

CEFET/RJ

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

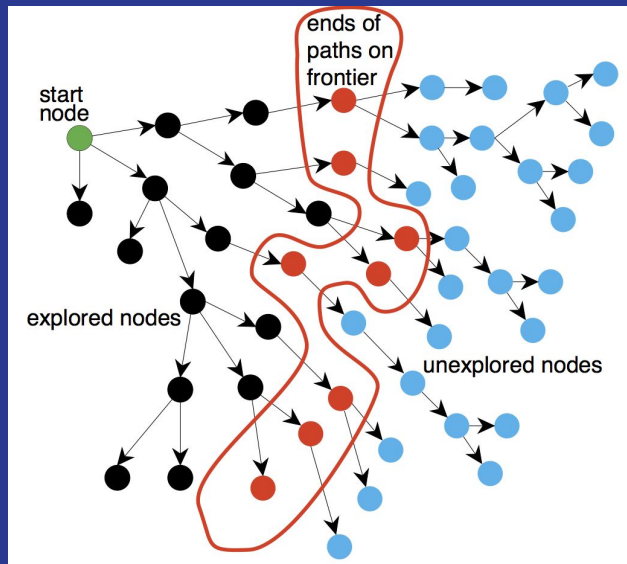
**GCC1734 - INTELIGÊNCIA ARTIFICIAL**

Eduardo Bezerra (CEFET/RJ)  
ebezerra@cefet-rj.br

# Créditos

- Essa apresentação é uma tradução e/ou adaptação feita pelo prof. Eduardo Bezerra (**ebezerra@cefet-rj.br**) do material cuja autoria é dos professores Dan Klein e Pieter Abbeel (UC Berkeley).
- O material original é usado no curso CS188 (Introduction to Artificial Intelligence).
  - <https://inst.eecs.berkeley.edu/~cs188>

# AGENTES E PROBLEMAS DE BUSCA



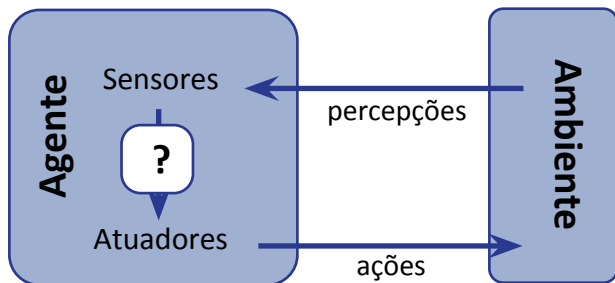
# Visão Geral

- Agentes
  - Tipos de agentes
  - Agentes planejadores
- Problemas de busca
- Estratégias de busca (conceito)

# Agentes

# Agentes

- Um **agente** é uma entidade que percebe e age em um ambiente.
- O **ambiente** é a parte do sistema que o agente não controla.
- Um agente interage com o ambiente por meio de seus sensores e atuadores.



# Agentes

- Agentes incluem humanos, robôs, softbots, termostatos, ...
- Agente humano:
  - Sensores: olhos, ouvidos, ...
  - Atuadores: mãos , pés, boca, ...
- Agente robô:
  - Sensores: câmeras e localizadores faixa do infravermelho
  - Atuadores: vários motores

# Agentes racionais

- Iremos usar o termo **racional** em um sentido específico:
  - Racional: diz-se daquele agente que maximamente alcança objetivos pré-definidos.
  - Racionalidade apenas diz respeito às decisões tomadas pelo agente (e não ao processo de raciocínio subjacente).
  - Objetivos são expressos em termos da **utilidade** dos resultados.
- Um **agente racional** é aquele que seleciona ações que maximizam a sua **função utilidade** (esperada).



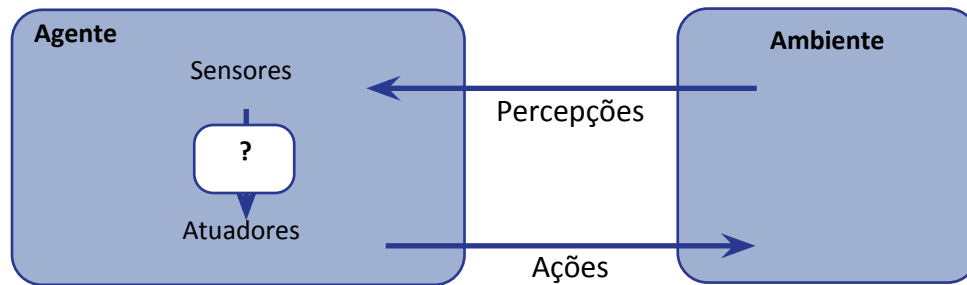
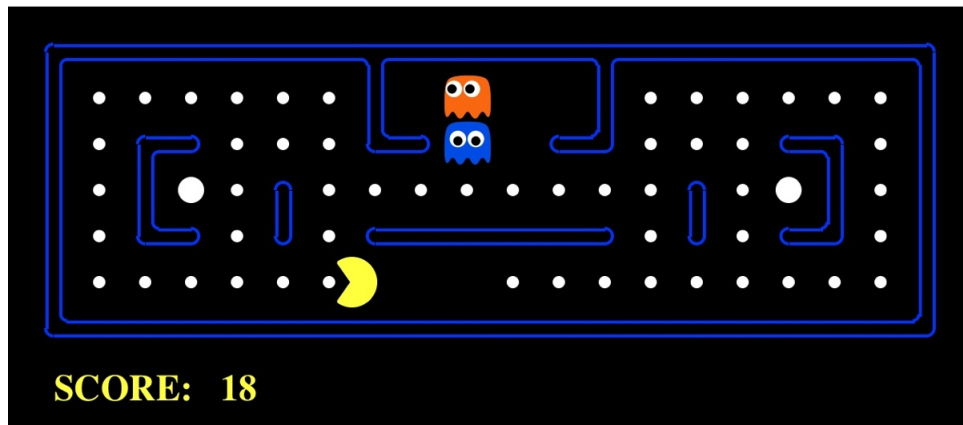
# Função utilidade

O objetivo de um agente racional é maximizar a **utilidade esperada** obtida.

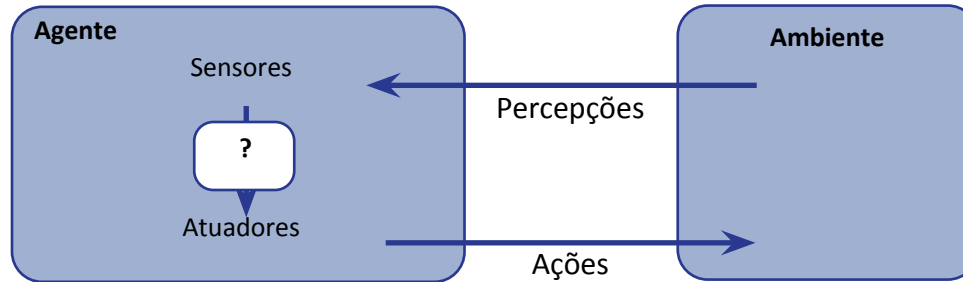


Uma **utilidade** é uma função que descreve os objetivos de um agente

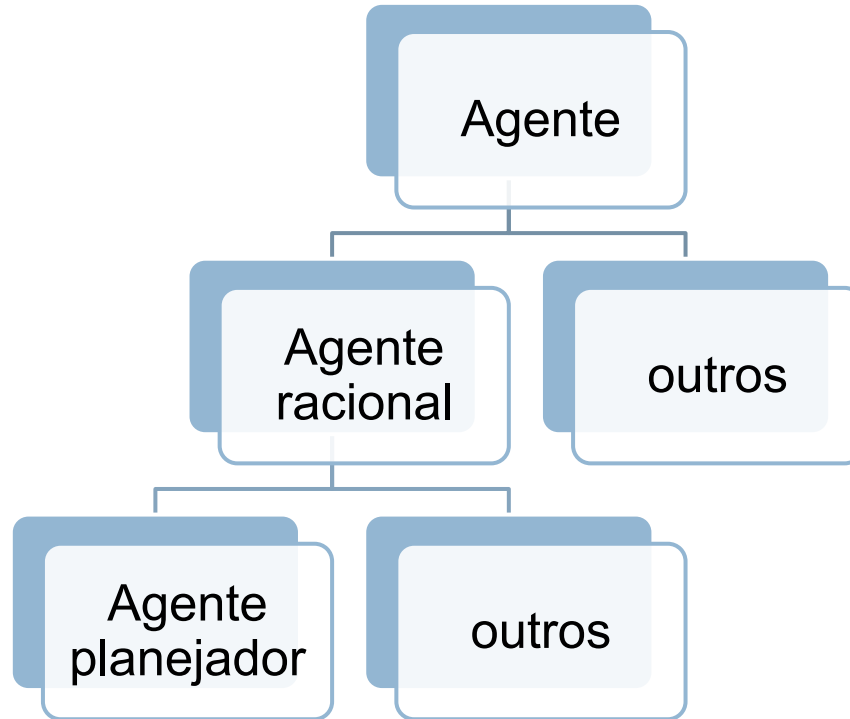
# Pac-Man como um agente racional



# Carro autônomo como um agente racional

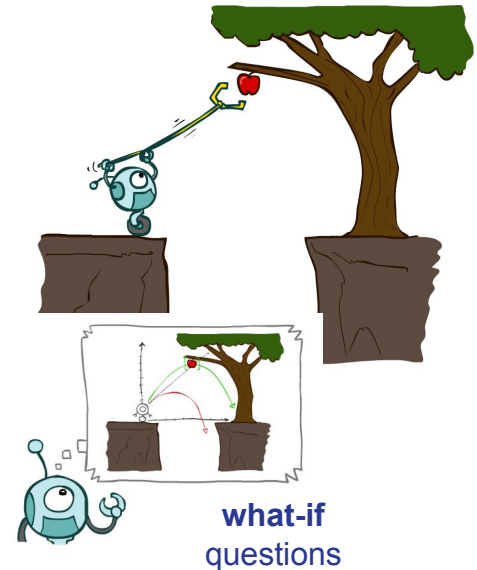


# Agentes planejadores (*planning agents*)



# Agentes planejadores (*planning agents*)

- Um **agente planejador**:
  - Toma decisões com base nas consequências de suas ações.
  - Deve obrigatoriamente ter um modelo de como o mundo evolui em resposta a suas ações.
  - Deve obrigatoriamente formular um **objetivo**.
- planejamento completo *versus* ótimo
  - Alguma solução versus a melhor solução

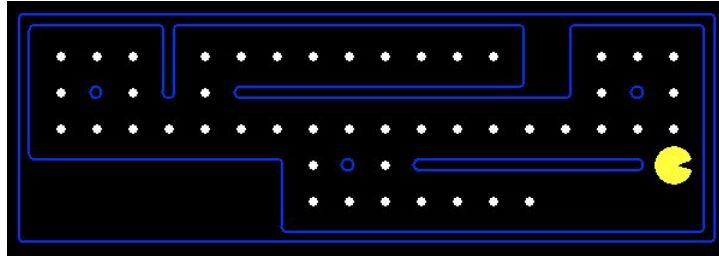


# Agentes planejadores (*planning agents*)

- Diante de um objetivo a ser alcançado (i.e., de um problema a ser resolvido), um agente planejador deve inicialmente encontrar (buscar) um plano para atingir seu objetivo (i.e., resolver o problema).
- Um plano é uma sequência de ações que o agente deve executar para que ele alcance seu objetivo.

# Agentes planejadores - exemplo

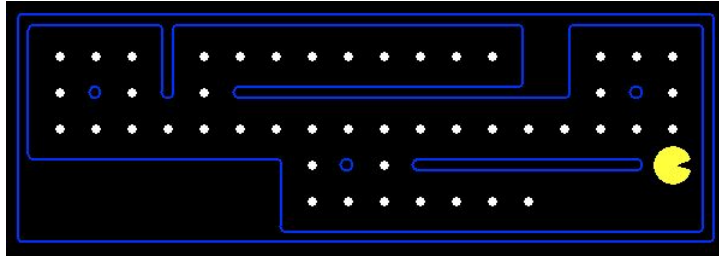
- Objetivo: capturar todas as pílulas em tempo mínimo possível.
- Estado inicial:



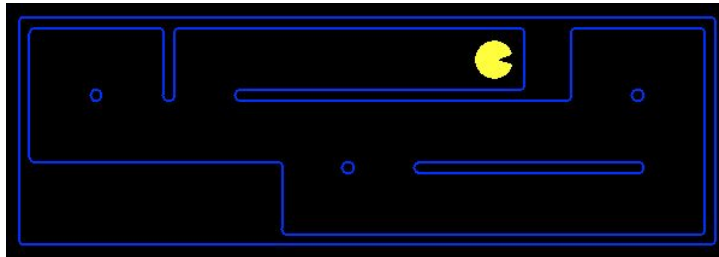
# Agentes planejadores - exemplo

- Objetivo: capturar todas as pílulas em tempo mínimo possível.

- Estado inicial:



- Estado objetivo:

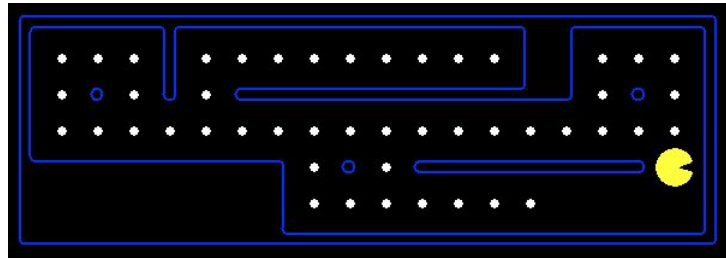




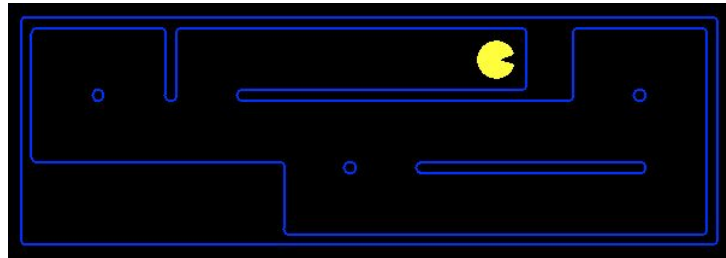
# Agentes planejadores - exemplo

- Objetivo: capturar todas as pílulas em tempo mínimo possível.

- Estado inicial:

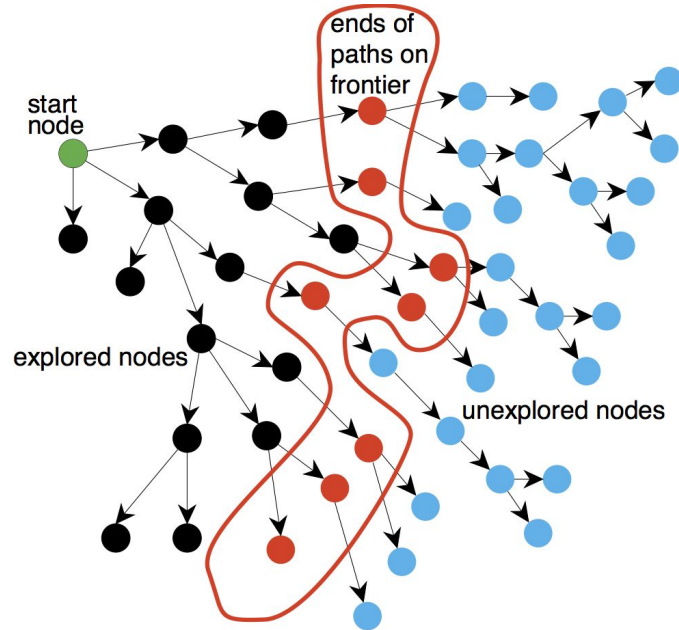


- Estado objetivo:



plano

# Problemas de Busca (*Search Problems*)



# Problema de Busca

- Definição (problema de busca). Um tipo de problema em que um agente precisa encontrar uma sequência de ações (plano) que leva de um estado inicial a um estado objetivo.
- Esse plano é então usado para atingir um determinado objetivo ou solução.

# Problema de Busca

- Para executar alguma tarefa, um agente planejador pode decidir o que fazer comparando diferentes sequências de ações possíveis.
  - para depois escolher a melhor sequência de ações para executar.
- Resolver um **problema de busca** (*search problem*) corresponde ao processo de procurar pela **melhor** sequência (de ações).
  - Nesse processo, o agente explora diferentes estados e ações possíveis para encontrar o plano que leva à solução desejada.

# Problema de Busca

- Resolver um problema de busca envolve 3 passos:
  - formular problema
  - buscar solução (em um espaço de possíveis soluções)
  - executar solução selecionada
- formular problema → buscar solução → executar solução

# Formulação de um Problema de Busca

- Um problema de busca é formulado pela definição de cinco componentes:

# Formulação de um Problema de Busca

- Um problema de busca é formulado pela definição de cinco componentes:
  1. Conjunto  $S$  (**espaço de estados**), com um **estado inicial**,

# Formulação de um Problema de Busca

- Um problema de busca é formulado pela definição de cinco componentes:
  1. Conjunto  $S$  (**espaço de estados**), com um **estado inicial**,
  2. Função  $ACTIONS(s)$ : produz as **ações** possíveis em cada estado,



# Formulação de um Problema de Busca

- Um problema de busca é formulado pela definição de cinco componentes:
  1. Conjunto  $S$  (**espaço de estados**), com um **estado inicial**,
  2. Função  $ACTIONS(s)$ : produz as **ações** possíveis em cada estado,
  3. Função  $RESULT(s, a)$ : **modelo de transição** ou **função sucessora**, produz o estado resultante de selecionar a ação  $a$  no estado  $s$ .

# Formulação de um Problema de Busca

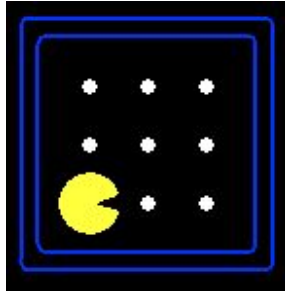
- Um problema de busca é formulado pela definição de cinco componentes:
  1. Conjunto  $S$  (**espaço de estados**), com um **estado inicial**,
  2. Função  $ACTIONS(s)$ : produz as **ações** possíveis em cada estado,
  3. Função  $RESULT(s, a)$ : **modelo de transição** ou **função sucessora**, produz o estado resultante de selecionar a ação  $a$  no estado  $s$ .
  4. Função **teste de objetivo**, que permite ao agente determinar se seu objetivo foi alcançado.

# Formulação de um Problema de Busca

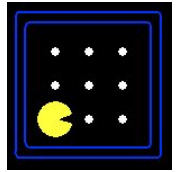
- Um problema de busca é formulado pela definição de cinco componentes:
  1. Conjunto  $S$  (**espaço de estados**), com um **estado inicial**,
  2. Função  $ACTIONS(s)$ : produz as **ações** possíveis em cada estado,
  3. Função  $RESULT(s, a)$ : **modelo de transição** ou **função sucessora**, produz o estado resultante de selecionar a ação  $a$  no estado  $s$ .
  4. Função **teste de objetivo**, que permite ao agente determinar se seu objetivo foi alcançado.
  5. Função **custo de caminho** (função aditiva e cumulativa), que permite ao agente comparar planos alternativos.

# Exemplo: PacMan “simplificado”

- Objetivo: “coletar todas as pílulas em tempo mínimo possível.”

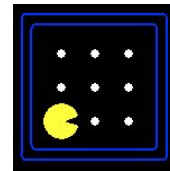


# Exemplo: PacMan “simplificado”

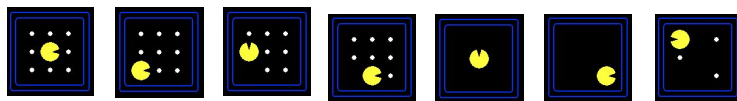


- Espaço de estados?

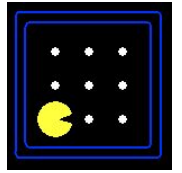
# Exemplo: PacMan “simplificado”



- Espaço de estados?
  - Estrutura de dados: *[posição agente, dot booleans]*
- Você consegue calcular qual o tamanho desse espaço de estados?

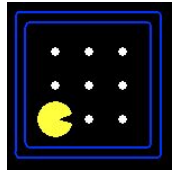


# Exemplo: PacMan “simplificado”



- Ações?

# Exemplo: PacMan “simplificado”

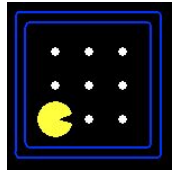


- Ações?

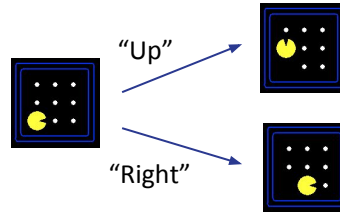
- ▣  $ACTIONS(s) \subset \{Up, Down, Left, Right\}$





# Exemplo: PacMan “simplificado”

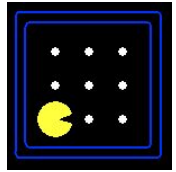


- Modelo de transições?

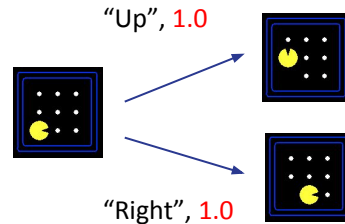


RESULT(  , "Up" ) = 

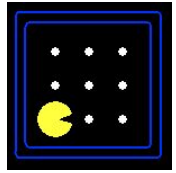
# Exemplo: PacMan “simplificado”



- Custo de caminho (*path cost*)?

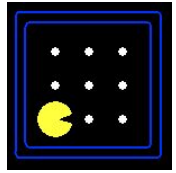


# Exemplo: PacMan “simplificado”




- Teste de objetivo (*goal test*)?

# Exemplo: PacMan “simplificado”

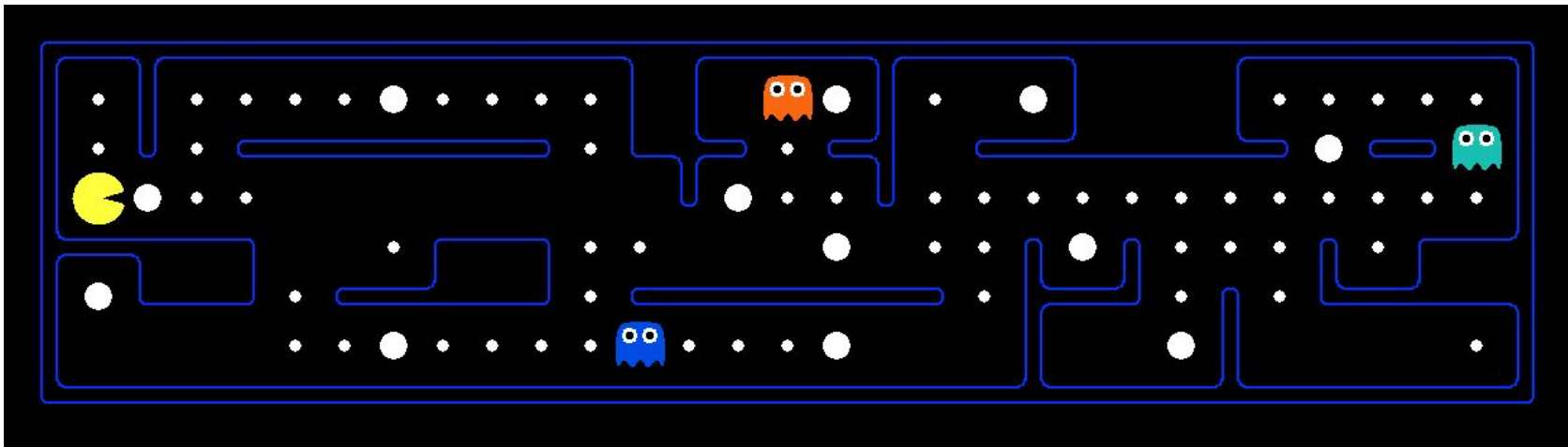


- Teste de objetivo (*goal test*)?
  - Teste de objetivo: “todas as pílulas foram coletadas?”.

teste(  ) retorna False

teste(  ) retorna True

# Quiz: Travessia Segura

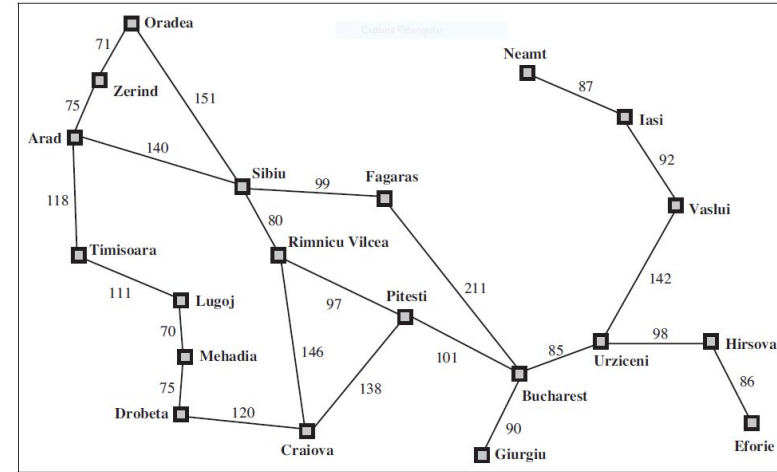


- Problema: comer todas as pílulas e vitaminas e, ao mesmo tempo, manter os fantasmas permanentemente “assustados”.
- Qual uma possível especificação do espaço de estados?
  - (posição agente, dot booleans, power pellet booleans, remaining scared time)

# Exemplo: Romênia

- De férias na Romênia; atualmente em Arad.
- Voo sai amanhã de Bucareste.
- Formular problema:
  - espaço de estados: cidades
  - estado inicial: In(Arad)
  - modelo de transições: permite dirigir entre cidades vizinhas;
  - Teste de objetivo: estar em Bucareste
  - Função custo = distância.
- Solução: uma sequência de cidades.
  - ex., Arad → Sibiu → Fagaras → Bucareste.

Figura 3.2 (AIMA3ed)



# Outros exemplos

# Exemplo 1: jarros

- Dados:
  - uma fonte de água,
  - dois jarros de capacidades 3L e 4L (ambos inicialmente vazios).
- Problema: como preencher 2L de água no jarro de capacidade 4L?





# Exemplo 1: jarros

- Espaço de estados?

## Exemplo 1: jarros

- Espaço de estados: cada elemento desse conjunto pode ser representado (modelado) como um par ordenado.

## Exemplo 2: jarros

- Espaço de estados: cada elemento desse conjunto pode ser representado (modelado) como um par ordenado.

(0,0)

(4,0)

(0,3)

(4,3)

...



## Exemplo 2: jarros

- Ações?

# Exemplo 1: jarros

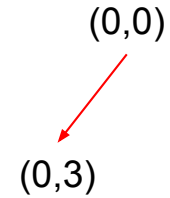
- Ações:
  - Encher J3
  - Encher J4
  - Despejar J3 em J4
  - Despejar J4 em J3
  - Esvaziar J3
  - Esvaziar J4

# Exemplo 1: jarros

- Transições?

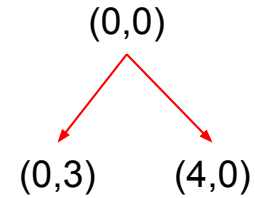
# Exemplo 1: jarros

- Transições:
  - No estado inicial, encher J3



# Exemplo 1: jarros

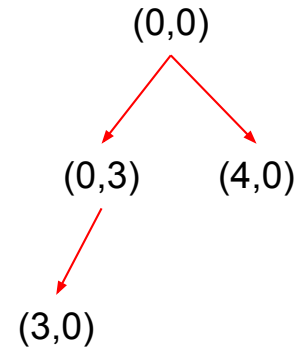
- Transições:
  - No estado inicial, encher J3
  - No estado inicial, encher J4



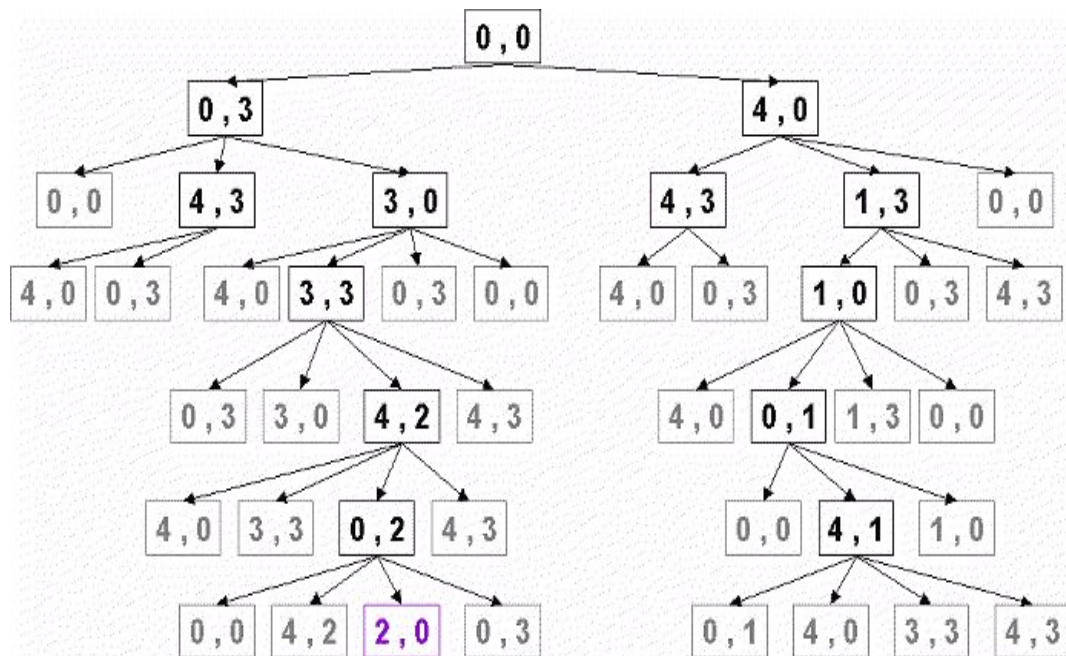


# Exemplo 1: jarros

- Transições:
  - No estado inicial, encher J3
  - No estado inicial, encher J4
  - No estado (0,3), despejar J3 em J4
  - Várias outras...

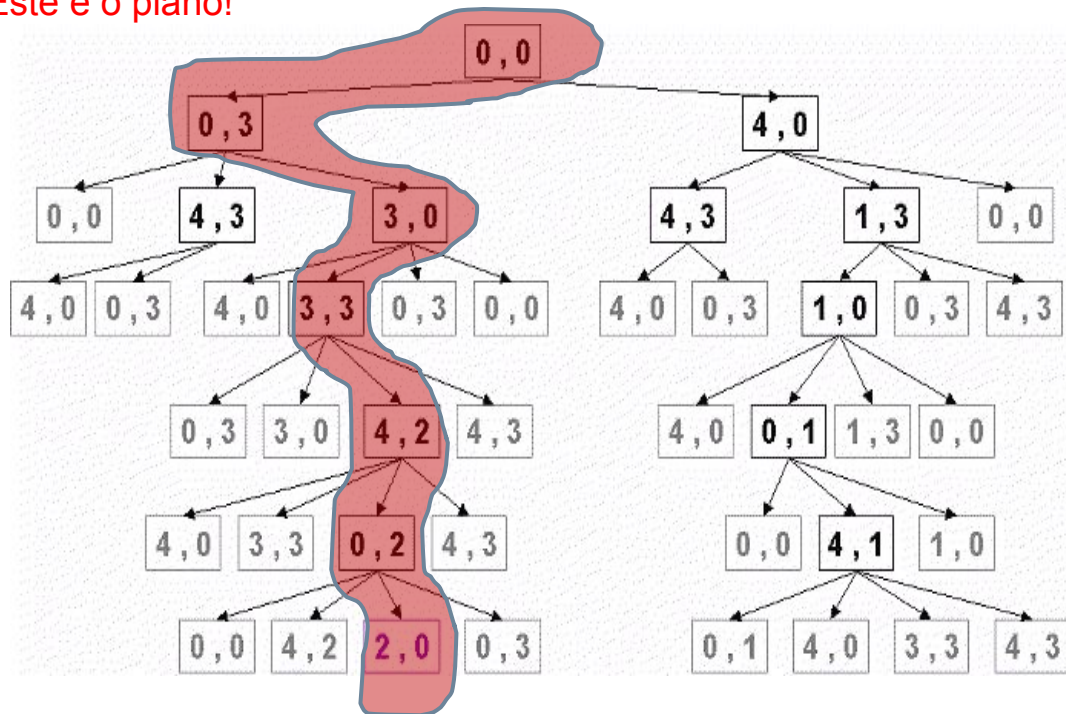


# Exemplo 1: jarros

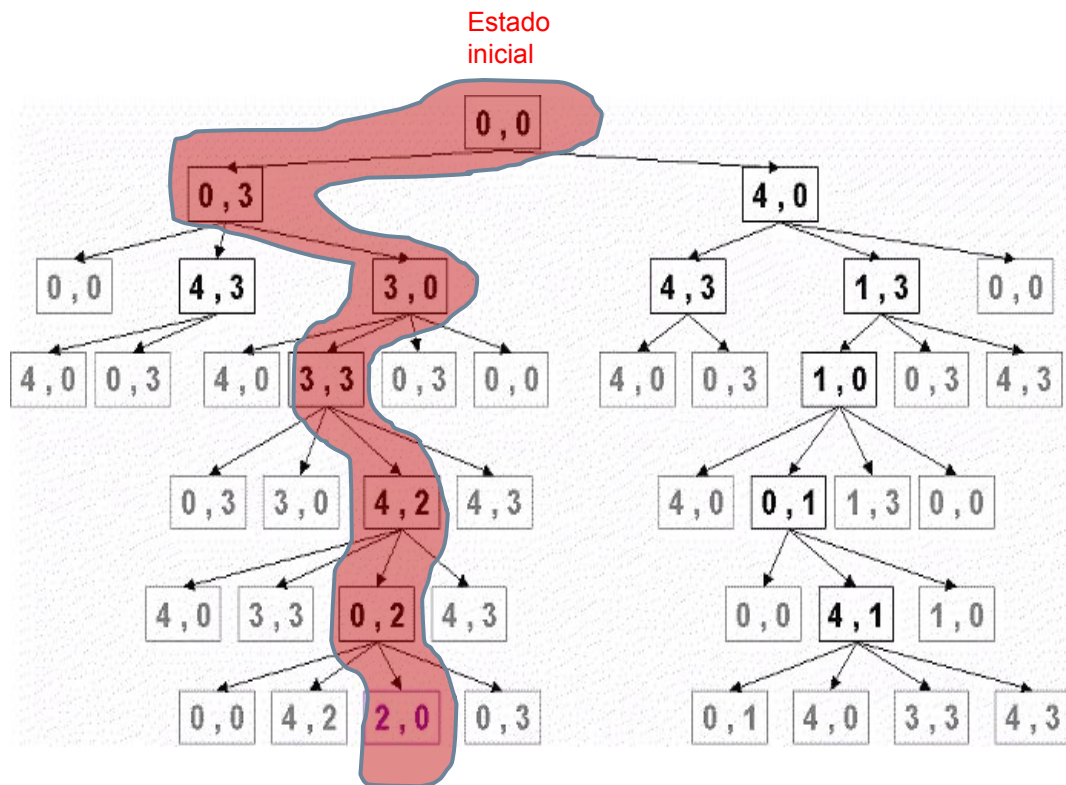


# Exemplo 1: jarros

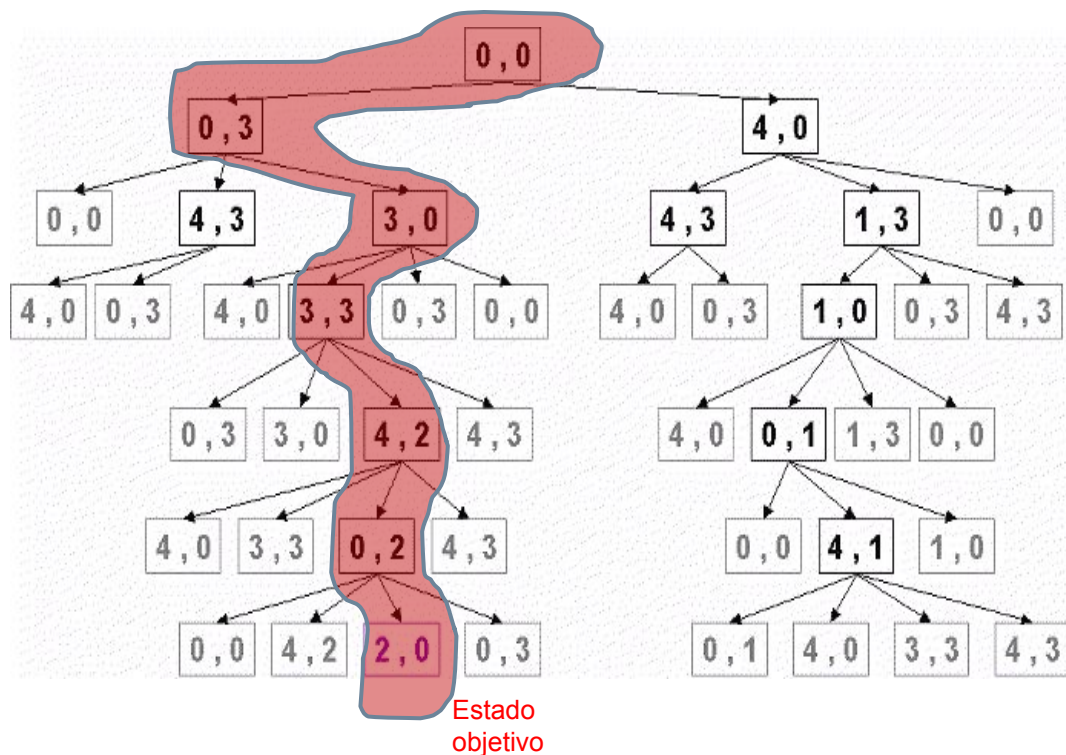
Este é o plano!



# Exemplo 1: jarros



# Exemplo 1: jarros



## Exemplo 2: Mundo do aspirador de pó

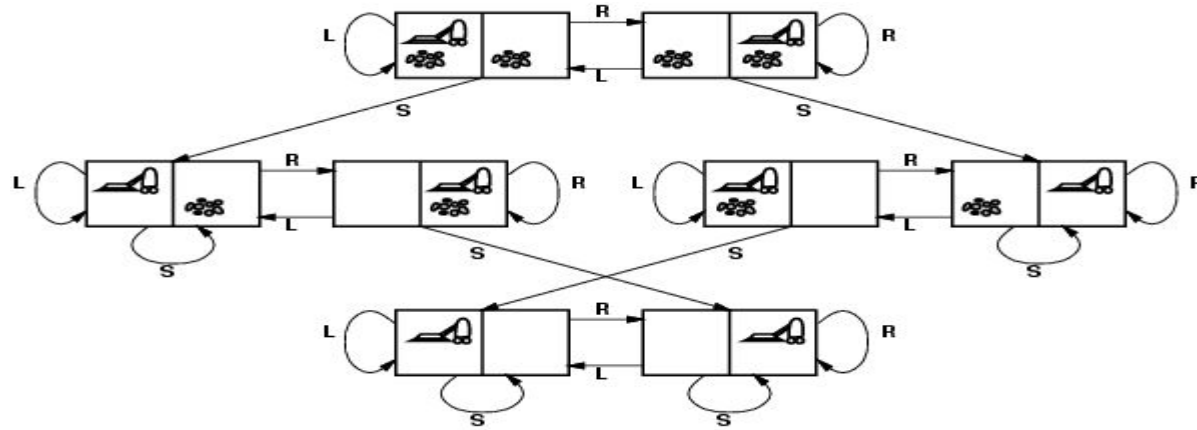


Figura 3.3  
(AIMA3ed)

1. **Espaço de estados:** elementos definidos pela posição do robô e sujeira (8 estados). **Estado inicial:** pode ser qualquer um.
2. **Conjunto de ações:** {Suck, Left, Right}.  $ACTIONS(s1) = \{R, L, S\}$
3. **Modelo de transição:** pode-se executar qualquer uma das ações em cada estado (S, L, ou R)
4. **Teste de objetivo:** verifica se todos os quadrados estão limpos
5. **Custo de caminho:** cada passo (ação) tem custo 1, e assim o custo do caminho é o número de passos do caminho.

## Exemplo 2: Quebra-cabeça de 8 peças

7	2	4
5		6
8	3	1

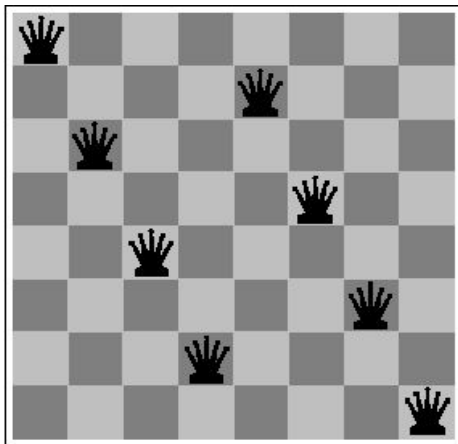
Start State

	1	2
3	4	5
6	7	8

Goal State

- **Espaço de estados:** cada elemento especifica a posição de cada uma das peças e do espaço vazio
- **Estado inicial:** Qualquer um
- **Modelo de transição:** gera os estados válidos que resultam da tentativa de executar as quatro ações (mover espaço vazio para esquerda, direita, acima ou abaixo)
- **Teste de objetivo:** Verifica se o estado corrente corresponde à configuração objetivo.
- **Custo do caminho:** Cada passo custa 1, e assim o custo do caminho é o número de passos do caminho

# Exemplo 4: Oito Rainhas

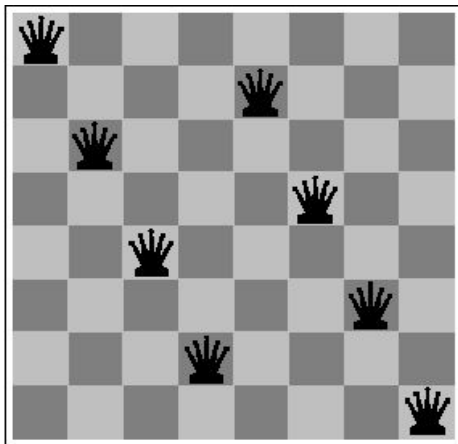


Tamanho do espaço de estados:  
 $64 \times 63 \times \dots \times 57 \approx 1,8 \times 10^{14}$  sequências para investigar

- *Formulação incremental*
  - **Estados:** qualquer disposição de 0 a 8 rainhas no tabuleiro é um estado
  - **Estado inicial:** arranjo com nenhuma rainha
  - **Modelo de transição:** adicionar uma rainha em qualquer quadrado desocupado
  - **Teste de objetivo:** estado contém 8 rainhas no tabuleiro, nenhuma atacada
  - **Custo do caminho?**



## Exemplo 4: Oito Rainhas



Tamanho do espaço de estados: 2.057

- *Formulação de estados completos*
  - **Estados:** disposições de  $n$  rainhas, uma por coluna, nas  $n$  colunas mais à esquerda sem que nenhuma rainha ataque outra
  - **Modelo de transição:** adicionar uma rainha a qualquer quadrado na coluna vazia mais à esquerda, de tal modo que ela não seja atacada

## E no mundo real?

- Todos os exemplos apresentados até aqui são problemas de brinquedo (*toy problems*)!
- Há aplicações no mundo real?

# E no mundo real?

- Todos os exemplos apresentados até aqui são problemas de brinquedo (*toy problems*)!
- Há aplicações no mundo real!
  - Problemas de roteamento (encontrar a melhor rota de um ponto a outro): redes de computadores, planejamento militar, planejamento de viagens aéreas
  - Problemas de tour: visitar cada ponto pelo menos uma vez
  - Problema do caixeiro viajante: visitar cada cidade exatamente uma vez, encontrar o caminho mais curto.

# Problemas do mundo real

## 2ND CALL FOR PARTICIPATION: KDD-BR 2021

News (1): top three teams will get a free registration to present their solutions at BRACIS 2021. News (2): top three teams will be invited to submit a short paper to ENIAC 2021 describing their solutions.

5th KDD-BR (Brazilian Knowledge Discovery in Databases) competition: AI-based approaches to predict solutions of the Travelling Salesman Problem

Kaggle Site: <https://www.kaggle.com/c/kddbr-2021>

KDD-BR 2021 Site: <http://c4ai.inova.usp.br/bracis/kdd.htm>



# Espaço de estados (*State space*)

- O conjunto de todos os estados acessíveis a partir de um estado inicial é chamado de **espaço de estados**.
- Definido implicitamente pelos seguintes componentes:
  - estado inicial,
  - ações e
  - modelo de transições (que fornece os estados acessíveis a partir de cada ação).
- O espaço de estados forma um **grafo direcionado**, no qual os nós são estados e os arcos são ações.

# Selecionando um espaço de estados

- O mundo real é absurdamente complexo!
- Um espaço de estados é uma **abstração**
  - Estado (abstrato) = conjunto de estados reais
  - Ação (abstrata) = combinação complexa de ações reais
    - ex., "Arad  $\rightarrow$  Zerind" representa um conjunto complexo de rotas, desvios, paradas, etc.
    - Qualquer estado real do conjunto "em Arad" deve levar a algum estado real "em Zerind".
  - Solução (abstrata) = conjunto de caminhos reais que são soluções no mundo real
- A abstração escolhida é útil se cada ação abstrata é mais fácil de executar que o problema original.
- Consideração acerca da simplificação demasiada.



# Solução versus Solução Ótima

- Uma **solução** para um problema de busca é uma sequência de ações (plano) que levam do estado inicial para o estado objetivo.
- Uma **solução ótima** é uma solução com o menor custo de caminho dentre todas as possíveis.
- Pode haver mais de uma solução ótima para um dado problema de busca.

# Agente de resolução de problemas

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
               state, some description of the current world state
               goal, a goal, initially null
               problem, a problem formulation

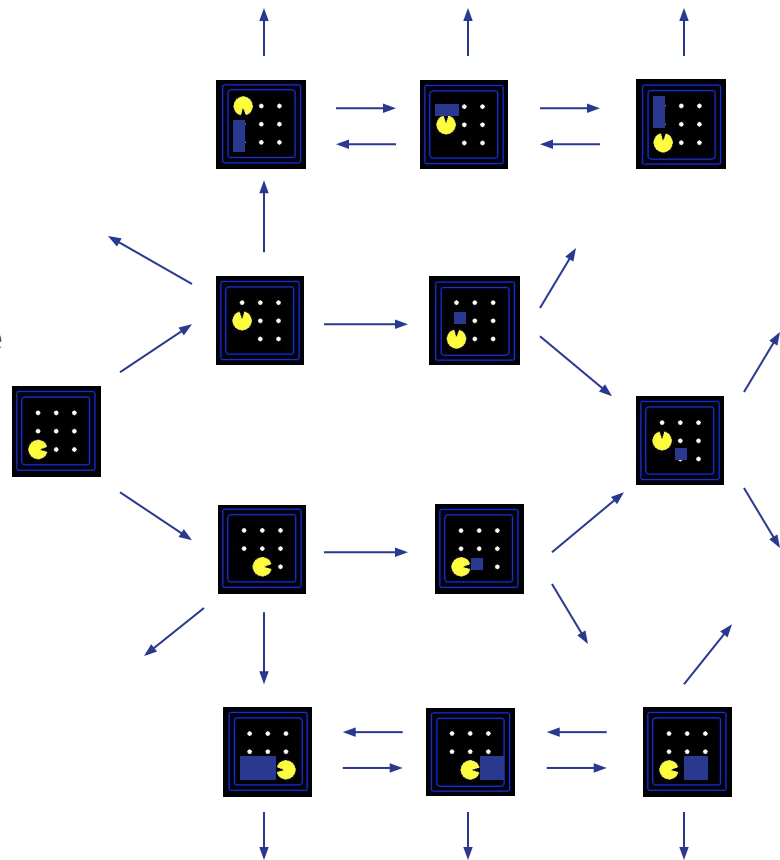
  state  $\leftarrow$  UPDATE-STATE(state, percept)
  if seq is empty then
    goal  $\leftarrow$  FORMULATE-GOAL(state)
    problem  $\leftarrow$  FORMULATE-PROBLEM(state, goal)
    seq  $\leftarrow$  SEARCH(problem)
    if seq = failure then return a null action
  action  $\leftarrow$  FIRST(seq)
  seq  $\leftarrow$  REST(seq)
  return action
```

Esse pseudocódigo supõe que ambiente é estático, observável, discreto e determinístico.



# Grafo de espaço de estados (*state space graph*)

- Grafo que representa um problema de busca.
  - Cada nó é uma (abstração de alguma) configuração do mundo
  - Arestas representam sucessores (resultados de ações)
  - O teste de objetivo (alvo) é satisfeito para um conjunto de nós objetivo (talvez apenas um)
- Em um grafo de espaço de estados, cada estado ocorre apenas uma vez.
- Para problemas práticos, não é possível construir esse grafo em memória.

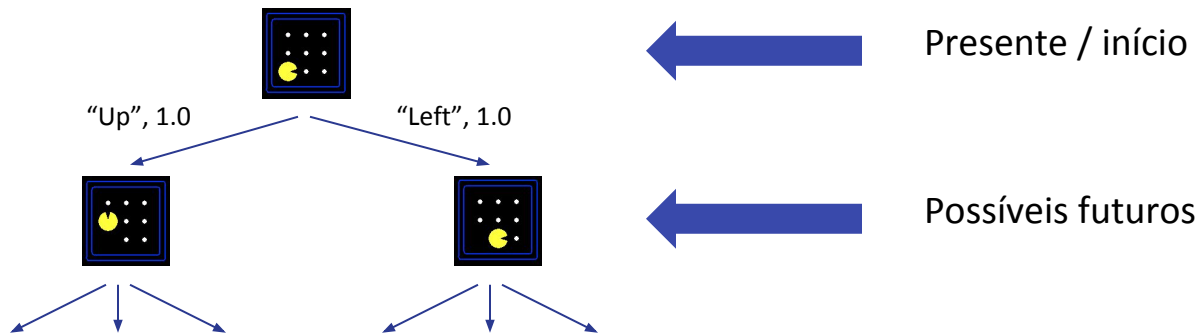


# Árvore de busca (*search tree*)

- Estrutura de dados na qual:
  - Cada nó (vértice) contém um elemento do espaço de estados.
    - Raiz contém o estado inicial.
  - Nós filhos correspondem a sucessores, pelo modelo de transição.
    - Cada aresta representa uma ação;
  - Cada nó corresponde a um **plano** para alcançar aquele estado.
- Para a maioria dos problemas de busca, é impossível construir a árvore inteira.

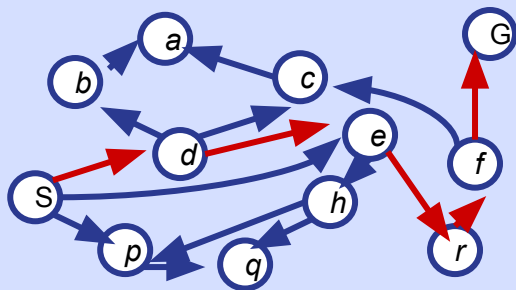
# Árvore de busca (search tree)

- Uma árvore de busca é a estrutura que permite a um agente planejador simular as consequências (possíveis futuros) de suas ações.



# Grafos de Espaço de Estados vs. Árvores de Busca

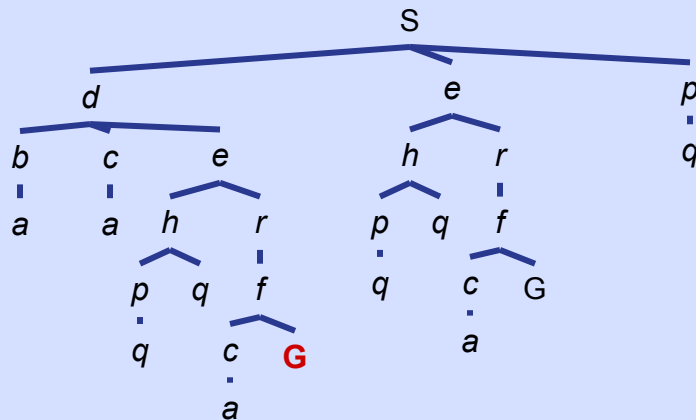
## Grafos de Espaço de Estados



*Cada nó na árvore de busca corresponde a um caminho no grafo de espaço de estados, começando do estado inicial.*

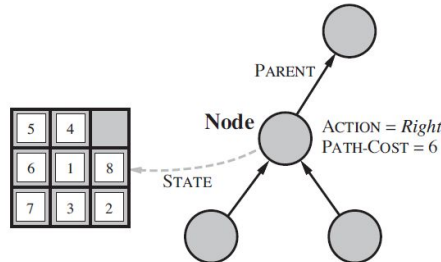
*Ambos são construídos sob demanda – tentamos construir o mínimo possível de cada um.*

## Árvore de Busca



# Estados vs. Nós

- Um **estado** é uma (representação de uma) configuração específica do sistema “agente/ambiente”.
- Um **nó** de uma árvore de busca inclui **estado**, **nó pai**, **ação**, **custo do caminho**.
  - Representado pela **estrutura de dados** denominada **Node**.
  - Note que o campo PARENT permite organizar os nós em uma árvore.



# Criação de um nó

- Dado um nó pai e uma ação, CHILD-NODE cria e retorna um novo nó filho, preenchendo os vários campos desse nó.

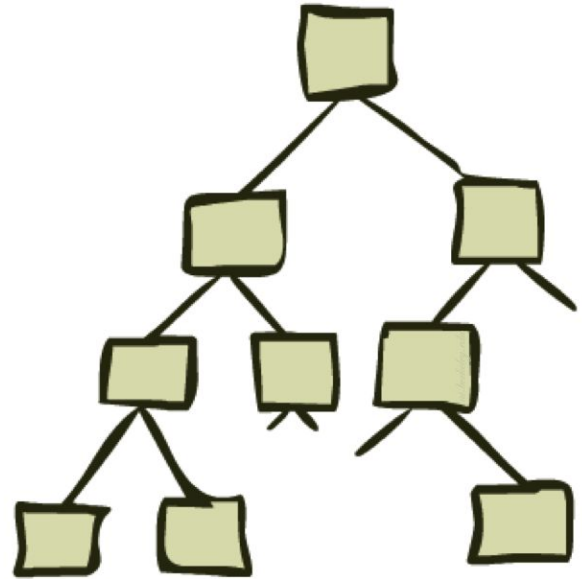
```
function CHILD-NODE(problem, parent, action) returns a node  
  return a node with  
    STATE = problem.RESULT(parent.STATE, action),  
    PARENT = parent, ACTION = action,  
    PATH-COST = parent.PATH-COST + problem.STEP-COST(parent.STATE, action)
```

# Estratégias de Busca

- São algoritmos para encontrar (buscar) um plano, a partir da formulação de um problema de busca.
- Ideia geral: percorrer o espaço de estados com o auxílio de uma **árvore de busca**.
- *Durante a execução do algoritmo, a árvore de busca é expandida.*
  - Expandir : aplicar o modelo de transição (sobre algum nó-folha) para gerar novos nós.

# Estratégias de Busca

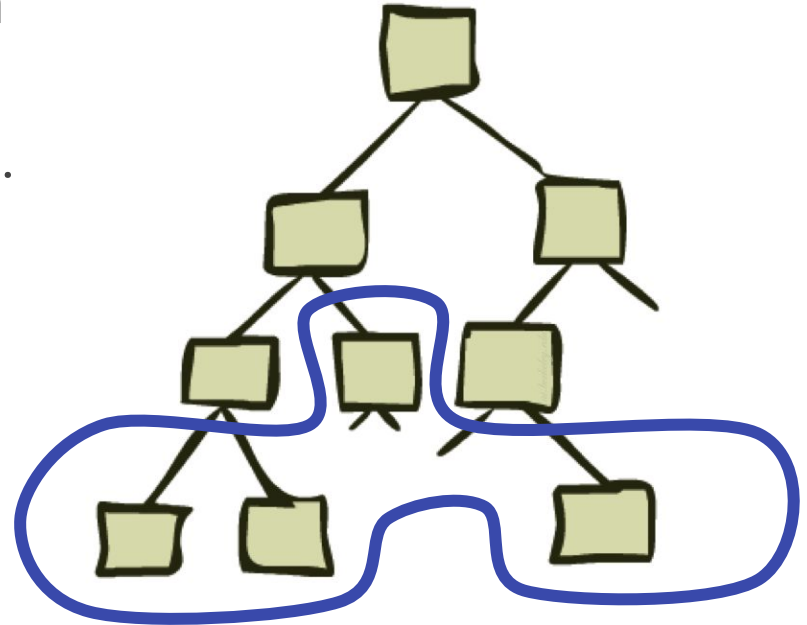
- O conjunto de nós folha disponíveis para expansão em um dado momento é denominado de **fronteira** ou **borda** (*frontier*, *fringe*).





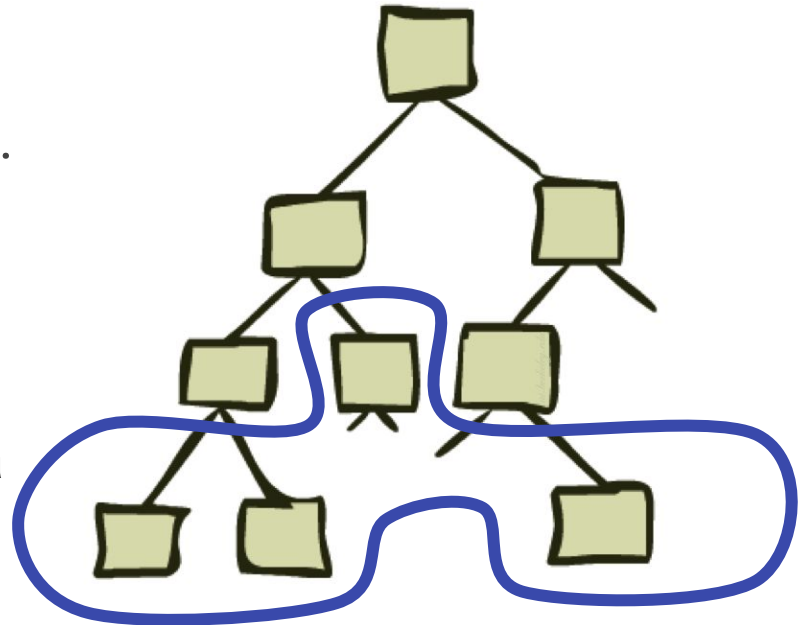
# Estratégias de Busca

- O conjunto de nós folha disponíveis para expansão em um dado momento é denominado de **fronteira** ou **borda** (*frontier*, *fringe*).

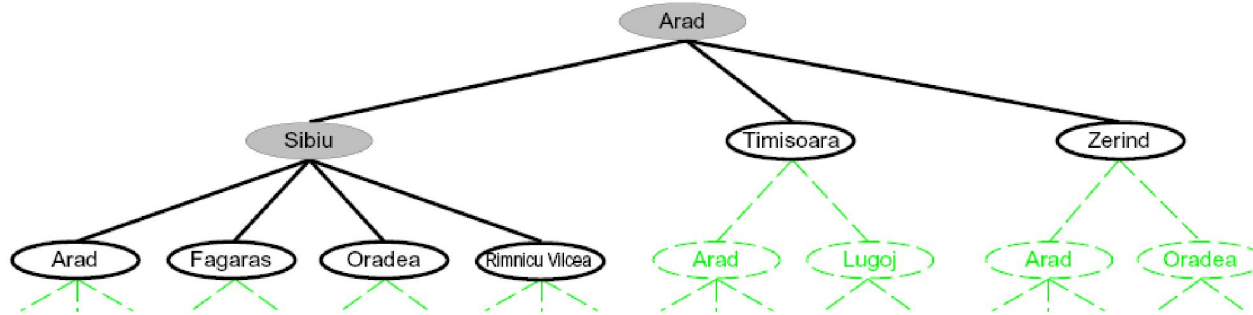


# Estratégias de Busca

- O conjunto de nós folha disponíveis para expansão em um dado momento é denominado de **fronteira** ou **borda** (*frontier, fringe*).
- Busca: seguir um caminho, guardando os outros para tentar depois.
- A **estratégia de busca** selecionada determina qual caminho seguir.

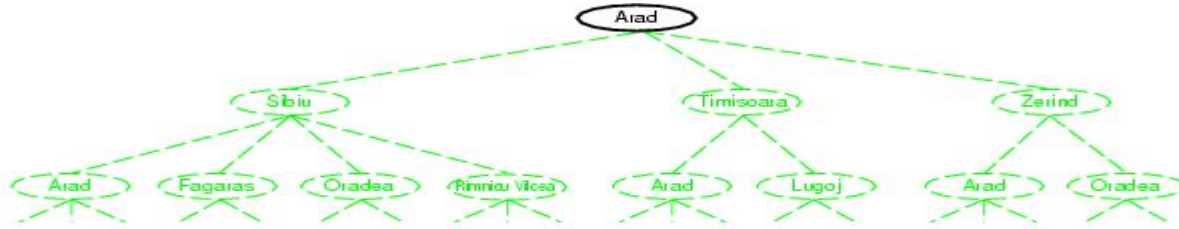


# Estratégias de Busca



- Como estratégias de busca funcionam:
  - Expandem uma árvore de busca à procura de um plano;
  - Mantêm uma **borda** (fronteira) de planos parciais sendo considerados;
  - Tentam expandir o mínimo de nós possível.

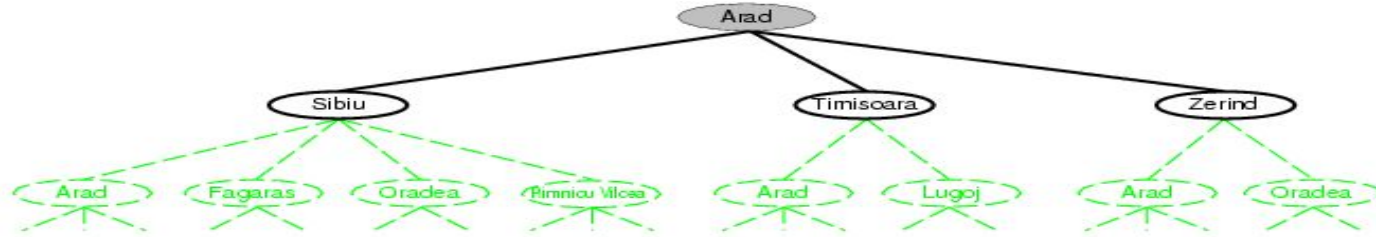
# Exemplo de árvore de busca



Estado inicial

Fronteira = {Arad}

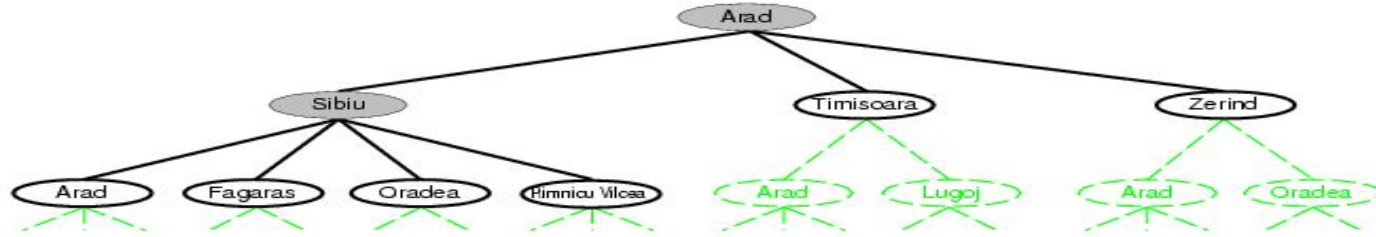
# Exemplo de árvore de busca



Depois de expandir Arad

Fronteira = {Sibiu, Timisoara, Zerind}

# Exemplo de árvore de busca



Depois de expandir Sibiu

Fronteira = {Arad, Fagaras, Oradea, Himnicu Vilcea, Timisoara, Zerind}

# Estratégias de busca - algoritmo genérico

```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end
```

A estratégia de seleção é o que diferencia os algoritmos de busca que iremos estudar

# Propriedades de uma Estratégia de Busca

- Estratégias são avaliadas de acordo com os seguintes critérios:
  - **Completa?** Sempre encontra a solução, se alguma existe?
  - **Complexidade de tempo:** Número de nós gerados no pior caso
  - **Complexidade de espaço:** Número máximo de nós na memória no pior caso
  - **Ótima?** Garante encontrar a solução ótima?
- Esboço de uma árvore de busca (pior caso):
  - $b$  é o máximo fator de ramificação
  - $m$  é a profundidade máxima
  - pode haver soluções (em rosa) em vários níveis
- Número de nós na árvore?
  - $1 + b + b^2 + \dots + b^m = O(b^{m+1})$

