

# Resolução de problemas

Inteligência artificial

Prof. Allan Rodrigo Leite

# Problema de busca

- Um problema de busca é composto por três elementos
  - Um espaço de estados denotado pelos conjuntos  $\mathcal{S}$  e  $\mathcal{A}$
  - Um estado inicial denotado por um estado particular  $s_0 \in \mathcal{S}$
  - Um conjunto de estados alvo (objetivo) denotado por um conjunto  $\mathcal{G} \subseteq \mathcal{S}$
- Tipos de objetivos
  - Objetivo explícito: estados alvos são conhecidos já no início da busca
    - Exemplo: ir de uma cidade a outra em um mapa
  - Objetivo implícito: estados alvos são desconhecidos
    - Exemplo: xeque-mate em um jogo de xadrez

# Problema de busca

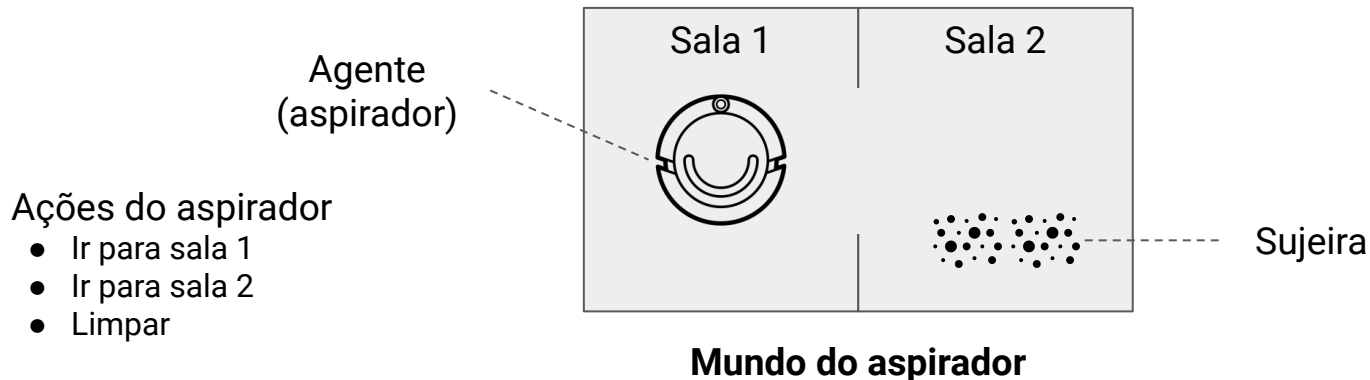
- Tipos de problema
  - Determinista (totalmente observável)
    - É conhecido o estado está e em qual irá estar após uma ação
  - Conhecimento parcial dos estados e ações
    - Não são observáveis, isto é, lida com incertezas e crenças
  - Não determinista e/ou não totalmente observável
    - Resultados das ações podem ser incertos
    - Dispõe-se de novas informações ao longo do procedimento de solução
    - Solução usualmente intercala entre busca e execução (planejamento)
    - Espaço de estados desconhecido e são explorados durante a busca

# Problema de busca

- Encontrar uma solução para um problema de busca consiste em
  - Identificar uma sequência de ações que leva do estado inicial a um dos estados alvo no espaço de estados do problema
- Características de uma solução
  - Sequência de ações que levam uma solução é conhecida por caminho
  - Custo do caminho é uma forma de mensurar a qualidade da solução
  - Uma solução é ótima ao apresentar o menor custo de caminho

# Problema de busca

- Espaço de estados
  - Representação de todos os estados alcançáveis a partir de um estado inicial após quaisquer sequências de ações
  - Formalmente definido por
    - Conjunto  $\mathcal{S}$  de estados
    - Conjunto  $\mathcal{A}$  de ações que mapeiam (relacionam) um estado a outro



# Problema de busca

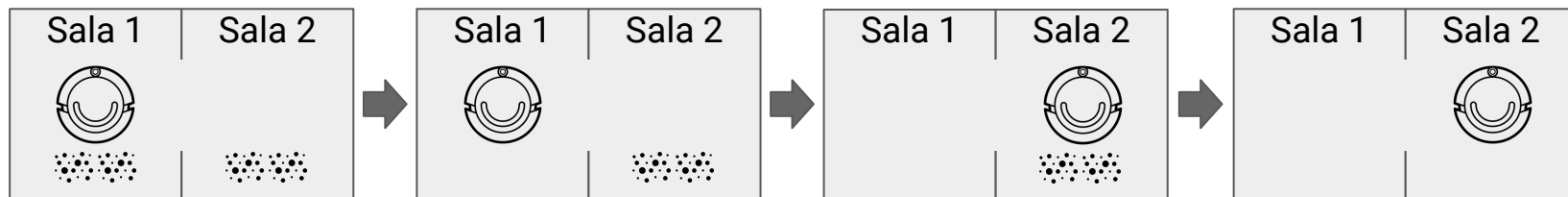
- Representação de estados
  - Estados são representados por estruturas, onde cada componente denota um atributo do estado representado
- Exemplo de representação do mundo do aspirador
  - Estados podem ser representados pela tupla  $[X, Y, Z]$ 
    - $X \in \{\text{sala1}, \text{sala2}\}$ : indica a localização do aspirador
    - $Y \in \{0, 1\}$ : indica a situação da sala1 (0 está limpa e 1 está suja)
    - $Z \in \{0, 1\}$ : indica a situação da sala2 (0 está limpa e 1 está suja)
  - Espaço de estados (todos os estados possíveis)
    - $[\text{sala1}, 0, 0], [\text{sala1}, 0, 1], [\text{sala1}, 1, 0], [\text{sala1}, 1, 1],$   
 $[\text{sala2}, 0, 0], [\text{sala2}, 0, 1], [\text{sala2}, 1, 0], [\text{sala2}, 1, 1]$

# Problema de busca

- Representação de ações
  - Representadas por operadores da forma  $\text{oper}(\alpha, s, s') \leftarrow \beta$ 
    - $\alpha$ : ação que transforma o estado  $s$  em estado  $s'$
    - $\beta$ : condição a ser satisfeita para realizar a ação
- Exemplo de representação do mundo do aspirador
  - $\text{oper}(\text{aspirar}, [\text{sala1}, Y, Z], [\text{sala1}, 0, Z]) \leftarrow Y = 1$
  - $\text{oper}(\text{aspirar}, [\text{sala2}, Y, Z], [\text{sala2}, Y, 0]) \leftarrow Z = 1$
  - $\text{oper}(\text{entrarSala1}, [X, Y, Z], [\text{sala1}, Y, Z]) \leftarrow X = \text{sala2}$
  - $\text{oper}(\text{entrarSala2}, [X, Y, Z], [\text{sala2}, Y, Z]) \leftarrow X = \text{sala1}$

# Problema de busca

- Exemplo do problema de busca para o mundo do aspirador
  - Estado inicial:  $[sala1, 1, 1]$
  - Estados alvo:  $\{[sala1, 0, 0], [sala2, 0, 0]\}$
  - Caminho para uma possível solução
    - $oper(aspirar, [sala1, 1, 1], [sala1, 0, 1])$
    - $oper(entrarSala2, [sala1, 0, 1], [sala2, 0, 1])$
    - $oper(aspirar, [sala2, 0, 1], [sala2, 0, 0])$





# Algoritmos de busca

- Objetivo dos algoritmos de busca
  - Motor de inferência sobre o espaço de estados
  - Visa encontrar um ou mais estados alvo a partir do estado inicial
- Propriedades de um algoritmo de busca
  - Completo
    - Garante que a busca encontrará uma solução para o problema (se existir)
  - Ótimo
    - Garante que a solução é encontrada é a melhor possível

# Algoritmos de busca

- Propriedades de um algoritmo de busca (cont.)
  - Complexidade temporal
    - Quantidade espaços em memória requeridos pelo processo de busca
  - Complexidade espacial
    - Quantidade de estados visitados até encontrar uma solução
  - Fator de ramificação
    - Número máximo de estados sucessores a partir de um estado
  - Profundidade
    - Profundidade máxima do espaço de estados (pode ser infinita)

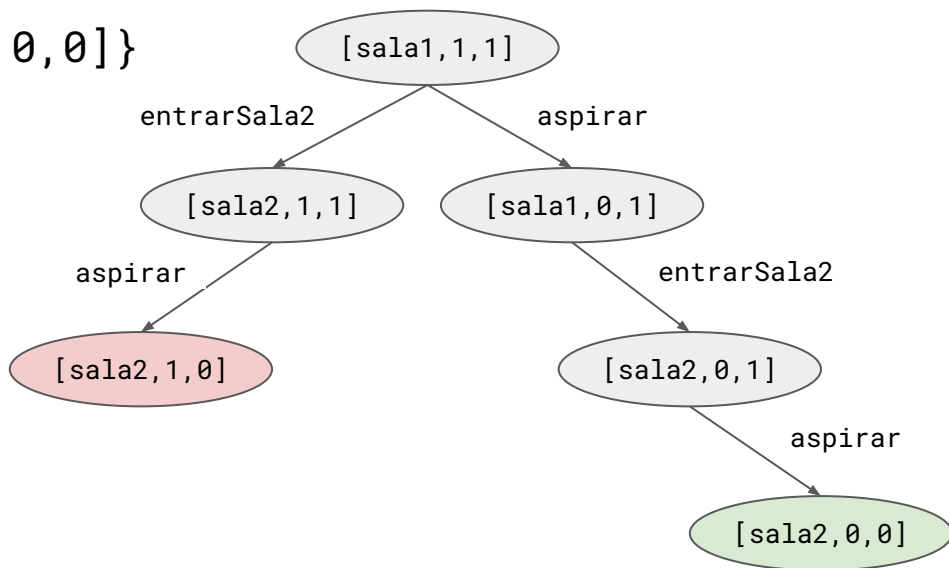
# Algoritmos de busca

- Processo de exploração do espaço de estados
  - Em geral não é especificado explicitamente o conjunto de estados  $\mathcal{S}$
  - Os estados subsequentes são gerados sob demanda ao visitar novos nós
- Algoritmo de busca não determinístico

```
Busca( $\mathcal{A}$ ,  $s_0$ ,  $\mathcal{G}$ )  
   $\Sigma \leftarrow \{s_0\}$  //lista com os estados a serem visitados  
  enquanto  $\Sigma \neq \emptyset$  faça //enquanto  $\Sigma$  não estiver vazia  
     $s \leftarrow \text{remove}(\Sigma)$  //retira o primeiro estado da lista  
    se  $s \in \mathcal{G}$  então //verifica se o estado atual é um dos estados alvo  
      retorna caminho( $s_0, s$ ) //retorna caminho realizado entre  $s_0$  e  $s$   
    fim se  
     $\Sigma \leftarrow \Sigma \cup \text{sucessores}(s, \mathcal{A})$  //retorna os próximos estados a partir de  $\mathcal{A}$   
  fim enquanto  
  retorna erro
```

# Algoritmos de busca

- Algoritmo de busca não determinístico (cont.)
  - A execução deste algoritmo produzirá uma árvore de busca
  - Exemplo para o mundo do aspirador
    - $s_0$ : [sala1, 1, 1]
    - $\mathcal{G}$ : {[sala1, 0, 0], [sala2, 0, 0]}
    - Resultado (caminho até s)
      - aspirar
      - entrarSala2
      - aspirar



# Algoritmos de busca

- Algoritmo de busca não determinístico (cont.)
  - Poderá executar infinitamente se o espaço de estados gerar ciclos
  - Para evitar isto, é necessário manter uma lista de estados já visitados

**Busca**( $\mathcal{A}$ ,  $s_0$ ,  $\mathcal{G}$ )

$\Gamma \leftarrow \emptyset$  //lista dos estados já visitados

$\Sigma \leftarrow \{s_0\}$  //lista com os estados a serem visitados

**enquanto**  $\Sigma \neq \emptyset$  **faça** //enquanto  $\Sigma$  não estiver vazia

$s \leftarrow \text{remove}(\Sigma)$  //retira o primeiro estado da lista

**se**  $s \in \mathcal{G}$  **então** //verifica se o estado atual é um dos estados alvo

**retorna** caminho( $s_0, s$ ) //retorna caminho realizado entre  $s_0$  e  $s$

**fim se**

$\Gamma \leftarrow \Gamma \cup \{s\}$  //adiciona estado atual na lista de estados visitados

$\Sigma \leftarrow \Sigma \cup (\text{sucessores}(s, \mathcal{A}) - \Gamma)$  //desconsidera estados já visitados

**fim enquanto**

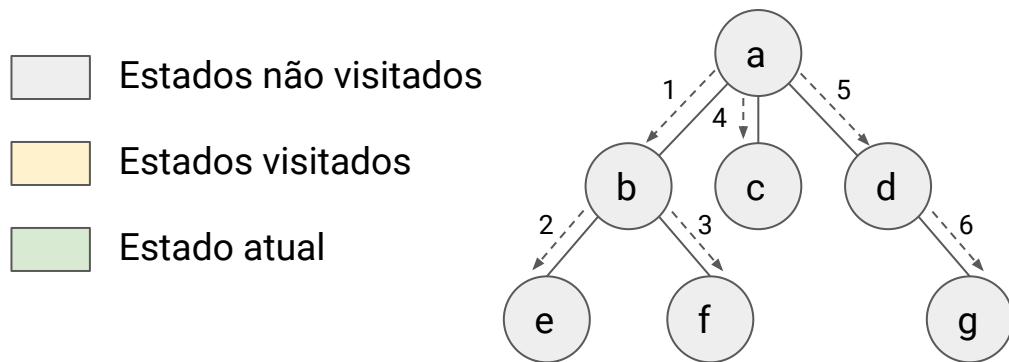
**retorna** erro

# Estratégia de busca cega

- Também conhecida como busca não informada
  - Usa somente informações providas pelo problema
  - Regras para um comportamento sistêmico em um algoritmo de busca
    - Porém, as regras não levam em conta a qualidade da solução encontrada
    - A busca simplesmente explora o espaço de estados
    - Não há mecanismos para guiar a busca para regiões promissoras do espaço
- Existem duas estratégias de busca cega utilizadas na prática
  - Busca em profundidade (*depth-first search*)
  - Busca em largura (*breadth-first search*)
  - Busca em profundidade limitada
  - Busca em largura iterativa

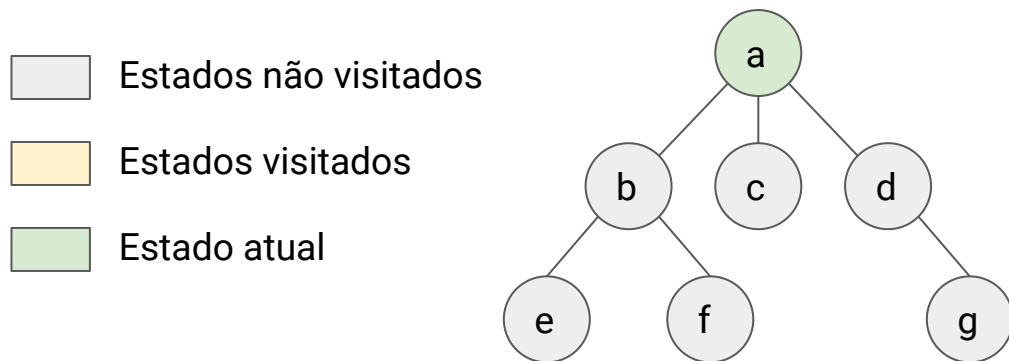
# Busca em profundidade

- Estratégia de busca
  - Os estados são visitados intercalando os níveis da árvore
  - O resultado da busca a partir da raiz forma múltiplos caminhos
    - A busca ocorre visitando os nós filhos da esquerda para a direita
    - O processo ocorre iterativamente a cada subárvore explorada



# Busca em profundidade

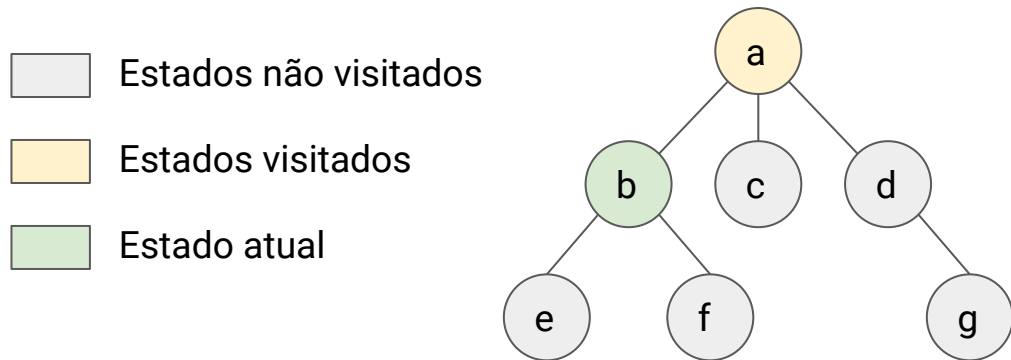
- Estratégia de busca
  - Os estados são visitados intercalando os níveis da árvore
  - O resultado da busca a partir da raiz forma múltiplos caminhos
    - A busca ocorre visitando os nós filhos da esquerda para a direita
    - O processo ocorre iterativamente a cada subárvore explorada





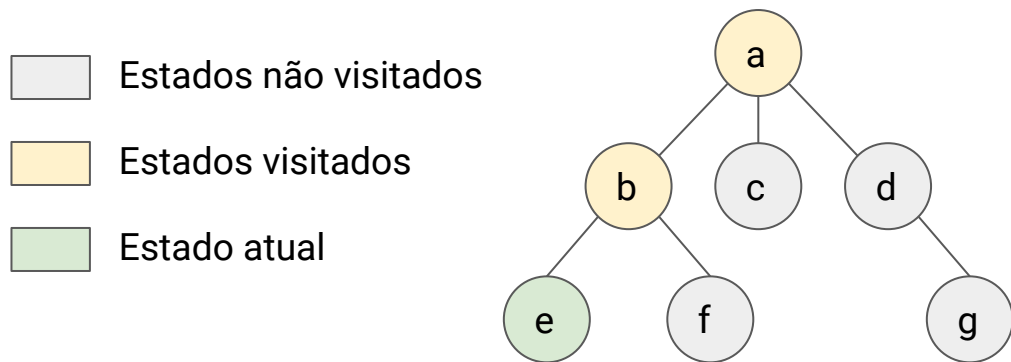
# Busca em profundidade

- Estratégia de busca
  - Os estados são visitados intercalando os níveis da árvore
  - O resultado da busca a partir da raiz forma múltiplos caminhos
    - A busca ocorre visitando os nós filhos da esquerda para a direita
    - O processo ocorre iterativamente a cada subárvore explorada



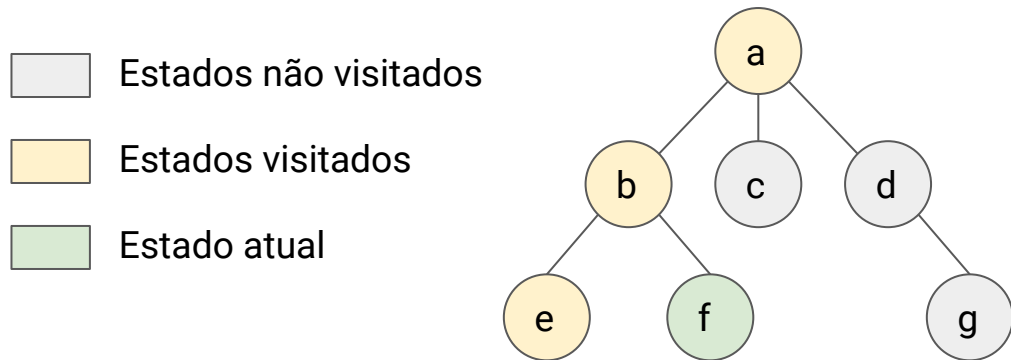
# Busca em profundidade

- Estratégia de busca
  - Os estados são visitados intercalando os níveis da árvore
  - O resultado da busca a partir da raiz forma múltiplos caminhos
    - A busca ocorre visitando os nós filhos da esquerda para a direita
    - O processo ocorre iterativamente a cada subárvore explorada



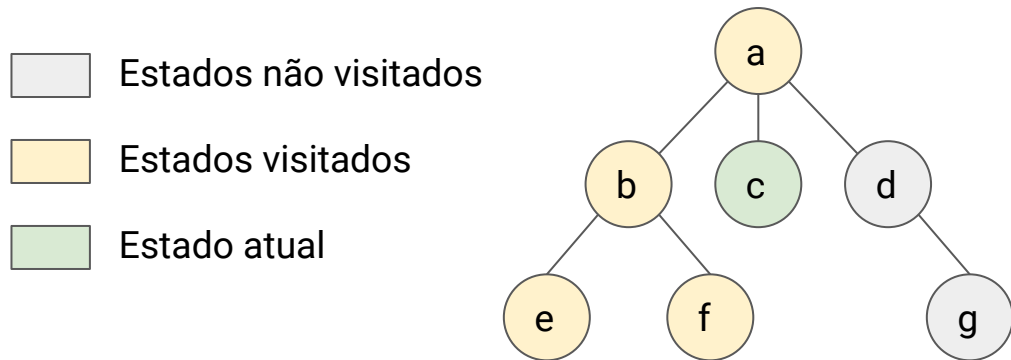
# Busca em profundidade

- Estratégia de busca
  - Os estados são visitados intercalando os níveis da árvore
  - O resultado da busca a partir da raiz forma múltiplos caminhos
    - A busca ocorre visitando os nós filhos da esquerda para a direita
    - O processo ocorre iterativamente a cada subárvore explorada



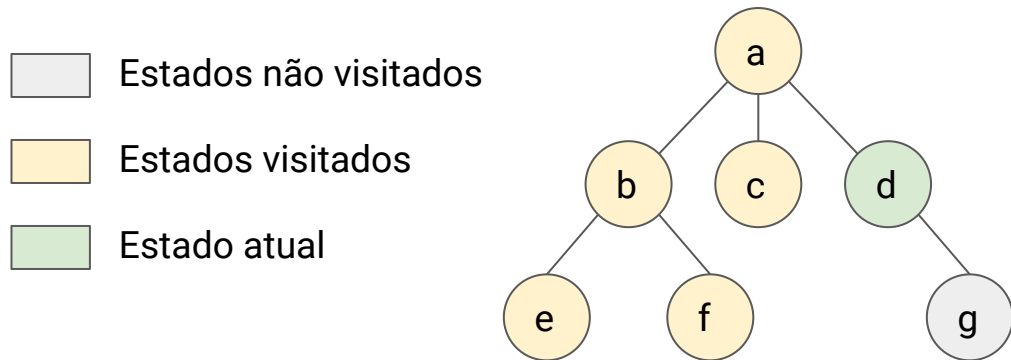
# Busca em profundidade

- Estratégia de busca
  - Os estados são visitados intercalando os níveis da árvore
  - O resultado da busca a partir da raiz forma múltiplos caminhos
    - A busca ocorre visitando os nós filhos da esquerda para a direita
    - O processo ocorre iterativamente a cada subárvore explorada



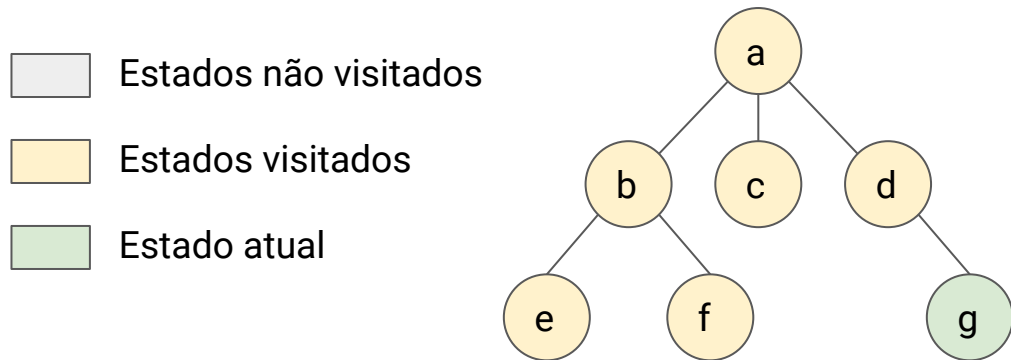
# Busca em profundidade

- Estratégia de busca
  - Os estados são visitados intercalando os níveis da árvore
  - O resultado da busca a partir da raiz forma múltiplos caminhos
    - A busca ocorre visitando os nós filhos da esquerda para a direita
    - O processo ocorre iterativamente a cada subárvore explorada



# Busca em profundidade

- Estratégia de busca
  - Os estados são visitados intercalando os níveis da árvore
  - O resultado da busca a partir da raiz forma múltiplos caminhos
    - A busca ocorre visitando os nós filhos da esquerda para a direita
    - O processo ocorre iterativamente a cada subárvore explorada



# Busca em profundidade

- Algoritmo de busca

**BuscaEmProfundidade**( $\mathcal{A}$ ,  $s_0$ ,  $\mathcal{G}$ )

$\Gamma \leftarrow \emptyset$  //lista com os estados já visitados

$\Sigma \leftarrow \{s_0\}$  //lista LIFO (pilha) com os estados a serem visitados

**enquanto**  $\Sigma \neq \emptyset$  **faça** //enquanto  $\Sigma$  não estiver vazia

$s \leftarrow \text{remove}(\Sigma)$  //retira o estado no topo da lista

**se**  $s \in \mathcal{G}$  **então** //verifica se o estado atual for um dos estados alvo

**retorna** caminho( $s_0, s$ ) //retorna caminho realizado entre  $s_0$  e  $s$

**fim se**

$\Gamma \leftarrow \Gamma \cup \{s\}$  //adiciona estado atual na lista de estados visitados

    adiciona(sucessores( $s, \mathcal{A}$ ) -  $\Gamma$ ,  $\Sigma$ ) //adiciona sucessores no topo de  $\Sigma$

**fim enquanto**

**retorna** erro

# Busca em profundidade

- Algoritmo recursivo de busca

```
BuscaEmProfundidade( $\mathcal{A}$ ,  $s_0$ ,  $\mathcal{G}$ )  
  retorna BuscaEmProfundidadeRecursiva( $\mathcal{A}$ ,  $s_0$ ,  $\{s_0\}$ ,  $\mathcal{G}$ )
```

```
BuscaEmProfundidadeRecursiva( $\mathcal{A}$ ,  $s$ ,  $\Gamma$ ,  $\mathcal{G}$ )  
  se  $s \in \mathcal{G}$  então //verifica se o estado atual for um dos estados alvo  
    retorna caminho( $s_0, s$ ) //retorna caminho realizado entre  $s_0$  e  $s$   
  fim se  
   $\Sigma \leftarrow \emptyset$  //lista com os estados a serem visitados  
  adiciona(sucessores( $s, \mathcal{A}$ ) -  $\Gamma$ ,  $\Sigma$ ) //adiciona sucessores não visitados  
  enquanto  $\Sigma \neq \emptyset$  faça  
     $s' \leftarrow \text{remove}(\Sigma)$  //retira o primeiro estado da lista  
    retorna BuscaEmProfundidadeRecursiva( $\mathcal{A}$ ,  $s'$ ,  $\Gamma \cup \{s\}$ ,  $\mathcal{G}$ )  
  fim enquanto  
  retorna erro
```

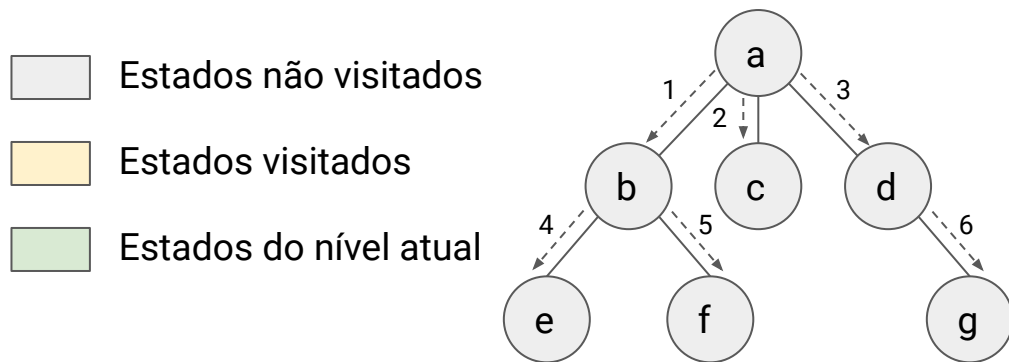


# Busca em profundidade

- Características
  - Não é completo
    - Caminhos infinitos pode impedir a exploração em outras ramificações
  - Não é ótimo
    - Quando encontra uma solução, não é garantido que seja ótima

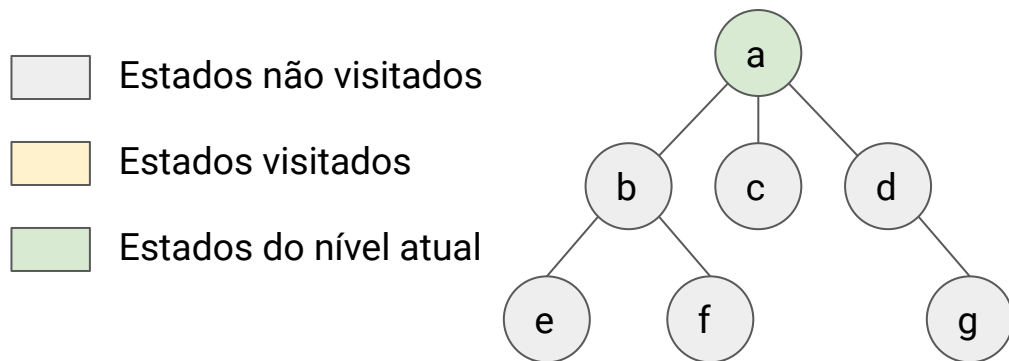
# Busca em largura

- Estratégia de busca
  - Examina todos os estados do mesmo nível antes de prosseguir adiante
    - Estados são visitados no mesmo nível da árvore
    - Em seguida é realizada a mesma busca no próximo nível da árvore



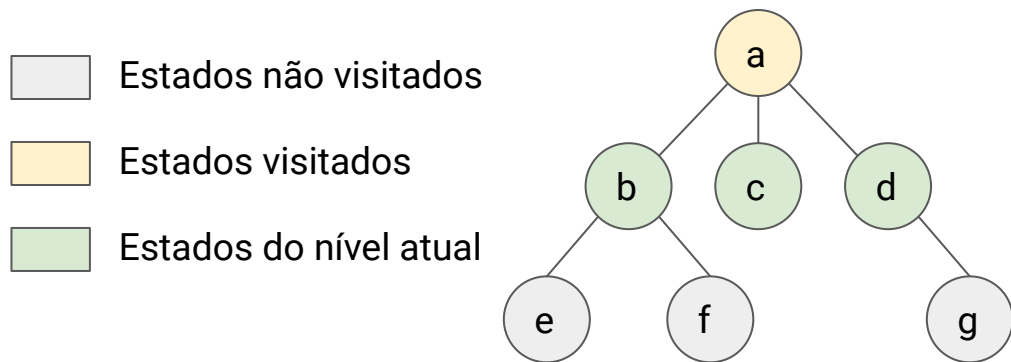
# Busca em largura

- Estratégia de busca
  - Examina todos os estados do mesmo nível antes de prosseguir adiante
    - Estados são visitados no mesmo nível da árvore
    - Em seguida é realizada a mesma busca no próximo nível da árvore



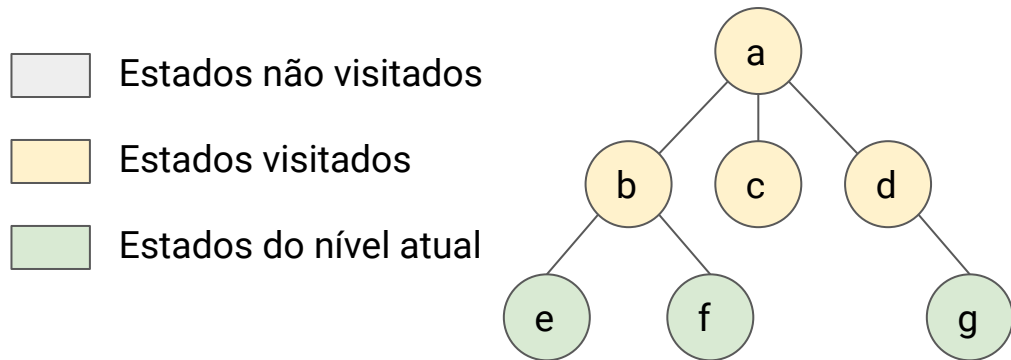
# Busca em largura

- Estratégia de busca
  - Examina todos os estados do mesmo nível antes de prosseguir adiante
    - Estados são visitados no mesmo nível da árvore
    - Em seguida é realizada a mesma busca no próximo nível da árvore



# Busca em largura

- Estratégia de busca
  - Examina todos os estados do mesmo nível antes de prosseguir adiante
    - Estados são visitados no mesmo nível da árvore
    - Em seguida é realizada a mesma busca no próximo nível da árvore



# Busca em largura

- Algoritmo de busca

**BuscaEmLargura**( $\mathcal{A}$ ,  $s_0$ ,  $\mathcal{G}$ )

$\Gamma \leftarrow \emptyset$  //lista FIFO (fila) com os estados já visitados

$\Sigma \leftarrow \{s_0\}$  //lista com os estados a serem visitados

**enquanto**  $\Sigma \neq \emptyset$  **faça** //enquanto  $\Sigma$  não estiver vazia

$s \leftarrow \text{remove}(\Sigma)$  //retira o primeiro estado da lista

**se**  $s \in \mathcal{G}$  **então** //verifica se o estado atual for um dos estados alvo

**retorna** caminho( $s_0, s$ ) //retorna caminho realizado entre  $s_0$  e  $s$

**fim se**

$\Gamma \leftarrow \Gamma \cup \{s\}$  //adiciona estado atual na lista de estados visitados

    adiciona(sucessores( $s, \mathcal{A}$ ) -  $\Gamma$ ,  $\Sigma$ ) //adiciona sucessores no fim de  $\Sigma$

**fim enquanto**

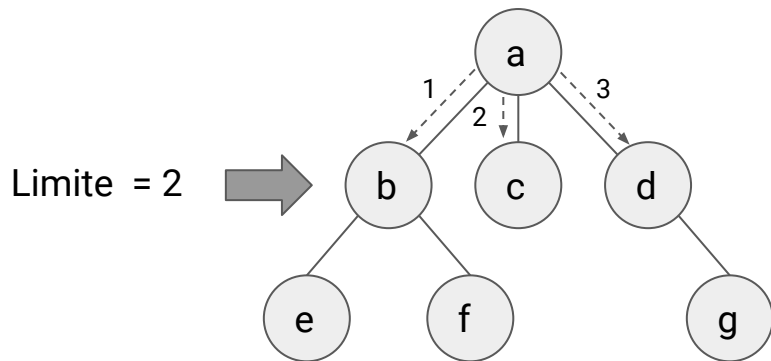
**retorna** erro

# Busca em largura

- Características
  - É completo
    - Condição é que os estados sucessores de todos os estados sejam finitos
  - Não é ótimo
    - Só é ótimo se os custos dos caminhos em um dado nível na árvore de busca forem iguais ou menores do que aqueles em níveis anteriores
    - Por exemplo, quando todas as ações possuem o mesmo custo

# Busca em profundidade limitada

- Estratégia de busca
  - Faz uma poda na árvore ao atingir uma dada profundidade no caminho
  - Usa-se esta técnica em problemas com caminhos infinitos
    - Não há garantia de uma busca completa
    - Não há garantia de uma solução ótima
  - Conhecimento do domínio da aplicação ajuda na definição do limite





# Busca em profundidade limitada

- Algoritmo de busca

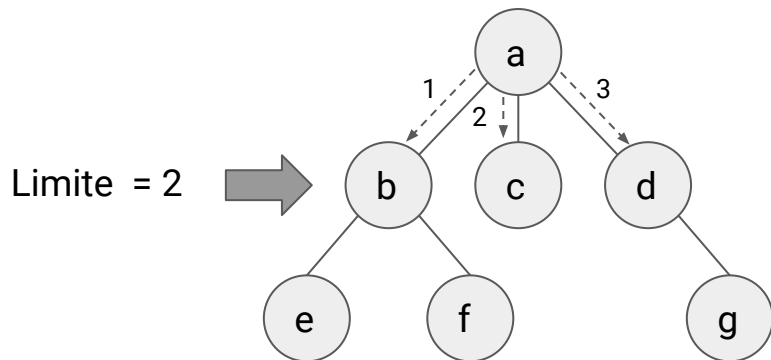
```
BuscaEmProfundidadeLimitada( $\mathcal{A}$ ,  $s_0$ ,  $\mathcal{G}$ ,  $\ell$ )
   $\Gamma \leftarrow \emptyset$  //lista com os estados já visitados
   $\Sigma \leftarrow \{s_0\}$  //lista LIFO (pilha) com os estados a serem visitados
  enquanto  $\Sigma \neq \emptyset$  faça //enquanto  $\Sigma$  não estiver vazia
     $s \leftarrow \text{remove}(\Sigma)$  //retira o estado do topo da lista
    se  $s \in \mathcal{G}$  então //verifica se o estado atual for um dos estados alvo
      retorna caminho( $s_0, s$ ) //retorna caminho realizado entre  $s_0$  e  $s$ 
    fim se
    se profundidade( $s$ ) =  $\ell$  então //se a profundidade limite foi atingida
      retorna erro
    fim se
     $\Gamma \leftarrow \Gamma \cup \{s\}$  //adiciona estado atual na lista de estados visitados
    adiciona(sucessores( $s, \mathcal{A}$ ) -  $\Gamma$ ,  $\Sigma$ ) //adiciona sucessores no topo de  $\Sigma$ 
  fim enquanto
  retorna erro
```

# Busca em profundidade limitada

- Características
  - Não é completo
    - O limite  $\ell$  na profundidade resolve o problema de caminhos infinitos
    - Porém, só é completo quando  $\ell < d$
    - Onde  $d$  que representa o menor caminho para uma solução ótima
  - Não é ótimo
    - Quando encontra uma solução, não é garantido que seja ótima

# Busca em largura iterativa

- Estratégia de busca
  - Faz uma poda na árvore ao atingir uma dada profundidade no caminho
  - Usa-se esta técnica em problemas com caminhos infinitos
    - Não há garantia de uma busca completa
    - Não há garantia de uma solução ótima
  - Conhecimento do domínio da aplicação ajuda na definição do limite



# Busca em largura iterativa

- Algoritmo de busca

**BuscaEmProfundidadeLimitada**( $\mathcal{A}$ ,  $s_0$ ,  $\mathcal{G}$ ,  $\ell$ )

$\Gamma \leftarrow \emptyset$  //lista com os estados já visitados

$\Sigma \leftarrow \{s_0\}$  //lista FIFO (fila) com os estados a serem visitados

**enquanto**  $\Sigma \neq \emptyset$  **faça** //enquanto  $\Sigma$  não estiver vazia

$s \leftarrow \text{remove}(\Sigma)$  //retira o primeiro estado da lista

**se**  $s \in \mathcal{G}$  **então** //verifica se o estado atual for um dos estados alvo

**retorna** caminho( $s_0, s$ ) //retorna caminho realizado entre  $s_0$  e  $s$

**fim se**

**se** profundidade( $s$ ) ==  $\ell$  **então** //se a profundidade limite foi atingida

**retorna** erro

**fim se**

$\Gamma \leftarrow \Gamma \cup \{s\}$  //adiciona estado atual na lista de estados visitados

    adiciona(sucessores( $s, \mathcal{A}$ ) -  $\Gamma$ ,  $\Sigma$ ) //adiciona sucessores no fim de  $\Sigma$

**fim enquanto**

**retorna** erro

# Busca largura iterativa

- Características

- Não é completo

- O limite  $\ell$  na profundidade resolve o problema de caminhos infinitos
    - Porém, só é completo quando  $\ell < d$
    - Onde  $d$  que representa o menor caminho para uma solução ótima

- Não é ótimo

- Só é ótimo se os custos dos caminhos em um dado nível na árvore de busca forem iguais ou menores do que aqueles em níveis anteriores
    - Por exemplo, quando todas as ações possuem o mesmo custo

# Exercícios

1. Implemente um algoritmo de busca para o problema do mundo do aspirador, mas neste caso, prevendo três salas a serem aspiradas. Utilize uma estratégia que evite ciclos durante o processo de busca.
  - Exemplo de representação dos estados tupla  $[X, Y, Z, W]$ 
    - $X \in \{\text{sala1}, \text{sala2}, \text{sala3}\}$ : indica a localização do aspirador
    - $Y \in \{0, 1\}$ : indica a situação da sala1 (0 está limpa e 1 está suja)
    - $Z \in \{0, 1\}$ : indica a situação da sala2 (0 está limpa e 1 está suja)
    - $W \in \{0, 1\}$ : indica a situação da sala3 (0 está limpa e 1 está suja)
  - Exemplo de representação das ações
    - $\text{oper}(\text{aspirar}, [\text{sala1}, Y, Z, W], [\text{sala1}, 0, Z, W]) \leftarrow Y = 1$
    - $\text{oper}(\text{aspirar}, [\text{sala2}, Y, Z, W], [\text{sala2}, Y, 0, W]) \leftarrow Z = 1$
    - $\text{oper}(\text{aspirar}, [\text{sala3}, Y, Z, W], [\text{sala3}, Y, Z, 0]) \leftarrow W = 1$
    - $\text{oper}(\text{entrarSala1}, [X, Y, Z, W], [\text{sala1}, Y, Z, W]) \leftarrow X = \text{sala2}$
    - $\text{oper}(\text{entrarSala1}, [X, Y, Z, W], [\text{sala1}, Y, Z, W]) \leftarrow X = \text{sala3}$
    - $\text{oper}(\text{entrarSala2}, [X, Y, Z, W], [\text{sala2}, Y, Z, W]) \leftarrow X = \text{sala1}$
    - $\text{oper}(\text{entrarSala2}, [X, Y, Z, W], [\text{sala2}, Y, Z, W]) \leftarrow X = \text{sala3}$
    - $\text{oper}(\text{entrarSala3}, [X, Y, Z, W], [\text{sala3}, Y, Z, W]) \leftarrow X = \text{sala1}$
    - $\text{oper}(\text{entrarSala3}, [X, Y, Z, W], [\text{sala3}, Y, Z, W]) \leftarrow X = \text{sala2}$

# Exercícios

2. Implemente um algoritmo de busca para problema dos jarros. O problema envolve por dois jarros com capacidades de 3 e 4 litros, respectivamente. Sabendo-se que podemos encher, esvaziar ou transferir água de um jarro para outro, encontre uma sequência de passos que deixe o jarro de 4 litros com exatamente 2 litros de água. Utilize uma estratégia que evite ciclos durante o processo de busca.

- Representação dos estados tupla  $[X, Y]$ 
  - $X \in \{0, 1, 2, 3\}$ : indica o conteúdo do jarro com capacidade de 3 litros
  - $Y \in \{0, 1, 2, 3, 4\}$ : indica o conteúdo do jarro com capacidade de 4 litros
- Representação das ações
  - $\text{oper}(\text{enche1}, [X, Y], [3, Y]) \leftarrow X < 3$
  - $\text{oper}(\text{enche2}, [X, Y], [X, 4]) \leftarrow Y < 4$
  - $\text{oper}(\text{esvazia1}, [X, Y], [0, Y]) \leftarrow X > 0$
  - $\text{oper}(\text{esvazia2}, [X, Y], [X, 0]) \leftarrow Y > 0$
  - $\text{oper}(\text{transfere1para2}, [X, Y], [0, X + Y]) \leftarrow X > 0, X + Y \leq 4$
  - $\text{oper}(\text{transfere1para2}, [X, Y], [X + Y - 4, 4]) \leftarrow X > 0, Y < 4, X + Y > 4$
  - $\text{oper}(\text{transfere2para1}, [X, Y], [X + Y, 0]) \leftarrow Y > 0, X + Y \leq 3$
  - $\text{oper}(\text{transfere2para1}, [X, Y], [3, X + Y - 3]) \leftarrow X < 3, Y > 0, X + Y > 3$

# Exercícios

3. Implemente um algoritmo de busca para problema do fazendeiro, qual encontra-se na margem esquerda de um rio, levando consigo uma raposa, uma galinha e milho. O fazendeiro precisa ir a outra margem do rio com toda a sua carga intacta, mas somente dispõe de um bote com capacidade para levar apenas ele e uma carga. O fazendeiro pode cruzar o rio quantas vezes forem necessárias, mas em sua ausência a raposa pode comer a galinha ou a galinha pode comer o milho.

- Representação dos estados tupla  $[F, R, G, M]$ 
  - $F, R, G, M \in \{e, d\}$ : lado que está o fazendeiro, a raposa, a galinha e o milho, respectivamente
- Representação das ações
  - $\text{oper}(\text{vaiSozinho}, [e, R, G, M], [d, R, G, M]) \leftarrow F = e, R \neq G, G \neq M$
  - $\text{oper}(\text{levaRaposa}, [e, e, G, M], [d, d, G, M]) \leftarrow F = e, R = e, G \neq M$
  - $\text{oper}(\text{levaGalinha}, [e, R, e, M], [d, R, d, M]) \leftarrow F = e, G = e$
  - $\text{oper}(\text{levaMilho}, [e, R, G, e], [d, R, G, d]) \leftarrow F = e, M = e, R \neq G$
  - $\text{oper}(\text{voltaSozinho}, [d, R, G, M], [e, L, G, M]) \leftarrow F = d, R \neq G, G \neq M$
  - $\text{oper}(\text{trazRaposa}, [d, d, G, M], [e, e, G, M]) \leftarrow F = d, R = d, G \neq M$
  - $\text{oper}(\text{trazGalinha}, [d, R, d, M], [e, R, e, M]) \leftarrow F = d, G = d$
  - $\text{oper}(\text{trazMilho}, [d, R, G, d], [e, R, G, e]) \leftarrow F = d, M = d, R \neq G$



# Exercícios

4. Crie uma representação do espaço de estados para o problema do quebra-cabeça de 8 peças. O problema consiste em mover peças do quebra-cabeça na horizontal ou vertical, de modo que a configuração final seja alcançada. Os movimentos realizados devem fazer com que a peça em questão ocupe a posição vazia adjacente à ela. Os movimentos devem ser sequencialmente realizados modo que a configuração final seja alcançada. Após criar a representação do espaço de estados, crie um algoritmo de busca para solucionar o problema.

**Estado inicial**

4	2	7
	8	6
3	5	1



**Estado final**

1	4	7
2	5	8
3	6	

# Estratégia de busca heurística

- Limitações da estratégia de busca cega
  - Encontra soluções, testa e expande estados, sistematicamente
    - Esta estratégia não garante encontrar soluções de custo mínimo
  - Realizam uma busca exaustiva até uma solução ser encontrada ou falhar
    - Não utilizam informações sobre o problema para guiar a busca
- Estratégia de busca heurística
  - Introduzir conhecimento do domínio do problema no processo de busca
  - Objetiva reduzir a quantidade dos estados explorados do problema
    - Explora as regiões mais promissoras na busca
    - Guia a busca através de uma estimativa de custo

# Estratégia de busca heurística

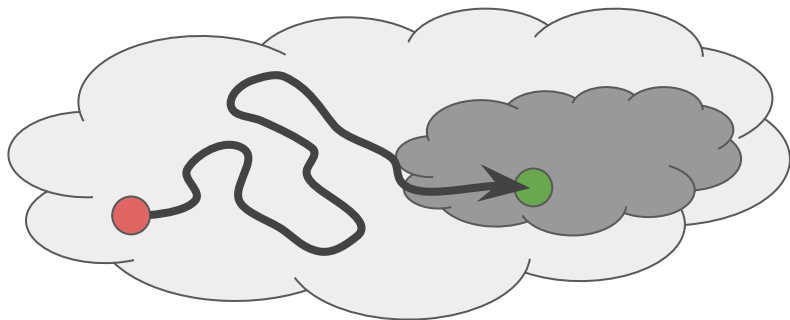
- Estratégia de busca heurística (cont.)
  - Para guiar o processo de busca usa-se duas funções  $g(n)$  e  $h(n)$
  - A função  $g(n)$  representa o custo ou qualidade até o estado atual
    - Leva em consideração o custo do estado inicial até o estado  $n$
    - Também chamado de custo do caminho percorrido até então
  - O cálculo da função  $g(n)$  requer uma informação adicional nas ações
    - $\text{oper}(\alpha, s, s', g(\alpha)) \leftarrow \beta$
    - $g(\alpha)$  é o custo da ação  $\alpha$ ,
    - Ou seja, o custo para transformar o estado  $s$  para o estado  $s'$

# Estratégia de busca heurística

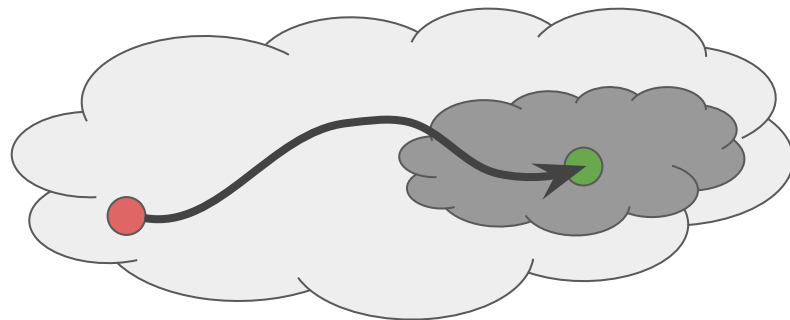
- Estratégia de busca heurística (cont.)
  - Cálculo do custo de caminho  $g(n)$ 
    - Dadas as ações  $[a_1, a_2, \dots, a_n]$  para alcançar o estado  $n$  (caminho até  $n$ )
    - Ou seja, é a sequência de ações realizadas do estado inicial até o atual
    - $g(n) = \sum_1^n g(a_i)$
    - Onde  $g(a_i)$  representa o custo da ação  $a_i$

# Estratégia de busca heurística

- Estratégia de busca heurística (cont.)
  - A função  $h(n)$  representa uma estimativa de custo
    - Leva em consideração o custo a partir do estado  $n$  até o estado objetivo
    - Não deve existir outro caminho melhor que tenha um custo  $h(n)$  menor
    - Mais especificamente,  $h(n)$  deve ser consistente (monotônica)
    - Deve ser definida usando conhecimento do domínio do problema



Exploração não guiada do espaço de estados

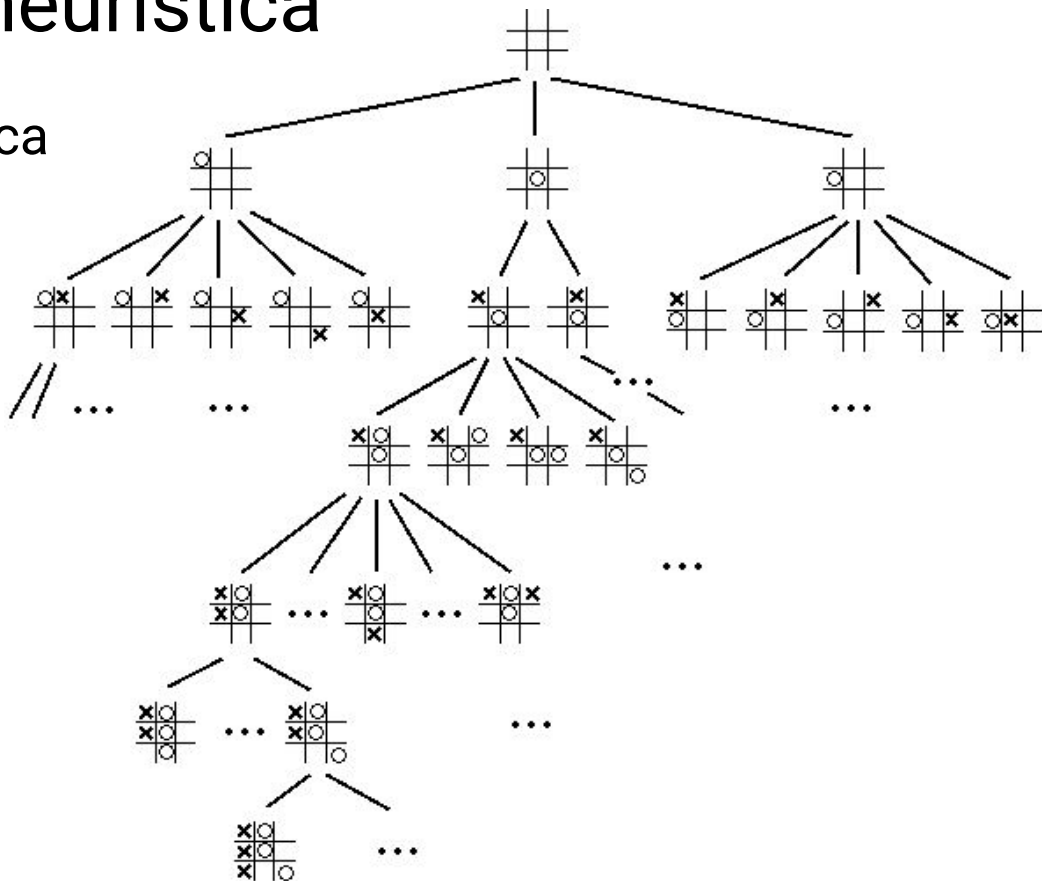


Exploração guiada do espaço de estados

# Estratégia de busca heurística

- Exemplo de função heurística para o jogo da velha

- Qual é a melhor jogada?
- Como representar a melhor jogada?
- Qual seria uma função heurística adequada?



# Estratégia de busca heurística

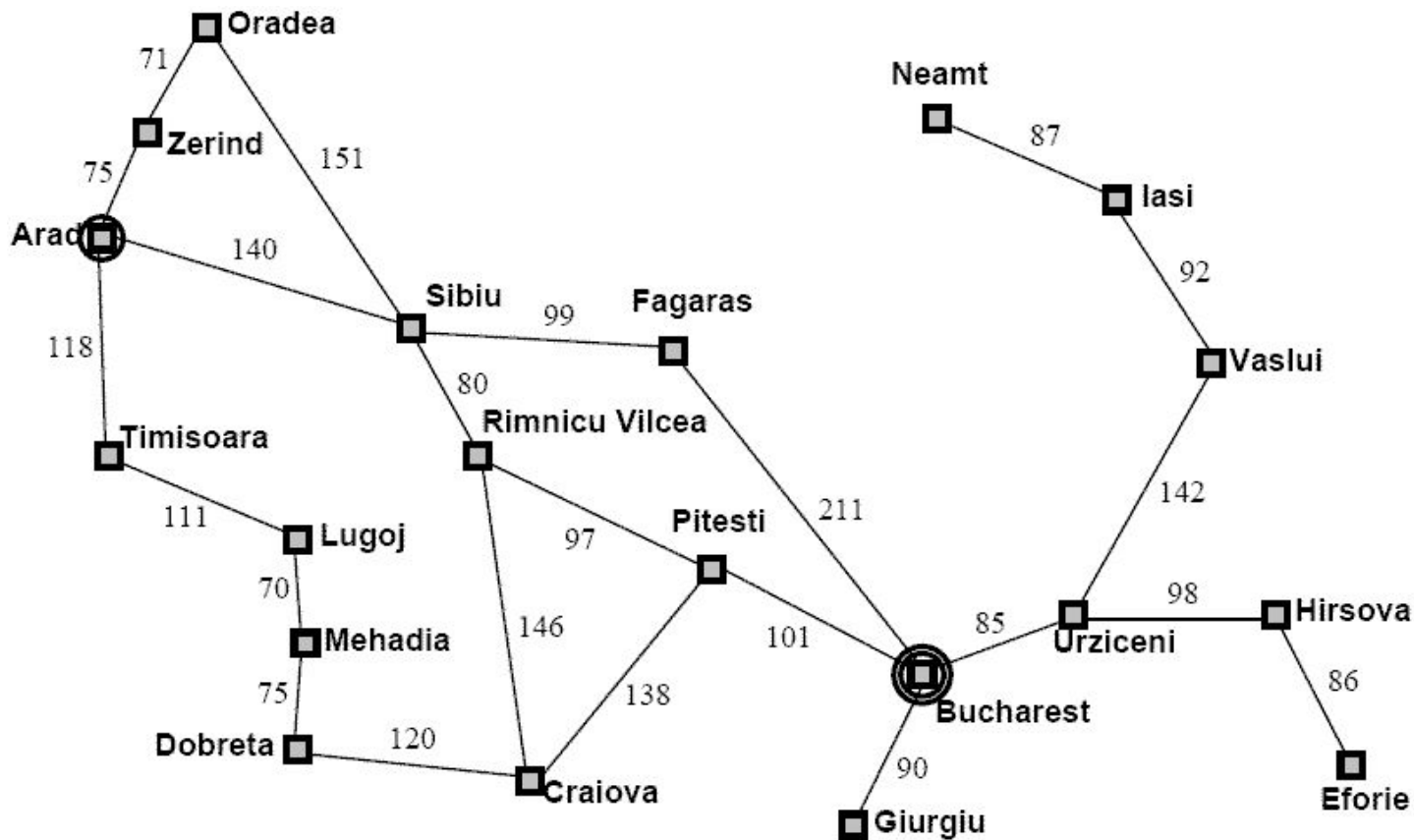
- Exemplo de função heurística para *path finding*
  - Encontrar uma rota mais próxima entre dois pontos em um mapa
    - Os estados são pontos em um plano cartesiano (mapa)
    - custo é a distância entre os pontos
    - $h(n)$  pode ser a distância em linha reta de  $n$  até o objetivo
  - Encontrar uma rota mais rápida entre dois pontos em um mapa
    - Os estados são pontos em um plano cartesiano (mapa)
    - Custo é tempo de deslocamento
    - $h(n)$  pode ser distância até o objetivo, dividida pela velocidade máxima

# Estratégia de busca heurística

- Exemplo de função heurística para *path finding* (cont.)
  - Ir de Arad até Bucharest com o menor caminho
    - $g(n)$  é a distância entre dois pontos considerando vias de acesso (estradas)
    - $h(n)$  representa a distância em linha reta entre dois pontos
  - Propriedade importante para uma heurística monotônica
    - $h(n) \leq g(n)$



## Distância entre cidades a partir das estradas existentes



### Distância em linha reta

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

# Estratégia de busca heurística

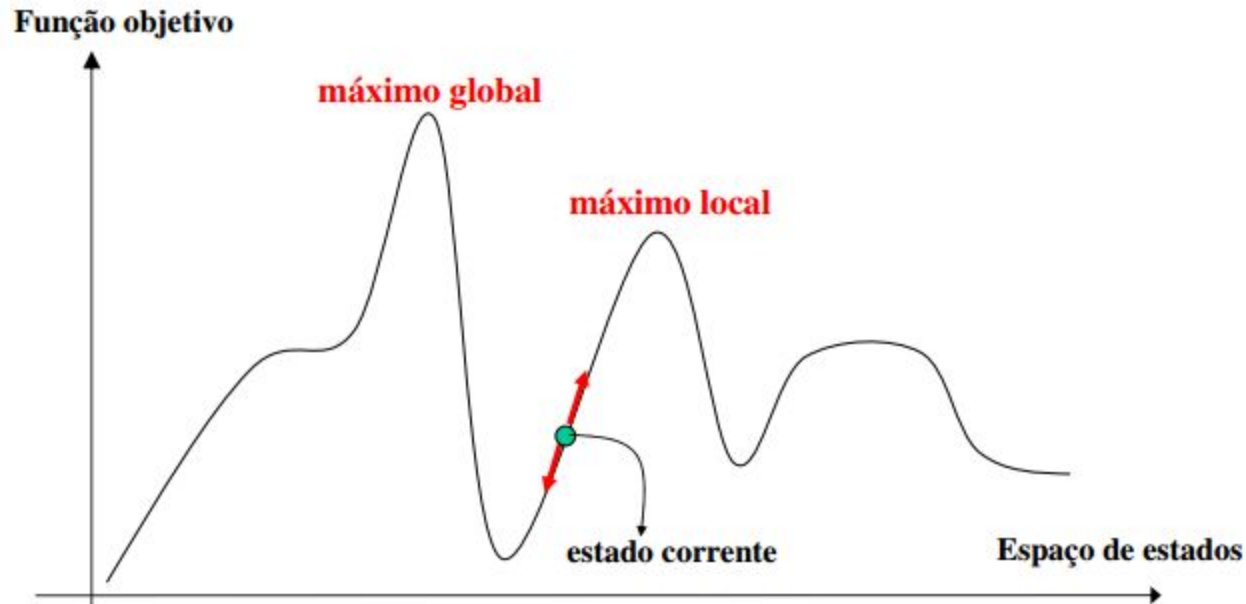
- Algoritmos de busca heurística
  - Principais características
    - Conhecimento domínio e do problema
    - $g(n)$ : avaliação do custo ou qualidade do estado
    - $h(n)$ : heurística para guiar a busca
    - $f(n)$ : função de avaliação do estado
- Algoritmos conhecidos como *best-first search*
  - Busca uniforme (menor custo):  $f(n) = g(n)$
  - Busca gulosa (*greedy search*):  $f(n) = h(n)$
  - $A^*$  (*A star*):  $f(n) = g(n) + h(n)$

# Estratégia de busca heurística

- Máximos local e global
  - São limites superiores alcançados durante a convergência de solução
  - Estes limites podem se usados para determinar se um algoritmo é ótimo
- Máximo global
  - Representa o custo ou qualidade da solução ótima
- Máximo local
  - Melhor custo encontrado até então em um ramificação da árvore de busca

# Estratégia de busca heurística

- Máximos local e global (cont.)



# Busca uniforme

- Algoritmo de busca
  - Usa uma lista de prioridades com custos dos estados a serem visitados
    - Em geral a lista ordenada é ascendente para privilegiar estados favoráveis

**BuscaUniforme**( $\mathcal{A}$ ,  $s_0$ ,  $\mathcal{G}$ )

$\Gamma \leftarrow \emptyset$  //lista dos estados já visitados

$\Sigma \leftarrow \{s_0\}$  //lista de prioridade  $g(n)$  com os estados a serem visitados

**enquanto**  $\Sigma \neq \emptyset$  **faça** //enquanto  $\Sigma$  não estiver vazia

$s \leftarrow \text{remove}(\Sigma)$  //retira o primeiro estado da lista

**se**  $s \in \mathcal{G}$  **então** //verifica se o estado atual é um dos estados alvo

**retorna** caminho( $s_0, s$ ) //retorna caminho realizado entre  $s_0$  e  $s$

**fim se**

$\Gamma \leftarrow \Gamma \cup \{s\}$  //adiciona estado atual na lista de estados visitados

$\Sigma \leftarrow \Sigma \cup (\text{sucessores}(s, \mathcal{A}) - \Gamma)$  //insere ordenado por  $g(n)$

**fim enquanto**

**retorna** erro

# Busca uniforme

- Características
  - É completo
    - Condição é ter custo de ação positivo
    - Os estados serão expandidos em ordem crescente de custo de caminho
  - É ótimo
    - Similar à busca em largura quando todos os custos das ações são iguais

# Busca gulosa

- Algoritmo de busca
  - Usa lista de prioridades com estimativas dos estados a serem visitados
    - Tem baixo custo, mas nem sempre é eficiente pois não evita máximos locais

**BuscaGulosa**( $\mathcal{A}$ ,  $s_0$ ,  $\mathcal{G}$ )

$\Gamma \leftarrow \emptyset$  //lista dos estados já visitados

$\Sigma \leftarrow \{s_0\}$  //lista de prioridade  $h(n)$  com os estados a serem visitados

**enquanto**  $\Sigma \neq \emptyset$  **faça** //enquanto  $\Sigma$  não estiver vazia

$s \leftarrow \text{remove}(\Sigma)$  //retira o primeiro estado da lista

**se**  $s \in \mathcal{G}$  **então** //verifica se o estado atual é um dos estados alvo

**retorna** caminho( $s_0, s$ ) //retorna caminho realizado entre  $s_0$  e  $s$

**fim se**

$\Gamma \leftarrow \Gamma \cup \{s\}$  //adiciona estado atual na lista de estados visitados


$\Sigma \leftarrow \Sigma \cup (\text{sucessores}(s, \mathcal{A}) - \Gamma)$  //insere ordenado por  $h(n)$

**fim enquanto**

**retorna** erro

# Busca gulosa

- Exemplo

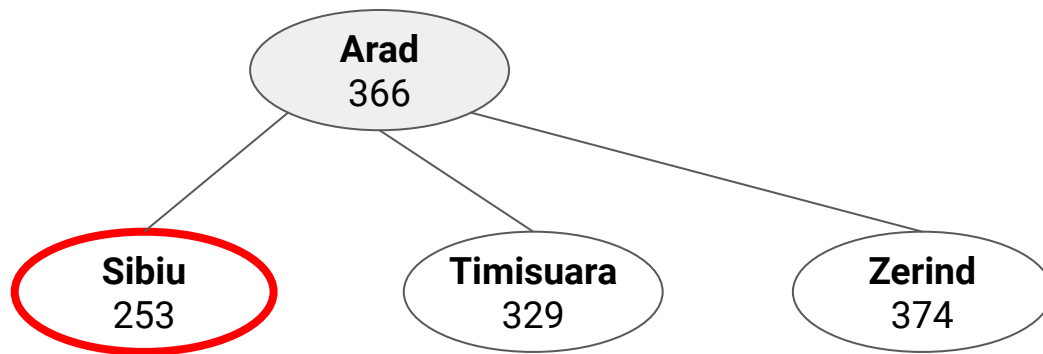


**Arad**  
366



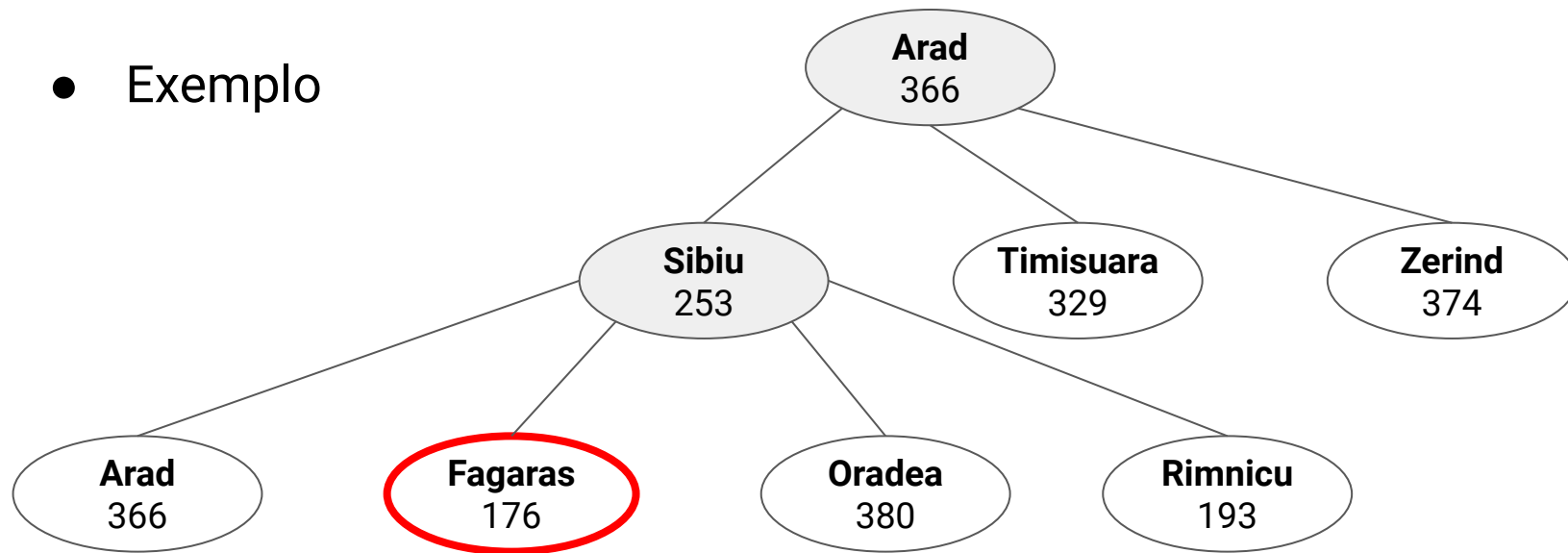
# Busca gulosa

- Exemplo



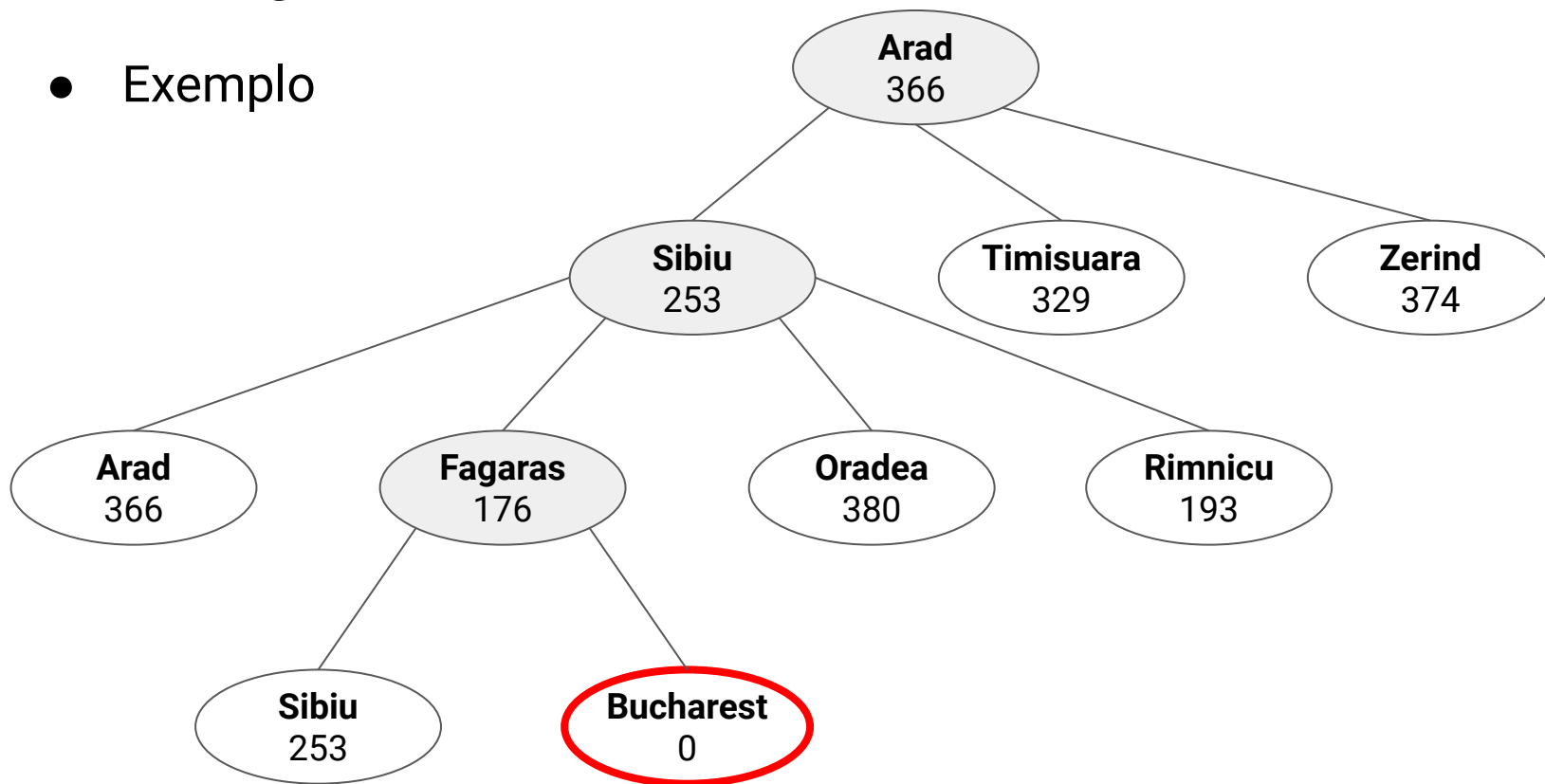
# Busca gulosa

- Exemplo



# Busca gulosa

- Exemplo



# Busca gulosa

- Características
  - É completo
    - Sempre encontrará uma solução, se existir
    - A heurística deve garantir esta propriedade
  - Não é ótimo
    - Não evita máximos locais

# Busca A\*

- Algoritmo de busca
  - Usa lista de prioridades com estimativas dos estados a serem visitados
    - É eficiente, completo e ótimo

**BuscaAStar**( $\mathcal{A}$ ,  $s_0$ ,  $\mathcal{G}$ )

$\Gamma \leftarrow \emptyset$  //lista dos estados já visitados

$\Sigma \leftarrow \{s_0\}$  //lista de prioridade  $g(n) + h(n)$  com estados a serem visitados

**enquanto**  $\Sigma \neq \emptyset$  **faça** //enquanto  $\Sigma$  não estiver vazia

$s \leftarrow \text{remove}(\Sigma)$  //retira o primeiro estado da lista

**se**  $s \in \mathcal{G}$  **então** //verifica se o estado atual é um dos estados alvo

**retorna** caminho( $s_0, s$ ) //retorna caminho realizado entre  $s_0$  e  $s$

**fim se**

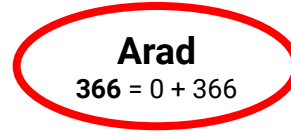
$\Gamma \leftarrow \Gamma \cup \{s\}$  //adiciona estado atual na lista de estados visitados

$\Sigma \leftarrow \Sigma \cup (\text{sucessores}(s, \mathcal{A}) - \Gamma)$  //insere ordenado por  $g(n) + h(n)$

**fim enquanto**

**retorna** erro

# Busca A\*

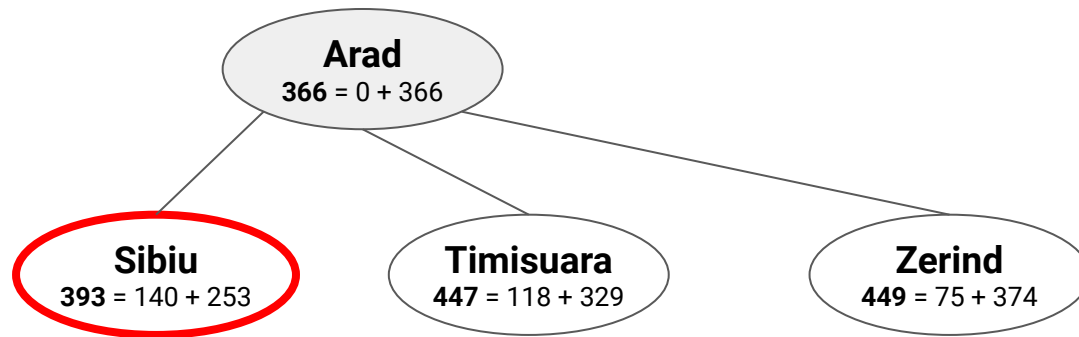


**Arad**  
 $366 = 0 + 366$

- Exemplo

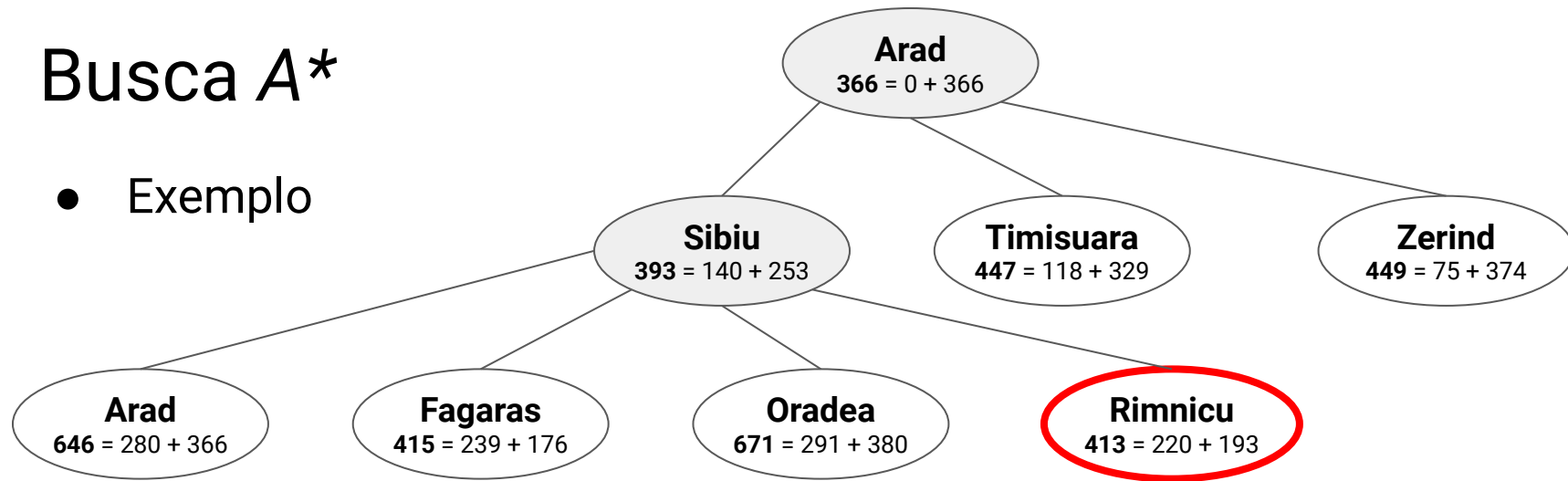
# Busca A\*

- Exemplo



# Busca A\*

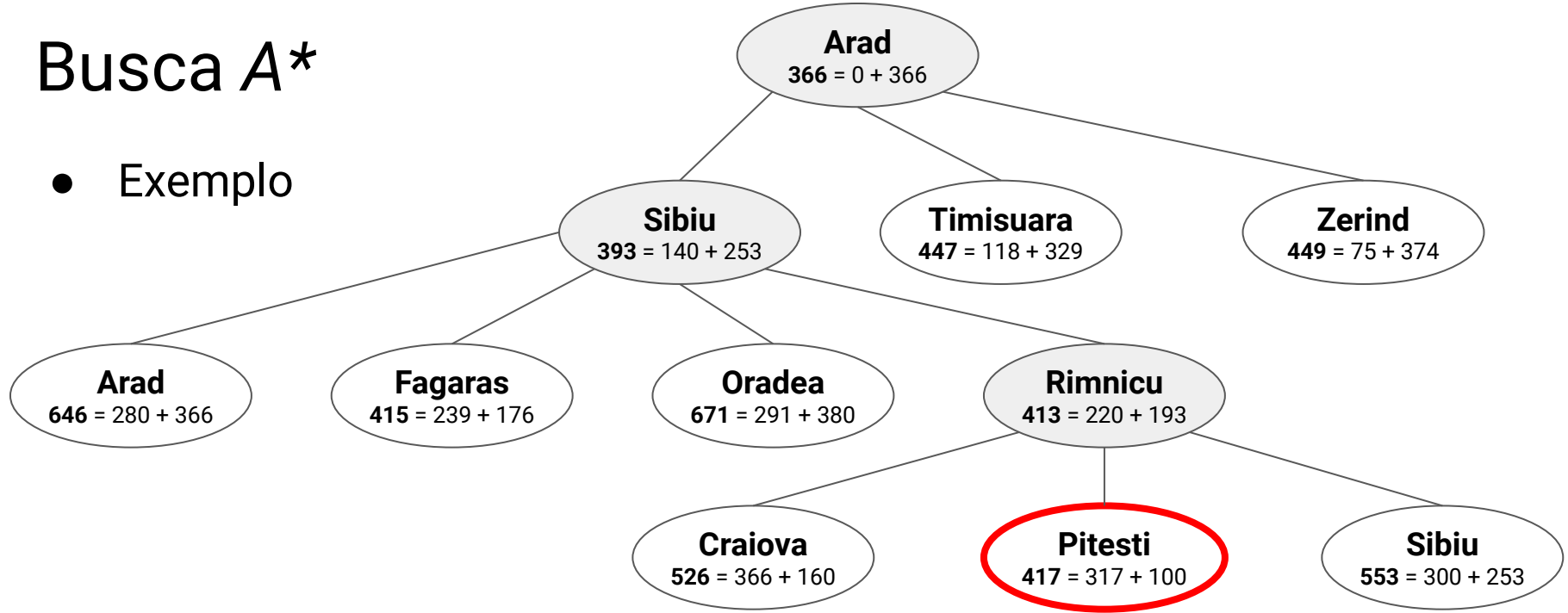
- Exemplo





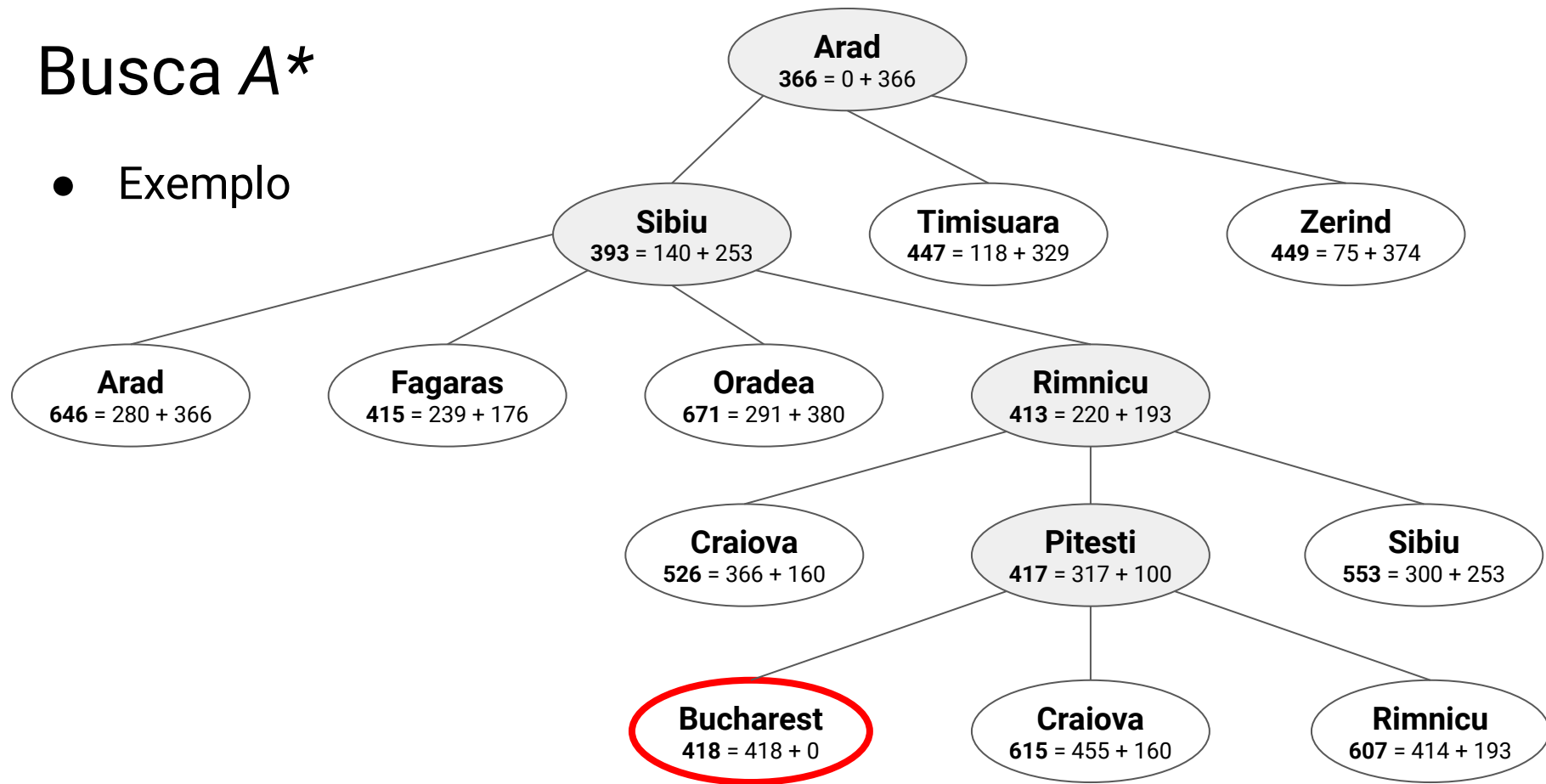
# Busca A\*

- Exemplo



# Busca A\*

- Exemplo



# Busca A\*

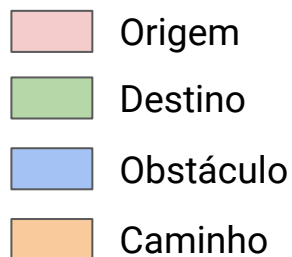
- Características
  - É completo
    - Sempre encontrará uma solução, se existir
    - A heurística deve garantir esta propriedade
  - É ótimo
    - É capaz de evitar máximos locais
    - Igualmente, a heurística deve garantir esta propriedade
    - $h(n)$  deve ser otimista e nunca deve superestimar o custo do objetivo

# Exercícios

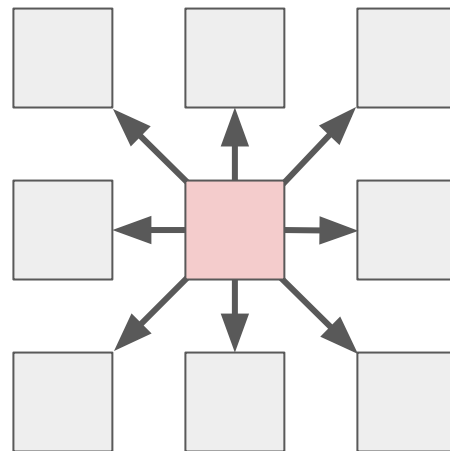
5. Implemente um algoritmo de busca gulosa para problema de deslocamento pelo mapa da Romênia. O objetivo do problema é ir de Arad até Bucharest no menor caminho possível. Além disso, o algoritmo deve visitar o menor número possível de estados candidatos.
6. Implemente um algoritmo de busca  $A^*$  para problema de deslocamento pelo mapa da Romênia. O objetivo do problema é ir de Arad até Bucharest no menor caminho possível. Além disso, o algoritmo deve visitar o menor número possível de estados candidatos.

# Exercícios

7. Implemente a busca A\* para o problema *path finding* utilizando distância Euclideana. O mapa deve ser representado por uma matriz  $M \times N$  e deve ser estabelecido o ponto de partida e o ponto de destino. Além disso, no caminho podem existir obstáculos que devem ser desviados. Para a heurística, considere as ações de movimentos laterais com o custo de 10 e as ações de movimentos na diagonal com custo de 14.



70	66	62	58	54	50
66	56	52	48	44	40
62	52	42	38	34	30
58	48	38	28	24	20
54	44	34	24	14	10
50	40	30	20	10	0



# Resolução de problemas

Inteligência artificial  
Prof. Allan Rodrigo Leite