

# **316024 Inteligência Artificial**

## **Turma A, 01/2022**

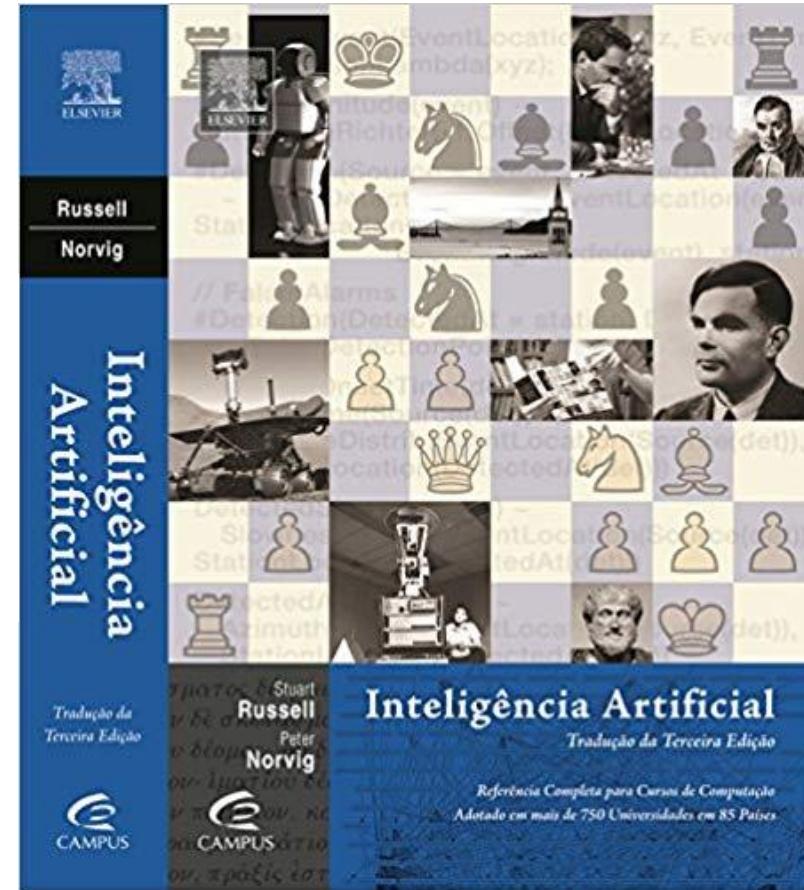
PPGInf - CIC/UnB

Prof. Li Weigang

[weigang@unb.br](mailto:weigang@unb.br)

# Resolução de problemas por meio de busca

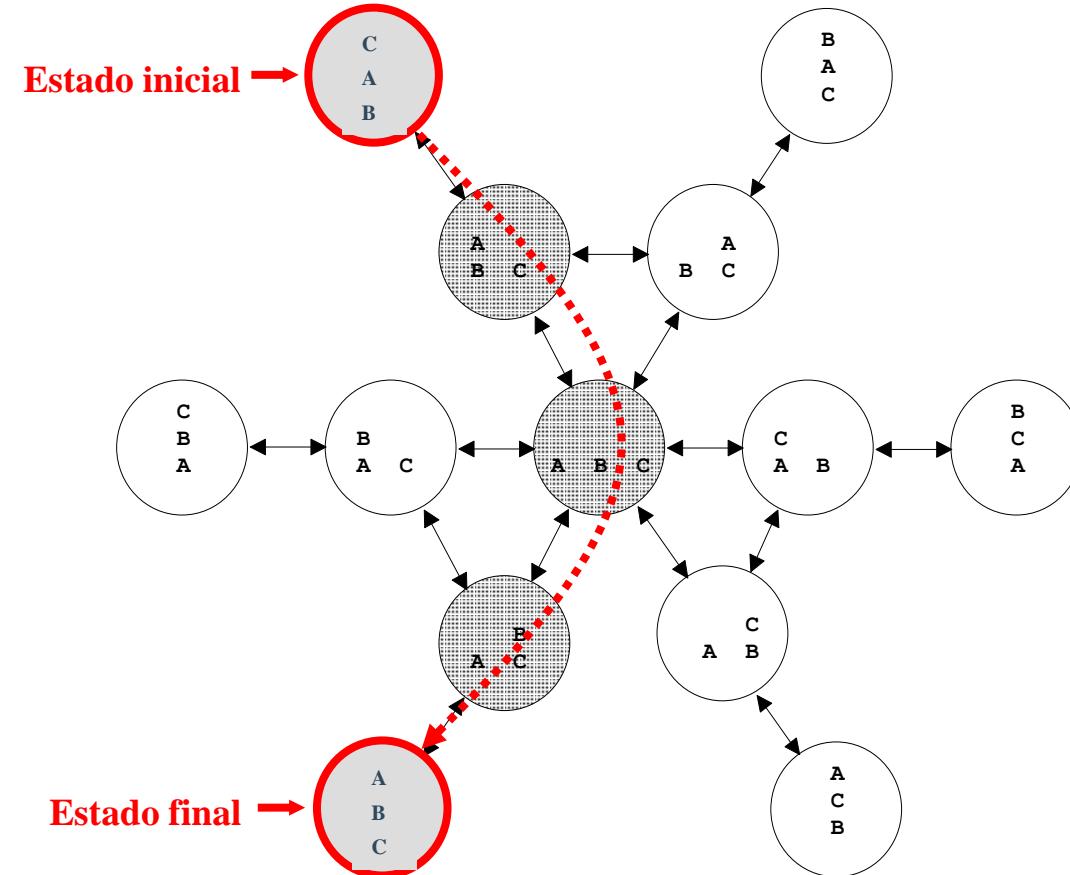
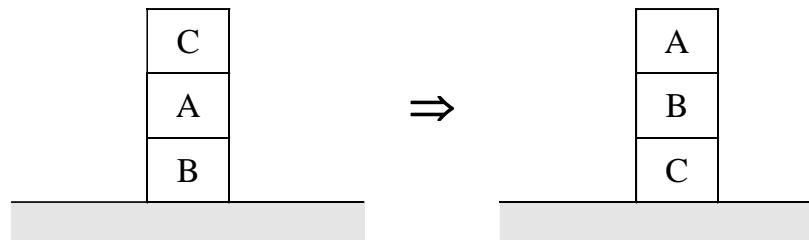
- Material:
  - Agentes inteligentes (Cap. 1 e 2)
  - Capítulo 3 – Russell & Norvig
    - Seções 3.1, 3.2 e 3.3
- Contato:
  - weigang@unb.br
  - weigangbr@gmail.com
- Página web:  
<https://cic.unb.br/~weigang/>



# Problema de IA

# O agente solucionador de problemas ...

- busca uma seqüência de ações que leve a estados desejáveis (*objetivos*)



# Métodos de Busca

- Busca é uma das mais poderosas abordagens para resolução de problemas em IA
- Busca é um mecanismo de resolução de problemas universal que:
  - Sistematicamente explora as *alternativas*
  - Encontra a seqüência de *passos para uma solução*

# Espaço de problemas

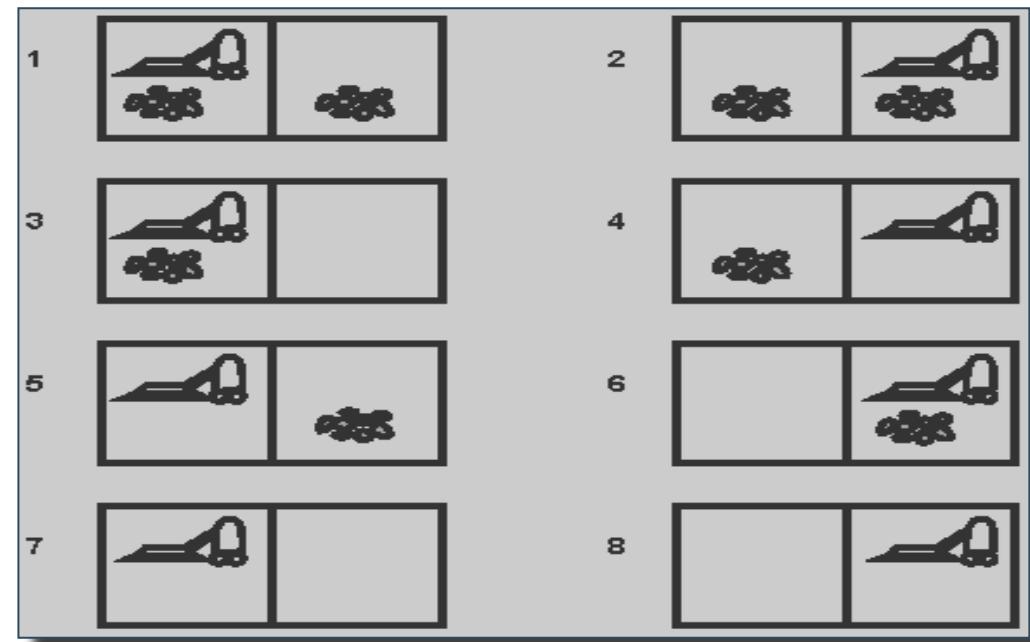
- Uma atividade simbólica orientada a objetivo ocorre em um espaço de problemas
- Busca em um espaço de problemas é visto como um *modelo geral de inteligência*

# Problemas clássicos

- Jogo dos 8 números
- Torre de Hanoi
- Missionários e Canibais
- Jarro d'água
- Mundo do aspirador
- Mundo de Wumpus
- Mundo dos blocos
- Caixeiro viajante
- Labirinto
- Cripto-aritmética
- Problema de Einstein
- Xadrez, Bridge, etc

# Exemplo: O mundo do aspirador

- O mundo consiste em **duas salas**
- Cada sala pode estar **suja**
- O agente pode estar em apenas **uma das salas de cada vez**
- Existem **8 estados possíveis**
- O agente pode efetuar **3 ações:**  
*Esquerda, Direita e Limpar*

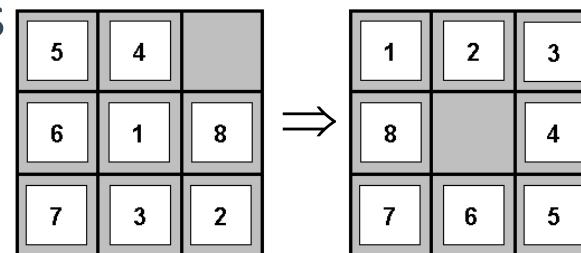


# O que é um problema em IA?

- Um **problema** em IA é definido em termos de...
  - (1) Um **espaço de estados** possíveis, incluindo:
    - um estado inicial
    - um (ou mais) estado final = objetivo
    - espaço de estados: *todas as cidades da região*

— Exemplo<sub>1</sub>: dirigir de Brasília a Goiânia

— Exemplo<sub>2</sub>: jogo de 8 números



# O que é um problema em IA?

- Um **problema** em IA é definido em termos de...
- (2) Um conjunto de **ações** (ou **operadores**) que permitem passar de um estado a outro
  - \_\_ **Exemplo<sub>1</sub>**: dirigir de uma cidade a outra
  - \_\_ **Exemplo<sub>2</sub>**: regras de movimentação de peças no jogo de 8 números
- (3) Um **objetivo**
  - Pode ser uma propriedade abstrata
    - \_\_ **Exemplo<sub>1</sub>**: condição de xeque-mate no Xadrez
    - Ou um conjunto de estados finais do mundo
  - \_\_ **Exemplo<sub>2</sub>**: estar em Goiânia

# O que é um problema em IA?

- Um **problema** em IA é definido em termos de...
- (4) Uma **solução**
  - caminho (seqüência de *ações* ou *operadores*) que leva do estado inicial a um estado final (objetivo).
- (5) Um **espaço de estados**
  - conjunto de todos os estados alcançáveis a partir do estado inicial por qualquer seqüência de ações.

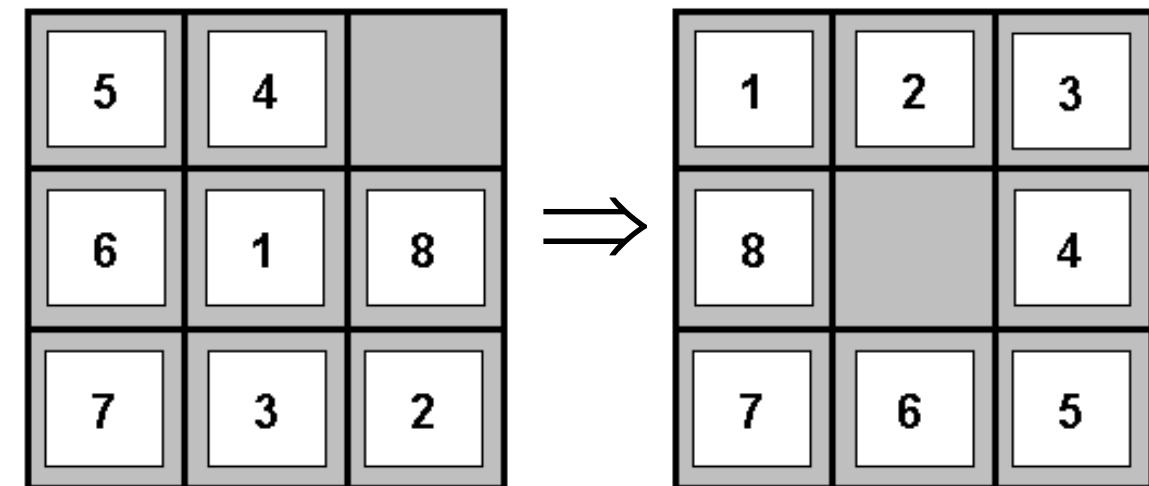
# Como formular um problema?

- **Formulação do problema e do objetivo**
  - Quais são os *estados* e as *ações* a serem consideradas?
  - Qual é (e como representar) o *objetivo*?
- **Busca (solução do problema)**
  - Processo que gera/analisa seqüências de ações para alcançar um objetivo
  - *Solução* = caminho entre estado inicial e estado final.
  - *Custo do caminho* = qualidade da solução
- **Execução**
  - Executar a solução completa encontrada ou
  - Intercalar execução com busca: exige *planejamento*

# Exemplos de formulação de problemas

- Jogo dos 8 números

- **Estados** = cada possível configuração do tabuleiro
- **Estado inicial** = qualquer um dos estados possíveis
- **Teste de término** = “ordenado, com vazio na posição [2,2]”
- **Operadores** = mover vazio (esquerda, direita, para cima e para baixo)
- **Custo do caminho** = número de passos da solução



# Exemplos de formulação de problemas

- Ida para Goiânia

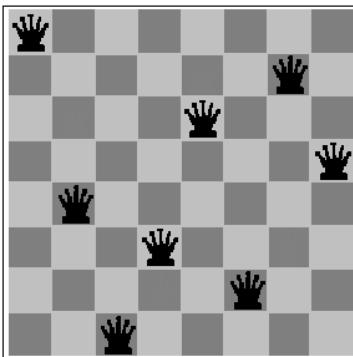
- Estados = cada possível cidade do mapa
- Estado inicial = Brasília
- Teste de término = estar em Goiânia
- Operadores = dirigir de uma cidade para outra
- Custo do caminho = número de cidades visitadas, distância percorrida, tempo de viagem, grau de divertimento, etc



# Exemplos de formulação de problemas

- 8 rainhas

- **Estado inicial** = qualquer arranjo de 0 a 8 rainhas no tabuleiro
- **Teste de término** = 8 rainhas sem ameaças mútuas
- **Operadores** = adicionar uma rainha em qualquer casa
- **Custo do caminho** = não aplicável ou zero, pois somente o estado final conta, mas o custo da busca, no entanto, pode ser de interesse



**Obs.:** Existem outras 91 soluções

```
solucao([]).
solucao([X/Y|Outros]) :-  
    solucao(Outros),  
    member(Y,[1,2,3,4,5,6,7,8]),  
    sem_ataque(X/Y,Outros).
sem_ataque(_,[]).
sem_ataque(X/Y,[X1/Y1|Outros]) :-  
    Y =\= Y1,  
    Y1 - Y =\= X1 - X,  
    Y1 - Y =\= X - X1,  
    sem_ataque(X/Y,Outros).
modelo([1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]).
```

# Alguns problemas do mundo real

- **Cálculo de rotas:** rotas em redes de computadores, planejamento de viagens, planejamento de rotas aéreas, caixeiro viajante
- **Alocação (*Scheduling*):** salas de aula, máquinas industriais
- **Projeto de VLSI:** um chip VLSI típico pode ter milhões de portas; os posicionamentos e as conexões de cada porta são críticos para o bom funcionamento de um chip
- **Navegação de robôs:**
  - geracaoalização do problema da navegação
  - Robôs movem-se em espaços contínuos, com um conjunto (infinito) de possíveis ações e estados
    - Controlar os movimentos do robô no chão, e de seus braços e pernas requer espaço multi-dimensional
- **Montagem de objetos complexos por robôs**

# Desempenho de um algoritmo de busca

- O algoritmo encontrou alguma solução?
- É uma boa solução?
  - **Custo de caminho** (qualidade da solução)
- É uma solução computacionalmente barata?
  - **Custo da busca** (tempo e memória)

# Como formular um problema?

- **Custo total** = custo do caminho + custo de busca
  - Exemplos de função de custo de caminho (caminho entre cidades)
    - número de cidades visitadas
    - distância entre as cidades
    - tempo de viagem
    - ...
- **Espaço de estados grande:**
  - melhor solução × solução mais barata

# Como escolher estados e ações?

- Abstrair-se dos detalhes irrelevantes
  - na formulação do problema (ações e estados)
  - nas funções de custo de caminho e de busca
- Exemplo: dirigir de Brasília a Goiânia
  - Não interessa:
    - número de passageiros
    - o que toca no rádio (estado)
    - cidades sem acesso rodoviário (estado)
    - o que se come e bebe dentro do carro (ações)
    - número de postos policiais no caminho (custo de caminho)
    - número de operações de adição (custo de busca)
- É tarefa do projetista fazer as boas escolhas

# Como formular um problema?

- **Como escolher estados e ações?**

- **Outro exemplo:** Jogo das 8 Rainhas
  - Dispor 8 rainhas no tabuleiro de forma que não possam se “atacar”
    - Não pode haver mais de uma rainha em uma mesma linha, coluna ou diagonal
  - Somente o custo da busca conta
    - Pois não existe custo de caminho
  - Existem diferentes estados e operadores possíveis
    - Essa escolha pode ter consequências boas ou nefastas na complexidade da busca ou no tamanho do espaço de estados

# Como formular um problema?

- **Como escolher estados e ações?**

- **Outro exemplo:** Jogo das 8 Rainhas

- **Formulação A**

- **estados:** qualquer disposição de  $n$  ( $n \leq 8$ ) rainhas
    - **operadores:** adicionar uma rainha a qualquer posição do tabuleiro
    - $64^8$  possibilidades: vai até o fim para testar se dá certo

- **Formulação B**

- **estados:** disposição de  $n$  ( $n \leq 8$ ) rainhas sem ataque mútuo (teste gradual)
    - **operadores:** adicionar uma rainha na coluna vazia mais à esquerda em que não possa ser atacada
    - melhor (2057 possibilidades), mas pode não haver ação possível

- **Formulação C**

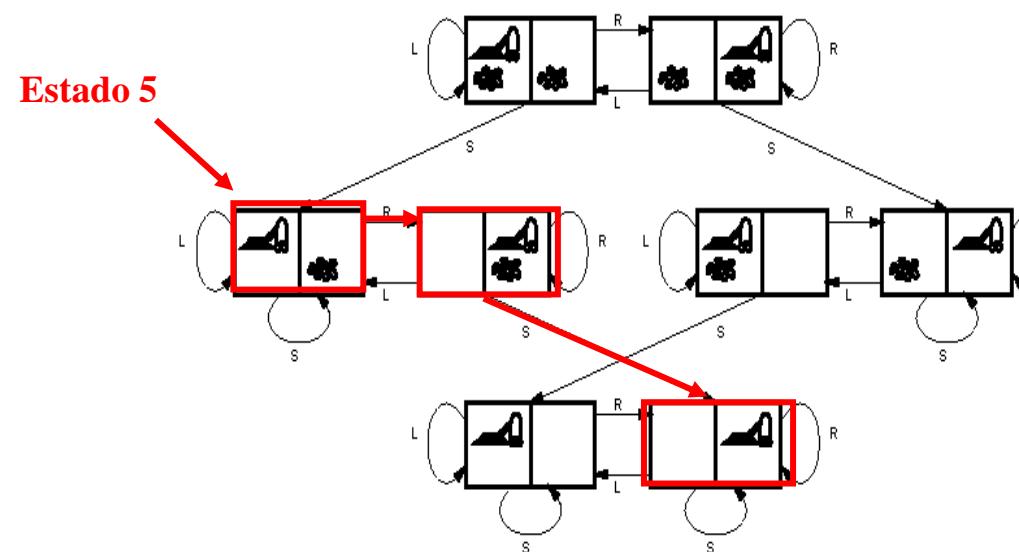
- **estados:** disposição com 8 rainhas, uma em cada coluna
    - **operadores:** mover rainha atacada para outra casa na mesma coluna

# Como formular um problema?

- **Problemas de estados simples**
  - **Conhecimento do agente**
    - Sabe em que estado está (mundo totalmente acessível)
    - Sabe o efeito de cada uma de suas ações
  - Cada ação leva a um único estado
  - **Técnica:** busca

# Problemas de estados simples

- **Exemplo: o mundo do aspirador**
  - **Estado inicial:** um dos 8 estados abaixo
  - **Operadores:** mover para a esquerda, mover para a direita, limpar
  - **Teste do objetivo:** nenhuma sujeira em qualquer sala
  - **Custo do caminho:** cada ação custa 1



# Problemas de estados múltiplos

- **Conhecimento do agente**
  - Não sabe seu estado inicial (percepção deficiente), mas sabe o efeito de suas ações
    - OU
    - Sabe seu estado inicial mas não o efeito de cada uma de suas ações
  - O agente deve raciocinar sobre os conjuntos de estados aos quais ele pode chegar pelas ações
- Nestes casos, sempre existe uma seqüência de ações que leva a um estado final
- **Técnica:** busca

# Problemas de estados múltiplos

## Exemplo: o mundo do aspirador

- **Conjunto de estados:** cada estado é um subconjunto dos 8 estados possíveis
- **Operadores:** mover para a esquerda, mover para a direita, limpar
- **Teste do objetivo:** nenhum dos estados do subconjunto de estados tem lixo
- **Custo do caminho:** cada ação custa 1

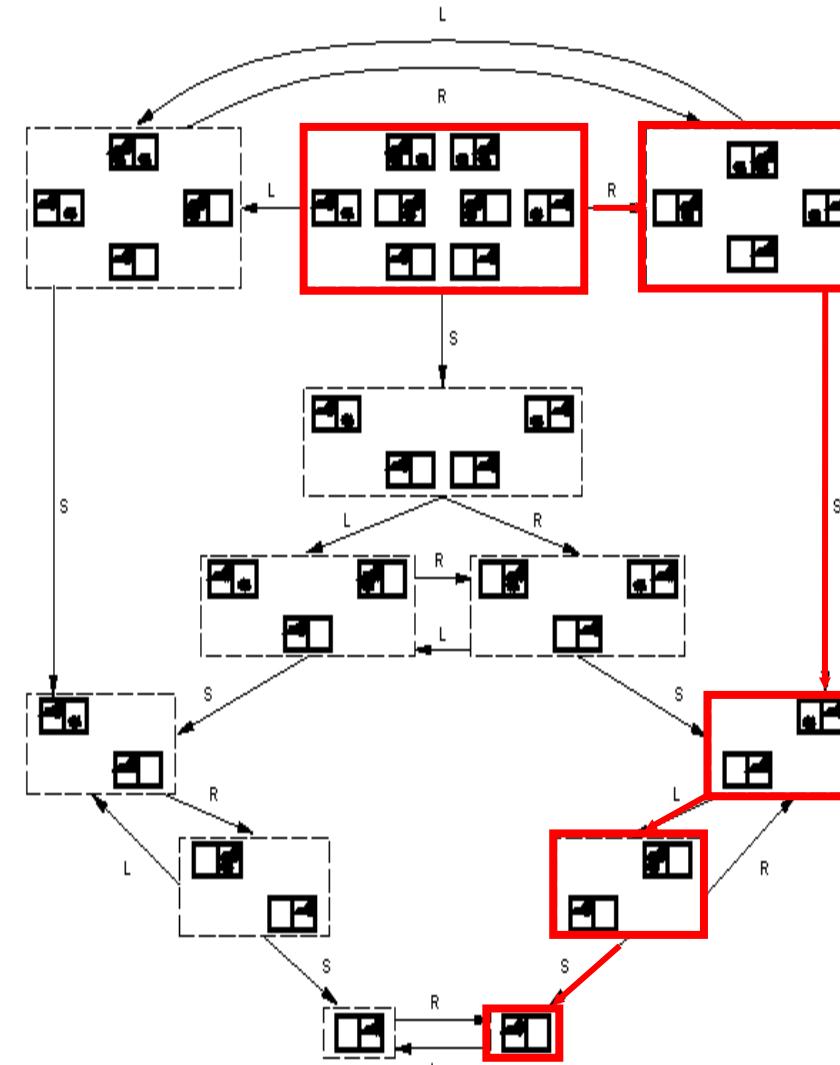
# Como formular um problema?

- **Problemas de estados múltiplos**

- Exemplo: o mundo do aspirador

No mundo do aspirador, quando não há sensores, o agente sabe que existem 8 estados iniciais

Pode ser calculado que uma ação para a **Direita** chegará a um dos estados **{2, 4, 6, 8}** e o agente pode descobrir que a seqüência de ações **[Direita, Limpar, Esquerda, Limpar]** garante que o objetivo é alcançado.



# Problemas contingenciais

- **Conhecimento do agente**

- O agente não enxerga o ambiente inteiro

OU

- O agente não sabe precisar o efeito das ações

**Exemplo:** agente enxerga apenas o quarto onde está; dirigir com mapa

- Não há uma seqüência prévia de ações que garanta a solução do problema

- O agente precisa intercalar busca e execução

**Exemplo:** o agente só pode decidir aspirar quando chegar ao quarto

{1,5} → [aspirar, direita, aspirar se existe poeira]

- **Técnica:** Planejamento

- O agente constrói uma **árvore de ações**, onde cada ramo lida com uma possível contingência.

# Problemas exploratórios

- **Conhecimento do agente**

- o agente não conhece seus possíveis estados

E

- o agente não sabe o efeito de suas ações

Exemplo: estar perdido em uma cidade desconhecida sem mapa

- O agente deve explorar seu ambiente, descobrindo gradualmente o resultado de suas ações e os estados existentes

Se o agente “sobreviver”, terá aprendido um mapa do ambiente que poderá ser reutilizado em outros problemas

- **Técnica:** Aprendizagem

# Implementação

- **Modelo genérico de agentes resolvedores de problemas**

função Agente-Simples-SP( $p$ ) retorna uma *ação*  
entrada:  $p$ , um dado perceptivo  
 $estado \leftarrow$  Atualiza-Estado( $estado, p$ )  
se  $s$  (seqüência de ações) está vazia  
então  
     $o(\text{objetivo}) \leftarrow$  Formula-Objetivo( $estado$ )  
     $problema \leftarrow$  Formula-Problema( $estado, o$ )  
     $s \leftarrow$  **Busca(*problema*)**  
     $ação \leftarrow$  Primeira( $s, estado$ )  
     $s \leftarrow$  Resto( $s, estado$ )  
retorna *ação*

# Implementação

- **Espaços de Estados** podem ser representados como uma árvore onde os estados são nós e as operações são arcos

Os nós da árvore podem guardar mais informação do que apenas o estado. Formam uma **estrutura** de dados com 5 componentes:

1. o estado correspondente
2. o seu nó pai
3. o operador aplicado para gerar o nó (a partir do pai)
4. a profundidade do nó
5. o custo do nó (desde a raiz)

# Implementação

- **Busca em espaço de estados**

função Busca-Genérica (*problema*, Função-Insere)

retorna uma solução ou falha

*fronteira*  $\leftarrow$  Faz-Fila(Faz-Nó(Estado-Inicial[*problema*]))

loop do

    se *fronteira* está vazia então retorna falha

*nó*  $\leftarrow$  Remove-Primeiro(*fronteira*)

    se Teste-Término[*problema*] aplicado a Estado[*nó*] tiver sucesso

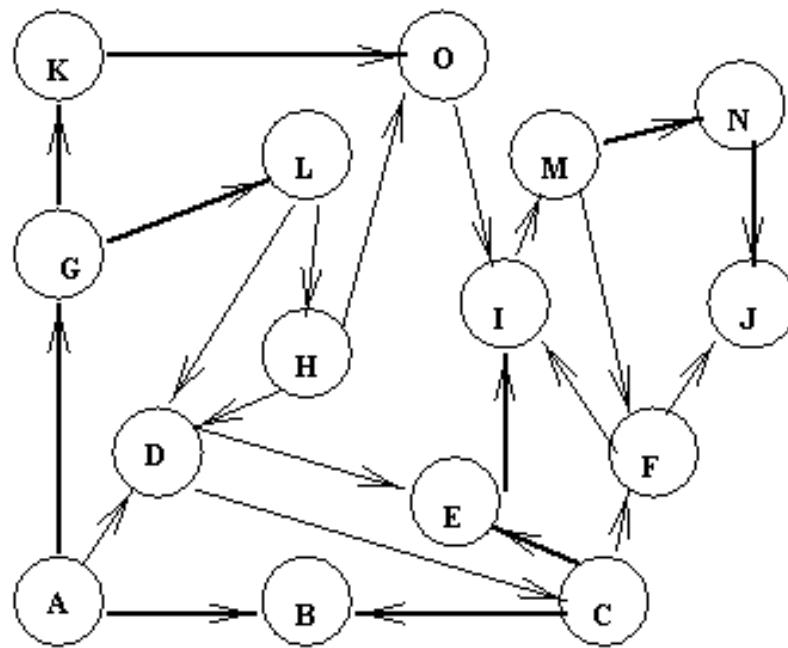
        então retorna *nó*

*fronteira*  $\leftarrow$  Função-Insere(*fronteira*, Operadores[*problema*])

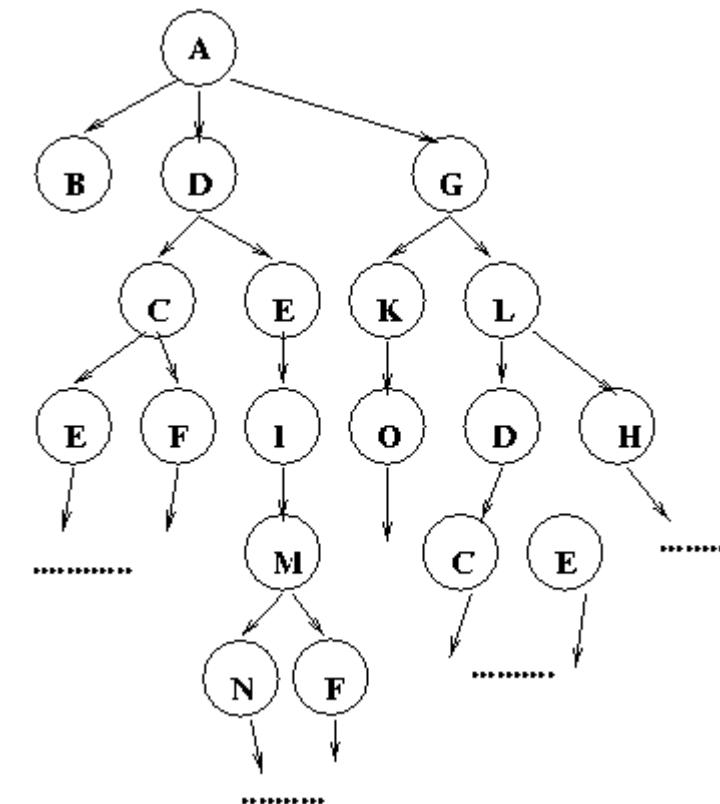
    end

# Como procurar a solução de um problema?

## ◆ Problema de busca geral



- Representação de uma árvore de busca



# Como procurar a solução de um problema?

- **Métodos para a resolução de problema**
  - Satisfação de restrições
  - Busca exaustiva (ou cega)
  - Busca heurística (ou informada)

# Satisfação de restrições

- Um **problema de satisfação de restrições** é composto de:
  1. **Estados**, que correspondem a valores de um conjunto de variáveis
  2. O **teste de objetivo** especifica um conjunto de **restrições** que os **valores devem obedecer**

# Satisfação de restrições

- **Problemas de busca comuns**
    - Estado é uma “caixa preta”, ou seja, qualquer velha estrutura de dados que possibilite teste de objetivo, avaliação e sucessor
  - **Satisfação de restrições**
    - Estado é definido por *variáveis*  $V_i$ , com *valores* de um *domínio*  $D_i$
    - O teste de objetivo é um conjunto de restrições que especificam combinações alcançáveis de valores através de subconjuntos de variáveis
- Exemplos: 8 rainhas, cripto-aritmética, problema de Einstein, projeto de circuitos, *scheduling*, ...

# Satisfação de restrições

- Exemplos

- Problema das 8 rainhas:

- Variáveis:  $R_1, R_2, \dots, R_8$ , onde  $R_i$  é a casa ocupada pela  $i^{\text{a}}$  rainha.
  - Domínio:  $\{1, 2, 3, 4, 5, 6, 7, 8\}$
  - Restrições:  $R_i \neq R_j$  (não pode estar na mesma linha)  
 $|R_i - R_j| \neq |i - j|$  (não pode estar na mesma diagonal)
  - Implementação:

```
solucao([]).
solucao([X/Y|Outros]) :-  
    solucao(Outros),  
    member(Y,[1,2,3,4,5,6,7,8]),  
    sem_ataque(X/Y,Outros).
sem_ataque(_,[]).
sem_ataque(X/Y,[X1/Y1|Outros]) :-  
    Y =\= Y1,  
    X - Y =\= X1 - X,  
    Y1 - Y =\= X - X1,  
    sem_ataque(X/Y,Outros).
modelo([1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]).
```

# Busca cega (ou exaustiva)

# Busca cega (ou exaustiva)

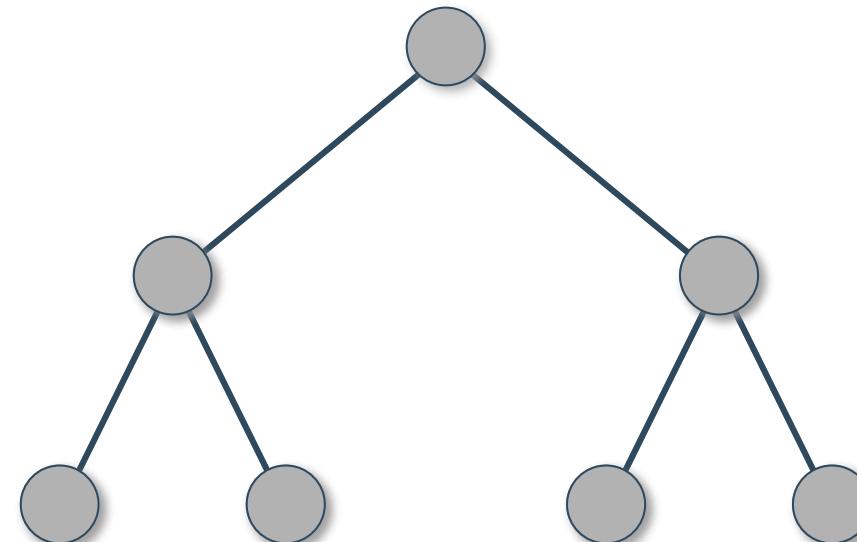
- Não sabe qual o melhor nó da fronteira a ser expandido = menor custo de caminho desse nó até um **nó final** (objetivo)
- **Estratégias de Busca** (ordem de expansão dos nós):
  - caminhamento em largura
  - caminhamento em profundidade
  - ... e suas variações.
- **Direção de Busca:**
  - Do estado inicial para o objetivo (*forward chaining*)
  - Do objetivo para o estado inicial (*backward chaining*)
  - Ambos ao mesmo tempo (*bidirecional*)

# Busca cega (ou exaustiva)

- Estratégias
  - Busca em largura
  - Busca com custo uniforme
  - Busca em profundidade
  - Busca com profundidade limitada
  - Busca com aprofundamento iterativo

# Busca cega (ou exaustiva)

- Busca em largura
  - Ordem de expansão dos nós:
    1. Nô raiz
    2. Todos os nós de profundidade 1
    3. Todos os nós de profundidade 2, etc...



# Busca cega (ou exaustiva)

- **Busca em largura**

- Encontra a solução de menor custo de caminho.
- *Completa sempre e ótima se condição<sub>1</sub> é satisfeita*

**Condição<sub>1</sub>:**

$$\forall n',n \quad \text{profundidade}(n') \geq \text{profundidade}(n) \Rightarrow \\ \text{custo de caminho}(n') \geq \text{custo de caminho}(n).$$

**Obs:** sempre é a solução geral mais barata, devido ao custo de busca associado

- **Custo de tempo:**

- *se o fator de expansão do problema* (número de nós gerados a partir de cada nó) é  $b$  e a primeira solução para o problema está no nível  $d$ , *então* o número máximo de nós gerados até se encontrar a solução é  $1 + b + b^2 + \dots + b^d$
- **Custo exponencial** =  $O(b^d)$ .

# Busca cega (ou exaustiva)

- **Busca em largura**

- **Custo de memória:** A fronteira do espaço de estados deve permanecer na memória, o que é um problema mais crucial do que o tempo de execução da busca

Profundidade	Nós	Tempo	Memória
0	1	1 milissegundo	100 bytes
2	111	0.1 segundo	11 quilobytes
4	11111	11 segundos	1 megabytes
6	$10^6$	18 minutos	111 megabytes
8	$10^8$	31 horas	11 gigabytes
10	$10^{10}$	128 dias	1 terabyte
12	$10^{12}$	35 anos	111 terabytes
14	$10^{14}$	3500 anos	11111 terabytes

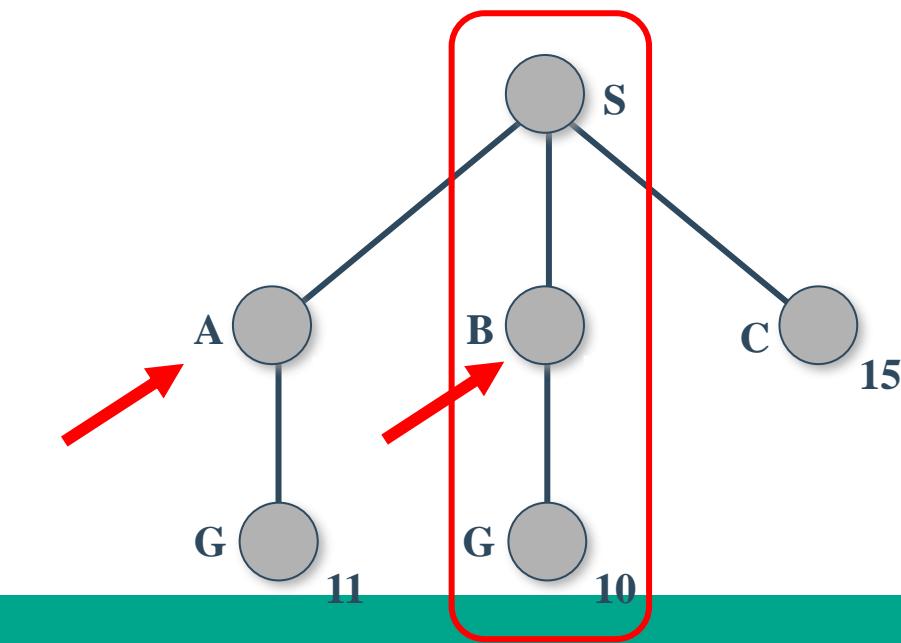
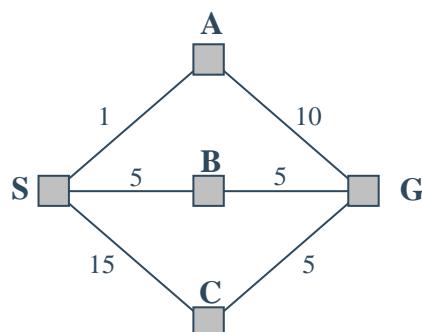
- **Conclusão:** *Esta estratégia só dá bons resultados quando a profundidade da árvore de busca é pequena*

# Busca cega (ou exaustiva)

- **Busca em largura - com custo uniforme**
  - Modifica a busca em largura
    - Expande o nó da fronteira com menor custo de caminho
    - Cada operador pode ter um custo associado diferente, medido pela função  $g(n)$ , para o nó  $n$ .
  - *Completa sempre e ótima se condição<sub>2</sub> é satisfeita.*  
Condição<sub>2</sub>:  $g(sucessor(n)) \geq g(n)$
  - **Complexidade:** teoricamente igual a busca em largura

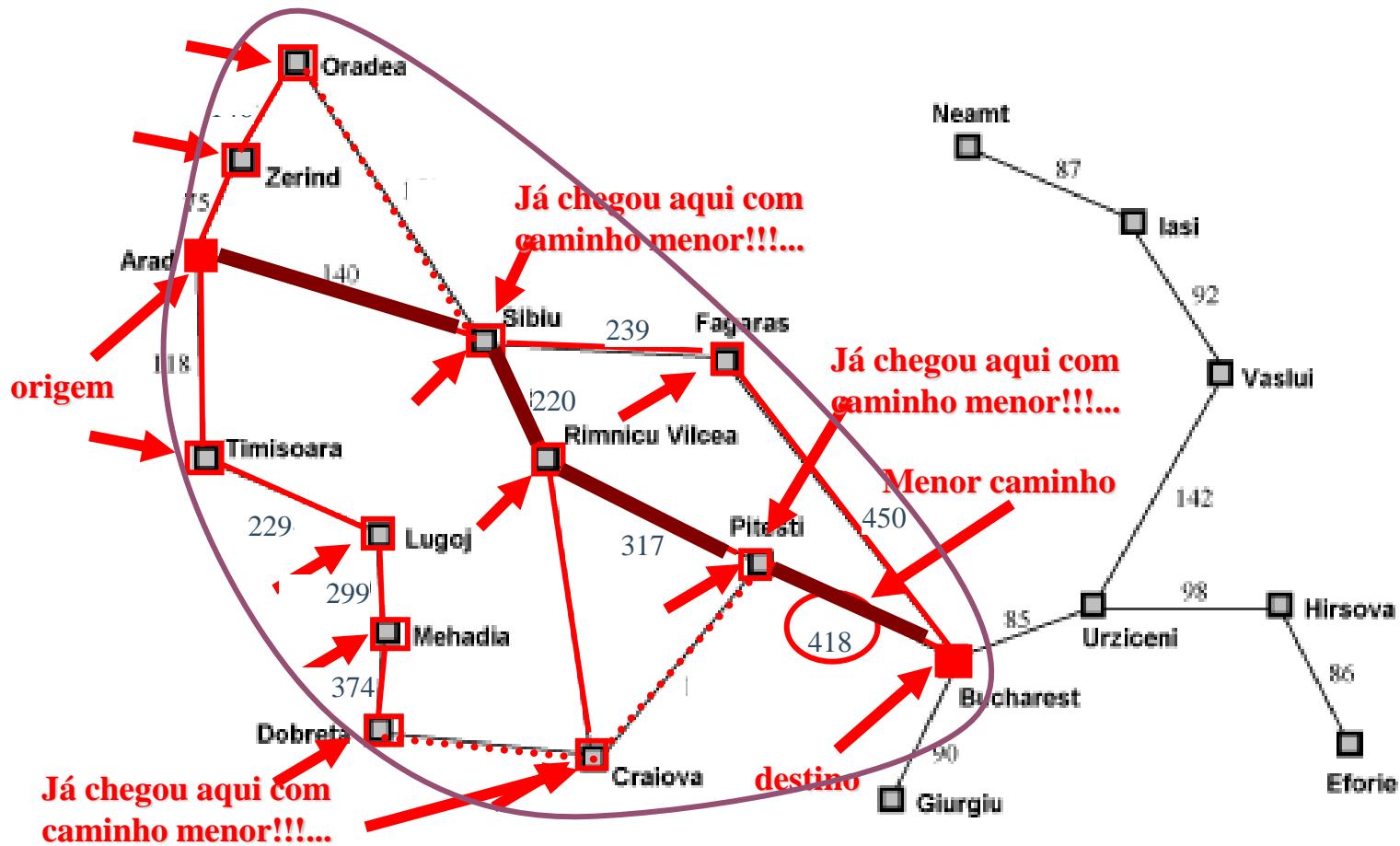
# Busca cega (ou exaustiva)

- Busca em largura com custo uniforme
  - Ordem de expansão dos nós:
    1. Nó raiz
    2. Nó profundidade 1 com menor custo
    3. Segundo nó profundidade 1 com menor custo, etc.



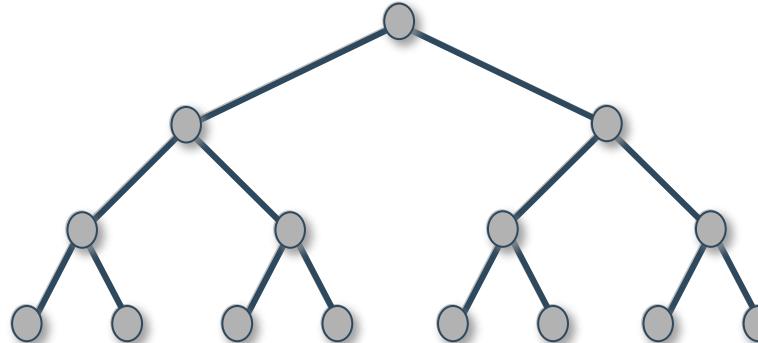
# Busca cega (ou exaustiva)

- Busca em largura



# Busca cega (ou exaustiva)

- Busca em profundidade
  - Ordem de expansão dos nós:
    - Sempre expande o nó no nível mais profundo da árvore
    - Quando um nó final não é solução, o algoritmo volta para expandir nós mais superficiais
  - 1. Nó raiz
  - 2. Primeiro nó de profundidade 1
  - 3. Primeiro nó de profundidade 2, etc.



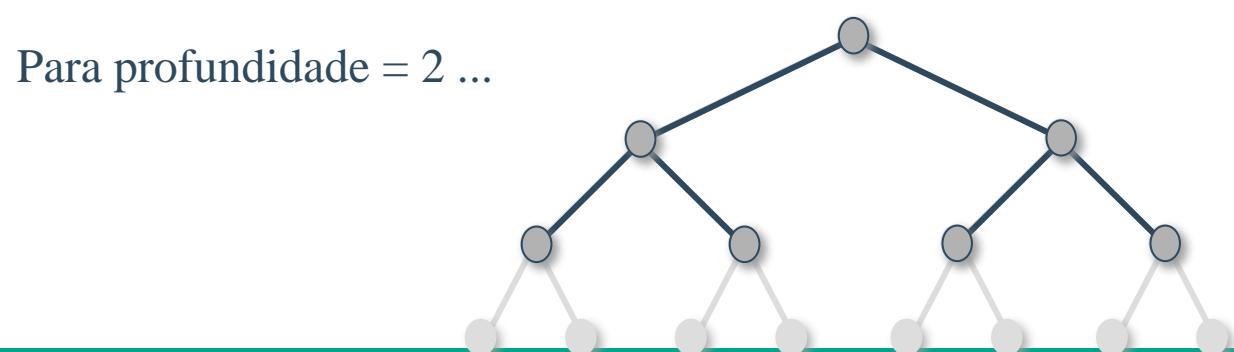
# Busca cega (ou exaustiva)

- **Busca em profundidade**

- Não é *completa nem é ótima*
- Deve ser evitada quando as árvores geradas são muito *profundas* ou geram *caminhos infinitos*
- Para problemas com várias soluções, pode ser bem mais rápida do que busca em largura
- **Custo de memória:**
  - mantém na memória o caminho que está sendo expandido no momento, e os nós irmãos dos nós no caminho
  - $O(b \cdot m)$ : necessita armazenar apenas  $b \cdot m$  nós para um espaço de estados com fator de expansão  $b$  e profundidade  $m$
  - $m$  pode ser maior que  $d$  (profundidade da 1a. solução)
- **Custo de tempo:**  $O(b^m)$ , no pior caso

# Busca cega (ou exaustiva)

- **Busca em profundidade limitada**
  - Evita o problema de caminhos muito longos ou infinitos impondo um limite máximo de profundidade para os caminhos gerados
  - Esse limite deve assegurar a completude da busca $I \geq d$ , onde  $I$  é o limite de profundidade e  $d$  é a profundidade da primeira solução do problema
  - Não é ótima (= busca em profundidade)

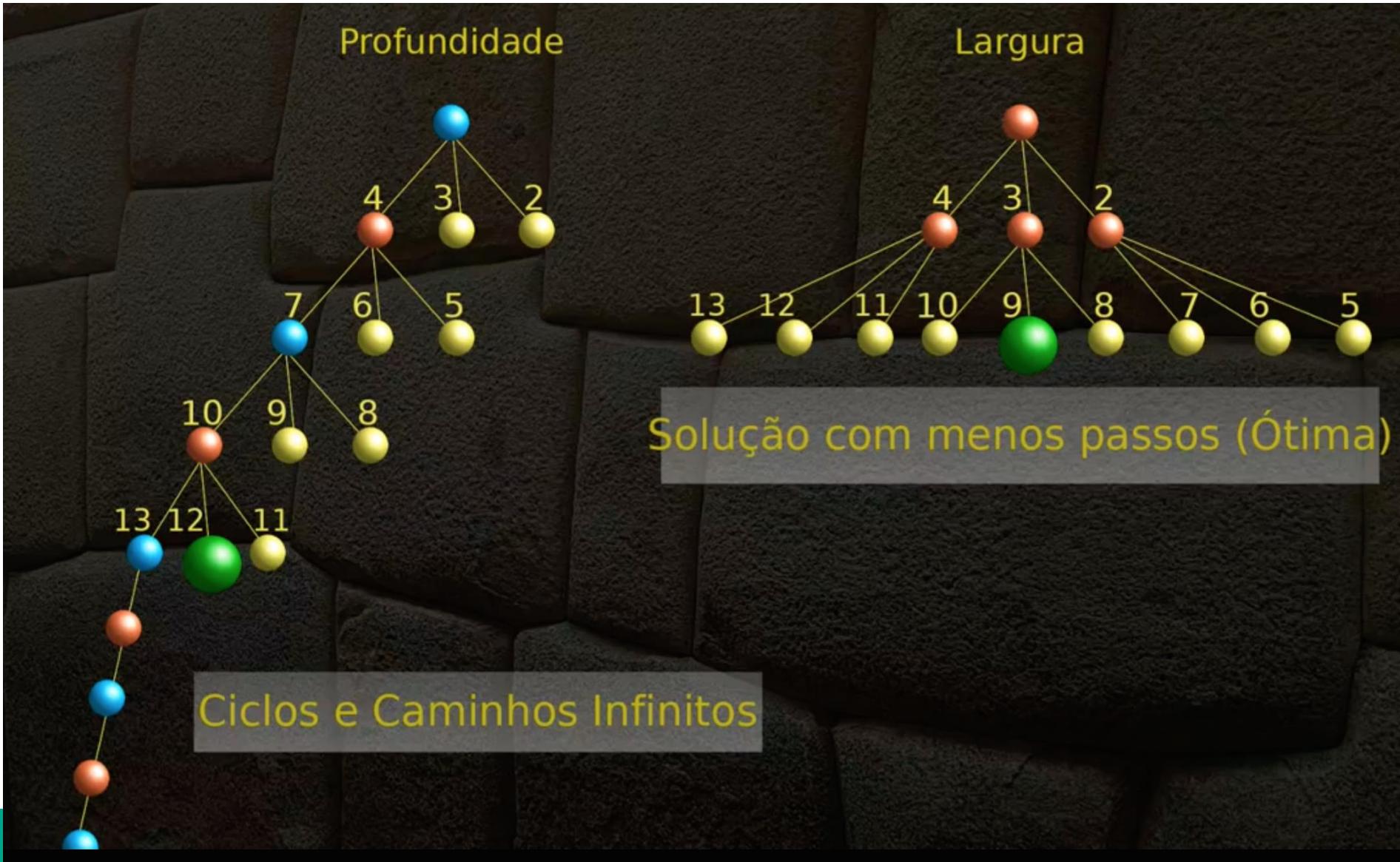


# Busca cega (ou exaustiva)

- **Busca em profundidade limitada**
  - **Complexidade de tempo e de espaço:**
    - Necessita armazenar apenas  $b.l$  nós para um espaço de estados com fator expansão  $b$
    - Similar a da busca em profundidade:  $O(b^l)$ , onde  $l$  é o limite de profundidade
  - **Problema:**

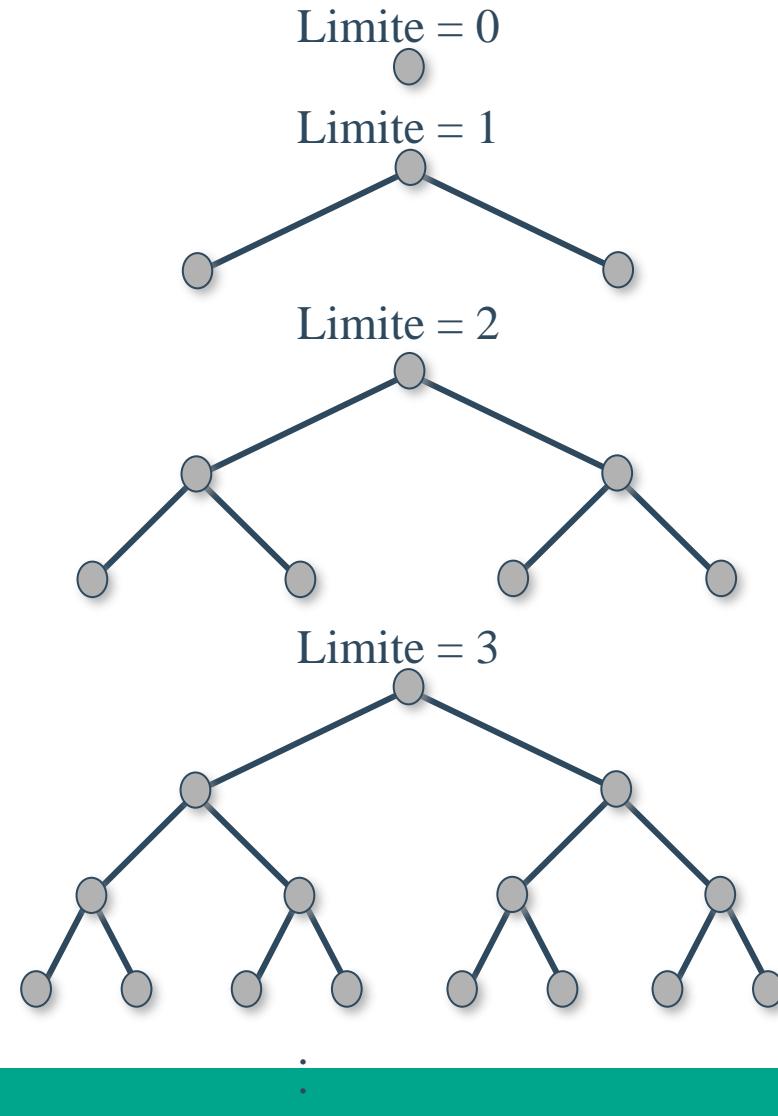
Em geral, é difícil de prever um bom limite de profundidade antes de se buscar uma solução do problema

# Busca cega (ou exaustiva)

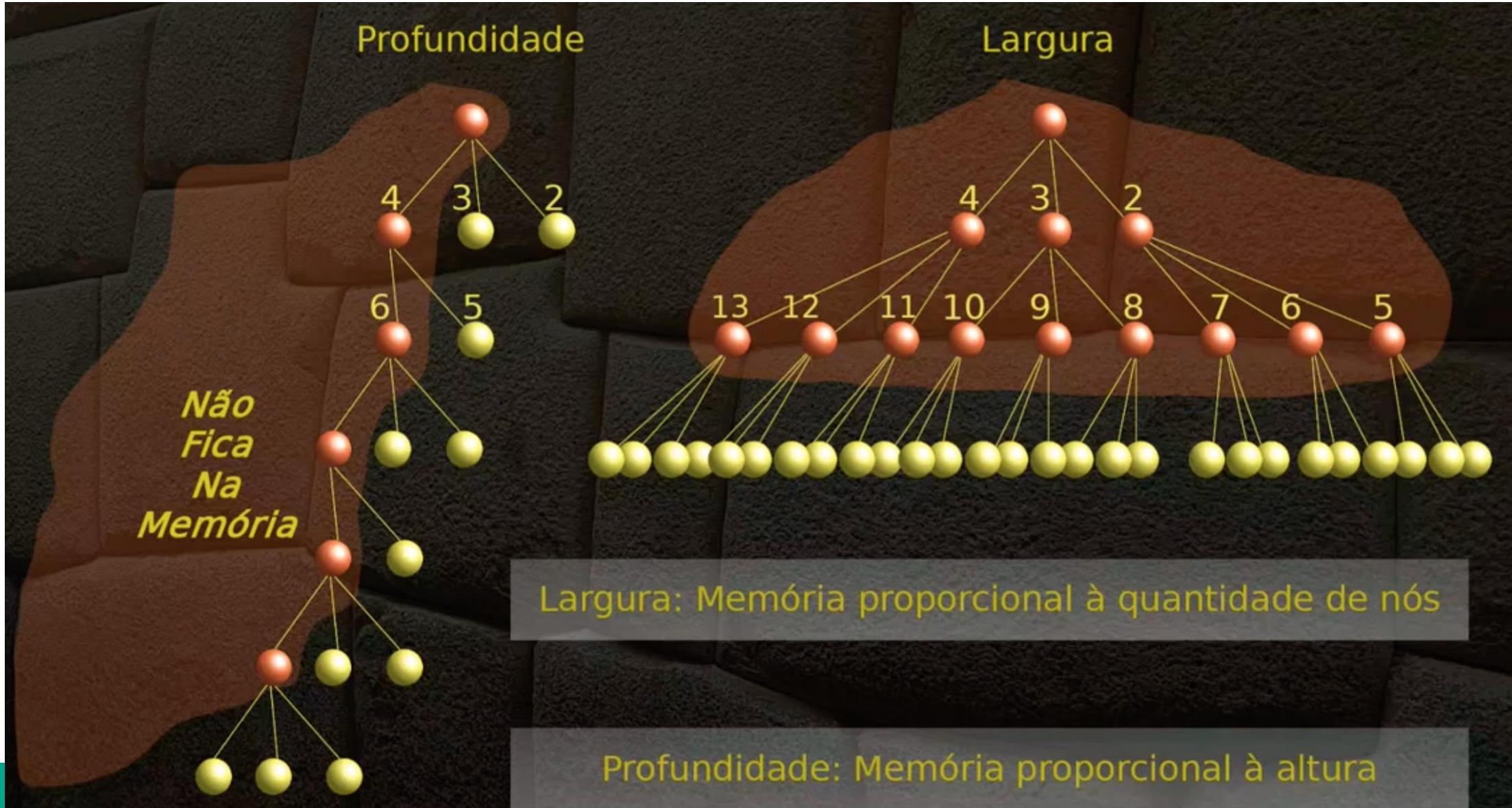


# Busca cega (ou exaustiva)

- **Busca em aprofundamento iterativo**
  - Tenta limites com valores crescentes, partindo de zero, até encontrar a primeira solução
    - fixa profundidade =  $i$ , executa busca
    - se não chegou a um objetivo, recomeça busca com profundidade =  $i + n$  ( $n$  qualquer)
  - Se  $i = 1$  e  $n = 1$ , é igual à busca em largura
  - Combina as vantagens de busca em largura com busca em profundidade
  - É *ótima e completa*



# Busca cega (ou exaustiva)



# Busca cega (ou exaustiva)

- **Busca em aprofundamento iterativo**
  - **Complexidade:**
    - *Fator de expansão:*  $1 + b + b^2 + \dots + b^{d-2} + b^{d-1} + b^d$
    - Nº expansões:  $(d+1).1 + (d).b + (d-1).b^2 + \dots + 3.b^{d-2} + 2.b^{d-1} + 1.b^d$
  - **Custo de memória:**  $b.d$ 
    - necessita armazenar apenas  $b.d$  nós para um espaço de estados com fator de expansão  $b$  e limite de profundidade  $d$
  - **Custo de tempo:**  $O(b^d)$
  - Bons resultados quando o espaço de estados é grande e a profundidade desconhecida

# Busca cega (ou exaustiva)

Busca por Aprofundamento Iterativo (Iterative Deepening)

Memória proporcional à altura !!!

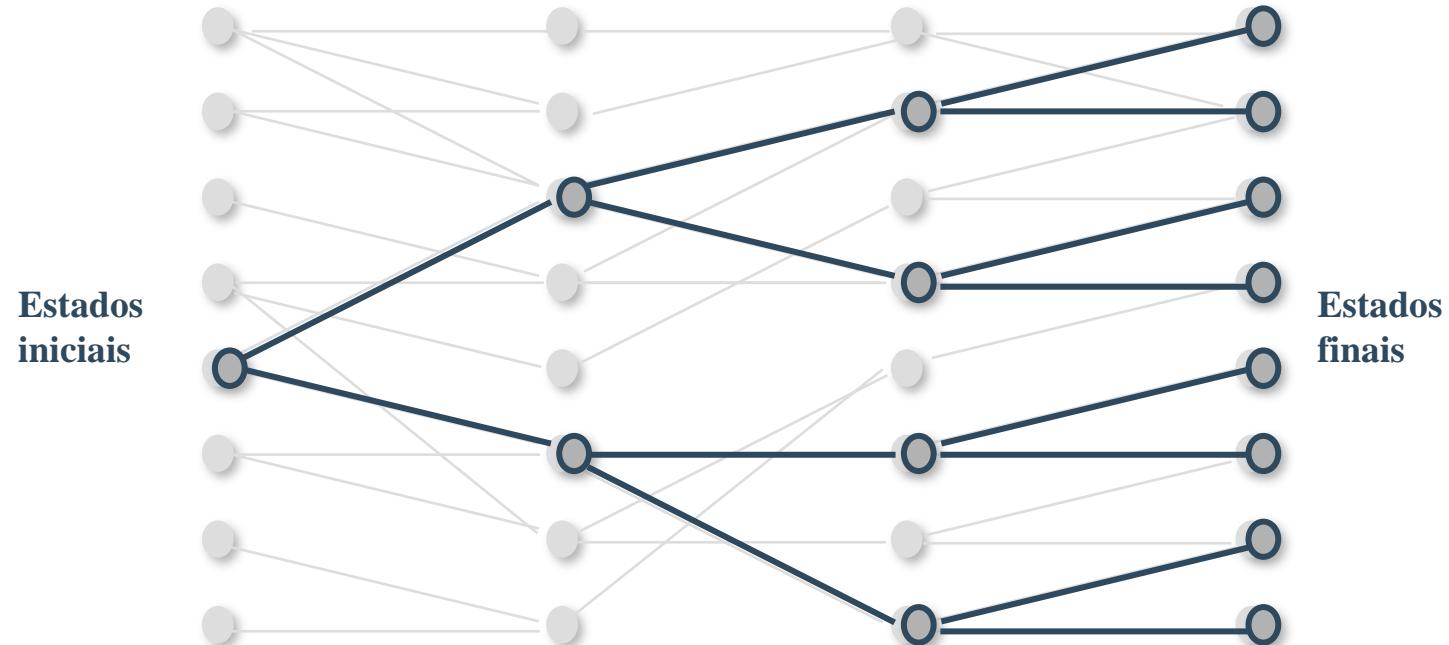
Solução ótima !!!

Tempo no pior caso é proporcional para largura, altura e aprofundamento iterativo

ID: Repete mais os nós nas camadas mais próximas da raiz (exponencialmente menos nós que camadas de baixo)

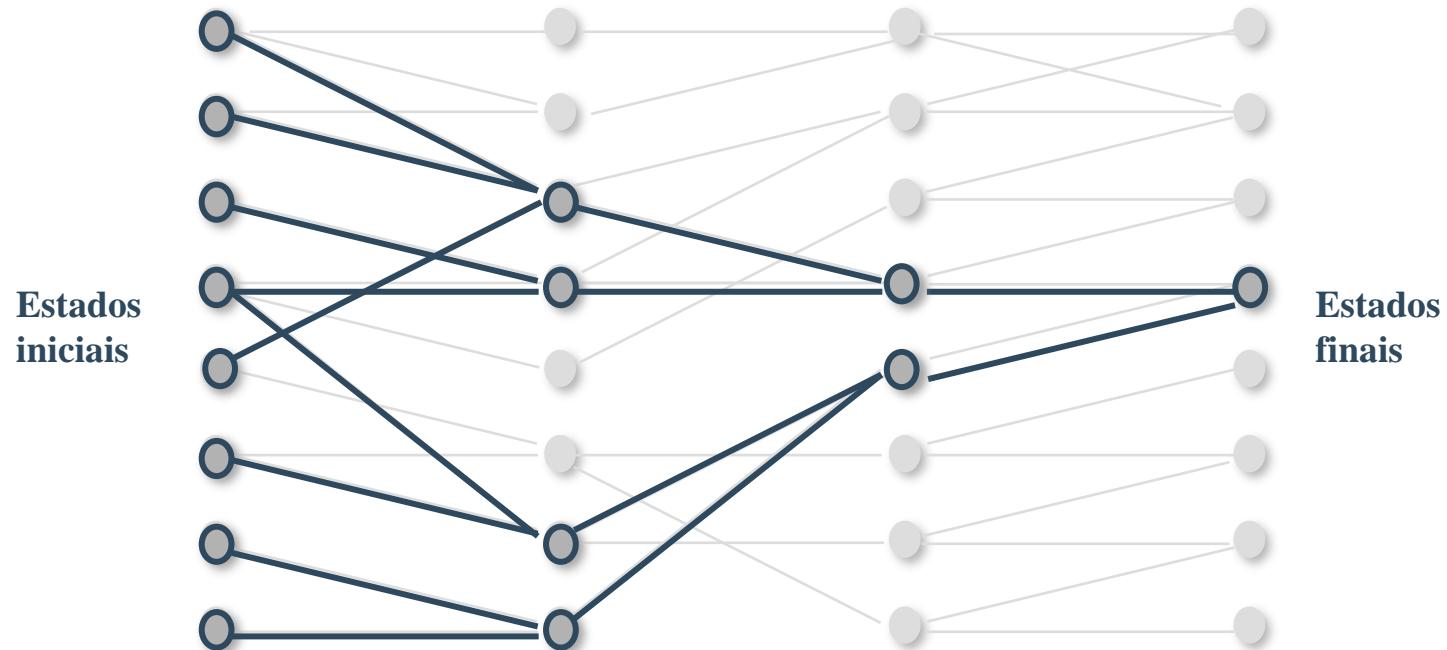
# Busca cega (ou exaustiva)

- Direção da busca
  - Encadeamento progressivo (*Forward chaining*)
    - Para frente, a partir do estado inicial
    - A busca é interrompida quando um estado final é encontrado



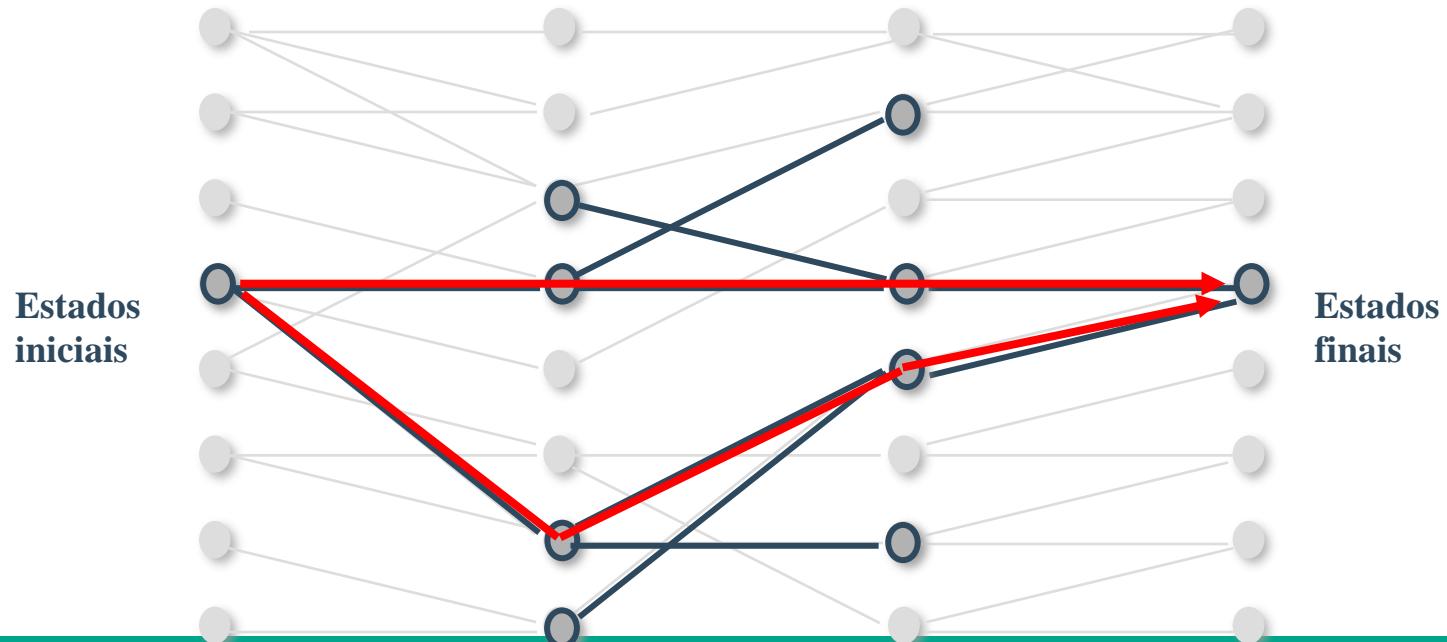
# Busca cega (ou exaustiva)

- Direção da busca
  - Encadeamento regressivo (**Backward chaining**)
    - Para trás, a partir do estado final desejado
    - A busca é interrompida quando o estado inicial é encontrado



# Busca cega (ou exaustiva)

- Direção da busca
  - Busca bidirecional
    - Para frente, a partir do nó inicial, e para trás, a partir do nó final
    - A busca é interrompida quando os dois processos geram um mesmo estado intermediário



# Busca cega (ou exaustiva)

- Direção da busca
  - Busca bidirecional
    - É possível utilizar estratégias diferentes (largura, profundidade, etc.) em cada direção da busca
    - Busca para trás
      - gera *predecessores* do nó final
      - Se os operadores são reversíveis: conjunto de predecessores do nó = conjunto de sucessores do nó
  - Custo:  $O(b^{d/2})$ 
    - Se o fator de expansão dos nós é  $b$  nas duas direções e a profundidade do último nó gerado é  $d$ , cada processo busca até metade da profundidade do nó

# Busca cega (ou exaustiva)

- **Evitando estados repetidos**
  - **Problema geral em busca:**

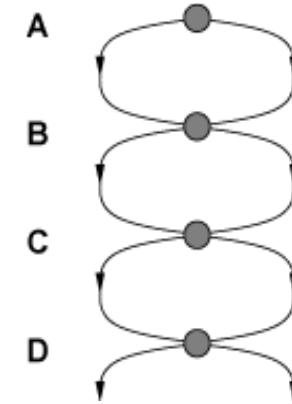
Expandir estados presentes em caminhos já explorados
  - É inevitável quando existe operadores reversíveis

A árvore de busca é potencialmente infinita

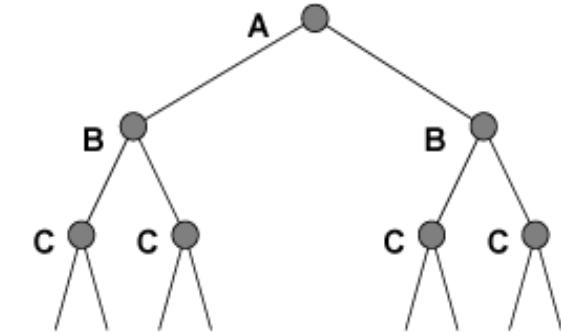
**Exemplo:** encontrar rotas, canibais e missionários
  - **Idéia:**
    - **podar (“prune”)** estados repetidos, para gerar apenas a parte da árvore que corresponde ao grafo do espaço de estados (que é finito!)
    - mesmo quando esta árvore é finita ... evitar estados repetidos pode reduzir exponencialmente o custo da busca

# Busca cega (ou exaustiva)

- Evitando estados repetidos
  - Exemplo:



*Espaço de estados*



*Árvore de busca*

$(m + 1)$  estados no espaço  $\Rightarrow 2^m$  caminhos na árvore

- Questão:

Como evitar expandir estados presentes em caminhos já explorados sem aumentar de forma considerável o custo computacional?

# Busca cega (ou exaustiva)

- **Evitando estados repetidos**
  - **Soluções:**
    - Não retornar ao estado “pai”
      - Função que rejeita geração de sucessor igual ao pai
    - Não criar caminhos com ciclos
      - Não gerar sucessores para qualquer estado que já apareceu no caminho sendo expandido
    - Não gerar qualquer estado que já tenha sido criado antes (em qualquer ramo)
      - Requer que todos os estados gerados permaneçam na memória
      - **Custo de memória:**  $O(b^d)$
      - Pode ser implementado mais eficientemente com tabelas *hash*

# Busca cega (ou exaustiva)

- Considerações finais
  - Comparação entre os métodos:

Critério	Largura	Custo Uniforme	Profundidade	Profundidade Limitada	Aprofundamento Iterativo	Bidirecional
Tempo	$b^d$	$b^d$	$b^m$	$b^l$	$b^d$	$b^{d/2}$
Espaço	$b^d$	$b^d$	$bm$	$bl$	$bd$	$b^{d/2}$
Otima?	Sim	Sim	Não	Não	Sim	Sim
Completa?	Sim	Sim	Não	Sim, se $l \geq d$	Sim	Sim

*Onde*

$b$  = fator de expansão do problema

$d$  = nível da primeira solução para o problema

$l$  = limite de profundidade

$m$  = profundidade máxima

# Busca cega (ou exaustiva)

- Considerações finais

- Problema:

- Custo de armazenamento e verificação × Custo extra de busca

- Solução:

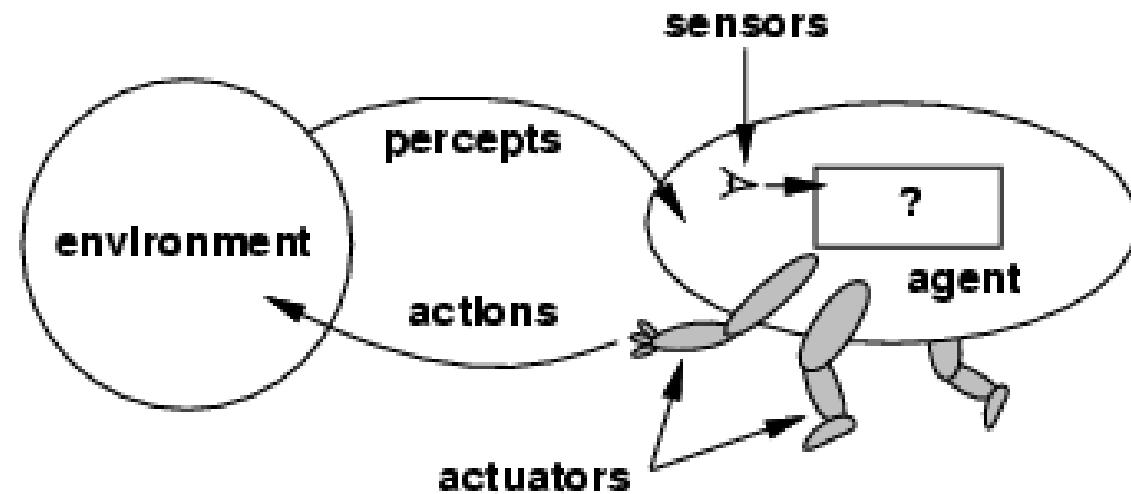
- Depende do problema
    - Quanto mais “loops”, mais vantagem em evitá-los!

# Agente

# Agentes

Um **agente** é algo capaz de perceber seu **ambiente** por meio de **sensores** e de agir sobre esse ambiente por meio de **atuadores**.

- Agente humano
  - Sensores: Olhos, ouvidos e outros órgãos.
  - Atuadores: Mão, pernas, boca e outras partes do corpo.
- Agente robótico
  - Sensores: câmeras e detectores de infravermelho.
  - Atuadores: vários motores.
- Agente de software
  - Sensores: entrada do teclado, conteúdo de arquivos e pacotes vindos da rede.
  - Atuadores: tela, disco, envio de pacotes pela rede.



# Mapeando percepções em ações

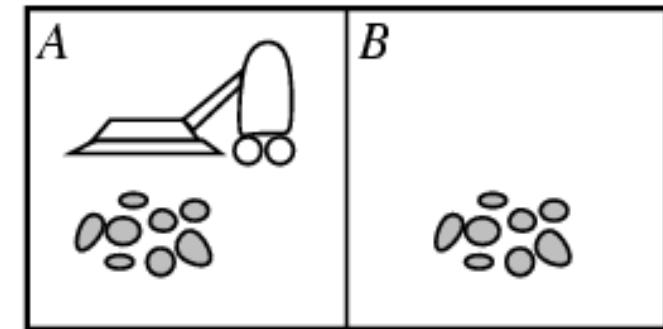
- Sequência de percepções: história completa de tudo que o agente percebeu.
- O comportamento do agente é dado abstratamente pela **função do agente**:

$$[f: \mathcal{P}^* \rightarrow \mathcal{A}]$$

onde é a  $\mathcal{P}^*$  é uma sequência de percepções e  $\mathcal{A}$  é uma ação.

- O **programa do agente** roda em uma arquitetura física para produzir  $f$ .
- Agente = arquitetura + programa.

Exemplo:  
O mundo do aspirador de pó



- Percepções: local e conteúdo
  - Exemplo: [A, sujo]
- Ações: Esquerda, Direita, Aspirar, NoOp

# Agentes Racionais (1)

PEAS

- Como preencher corretamente a tabela de ações do agente para cada situação?
- O agente deve tomar a ação “correta” baseado no que ele percebe para ter sucesso.
  - O conceito de sucesso do agente depende uma **medida de desempenho** objetiva.
    - Exemplos: quantidade de sujeira aspirada, gasto de energia, gasto de tempo, quantidade de barulho gerado, etc.
    - A medida de desempenho deve refletir o resultado realmente desejado.

Ao projetar um agente, a primeira etapa deve ser sempre especificar o ambiente de tarefa.

- Performance = Medida de Desempenho
- Environment = Ambiente
- Actuators = Atuadores
- Sensors = Sensores

# Agentes Racionais (2)

- Agente racional: para cada sequência de percepções possíveis deve selecionar uma ação que se espera venha a maximizar sua medida de desempenho, dada a evidência fornecida pela seqüência de percepções e por qualquer conhecimento interno do agente.
  - Exercício: para que medida de desempenho o agente aspirador de pó é racional?

# Agentes Racionais (3)

- Racionalidade é diferente de perfeição.
  - A racionalidade maximiza o desempenho esperado, enquanto a perfeição maximiza o desempenho real.
  - A escolha racional só depende das percepções até o momento.
- Mas os agentes podem (e devem!) executar ações para coleta de informações.
  - Um tipo importante de coleta de informação é a exploração de um ambiente desconhecido.
- O agente também pode (e deve!) aprender, ou seja, modificar seu comportamento dependendo do que ele percebe ao longo do tempo.
  - Nesse caso o agente é chamado de autônomo.
  - Um agente que aprende pode ter sucesso em uma ampla variedade de ambientes.

# Agentes Racionais (3)

- Racionalidade é diferente de perfeição.
  - A racionalidade maximiza o desempenho esperado, enquanto a perfeição maximiza o desempenho real.
  - A escolha racional só depende das percepções até o momento.
- Mas os agentes podem (e devem!) executar ações para coleta de informações.
  - Um tipo importante de coleta de informação é a exploração de um ambiente desconhecido.
- O agente também pode (e deve!) aprender, ou seja, modificar seu comportamento dependendo do que ele percebe ao longo do tempo.
  - Nesse caso o agente é chamado de autônomo.
  - Um agente que aprende pode ter sucesso em uma ampla variedade de ambientes.