Inteligência Artificial

Prof. Fabrício Olivetti de França, Prof. Denis Fantinato

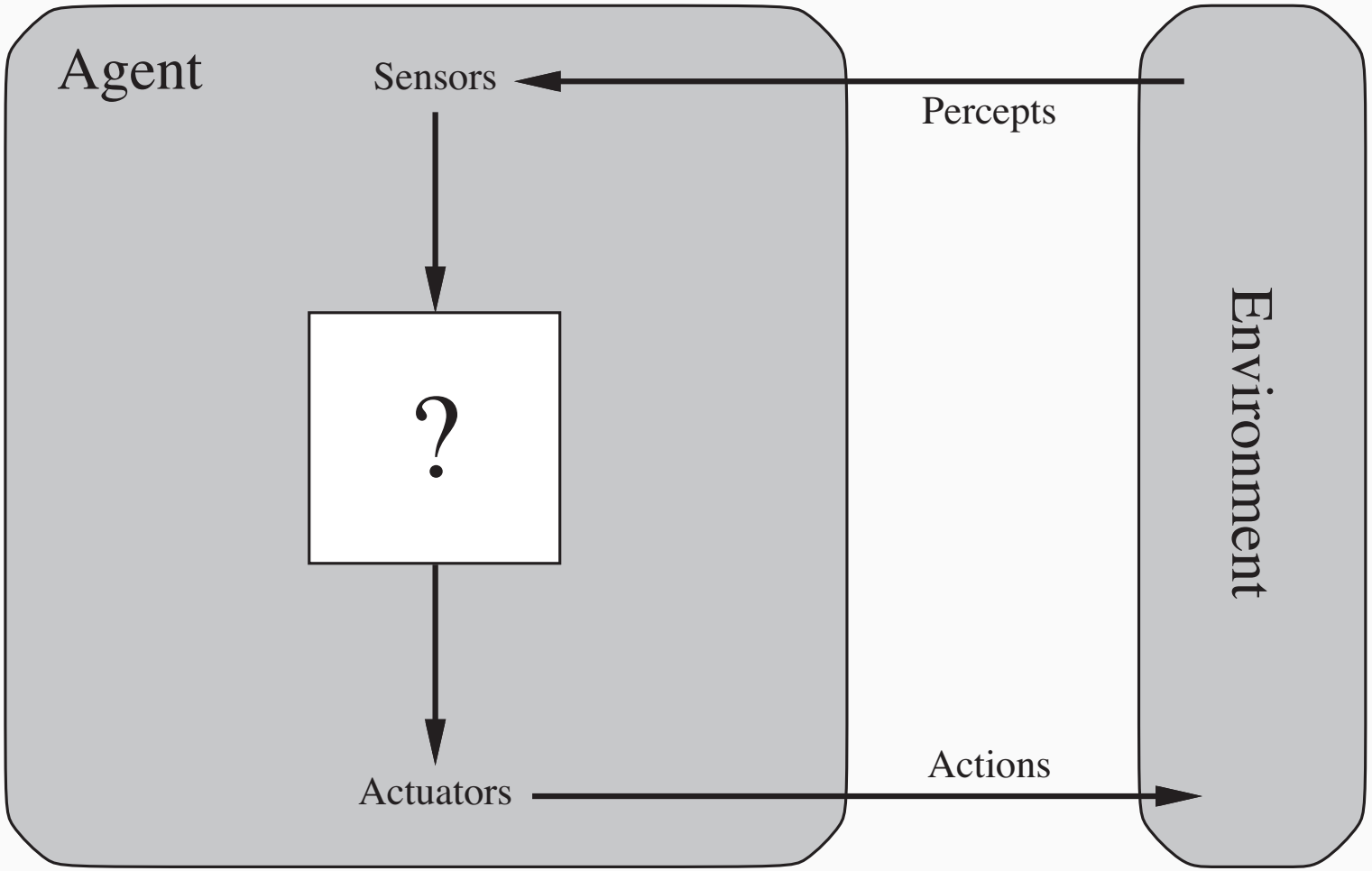
3º Quadrimestre de 2019

Agentes

Agente, Ambiente e Sensores

Um **agente** é definido pelo **ambiente** que ele consegue perceber através de seus **sensores** e as **ações** que ele pode escolher para atingir um certo objetivo.

Definição



Estado

Um **estado** é uma descrição de tudo que o agente pode perceber do ambiente em dado momento.

Uma **sequência de estados** é todo o histórico de estados pelo qual o agente percebeu durante um período de tempo.

Criando um Agente

Podemos criar um agente como um **mapa** de ações para cada possível estado do ambiente.

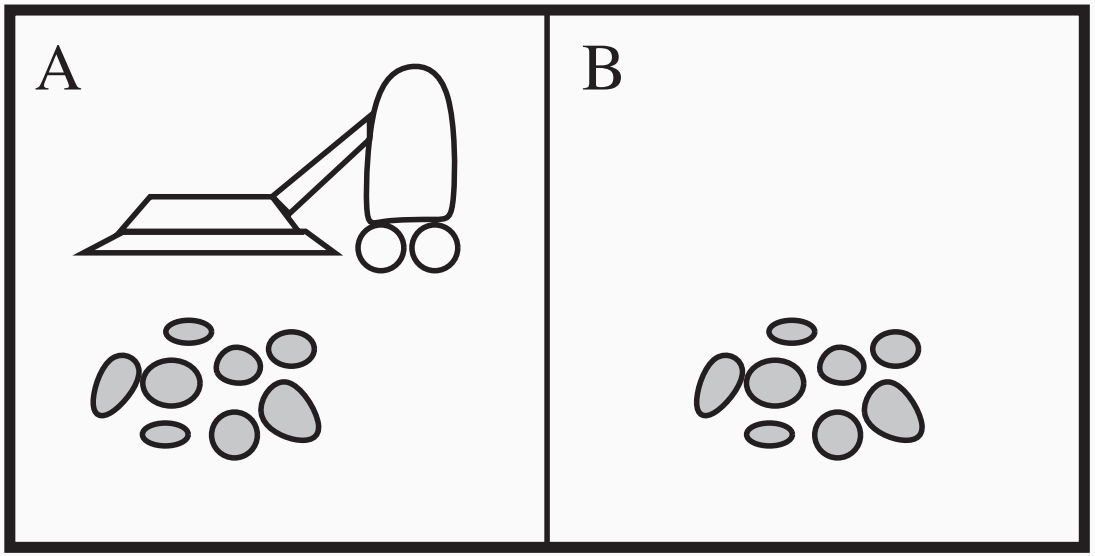
Matematicamente $f : \mathbb{E} \mapsto \mathbb{A}$

Robô-aspirador

Vamos definir o mundo do robô-aspirador que aspira o pó do chão de um ambiente.

O ambiente é definido como um quadriculado que pode ou não conter sujeira.

Robô-aspirador



Robô-aspirador

Imaginando que ele consegue perceber o local que ele se encontra e se esse local contém sujeira, um possível mapa de ações poderia ser:

Estado	Ação
(A, sujo)	Limpar
(A, limpo)	Direita
(B, sujo)	Limpar
(B, limpo)	Baixo
(C, sujo)	Limpar
(C, limpo)	Cima
(D, sujo)	Limpar
(D, limpo)	Esquerda

Racionalidade

Um agente é dito **racional** se cada ação que ele toma maximiza uma função de **desempenho**.

Agente Racional

Para cada possível sequência de estados, um agente racional deve escolher uma ação que espera-se maximizar uma medida de desempenho, dada a evidência obtida pela sequência de estados e o conhecimento construído pelo agente ao longo do tempo (Russell e Norvig, pg. 37)

PEAS

PEAS é a sigla para **P**erformance, **E**nvironment, **A**ctuators, **S**ensors e descreve as propriedades de cada tipo para que um tipo de agente adequado seja escolhido.

- Medida de desempenho, ambiente
- atuadores e sensores do agente

PEAS

Agente	Performance	Environment	Actuators	Sensors
Motorista Uber	Seguro, rápido, confortável	Ruas, estradas, pedestres, clientes	Direção, acelerar, brecar, sinalizar, buzinar	Cameras, velocímetro, GPS, sensores do veículo

Propriedades do Ambiente

Um ambiente é dito **totalmente observável** quando os sensores de um agente conseguem medir todo o estado atual do ambiente, caso contrário ele é **parcialmente observável**.

O ambiente pode ser parcialmente observável por conta de ruídos ou incapacidade dos sensores perceberem todo o ambiente.

Propriedades do Ambiente

Um ambiente é **determinístico** quando o próximo estado é completamente determinado pelo estado atual e a ação executada pelo agente.

Quando podem ocorrer incertezas, ou o estado é descrito probabilisticamente, dizemos que ele é **estocástico**.

Propriedades do Ambiente

Se um ambiente demanda mais do que um agente, dizemos que ele é **multiagente**. Exemplos:

- Jogo de Xadrez demanda dois agentes, levando a um cenário **competitivo**.
- Motoristas de Uber demandam múltiplos agentes cooperando para minimizar acidentes, cenário **cooperativo**.

Propriedades do Ambiente

O ambiente pode ser **episódico** quando em cada passo o agente observa o ambiente e age sem levar em conta os episódios anteriores.

Exemplo: para classificar se uma foto contém um gato, o agente não precisa levar em conta se a foto anterior foi classificada como verdadeiro.

O ambiente **sequencial** a decisão tomada em dado instante, pode afetar toda a sequência de observações e ações seguinte.

Exemplo: jogador autônomo. Considere um jogo de xadrez em que uma ação tomada no estado atual afeta as possíveis decisões futuras.

Propriedades do Ambiente

O ambiente é dito **dinâmico** se, enquanto o agente não decide por uma ação, o estado se altera. Caso contrário é dito como **estático**.

Exemplo: o ambiente do motorista de Uber é dinâmico, o de um jogo de Xadrez é estático.

Propriedades do Ambiente

O ambiente é **discreto** quando contém um conjunto finito de possíveis estados e/ou não leva em conta a dimensão do tempo. Caso contrário ele é dito **contínuo**.

Exemplo: o ambiente do motorista de Uber é contínuo, o de um jogo de Xadrez é discreto.

Propriedades do Ambiente

Se o agente tem todo o conhecimento do ambiente e das ações que pode fazer, o ambiente é **conhecido**, caso contrário, o agente deve aprender como atuar no ambiente e, então, esse é **desconhecido**.

Considere um novo jogo de video-game como um ambiente desconhecido. O jogador deve primeiro observar o que cada botão do controle faz, o que cada obstáculo causa, etc. até aprender a jogar.

Exercício

Descreva cada uma das tarefas abaixo utilizando o formato PEAS (Performance, Environment, Actuators, Sensors):

- Jogar futebol
- Fazer as compras do mercado
- Dar lances em um produto no leilão

Tipos de Agentes

Algoritmo geral

O pseudo-algoritmo abaixo será utilizado para explicar os algoritmos seguintes para criar um agente:

```
def agenteMapa(ambiente):
  acao = mapa[ambiente]
  return acao
```

Agente reflexivo

Mapeia as ações em forma de regras **SE-ENTÃO**:

```
def agenteReflex(ambiente):
  estado = interpreta(ambiente)
  regra  = escolheRegra(estado)
  return regra.acao
```

Robô-aspirador

A função `escolheRegra` no nosso exemplo do robô aspirador seria:

```
def escolheRegra(estado):
  pos, sujo = estado
  if sujo:
    return {'estado': estado, 'acao': 'limpa'}
  else:
    if pos == 'A':
      return {'estado': estado, 'acao': 'direita'}
    elif pos == 'B':
      return {'estado': estado, 'acao': 'baixo'}
    elif pos == 'C':
      return {'estado': estado, 'acao': 'cima'}
    else:
      return {'estado': estado, 'acao': 'esquerda'}
```

Agente reflexivo

Esse tipo de agente, embora simples, é limitado quando o ambiente não é totalmente observável ou contém incertezas.

Agente Reflexivo com Modelo

O agente reflexivo com modelo introduz uma função de mapeamento do estado atual, em que ele atualiza o estado percebido de acordo com a sequência de observações feitas durante seu histórico:

```
def agenteModelReflex(ambiente):
  estado, modelo = atualizaEstado(estado, ambiente, modelo)
  regra          = escolheRegra(estado)
  return regra.acao
```

Agente baseado em objetivos

Agentes com um objetivo bem definido e que guiam suas ações de tal forma a atingí-lo:

```
def agenteObjetivo(estado):
  acao = buscaMaisProximo(estado, objetivo)
  return acao
```

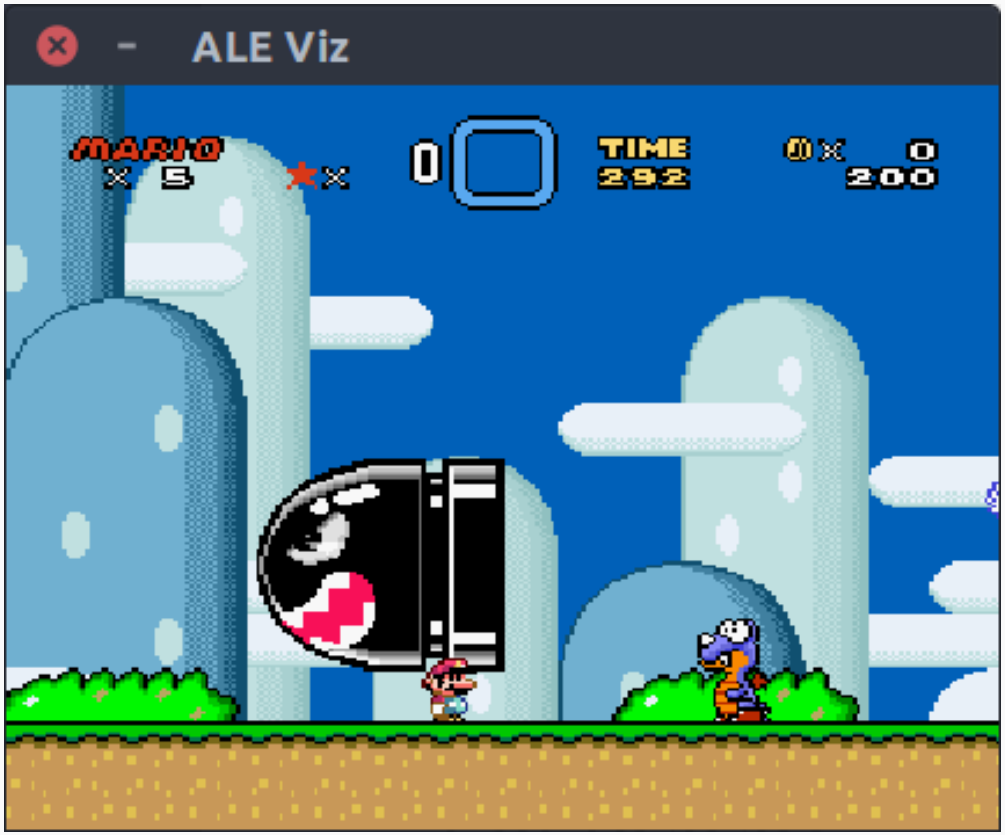
Agente baseado em Utilidade

Agentes com um objetivo quantificável:

```
def agenteObjetivo(estado):
  acao = argmax(acoes, estado, f)
  return acao
```

Exercício

Que tipo de regras você faria para um agente no jogo Super Mario World?



Problemas de Busca

Buscando soluções

Uma forma de criar um agente é formalizando o problema a ser resolvido e buscando por uma solução (sequência de ações) que atinge o objetivo.

Definindo um problema

Para formalizar um problema precisamos:

- **Estado inicial:** o estado que o agente inicia a busca por uma solução.
- **Possíveis ações:** uma função `acoes(estado)` que retorna o conjunto de ações factíveis naquele estado atual.

Definindo um problema

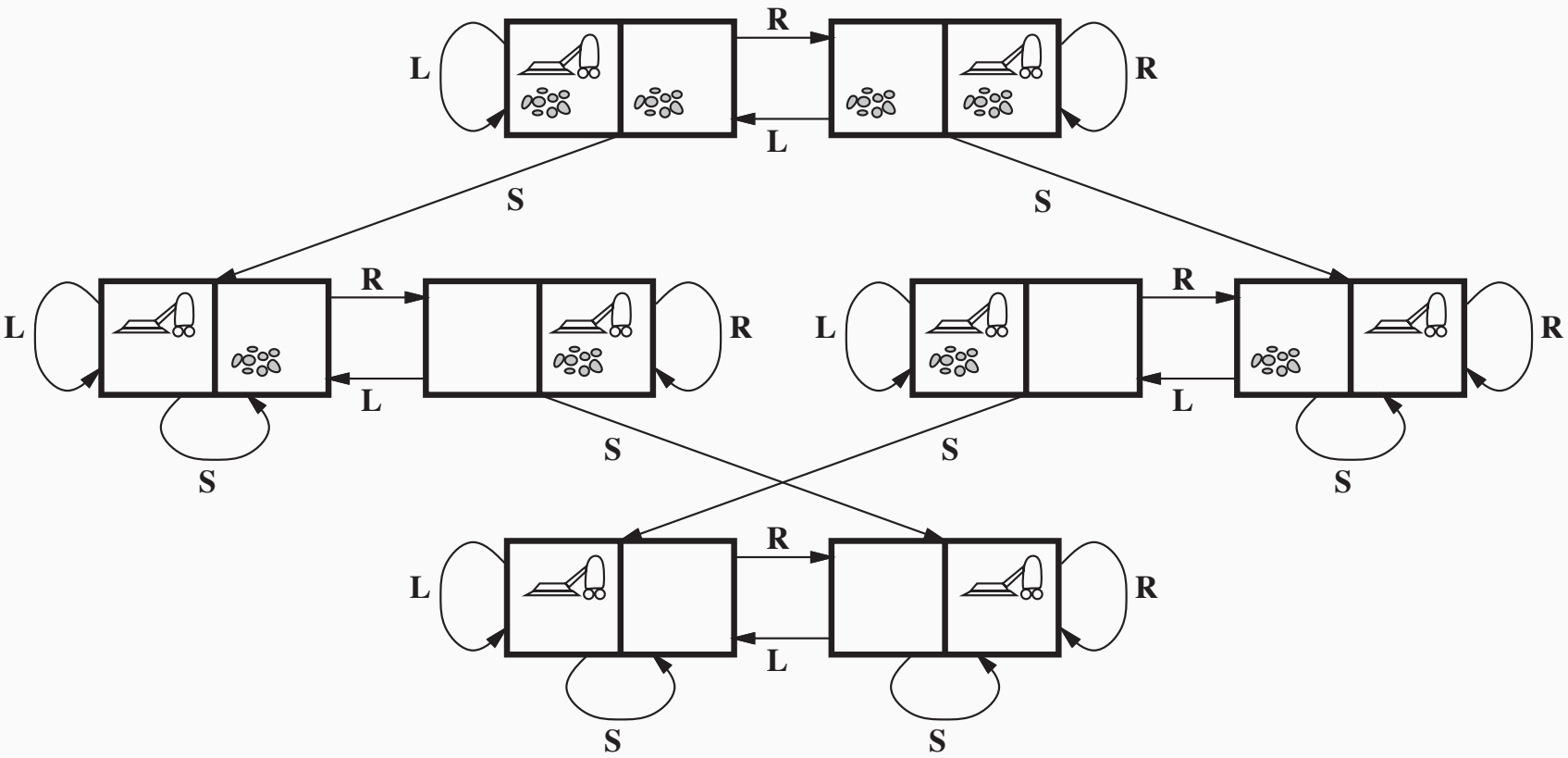
- **Modelo de transição:** uma função `resultado(s, a)` que retorna o estado `s'` sucessor de `s` ao executar a ação `a`.
- **Função de objetivo:** uma função `atingiuObj(s)` que retorna `True` se o agente atingiu o objetivo desejado.
- **Custo de caminho:** uma função `custo(s, a, s')` que indica o custo de sair do estado `s` e chegar no estado `s'` ao executar a ação `a`.

Robô-aspirador

No mundo do robô aspirador, podemos definir:

- **Estado inicial:** qualquer estado arbitrário.
- **Possíveis ações:** a lista `[esquerda, direita, cima, baixo, aspira]` excluindo os movimentos infactíveis.
- **Modelo de transição:** vide figura no próximo slide.
- **Objetivo:** todos os quadrados limpos.
- **Custo de caminho:** cada movimento tem custo 1, o custo total é a soma dos custos até atingir o objetivo.

Modelo de transição



Buscando soluções

Uma forma de encontrar a sequência de ações que leva ao objetivo é por uma **busca em árvore:**

```
def buscaArvore(raiz, problema):
  if vazio(raiz):
    return Falha
  for no in raiz:
    if atingiuObj(no):
      return solution(no)
    novosNos = expand(no)
  for no in novosNos:
    sol = buscaArvore(no, problema)
    if sol != Falha:
      return sol
  return Falha
```

Buscando soluções

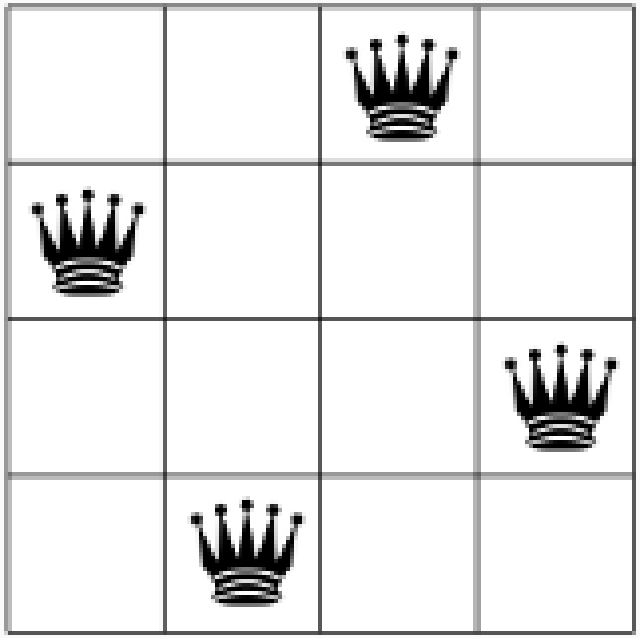
A ideia é percorrer todas as sequências de estado até que encontre uma que encontre o objetivo.

Caso encontre, retorna a sequência de ações para obter a solução.

Pode não ser viável dependendo do tamanho do espaço de busca.

Exercício

Considere o problema de colocar quatro rainhas no seguinte tabuleiro sem que elas se ataquem:



Faça a árvore de busca partindo do tabuleiro vazio.

Busca Cega

Busca sem informação

A **busca cega** ou **busca desinformada** é o conjunto de algoritmos de busca que procura por uma solução utilizando apenas o conhecimento provido pela descrição do problema.

Esses algoritmos se diferenciam um dos outros pela ordem que expandem os nós da árvore de busca.

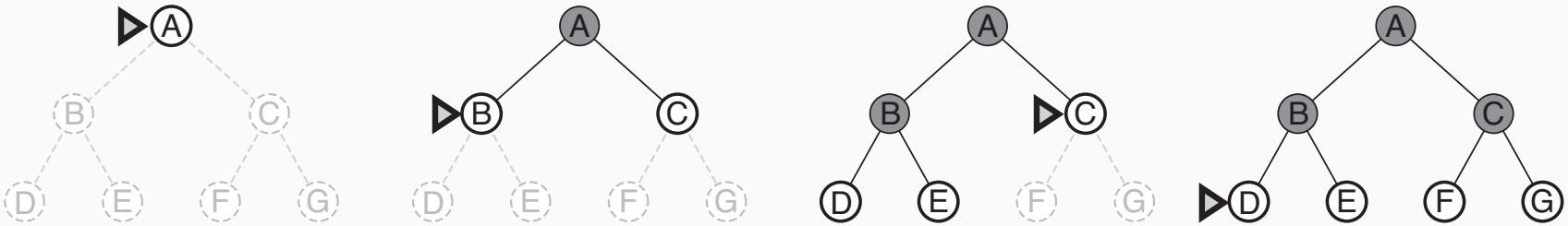
Busca em Largura

A estratégia adotada pela busca em largura é a de expandir os filhos do nó atual em apenas um nível, gerando uma lista de nós. Essa lista é então expandida em mais um nível e assim por diante até encontrar a solução.

Busca em Largura

```
def buscaLargura(estados):
  estados' = []
  for s in estados:
    s' = [resultado(s,a) for a in acoes(s)]
    estados' = estados' + s'
  if any(atingiuObj(s) for s in estados')
    return solucao(estados')
  return buscaLargura(estados')
```

Busca em Largura



Busca em Largura

A busca em largura garante encontrar a solução em um número limitado de passos se essa existir e o conjunto de estados não for infinito.

Busca em Largura

Uma forma de tornar o algoritmo mais rápido é expandir os nós na ordem de seu custo total:

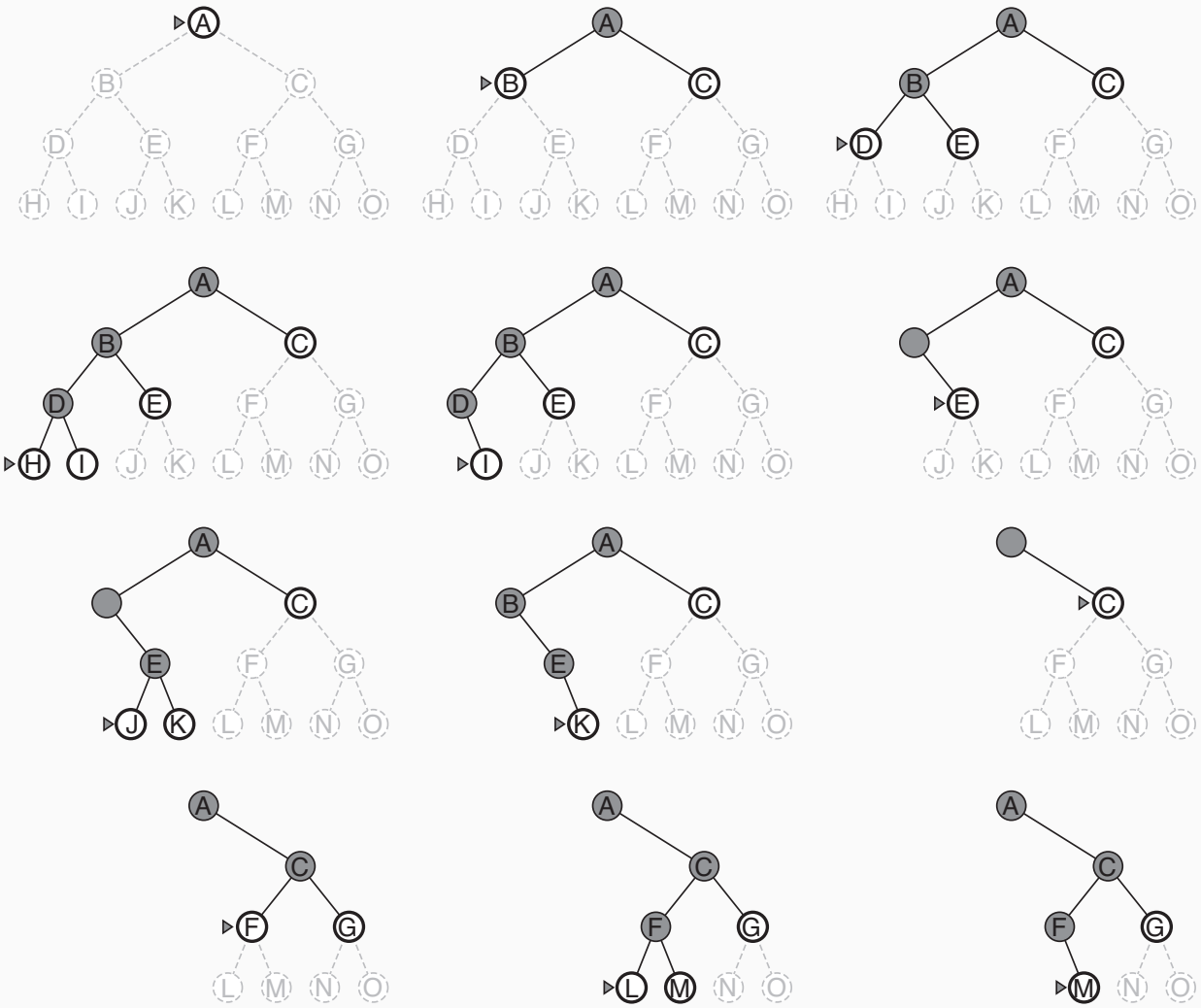
```
def buscaLargura(estados):
  s  = argmin(custo(s) for s in estados)
  s' = [resultado(s,a) for a in acoes(s)]
  estados' = estados.remove(s) + s'
  if any(atingiuObj(s) for s in s')
    return solucao(s')
  return buscaLargura(estados)
```

Busca em Profundidade

Na busca em profundidade primeiro expande toda a extensão de um nó para então dar prosseguimento na expansão do nó seguinte:

```
def buscaProfundidade(s):
  for a in acoes(s):
    sol = buscaProfundidade(resultado(s,a))
    if atingiuObj(sol):
      return sol
  return Falha
```

Busca em Profundidade



Busca em Profundidade

Deve-se tomar cuidado pois se o problema apresenta ciclos de transições de estado ou existem infinitos possíveis estados, o algoritmo pode não parar.

Para evitar essa situação costuma-se acrescentar um limite máximo de aprofundamento para forçar a parada, denominado **busca em profundidade limitada*.

Busca em Profundidade Limitada

```
def buscaProfundidadeLimite(s, profundidade):
  if profundidade > limite:
    return Falha
  for a in acoes(s):
    sol = buscaProfundidade(resultado(s,a), profundidade+1)
    if atingiuObj(sol):
      return sol
  return Falha
```

Busca em Profundidade Limitada

Esse limite pode ser ajustado iterativamente até encontrar a melhor solução:

```
def buscaProfundidadeIterativo(s):
  for it in range(maxProfundidade):
    sol = buscaProfundidadeLimite(s,it)
    if sol != Falha:
      return sol
  return Falha
```

Exercício

No problema anterior das 4 rainhas, quantos passos são necessários para atingir o objetivo utilizando busca em largura e busca em profundidade?