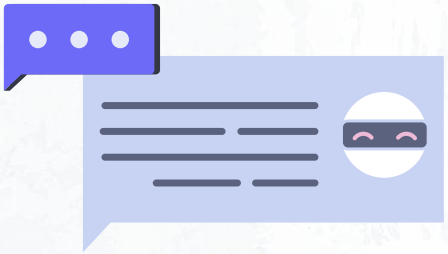
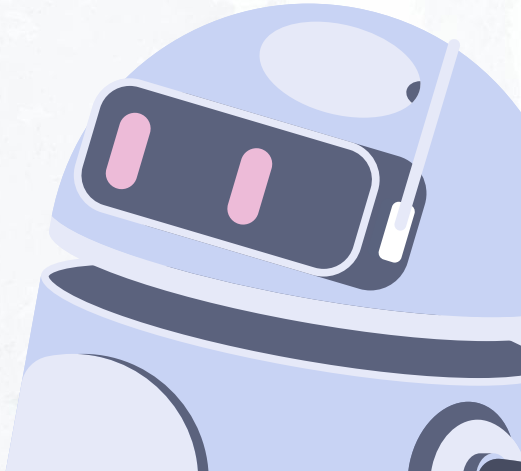
Introdução à Inteligência Artificial

João Paulo Aires

(IA)

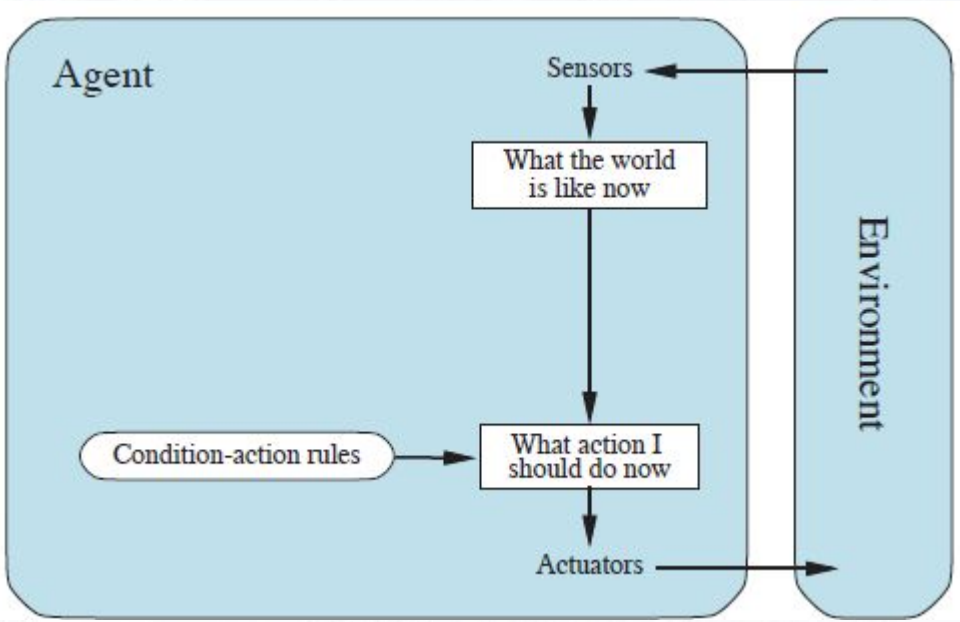




Recap

Tipos de Agentes

Agentes de Reflexos Simples



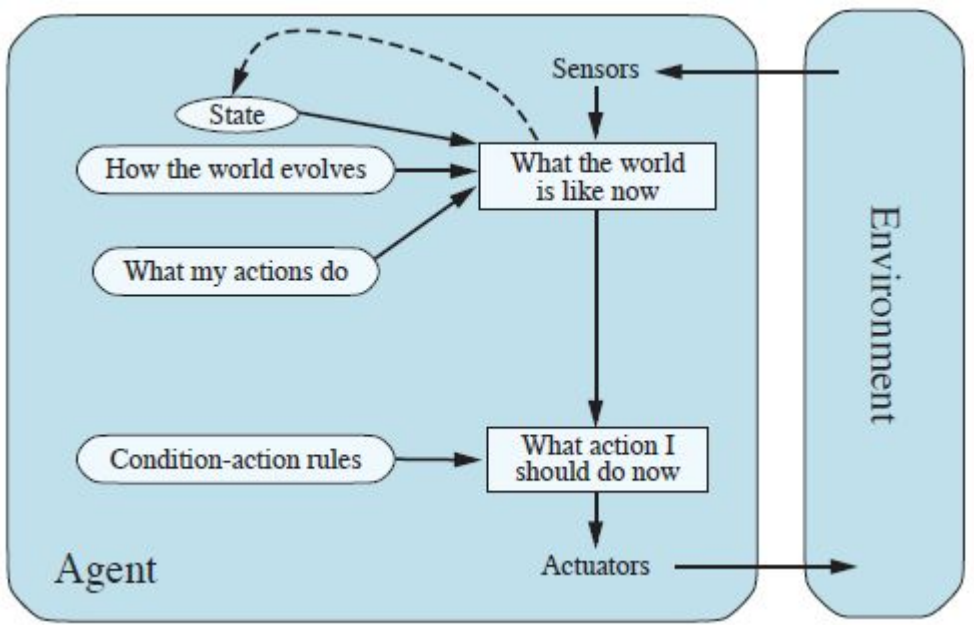
função Agente de Reflexo Simples (percepção) **retorna** uma ação

constante: regras → um conjunto de regras condição-ação

estado ← interpretaEntrada(percepção)
regra ← encontraRegra(estado)
ação ← regra.Ação
retorna ação

Tipos de Agentes

Agentes de Reflexos Baseado em Modelo

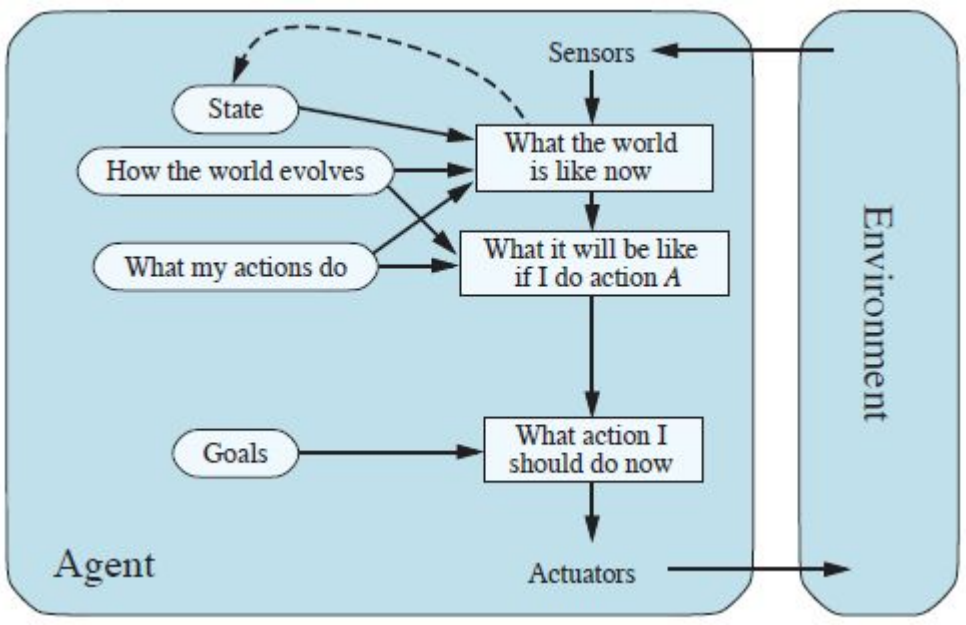


função Agente de Reflexo Modelo (percepção)
retorna uma ação
constante:
estado → concepção do mundo
t_m → modelo de dinâmica do ambiente
s_m → modelo de dinâmica do sensor
regras → um conjunto de regras condição-ação
ação → a ação mais recente

estado ← atualizaEstado(estado, ação, percepção, t_m, s_m)
regra ← encontraRegra(estado, regras)
ação ← regra.Ação
retorna ação

Tipos de Agentes

Agentes Baseados em Objetivos



- Objetivos (complexas) impulsionam o comportamento;
- Juntamente com um modelo permite o raciocínio de longo prazo
- Veremos isso em Busca e Planejamento

Tipos de Ambientes

Ambiente	Observável	Agentes	Determinístico	Episódico	Estático	Discreto
Jogo da Velha						
Caminhar até o fundo desta sala						

Tipos de Ambientes

Ambiente	Observável	Agentes	Determinístico	Episódico	Estático	Discreto
Jogo da Velha	Sim	Multi	Sim	Não	Não	Sim
Caminhar até o fundo desta sala	Não	Multi	Não	Não	Não	Não

Objetivos

- Agentes para resolução de problemas;
- Algoritmos de Busca (Árvore/Grafo);
- Estratégias de Busca Desinformada.

Índice

Índice

01 →

Agentes Problem-Solving

(IA)

Agentes Problem-Solving

```
1: function Simple-Problem-Solving-Agente(percepção, estado)
2:     static seq ← []                    # uma sequência de ações
3:     estado ← atualizaEstado(estado, percepção)
4:     if seq está vazia then
5:         goal ← formulaObjetivo(estado)
6:         problema ← formulaProblema(estado; objetivo )
7:         seq ← busca(problema)
8:     action ← first(seq)
9:     seq ← rest(seq)
10:    return action
```

Agentes Problem-Solving

```
1: function Simple-Problem-Solving-Agente(percepção, estado)
2:     static seq ← []                    # uma sequência de ações
3:     estado ← atualizaEstado(estado, percepção)
4:     if seq está vazia then
5:         goal ← formulaObjetivo(estado)
6:         problema ← formulaProblema(estado; objetivo )
7:         seq ← busca(problema)
8:     action ← first(seq)
9:     seq ← rest(seq)
10:    return action
```

Formular um objetivo (via formulaObjetivo) é o primeiro passo na resolução de problemas; O que estamos tentando alcançar?

Agentes Problem-Solving

```
1: function Simple-Problem-Solving-Agente(percepção, estado)
2:     static seq ← []                    # uma sequência de ações
3:     estado ← atualizaEstado(estado, percepção)
4:     if seq está vazia then
5:         goal ← formulaObjetivo(estado)
6:         problema ← formulaProblema(estado; objetivo )
7:         seq ← busca(problema)
8:     action ← first(seq)
9:     seq ← rest(seq)
10:    return action
```

A formulação do problema envolve a identificação das ações e estados que devem ser considerados.

Agentes Problem-Solving

```
1: function Simple-Problem-Solving-Agente(percepção, estado)
2:     static seq ← []                    # uma sequência de ações
3:     estado ← atualizaEstado(estado, percepção)
4:     if seq está vazia then
5:         goal ← formulaObjetivo(estado)
6:         problema ← formulaProblema(estado; objetivo )
7:         seq ← busca(problema)
8:     action ← first(seq)
9:     seq ← rest(seq)
10:    return action
```

A busca envolve examinar diferentes sequências possíveis de ações que levam a estados de valor conhecido e, em seguida, escolher a melhor sequência.

Agentes Problem-Solving

```
1: function Simple-Problem-Solving-Agente(percepção, estado)
2:     static seq ← []                    # uma sequência de ações
3:     estado ← atualizaEstado(estado, percepção)
4:     if seq está vazia then
5:         goal ← formulaObjetivo(estado)
6:         problema ← formulaProblema(estado; objetivo )
7:         seq ← busca(problema)
8:     action ← first(seq)
9:     seq ← rest(seq)
10:    return action
```

Um algoritmo de busca toma um problema como entrada e retorna uma solução na forma de uma sequência de ações.

Agentes Problem-Solving

Tipos de Problemas

- **Determinístico**, **Totalmente Observável** → Problema de Estado-único
  - O agente sabe exatamente em que estado estará; solução é uma sequência.

Agentes Problem-Solving

Tipos de Problemas

- **Determinístico**, **Totalmente Observável** → Problema de Estado-único
  - O agente sabe exatamente em que estado estará; solução é uma sequência.

- **Não observável** → Problema de Conformidade
  - O agente pode não ter ideia de onde está; solução (se houver) é uma sequência

Agentes Problem-Solving

Tipos de Problemas

- **Determinístico**, **Totalmente Observável** → Problema de Estado-único
  - O agente sabe exatamente em que estado estará; solução é uma sequência.

- **Não observável** → Problema de Conformidade
  - O agente pode não ter ideia de onde está; solução (se houver) é uma sequência

- **Não Determinístico** e/ou **Parcialmente Observável** → Problema Contingente
  - percepções fornecem **novas** informações sobre o estado atual. A solução é um **plano contingente** ou uma **política** frequentemente intercalada, busca, execução

Agentes Problem-Solving

Tipos de Problemas

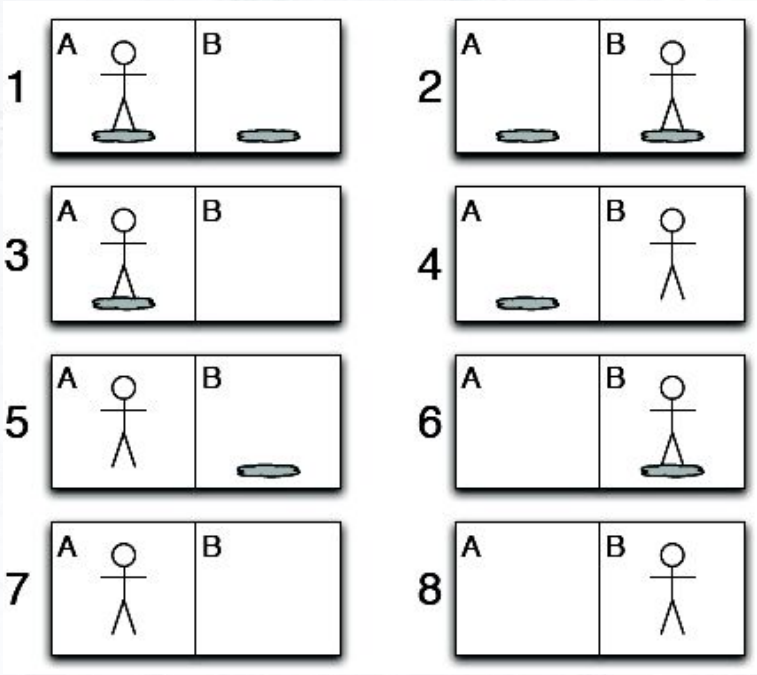
- **Determinístico**, **Totalmente Observável** → Problema de Estado-único
  - O agente sabe exatamente em que estado estará; solução é uma sequência.

- **Não observável** → Problema de Conformidade
  - O agente pode não ter ideia de onde está; solução (se houver) é uma sequência
- **Não Determinístico** e/ou **Parcialmente Observável** → Problema Contingente
  - percepções fornecem **novas** informações sobre o estado atual a solução é um **plano contingente** ou uma **política** frequentemente intercalada, busca, execução
- **Espaço de estados desconhecido** → Problema de exploração (“online”)

Agentes Problem-Solving

Exemplo: Vacuum World

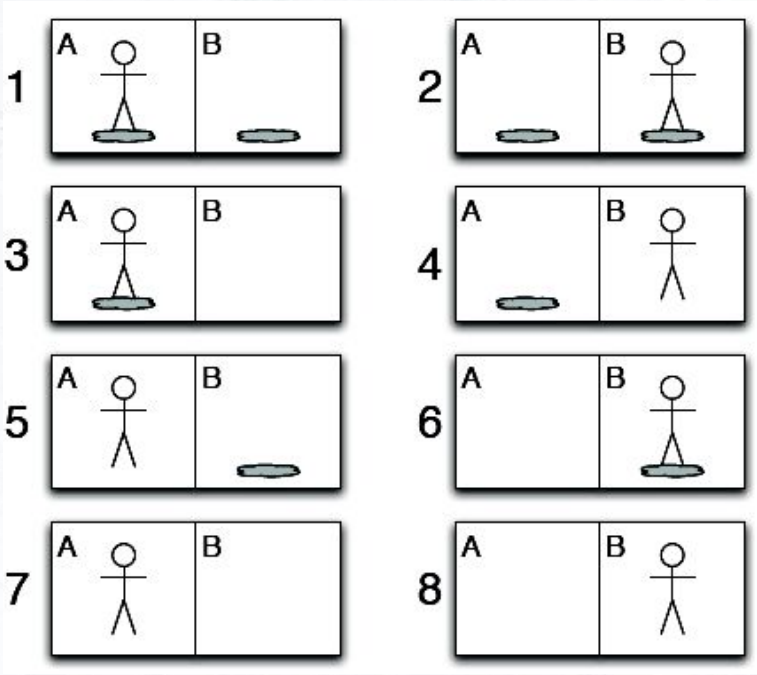
- Estado-único, começa no 5. Solução?



Agentes Problem-Solving

Exemplo: Vacuum World

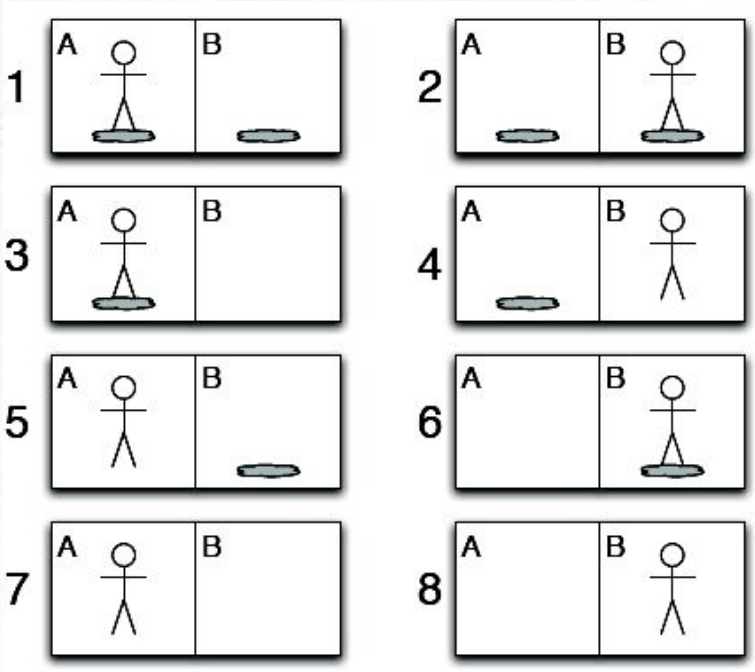
- Estado-único, começa no 5. Solução?
  - **[Direita, Aspira]**



Agentes Problem-Solving

Exemplo: Vacuum World

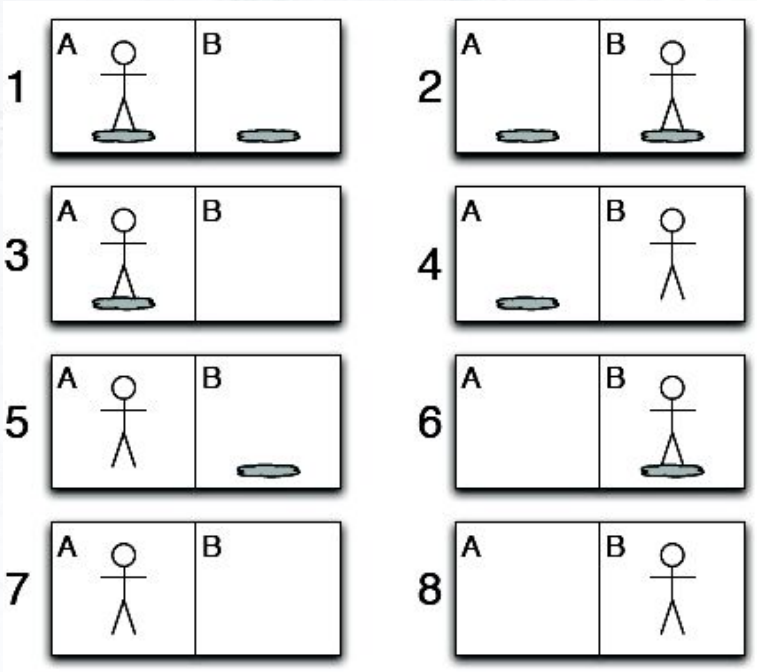
- Estado-único, começa no 5. Solução?
  - **[Direita, Aspira]**
- Conformativo, começa em {1, 2, 3, 4, 5, 6, 7, 8}, e.g., **Direta** vai para {2, 4, 6, 8}. Solução?



Agentes Problem-Solving

Exemplo: Vacuum World

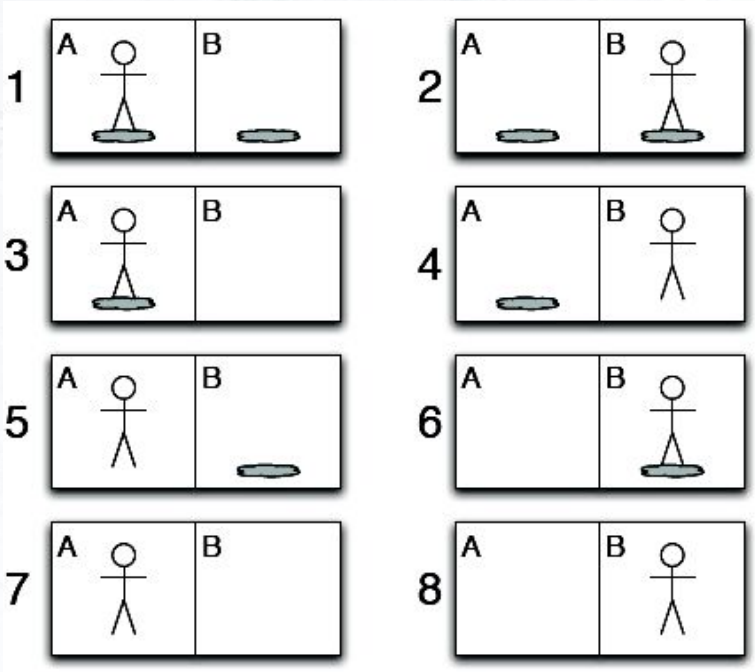
- Estado-único, começa no 5. Solução?
  - **[Direita, Aspira]**
- Conformativo, começa em {1, 2, 3, 4, 5, 6, 7, 8}, e.g., **Direta** vai para {2, 4, 6, 8}. Solução?
  - **[Direita, Aspira, Esquerda, Aspira]**



Agentes Problem-Solving

Exemplo: Vacuum World

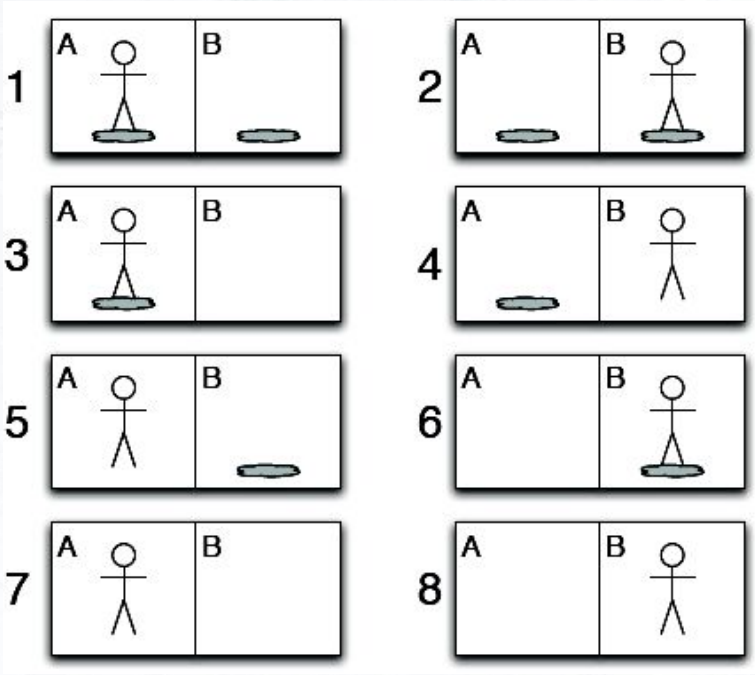
- Estado-único, começa no 5. Solução?
  - **[Direita, Aspira]**
- Conformativo, começa em {1, 2, 3, 4, 5, 6, 7, 8}, e.g., **Direta** vai para {2, 4, 6, 8}. Solução?
  - **[Direita, Aspira, Esquerda, Aspira]**
- Contingencial, começa no 5.
- Lei de Murphy: Aspirar o limpo suja
- Sensor: Sujeira, somente no local.
- Solução?



Agentes Problem-Solving

Exemplo: Vacuum World

- Estado-único, começa no 5. Solução?
  - **[Direita, Aspira]**
- Conformativo, começa em {1, 2, 3, 4, 5, 6, 7, 8}, e.g., **Direta** vai para {2, 4, 6, 8}. Solução?
  - **[Direita, Aspira, Esquerda, Aspira]**
- Contingencial, começa no 5.
- Lei de Murphy: Aspirar o limpo suja
- Sensor: Sujeira, somente no local.
- Solução?
  - **[Direita,** se **sujo** então **Aspira]**



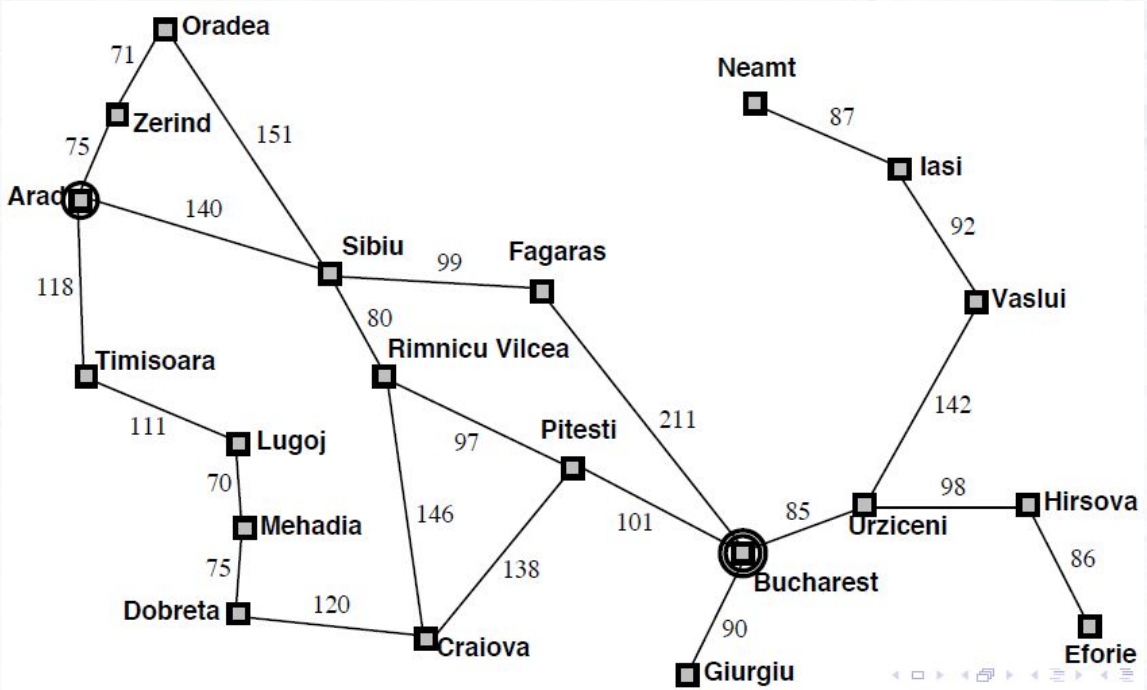
02 →

Formulação do Problema

(IA)

Formulação do Problema

Exemplo: Romênia



Formulação do Problema

Exemplo: Romênia

- De férias na Romênia
  - atualmente em **Arad**
  - voo sai amanhã de Bucareste
- Formular objetivo: estar em **Bucareste**
- Formule o problema:
  - **estados**: várias cidades
  - **ações**: dirigir entre cidades
- Encontre a solução:
  - sequência de cidades, por ex. Arad, Sibiu, Fagaras, Bucareste

Formulação do Problema

Formulação de um Problema de Estado Único

- Um problema é definido por cinco itens:
  - **Estado Inicial**, por exemplo, “em Arad”
  - **Ações**, por ex. dirigir a rota Arad → Zerind
  - **Função Sucessora** S(x) = conjunto de pares ação-estado

    Por exemplo, S(Arad) = {⟨Arad → Zerind, Zerind⟩, . . . }

Formulação do Problema

Formulação de um Problema de Estado Único

- **Teste de Objetivo**, pode ser:
  - explícito, por exemplo, x = “emBucareste”
  - implícito, por exemplo, SemSujeira(x)
- **Custo do Caminho** (aditivo)

  Por exemplo, soma de distâncias, número de ações executadas, etc.

  c(x, a, y) é o **custo da etapa**, assumido como ≥ 0
- Uma solução é uma sequência de ações que levam do estado inicial a um estado objetivo

Formulação do Problema

Selecionando um espaço de estado

- O mundo real é absurdamente **complexo**
  - o espaço de estados deve ser abstraído para resolução de problemas
- (Abstrato) **estado** = conjunto de estados reais
- (Abstrata) **ação** = combinação complexa de ações reais
  - por exemplo, “Arad → Zerind” representa um conjunto de possíveis rotas, desvios, paradas, etc.

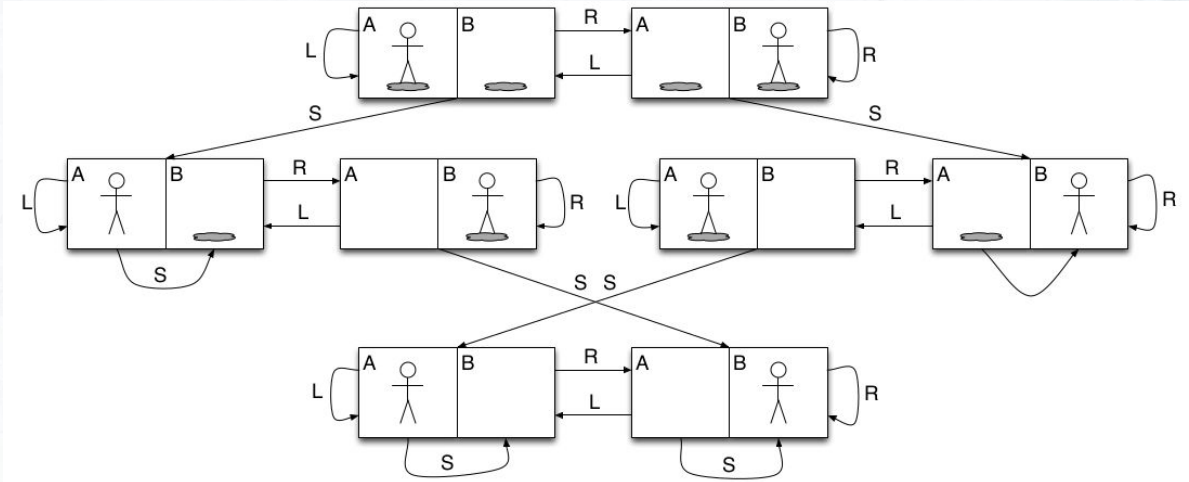
Formulação do Problema

Selecionando um espaço de estado

- Para ser viável, qualquer estado real “em Arad” deve chegar a algum estado real “em Zerind”
- (Abstrata) **Solução** = conjunto de caminhos reais que são soluções no mundo real
- Cada ação abstrata deve ser “mais fácil” que o problema original!

Formulação do Problema

Exemplo: grafo do espaço de estados do Vacuum world

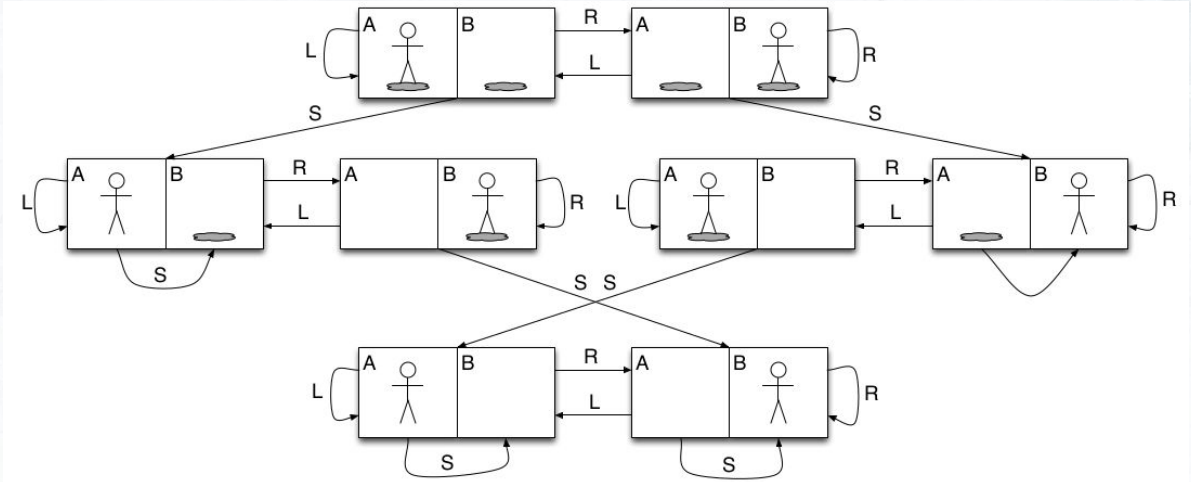


O que são os?

- Estados:
- Ações:
- Estado Objetivo:
- Custo do caminho:

Formulação do Problema

Exemplo: grafo do espaço de estados do Vacuum world

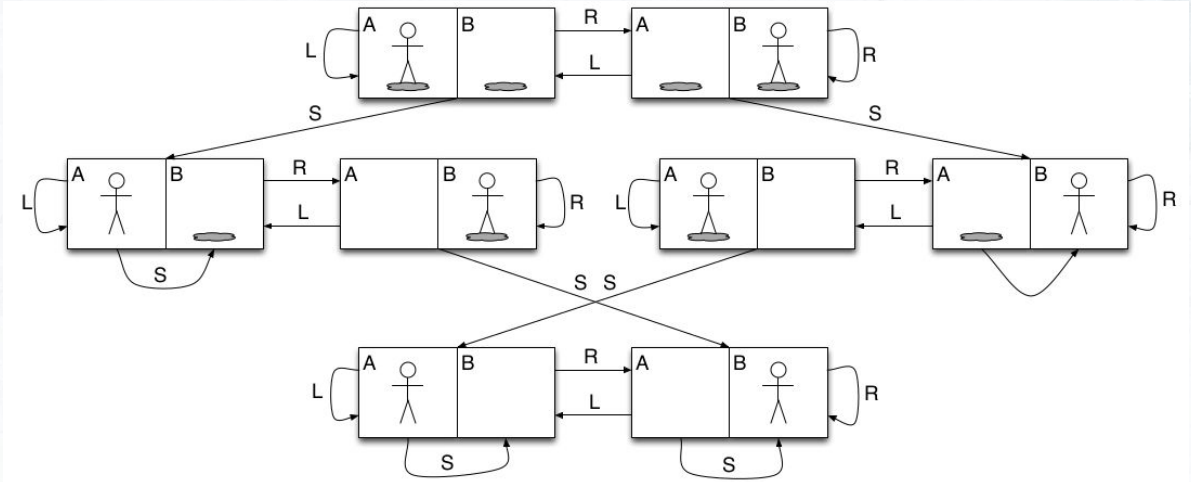


O que são os?

- Estados: sujeira inteira e localizações do robô (ignore quantidades de sujeira, etc.)
- Ações:
- Estado Objetivo:
- Custo do caminho:

Formulação do Problema

Exemplo: grafo do espaço de estados do Vacuum world

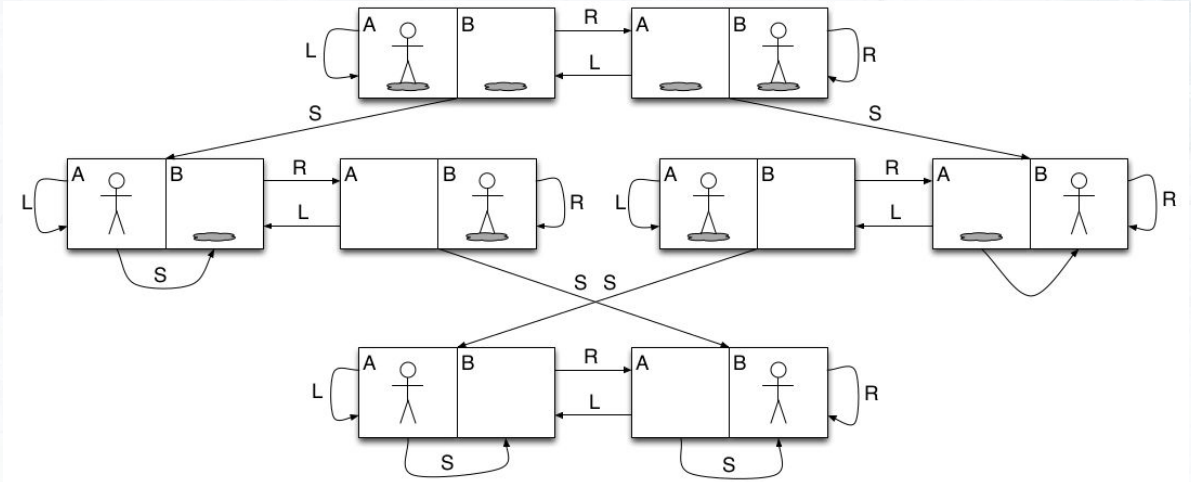


O que são os?

- Estados: sujeira inteira e localizações do robô (ignore quantidades de sujeira, etc.)
- Ações: **Esquerda, Direita, Aspirar, Não Agir**
- Estado Objetivo:
- Custo do caminho:

Formulação do Problema

Exemplo: grafo do espaço de estados do Vacuum world

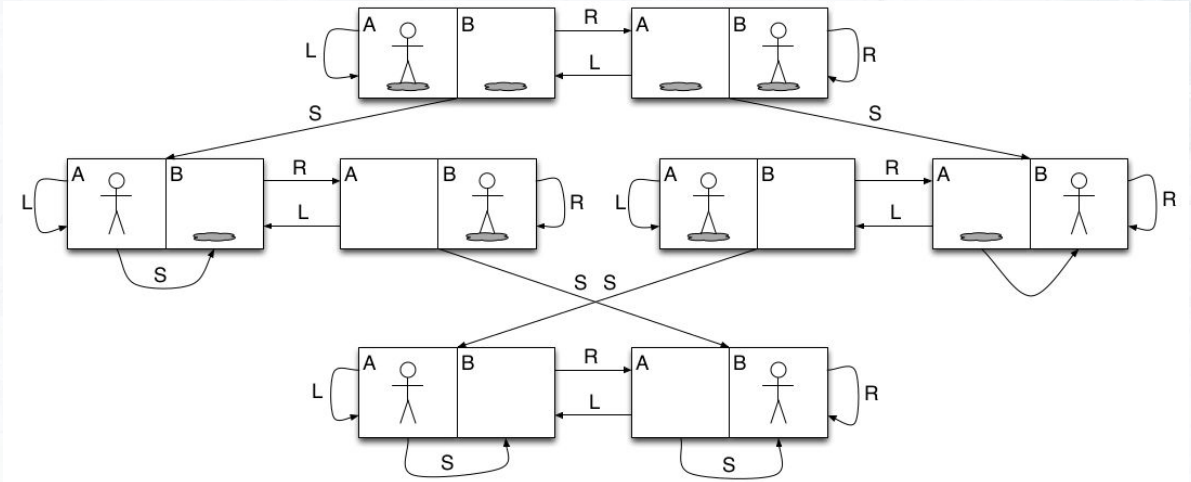


O que são os?

- Estados: sujeira inteira e localizações do robô (ignore quantidades de sujeira, etc.)
- Ações: **Esquerda, Direita, Aspirar, Não Agir**
- Estado Objetivo: sem sujeira
- Custo do caminho:

Formulação do Problema

Exemplo: grafo do espaço de estados do Vacuum world

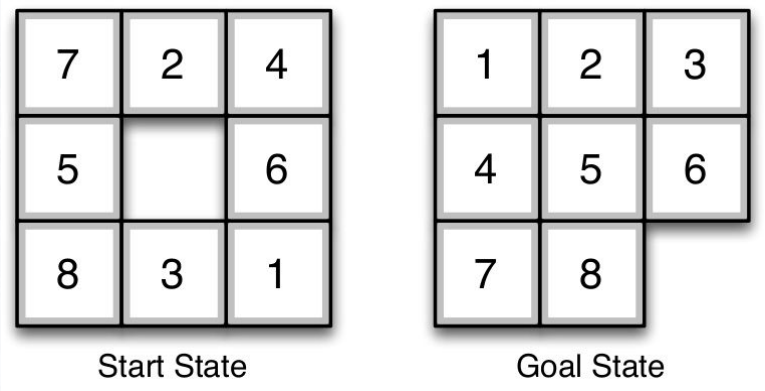


O que são os?

- Estados: sujeira inteira e localizações do robô (ignore quantidades de sujeira, etc.)
- Ações: **Esquerda, Direita, Aspirar, Não Agir**
- Estado Objetivo: sem sujeira
- Custo do caminho: 1 por ação (0 para **Não Agir**)

Formulação do Problema

Exemplo: The 8-puzzle

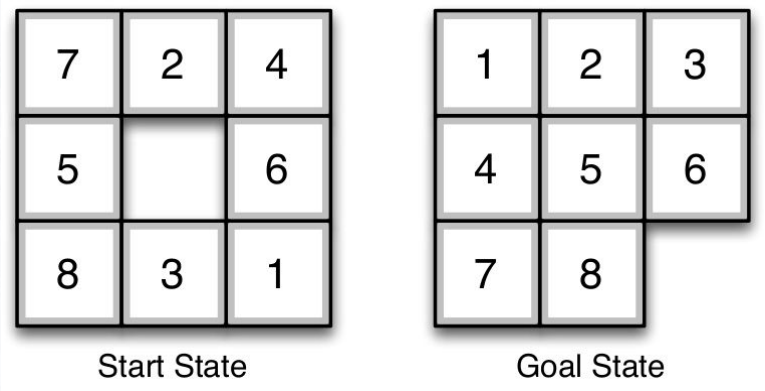


O que são os?

- Estados:
- Ações:
- Estado Objetivo:
- Custo do caminho:

Formulação do Problema

Exemplo: The 8-puzzle

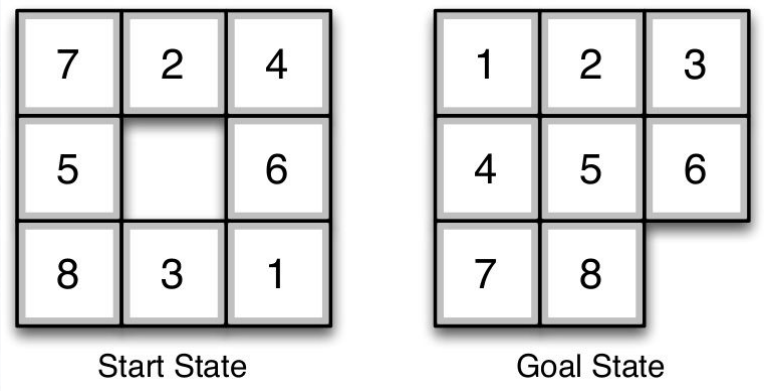


O que são os?

- Estados: localizações inteiras de blocos (ignore posições intermediárias)
- Ações:
- Estado Objetivo:
- Custo do caminho:

Formulação do Problema

Exemplo: The 8-puzzle

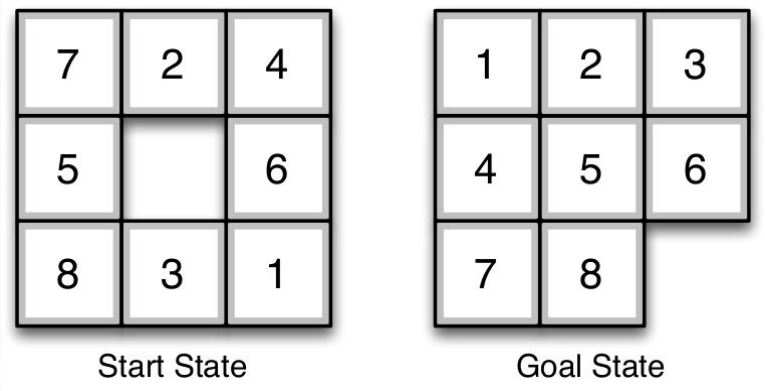


O que são os?

- Estados: localizações inteiras de blocos (ignore posições intermediárias)
- Ações: mova o espaço em branco para a esquerda, direita, para cima, para baixo
- Estado Objetivo:
- Custo do caminho:

Formulação do Problema

Exemplo: The 8-puzzle

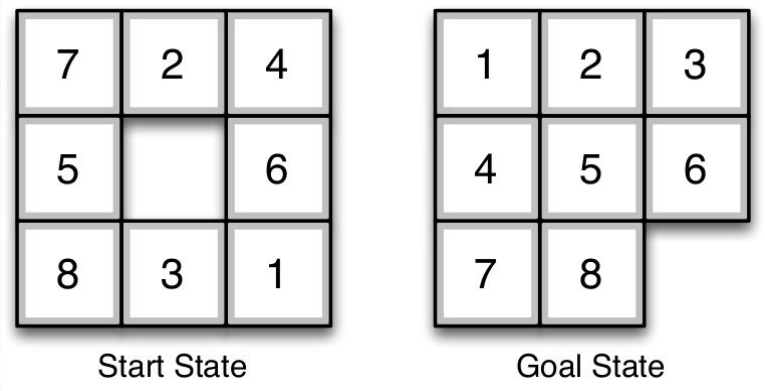


O que são os?

- Estados: localizações inteiras de blocos (ignore posições intermediárias)
- Ações: mova o espaço em branco para a esquerda, direita, para cima, para baixo
- Estado Objetivo: Goal State
- Custo do caminho:

Formulação do Problema

Exemplo: The 8-puzzle



O que são os?

- Estados: localizações inteiras de blocos (ignore posições intermediárias)
- Ações: mova o espaço em branco para a esquerda, direita, para cima, para baixo
- Estado Objetivo: Goal State
- Custo do caminho: 1 por ação

Formulação do Problema

Espaço de Estados

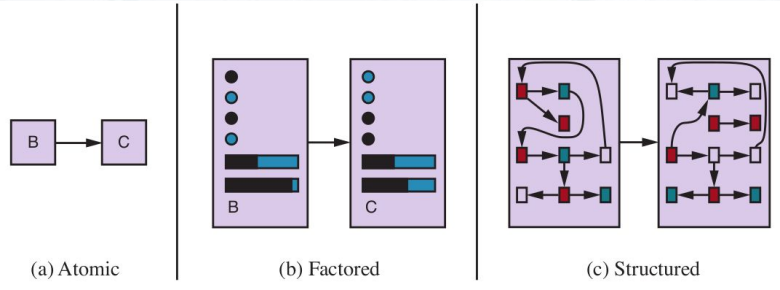
Como definimos um espaço de estados?

- Para espaços de estados finitos, podemos enumerar

  Vacuum World – 3 variáveis binárias: posição (A/B). sujaA, sujaB

  8-Puzzle – 9! estados possíveis

- Para espaços de estados infinitos, precisamos definir uma função que gere o próximo estado



03 →

Resolvendo Problemas

(IA)

Resolvendo Problemas

Tendo definido um problema, agora precisamos resolvê-lo. Fazemos isso pesquisando no espaço de estados

```
1.  function treeSearch(problema p, estratégia s)
2.      fronteira.add(p.inicial)
3.      loop
4.          if fronteira está vazia then
5.              return fail
6.          n ← s.removeEscolha(fronteira)
7.          if p.testaObjetivo(n.estado) then
8.              return getCaminho(n)
9.          for all a ∈ p.ações(n) do
10.             n' ← a.resultado(n.estado)
11.             fronteira.add(n')
```

Resolvendo Problemas

O que é um nó aqui?

- n.ESTADO: o estado correspondente no espaço de estados
- n.PARENT: o nó que gerou n
- n.AÇÃO: a ação aplicada a n.PARENT que resultou em n
- n.PATH-COST: o custo do caminho do estado inicial até n, conforme indicado pelos ponteiros PARENT

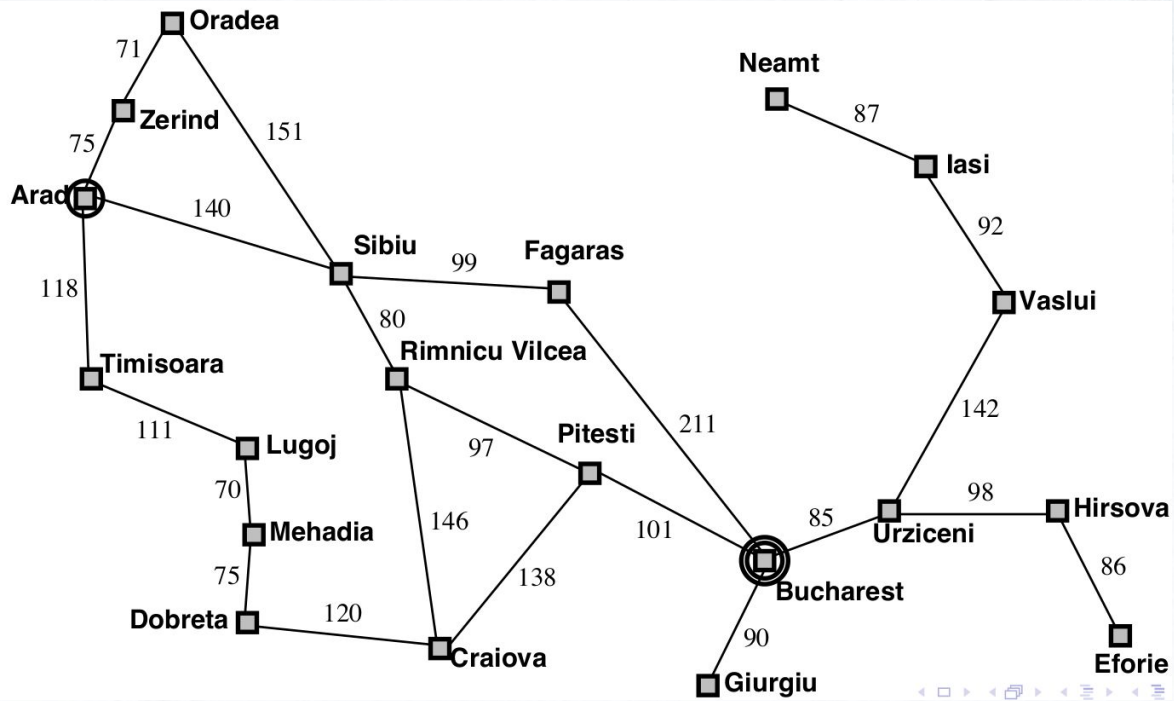
Resolvendo Problemas

Árvore de Busca

- A expansão ocorre aplicando funções sucessoras ao nó selecionado, fornecendo novos estados.
- Os nós que foram gerados, mas não expandidos, são chamados de fronteira ou franja.
- Diferentes estratégias de pesquisa comportam-se de maneira muito diferente.
- A árvore de busca não é igual ao espaço de estados. No exemplo de localização de rotas, existem 20 nós no espaço de estados, mas a árvore de busca possui um número infinito de nós.

Resolvendo Problemas

Exemplo de Fronteira



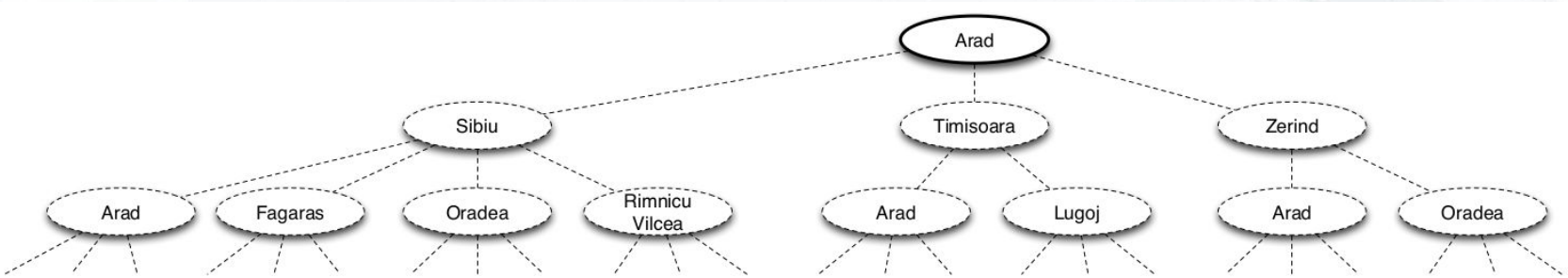
Resolvendo Problemas

Algoritmos de Busca em Árvore

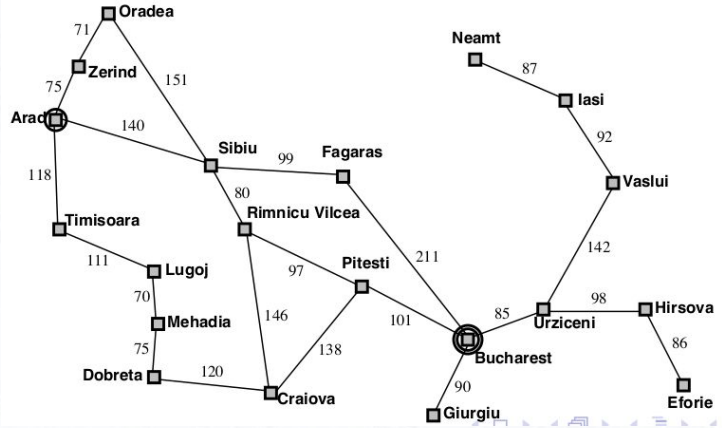
Ideia básica: Exploração simulada off-line do espaço de estados, gerando sucessores de estados já explorados (também conhecidos como estados em expansão)

Resolvendo Problemas

Exemplo de Busca em Árvore

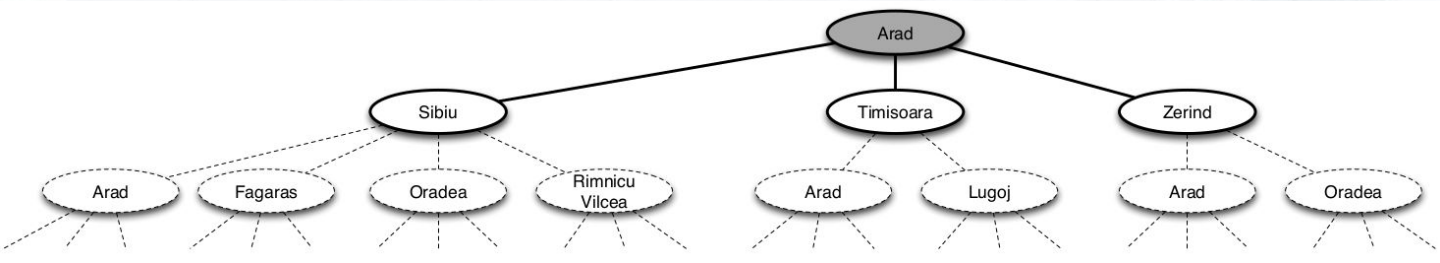


Estado Inicial

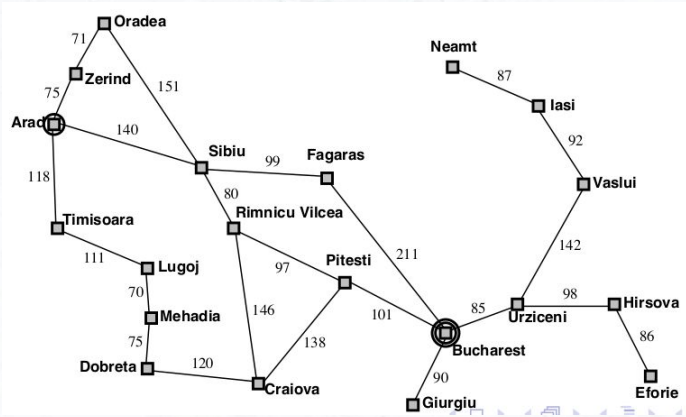


Resolvendo Problemas

Exemplo de Busca em Árvore

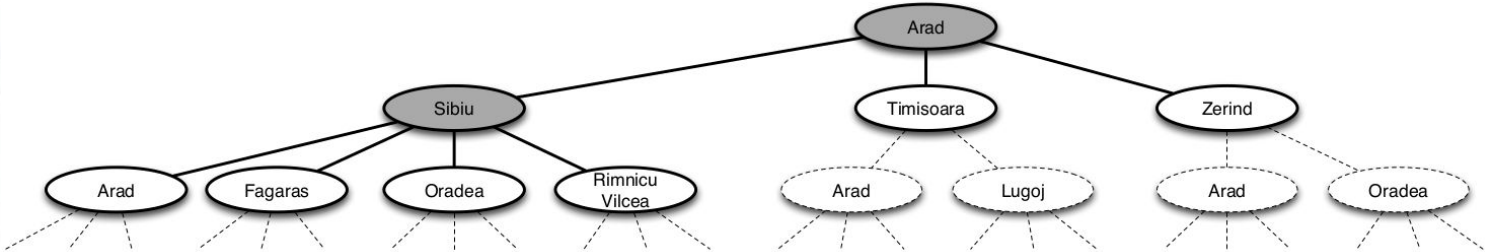


Após expandir Arad

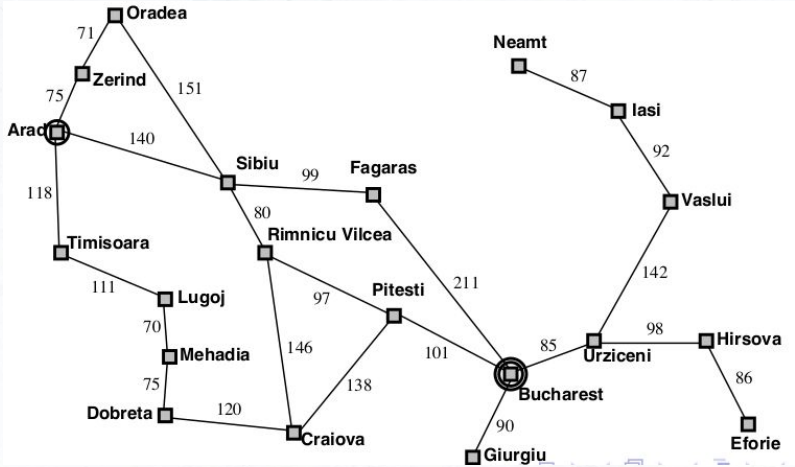


Resolvendo Problemas

Exemplo de Busca em Árvore



Após expandir Sibiu



04 →

Estratégias de Busca

(IA)

Estratégias de Busca

Quão boa é a busca?

- Podemos avaliar estratégias de busca de diversas maneiras
  - **Completeness**: Encontra sempre uma solução, se existir?
  - **Otimalidade**: a solução é ótima (ou seja, de menor custo)?
  - **Complexidade temporal**: Quanto tempo leva para encontrar uma solução?
  - **Complexidade do espaço**: quanta memória é necessária para realizar a pesquisa

Estratégias de Busca

Quão boa é a busca?

A complexidade do tempo e do espaço é medida em termos de:

- b – fator máximo de ramificação da árvore de busca
- d – profundidade da solução de menor custo
- m – profundidade máxima do espaço de estados (pode ser ∞)

Estratégias de Busca

Estratégias de Busca Desinformada

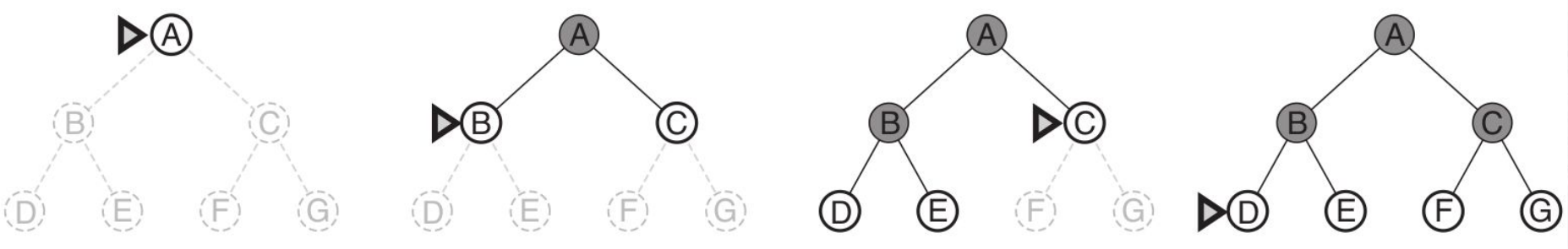
Estratégias Desinformadas usam apenas as informações disponíveis na definição do problema

- Pesquisa ampla (Breadth-first search)
- Pesquisa de custo uniforme (Uniform-cost search)
- Pesquisa em profundidade (Depth-first search)
- Pesquisa com profundidade limitada (Depth-limited search)
- Pesquisa iterativa de aprofundamento (Iterative deepening search)

Estratégias de Busca

Breadth-first search

- Nó raiz expandido primeiro
- Todos os sucessores do nó raiz são então expandidos, depois seus sucessores, etc.
- Todos os nós em uma determinada profundidade são expandidos antes que o próximo nível seja expandido.

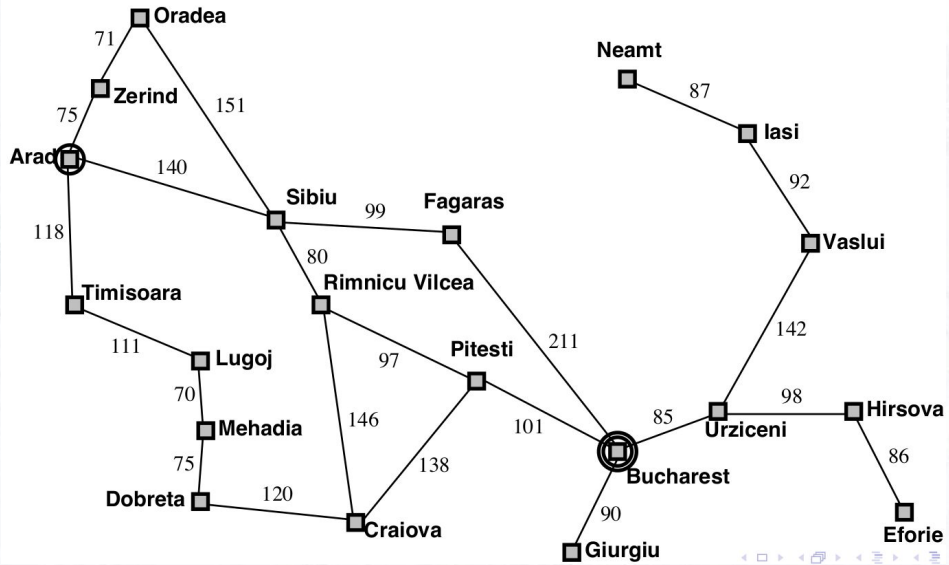


Implementação: franja é um FIFO (fila)

Estratégias de Busca

Breadth-first search

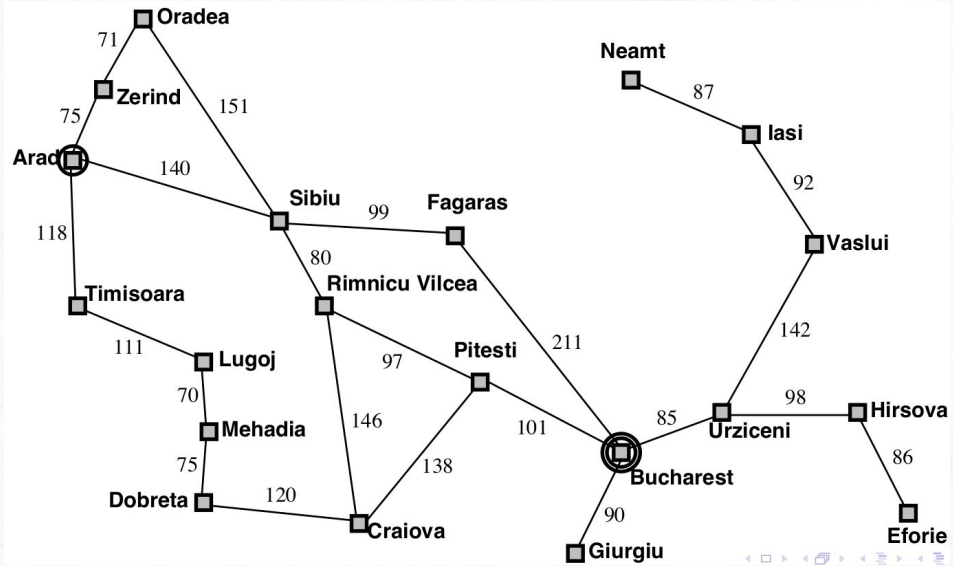
- Complete?
- Tempo?
- Espaço?
- Óptimo?



Estratégias de Busca

Breadth-first search

- Complete? Sim
- Tempo? $1+b+b^2+ \dots +b^d = O(b^d)$
- Espaço? $O(b^d)$ nós gerados
- Ótimo? somente se o custo do caminho for uma função não decrescente da profundidade do nó.



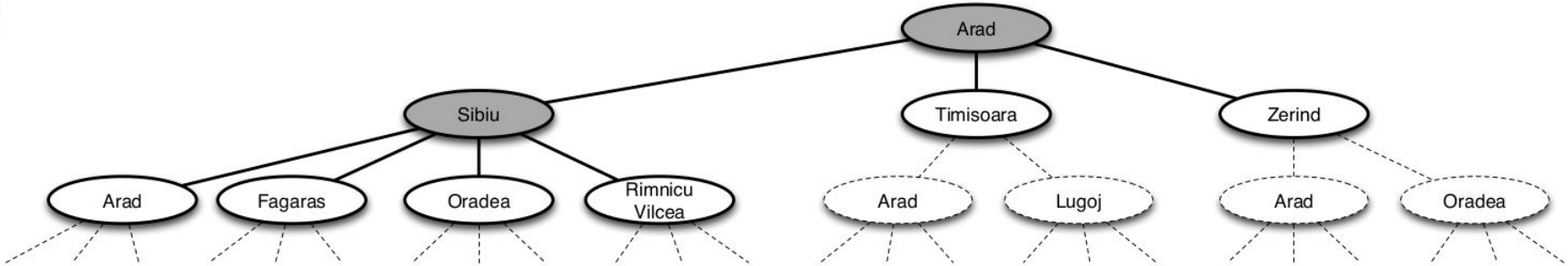
05 →

Busca em Grafo

(IA)

Busca em Grafo

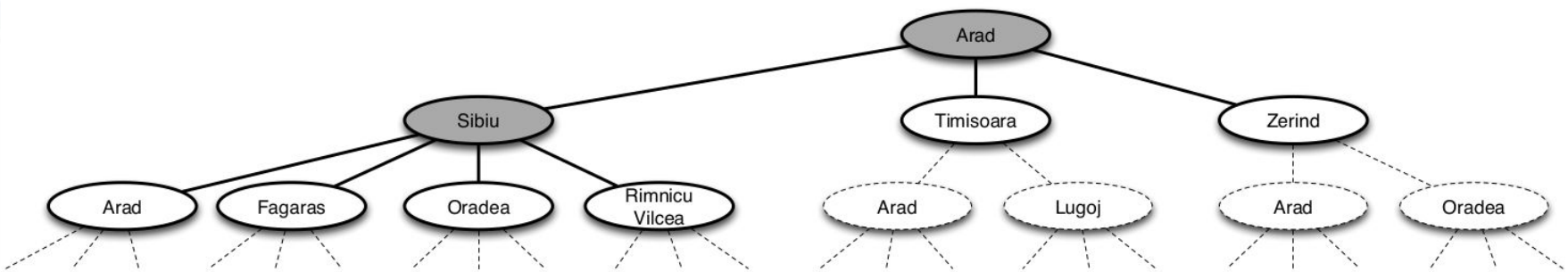
Exemplo de Busca em Árvore



- Lembra do problema da Romênia?
- É um problema finito?

Busca em Grafo

Exemplo de Busca em Árvore

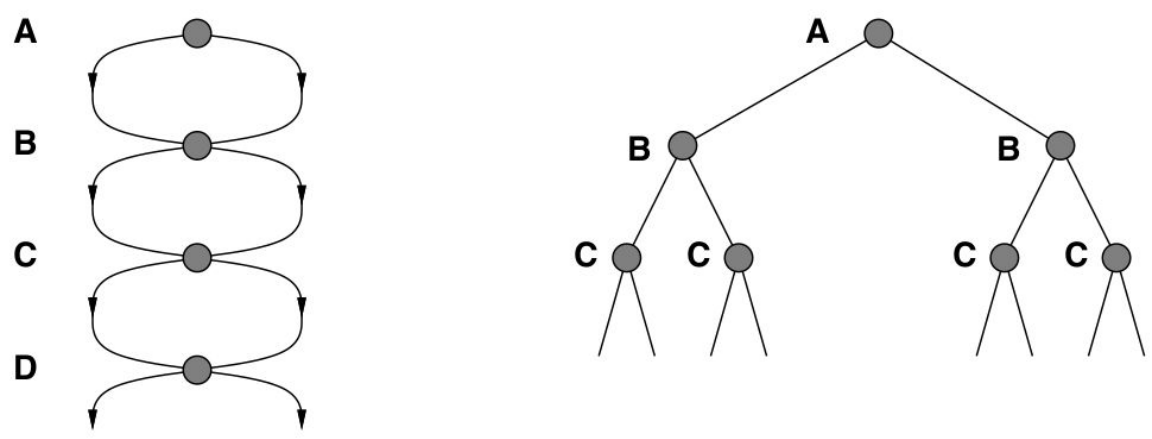


- Lembra do problema da Romênia?
- É um problema finito?
    - Estados repetidos na árvore tornam ele infinito.

Busca em Grafo

Estados repetidos

Não detectar estados repetidos pode tornar um problema linear em um problema exponencial!



Busca em Grafo

Busca em Grafo

- Temos procurado através das árvores até agora.
- Nos grafos, podemos expandir estados que já foram encontrados e expandidos antes.
- Sem detectar estados repetidos, podemos fazer um loop, tornando insolúveis problemas que de outra forma seriam solucionáveis.
- Lembrando cada estado expandido e descartando qualquer expansão que revisite esse estado, podemos evitar que esse problema ocorra.
- Armazenamos essas estatísticas expandidas em uma lista fechada.

Busca em Grafo

Busca em Grafo

```
1.  function graphSearch(problema p, estratégia s)
2.    fechado ← {}                # Mantemos uma lista de estados explorados.
3.    frontera.add(novoNode(p.inicial))
4.    loop
5.      if fronteira está vazia then
6.        return fail
7.      n ← s.removeEscolha(fronteira)
8.      fechado.add(n.estado)          # Onde salvamos estados já visitados.
9.      if p.testeObjetivo(n.estado) then
10.       return getCaminho(n)
11.     for all a ∈ p.ações(n) do
12.       n' ← a.resultado(n.estado)
13.       if n'.estado ∉ fechado then
14.         fronteira.add(n')          # E explore apenas os que não visitamos ainda.
```

Busca em Grafo

Busca em Grafo

```
1.  function graphSearch(problema p, estratégia s)
2.    fechado ← {}                # Mantemos uma lista de estados explorados.
3.    frontera.add(novoNode(p.inicial))
4.    loop
5.      if fronteira está vazia then
6.        return fail
7.      n ← s.removeEscolha(fronteira)
8.      fechado.add(n.estado)     # Onde salvamos estados já visitados.
9.      if p.testeObjetivo(n.estado) then
10.       return getCaminho(n)
11.     for all a ∈ p.ações(n) do
12.       n' ← a.resultado(n.estado)
13.       if n'.estado ∉ fechado then
14.         fronteira.add(n')     # E explore apenas os que não visitamos ainda.
```

Busca em Grafo

Eficiência

- Existem muitas estruturas de dados que podemos usar para tornar as coisas mais eficientes. Por exemplo:
  - Tabela hash para o conjunto fechado;
  - Fila de prioridade para determinar qual nó expandir na pesquisa de custo uniforme;
  - Acompanhar o caminho mais curto para cada nó;
  - etc...

06 →

Estratégias de Busca Continuadas

(IA)

Estratégias de Busca Continuada

Propriedades da busca Breadth-first

- A pesquisa em largura é ideal quando todos os custos das etapas são iguais, pois sempre expande os nós não expandidos mais rasos.
- E se tivermos diferentes funções de **custo escalonado**?

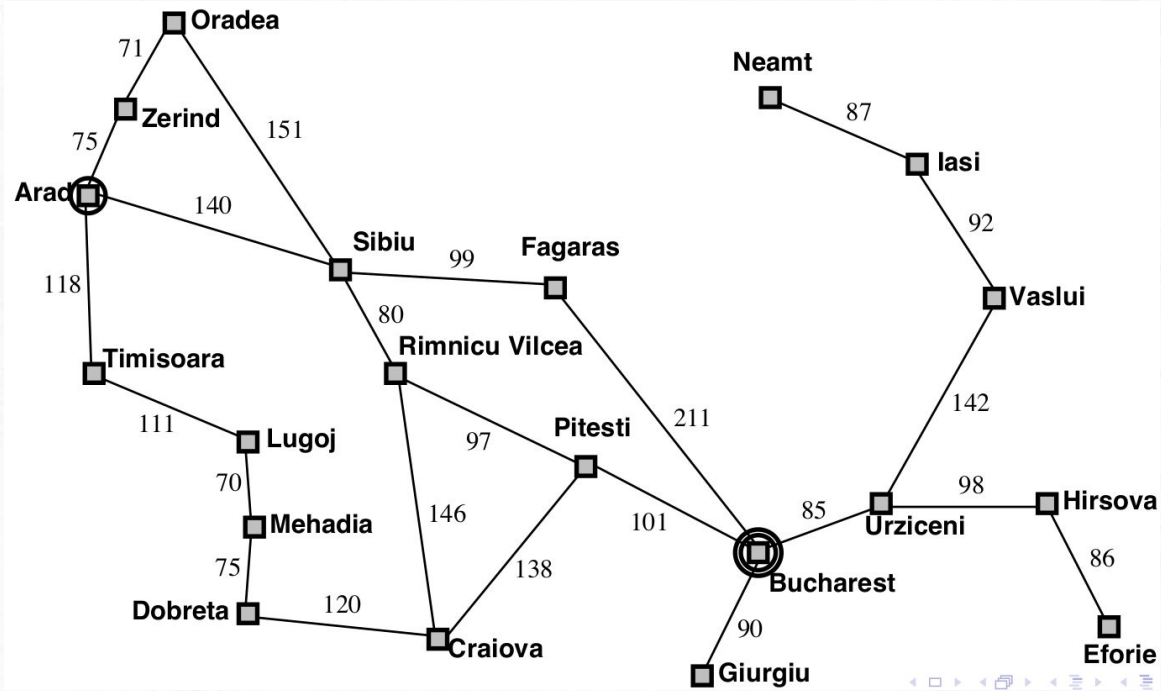
Estratégias de Busca Continuada

Busca de Custo Uniforme

- A pesquisa em largura é ideal quando todos os custos das etapas são iguais, pois sempre expande os nós não expandidos mais rasos.
- E se tivermos diferentes funções de **custo escalonado**?
- Expandimos o nó com menor custo de caminho.
- Para completar, precisamos sempre ter um custo de caminho (caso contrário, podemos fazer um loop para sempre).
- Implementação: a margem é um FIFO ordenado por custo, o menor custo primeiro (fila de prioridade)

Estratégias de Busca Continuada

Busca de Custo Uniforme



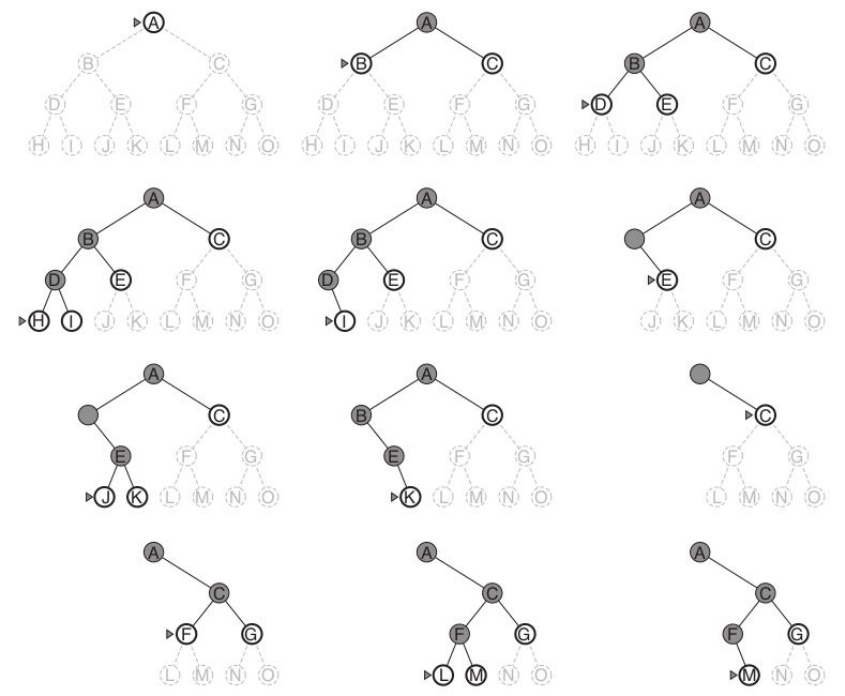
Estratégias de Busca Continuada

Busca Depth-first

- Este tipo de busca expande o nó mais profundo da fronteira atual.
- Quando o fim é alcançado, a busca volta para o próximo nó mais raso que ainda possui sucessores inexplorados.
- Implementação: a franja é um LIFO (pilha)
- O que precisamos armazenar na memória em comparação com BFS e UCS?

Estratégias de Busca Continuada

Busca Depth-first



Estratégias de Busca Continuada

Propriedades da Busca Depth-first

- Complete?
- Tempo?
- Espaço?
- Ótimo?

Estratégias de Busca Continuada

Propriedades da Busca Depth-first

- Complete? Não: falha em espaços de profundidade infinita, espaços com loops. Modifique para evitar estados repetidos ao longo do caminho → completo em espaços finitos
- Tempo – $O(b^m)$ terrível se **m** for muito maior que **d**, mas se as soluções forem densas, podem ser muito mais rápidas do que as soluções em largura
- Espaço – O(bm), ou seja, espaço linear!
- Ótima – Não

07 →

Estratégias de Busca Combinadas

(IA)

Estratégias de Busca Combinada

Busca Depth-limited

```
1.  function depthLimitedSearch(problema p, limite l)
2.    recursiveDLS(newNode(p.inicial), p, l)
3.  function recursiveDLS(node n, problema p, limite t)
4.    cutoffOcorreu ← false
5.    if p.testeObjetivo(n.estado) then
6.      return getCaminho(n)
7.    else if node.depth = l then
8.      return cutoff
9.    else
10.     for all sucessor in p.expandido(n) do
11.       resultado ← recursiveFLS(sucessor, p, l)
12.       if resultado = cutoff then
13.         cutoffOcorreu ← true
14.       else if result ≠ failure then
15.         return  resultado
16.     if cutoffOcorreu then
17.       return  cutoff
18.     else
19.       return failure
```

DFS com limite L de profundidade, ou seja, nodos na profundidade L não têm sucessores.

Estratégias de Busca Combinada

Busca Depth-limited

- Esta busca resolve o problema do caminho infinito do DFS
- Mas a incompletude é introduzida se o objetivo mais raso estiver além do limite de profundidade.
- No exemplo de encontrar a rota, se sabemos que existem 20 cidades, sabemos que a profundidade máxima é 19.
- Mas examinando o problema, vemos que qualquer cidade pode ser alcançada em profundidade 9. Este é o **diâmetro** do espaço de estados.

Estratégias de Busca Combinada

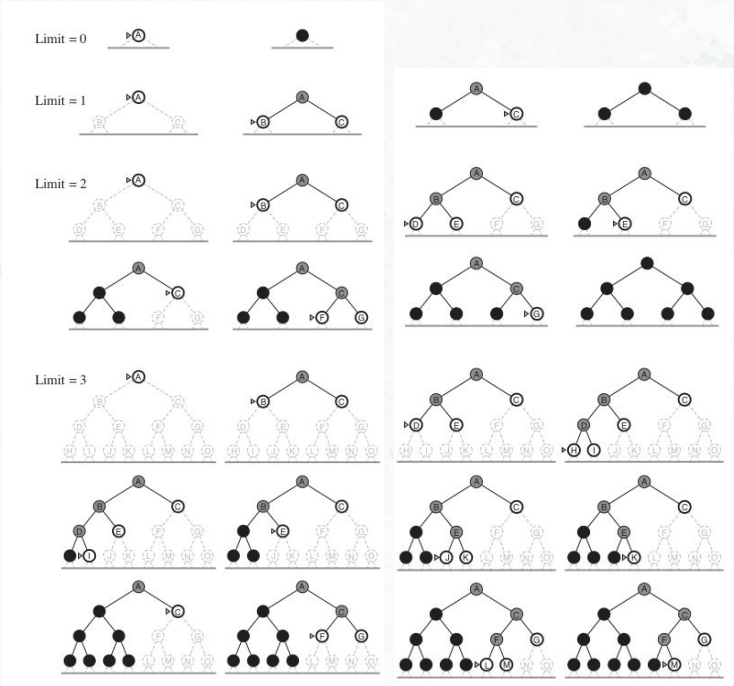
Busca Depth-first com Aprofundamento Iterativo

```
1.  function IterativeDeepeningSearch(problema p)
2.      for all depth ← 0 to ∞ do
3.          resultado ← DepthLimitedSearch(problema, depth)
4.          if resultado ≠ cutoff then
5.              return resultado
```

- Este atua como um DFS com um limite crescente gradualmente até que um objetivo seja encontrado.
- Combina benefícios de DFS e BFS.
- Observe que os estados são gerados diversas vezes. Mas isso não é caro, pois a maioria dos nós está no nível inferior.

Estratégias de Busca Combinada

Busca Depth-first com Aprofundamento Iterativo



Estratégias de Busca Combinada

Propriedades da Busca Aprofundada Iterativa

- Completo?
- Tempo?
- Espaço?
- Ótimo?

Estratégias de Busca Combinada

Propriedades da Busca Aprofundada Iterativa

- Completo? Sim
- Tempo? $(d)b^0 + (d-1)b^1 + (d-2)b^2 + (d-3)b^3 + \cdots + (!)b^d = O(b^d)$
- Espaço? $O(bd)$
- Ótimo? Sim, se o custo do passo = 1

Estratégias de Busca Combinada

Resumo da Busca Desinformada

- Agentes de resolução de problemas
  - Tipos de problemas
  - Formulação de problema
- Busca Desinformada
  - Busca em árvore e gráfico
  - Estratégias de busca desinformadas