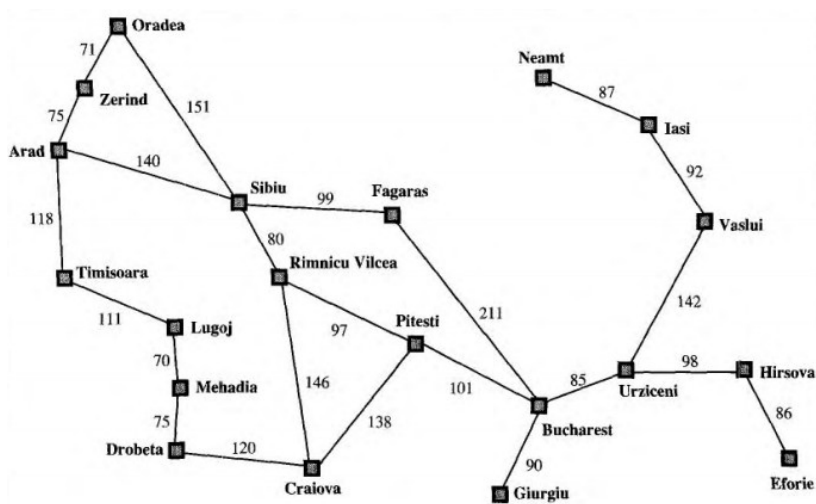


Como Encontrar a Solução?

- Uma vez 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

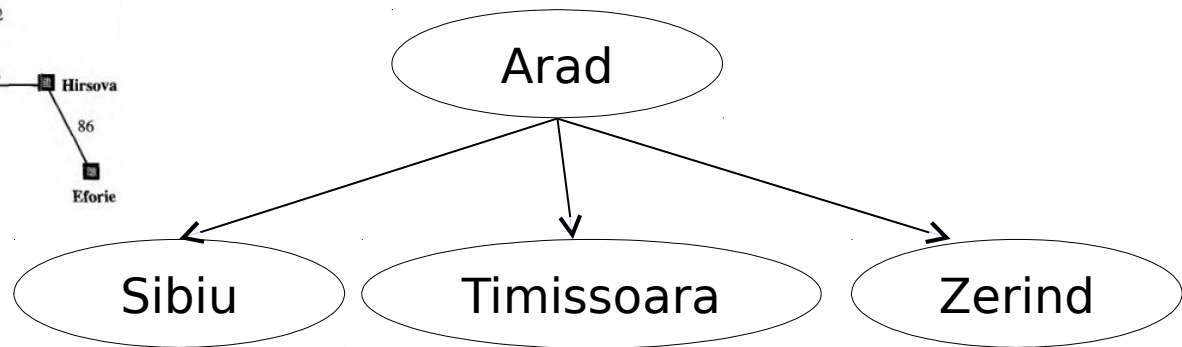
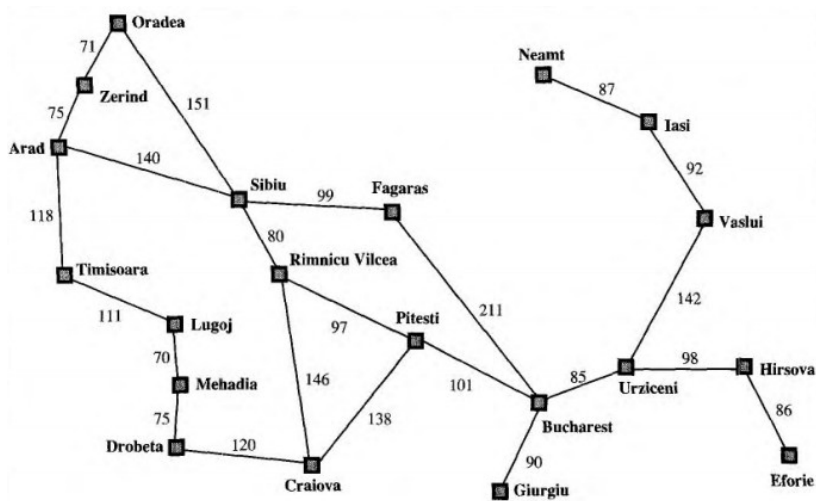
- **Exemplo:** Ir de **Arad** para **Bucharest**



Arad

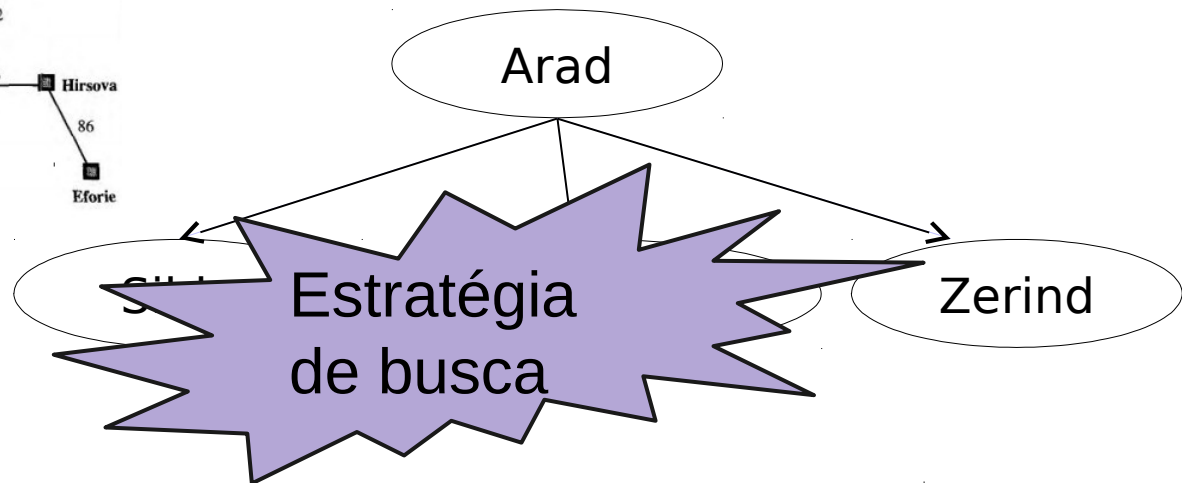
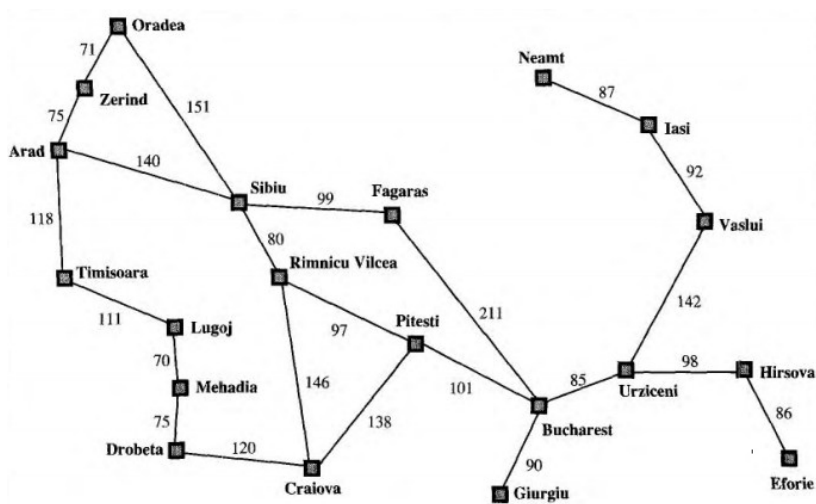
Buscando Soluções

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



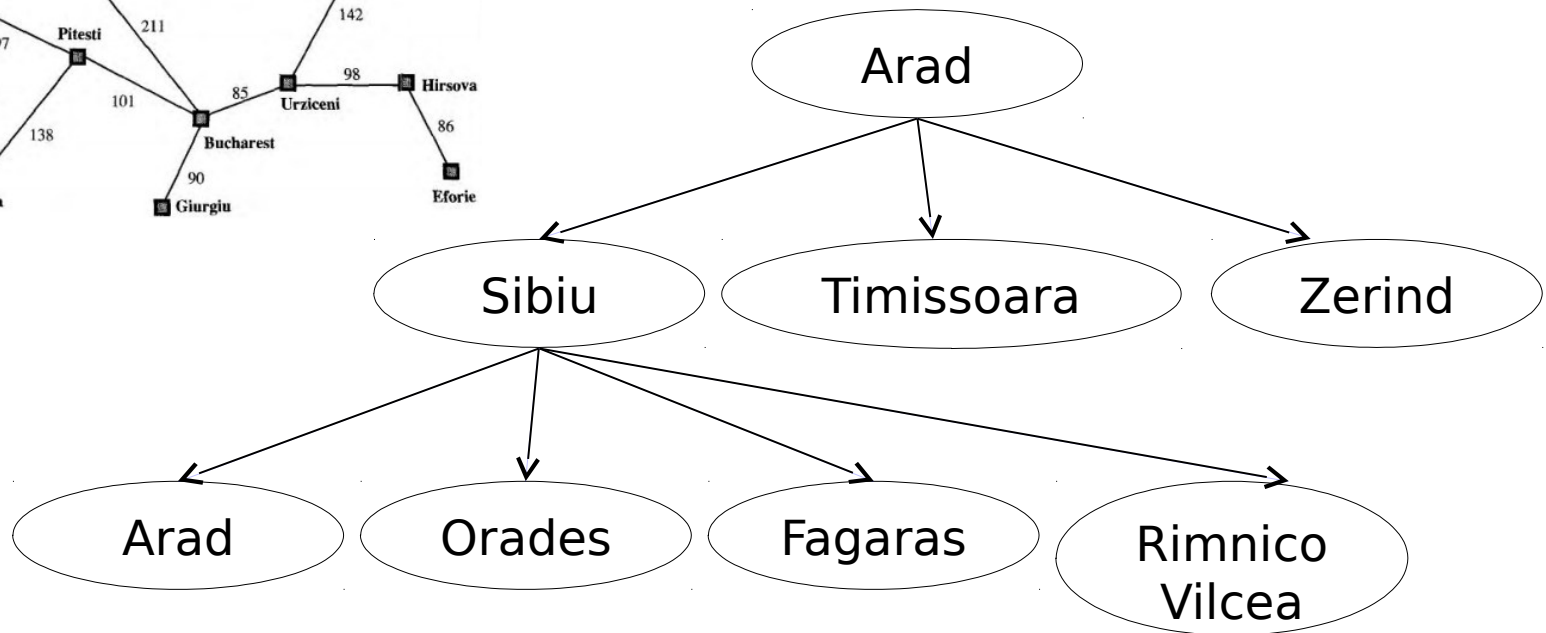
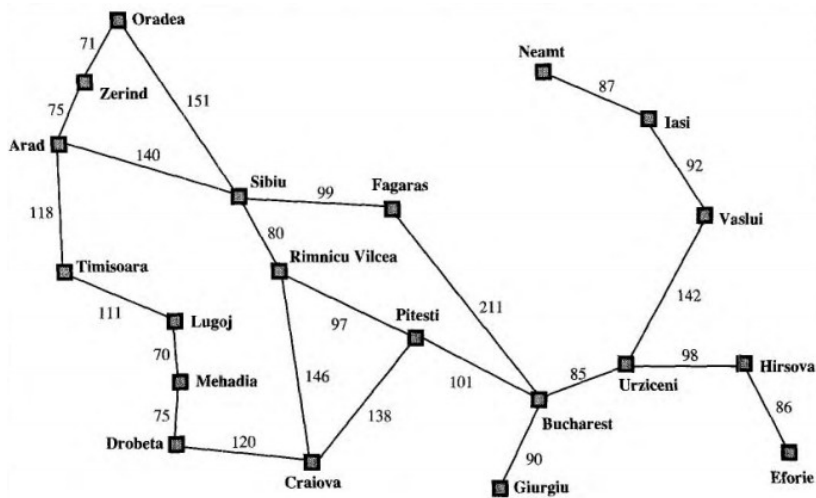
Buscando Soluções

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



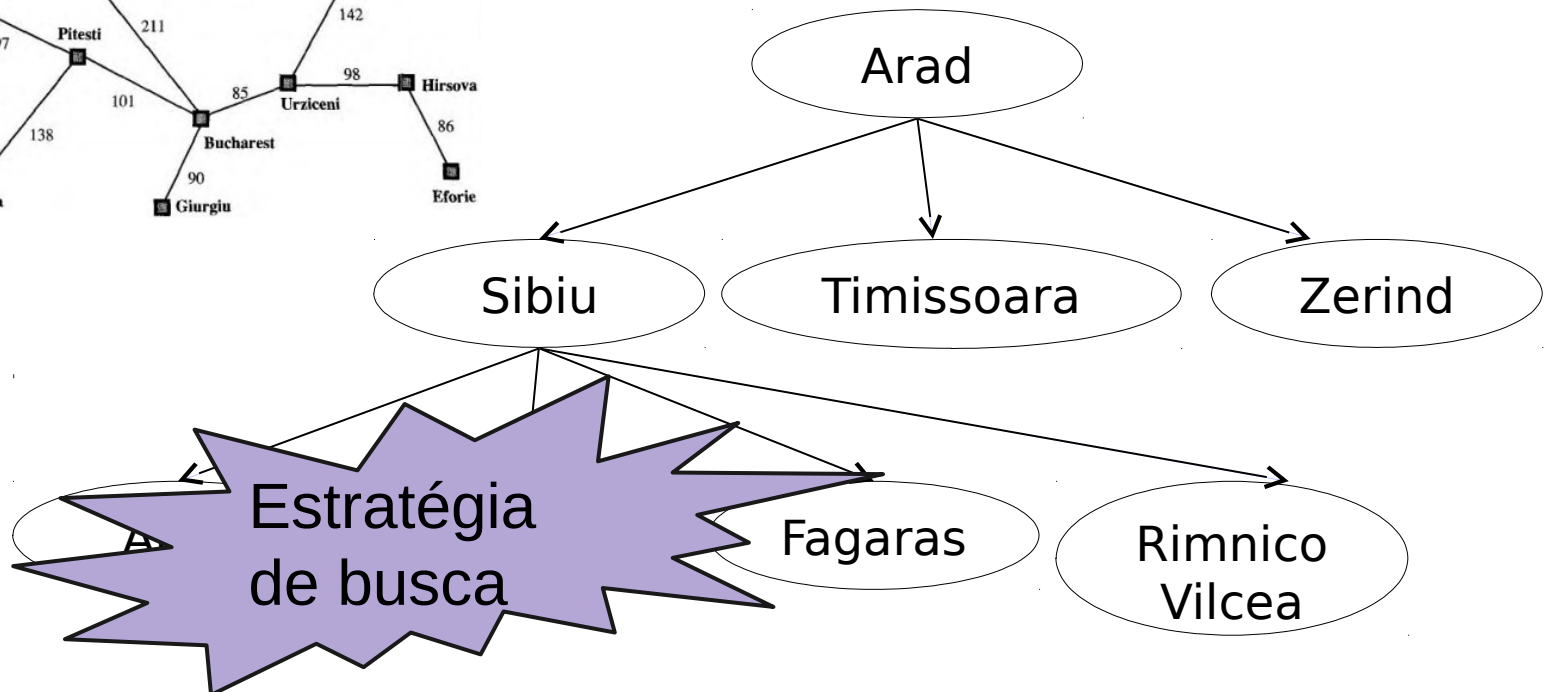
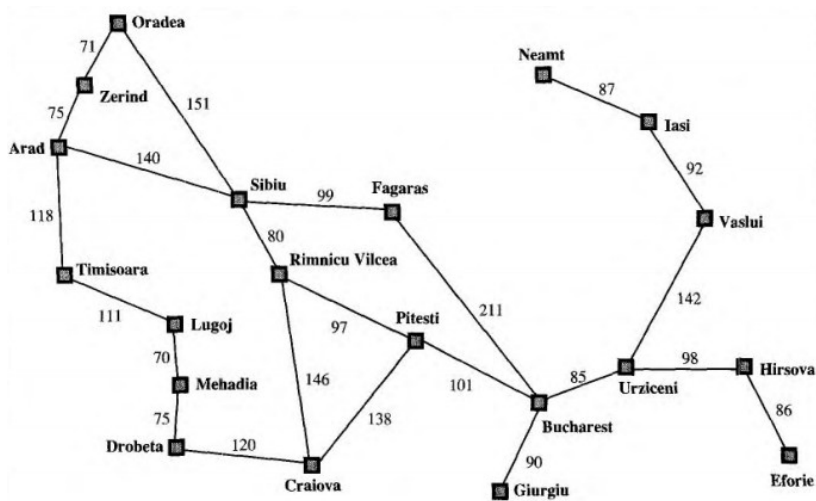
Buscando Soluções

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



Buscando Soluções

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

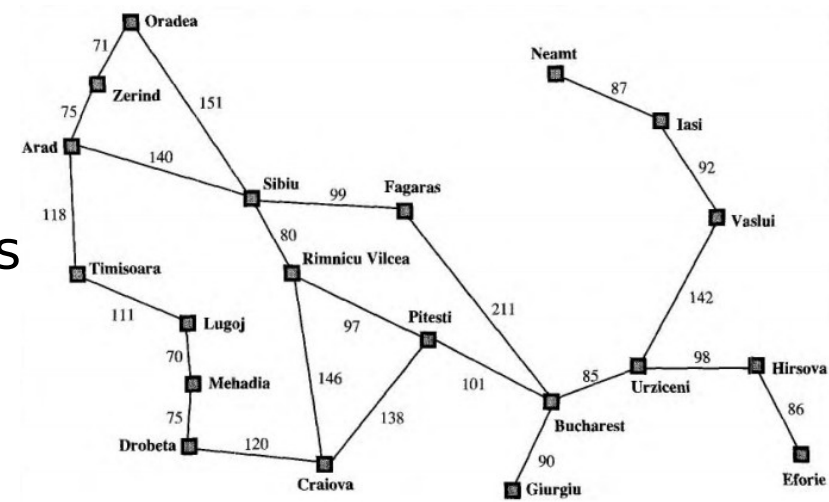


Buscando Soluções

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

- **Exemplo:**

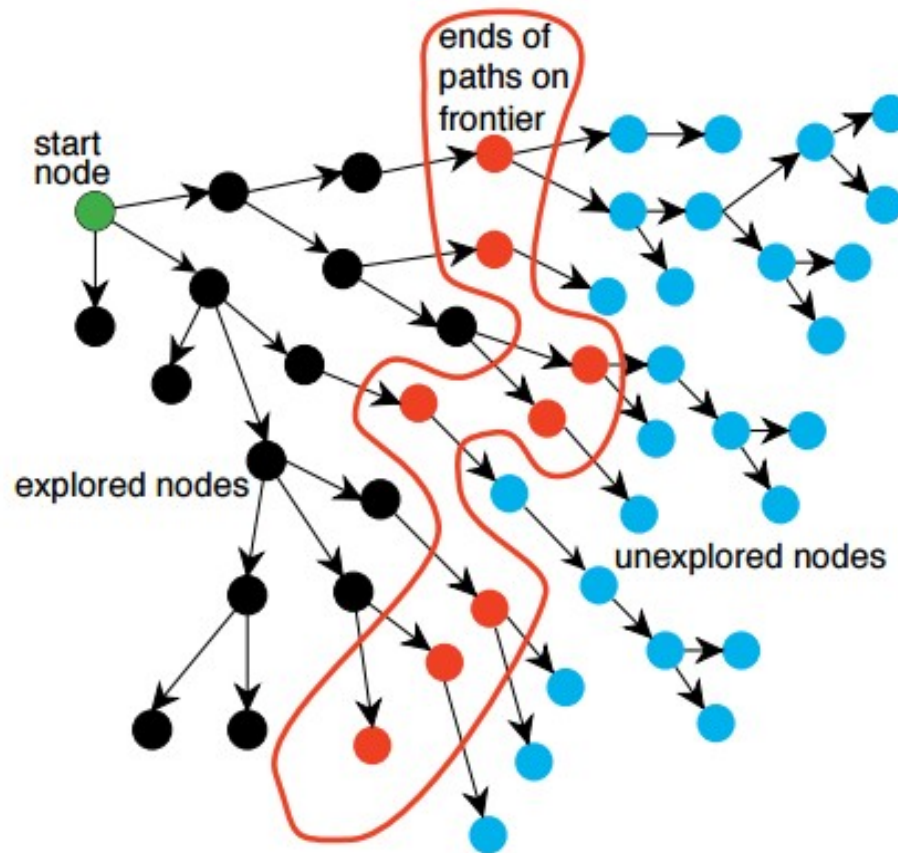
- 20 estados no espaço de estados
- Número de caminhos infinito;
- Árvore com infinitos nós;



Código Descritivo – Busca em Árvore

```
Função BuscaEmArvore(Problema, Estratégia) retorna solução ou falha
Início
  Inicializa a árvore 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 árvore de busca
Fim
```

Código Descritivo – Busca em Árvore



Código Descritivo – 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(ExpandFronteira(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:**

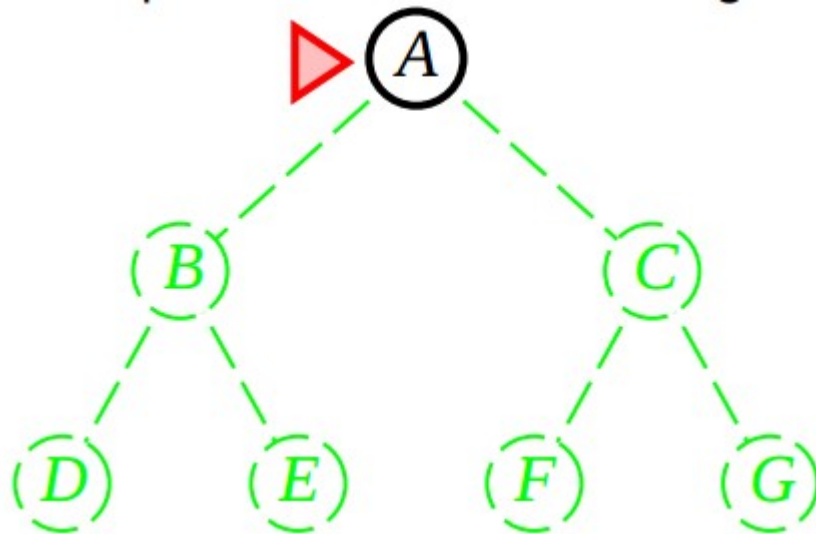
- Operam em um único estado e movem-se para a vizinhança deste estado.

Busca Cega

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

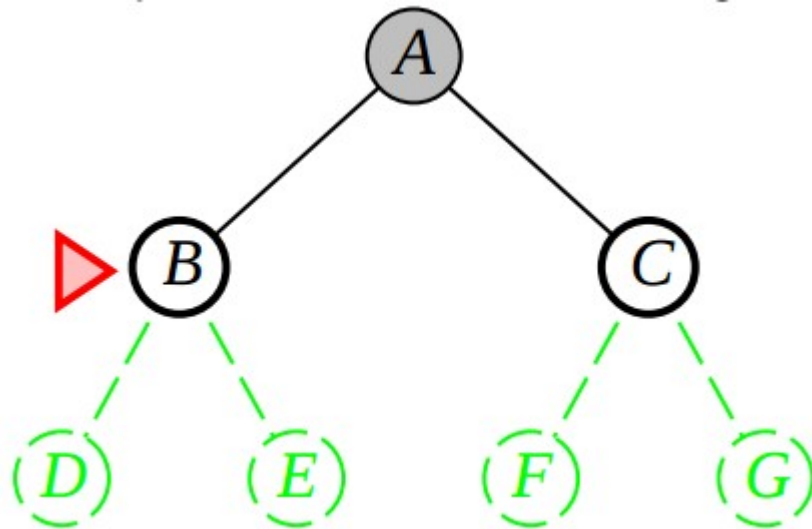
Busca em Largura

- 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 das fronteiras.



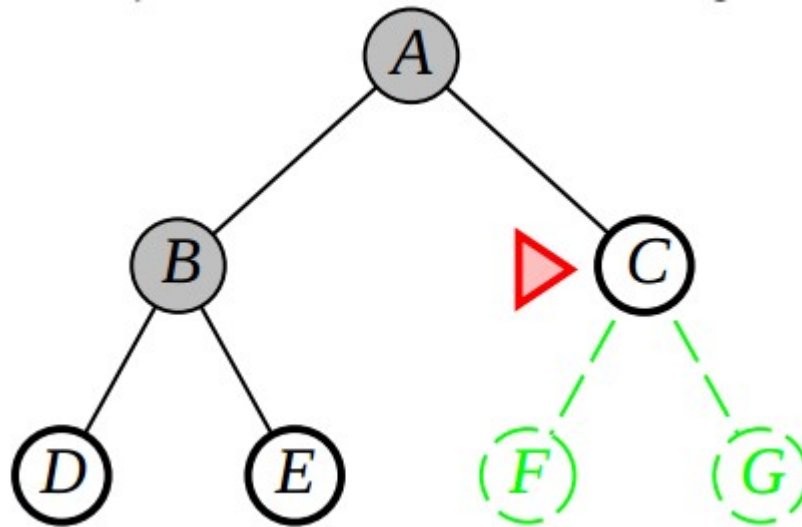
Busca em Largura

- 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 das fronteiras.



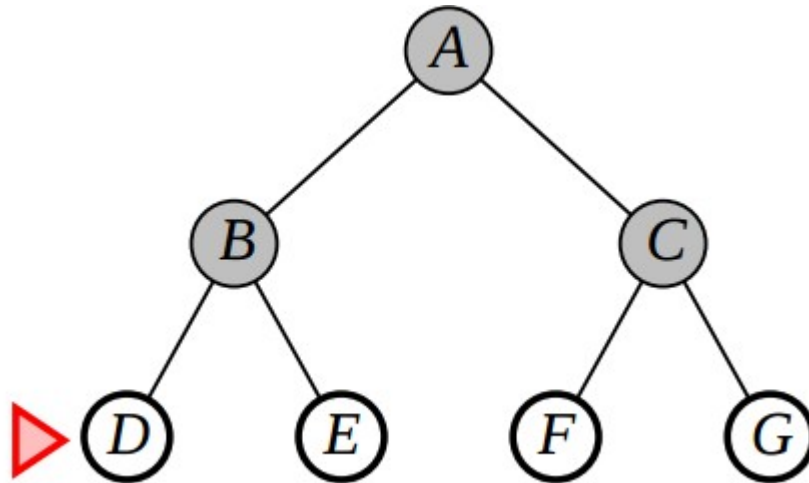
Busca em Largura

- 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 das fronteiras.



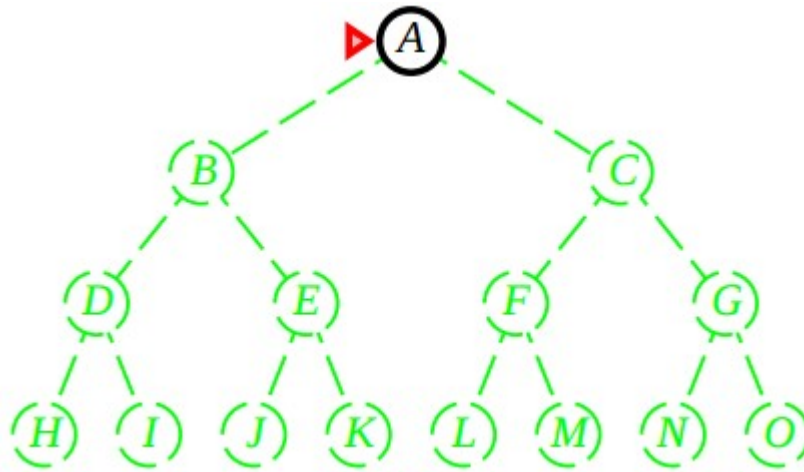
Busca em Largura

- 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 das fronteiras.



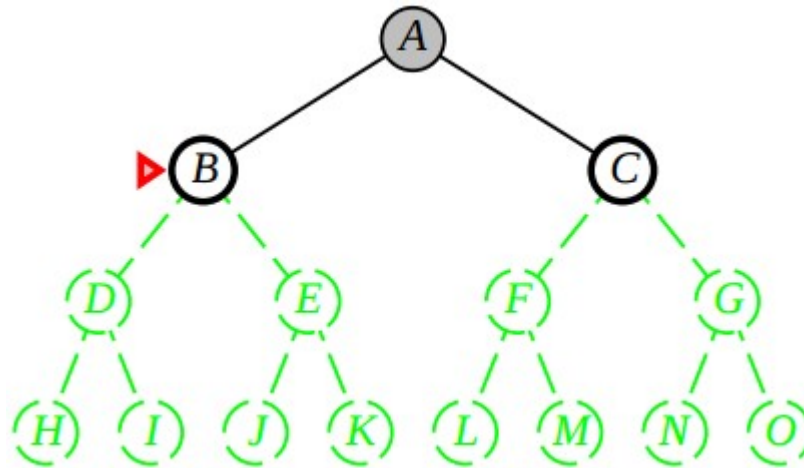
Busca em Profundidade

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



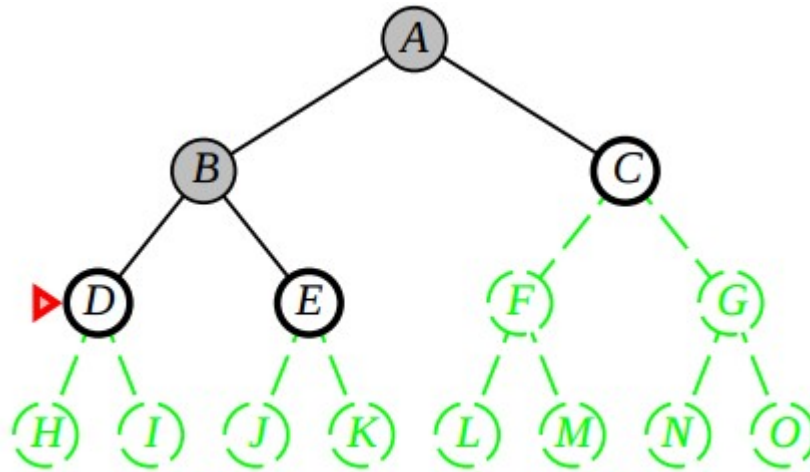
Busca em Profundidade

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



Busca em Profundidade

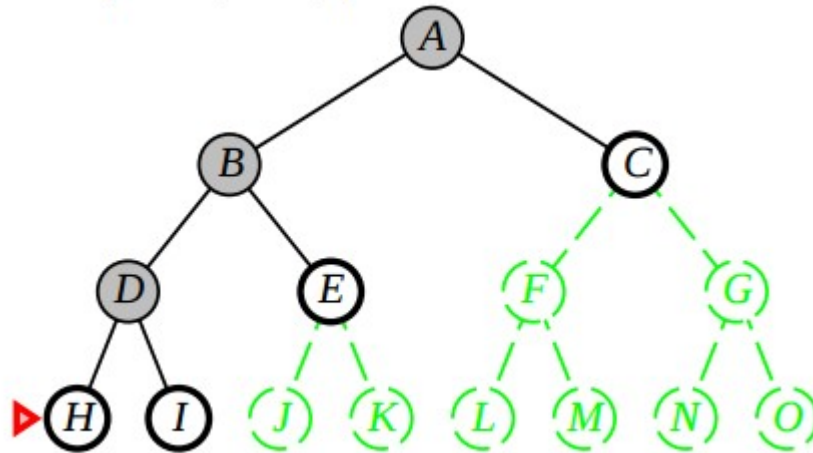
- **Estratégia:**
 - Expande os nós da vizinhança até o nó mais profundo.



Busca em Profundidade

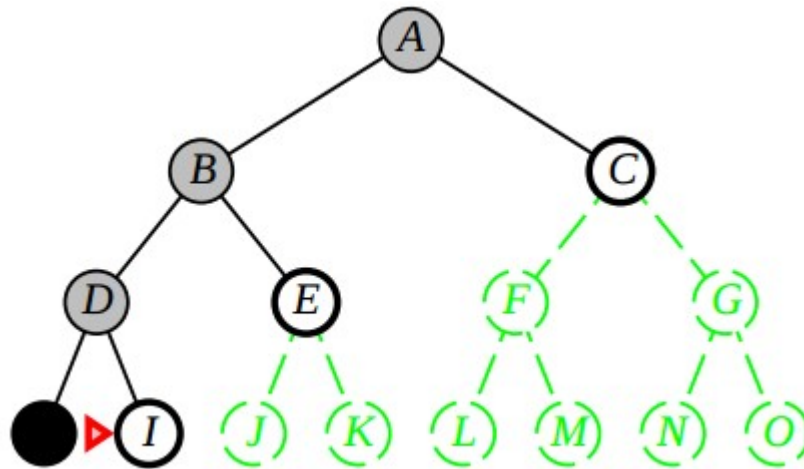
- **Estratégia:**

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



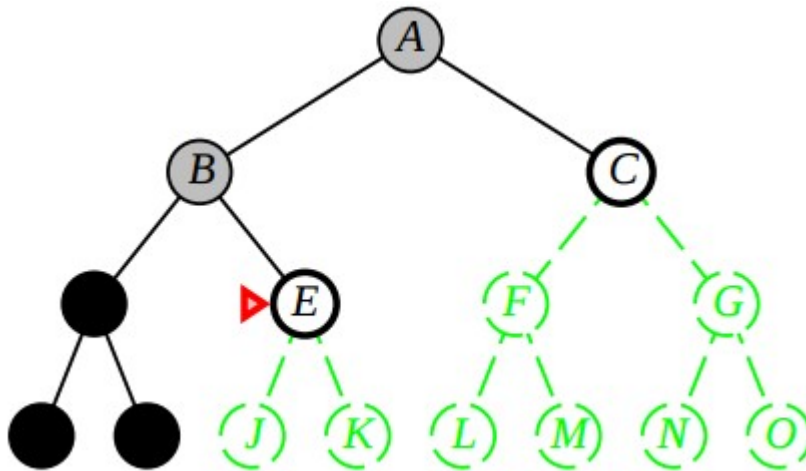
Busca em Profundidade

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



Busca em Profundidade

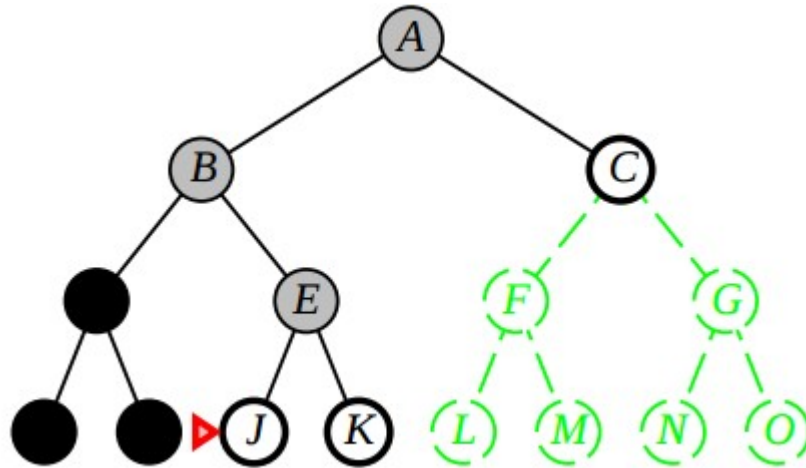
- **Estratégia:**
 - Expande os nós da vizinhança até o nó mais profundo.



Busca em Profundidade

- **Estratégia:**

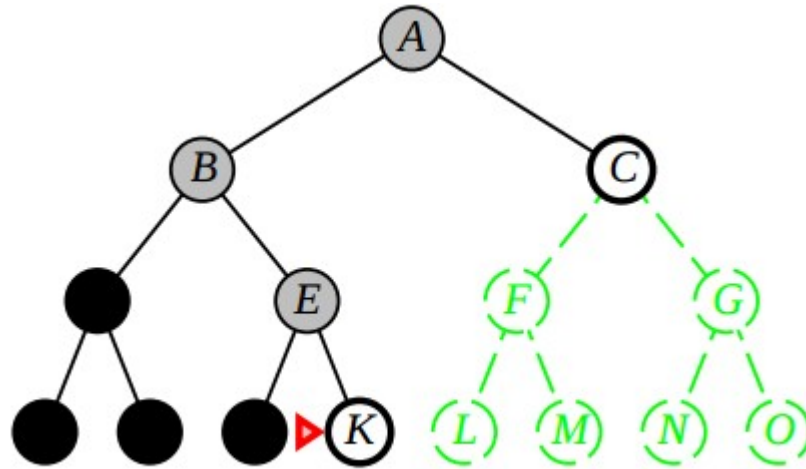
- Expande os nós da vizinhança até o nó mais profundo.



Busca em Profundidade

- **Estratégia:**

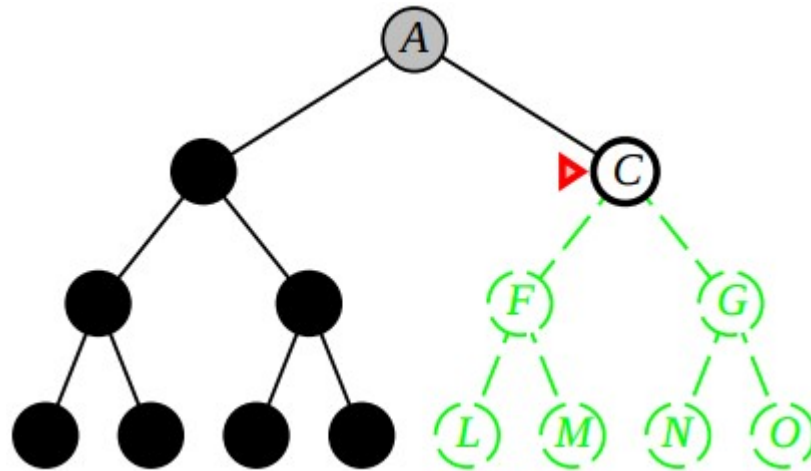
- Expande os nós da vizinhança até o nó mais profundo.



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 ser armazenado. Caminhos que já foram explorados podem ser descartados da memória.

Busca em Profundidade

- **Problema:** O algoritmo pode fazer uma busca muito longa mesmo quando a resposta do problema esta localizado a poucos nós da raiz da árvore.

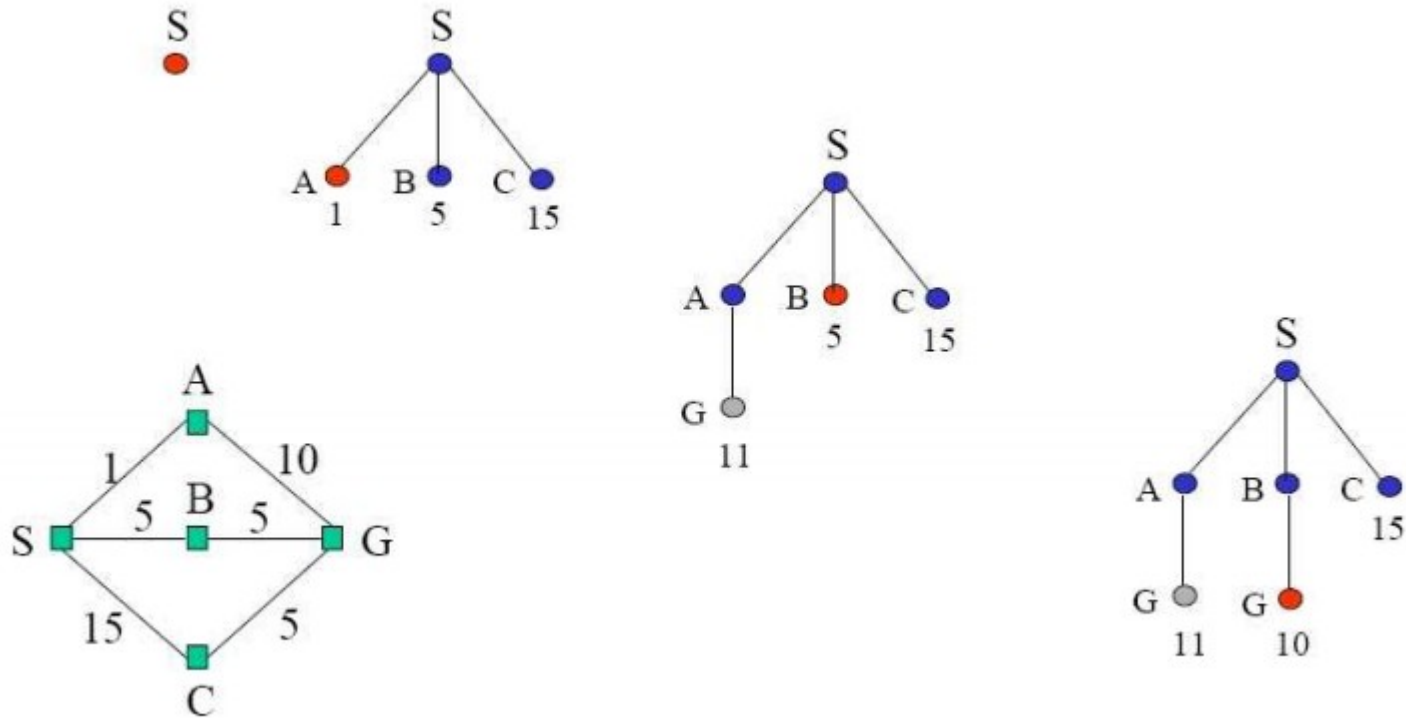
Busca de Custo Uniforme

- **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

- considere o problema de encontrar o menor caminho entre S e G num grafo.



Busca de Custo Uniforme

- Implementação semelhante a busca em largura. Adiciona-se uma **condição de seleção** dos nós a serem expandidos.

Busca com Aprofundamento Iterativo

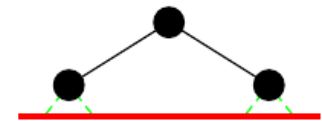
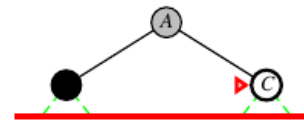
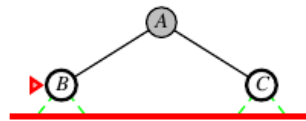
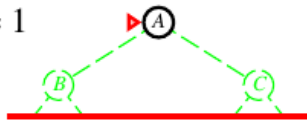
- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



Busca com Aprofundamento Iterativo

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

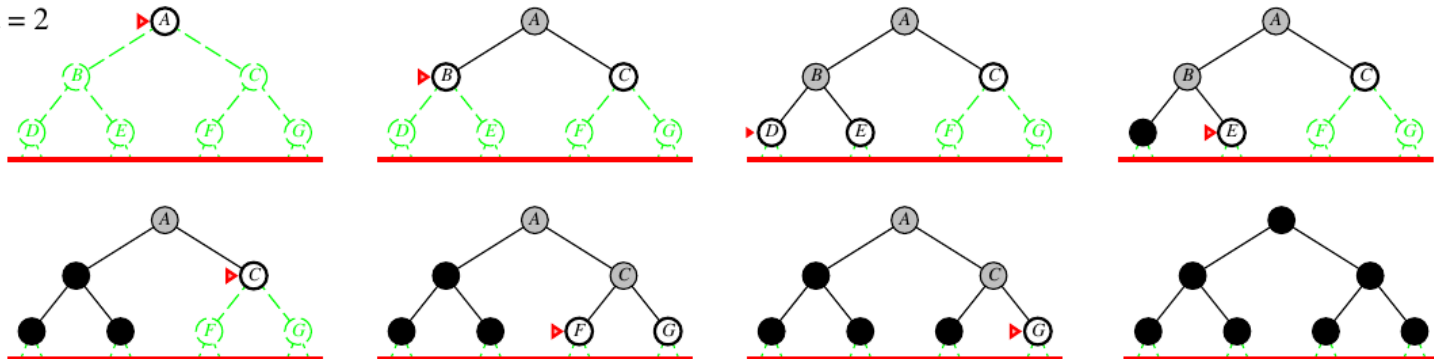
Limit = 1



Busca com Aprofundamento Iterativo

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

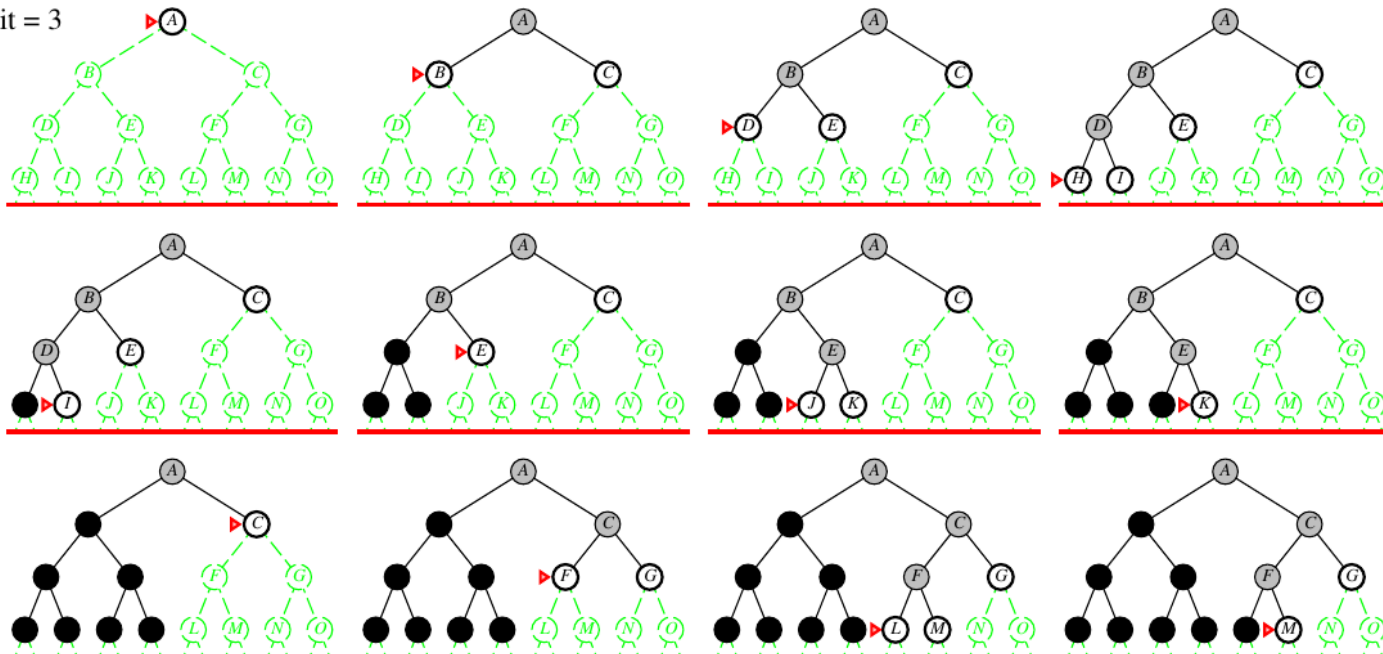
Limit = 2



Busca com Aprofundamento Iterativo

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

Limit = 3



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.

Leitura Complementar

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

