

Slide 54 →

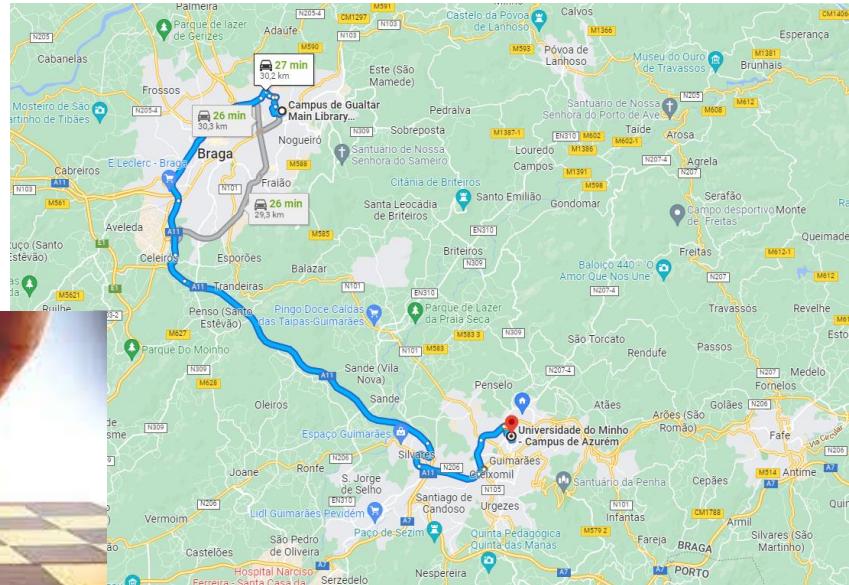
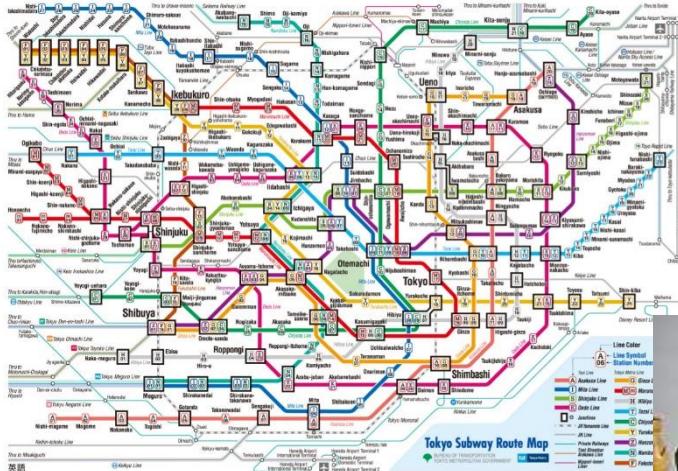
MÉTODOS DE RESOLUÇÃO DE PROBLEMAS E DE PROCURA

LICENCIATURA EM ENGENHARIA INFORMÁTICA
MESTRADO integrado EM ENGENHARIA INFORMÁTICA

Inteligência Artificial
2022/23

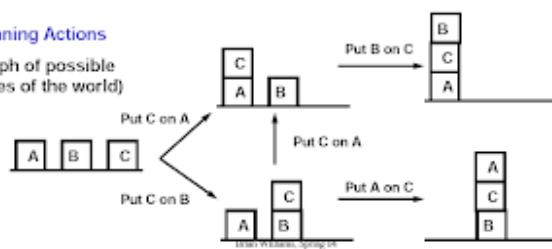
- Formulação de Problemas
- Resolução de problemas
- Tipos de problemas
- Exemplos de problemas do mundo real
- Procura de soluções
- Estratégias de procura
 - Procura Não-Informada (cega)
 - Procura Informada (heurística)
- Para além da procura Clássica



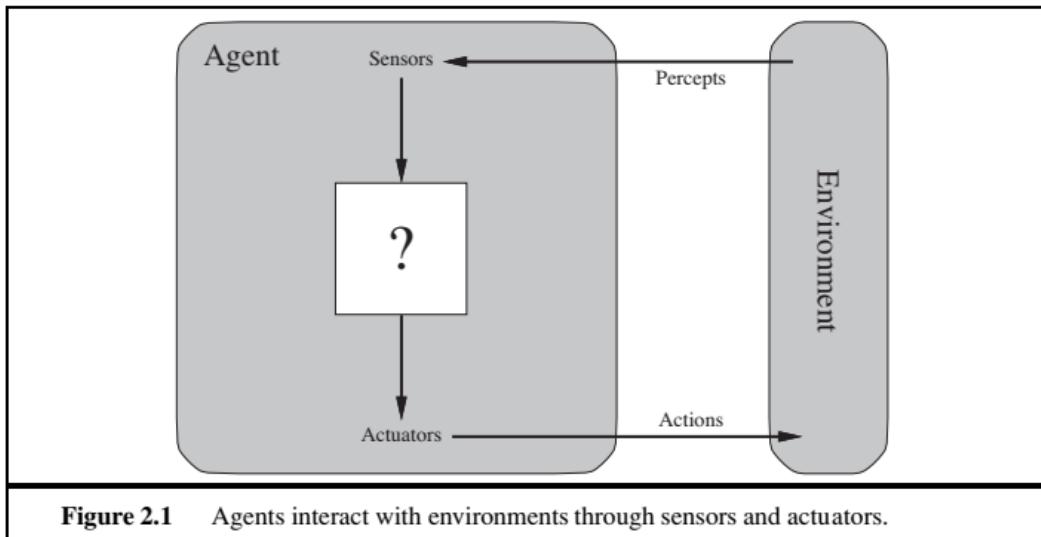


Planning Actions

(graph of possible states of the world)



- **Agente:** é algo que que **perceciona o ambiente** através de sensores e atua no ambiente através de atuadores



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- O termo *perceção* refere-se às percepções do agente num dado instante. Uma *sequência de percepções* do agente é o histórico total de tudo o que agente percepcionou.
- O comportamento do agente é descrito matematicamente pela *função do agente* que mapeia qualquer sequência de percepções numa ação.
- Agente : arquitetura* + programa**

* dispositivo computacional com sensores e atuadores

** materialização da função do agente

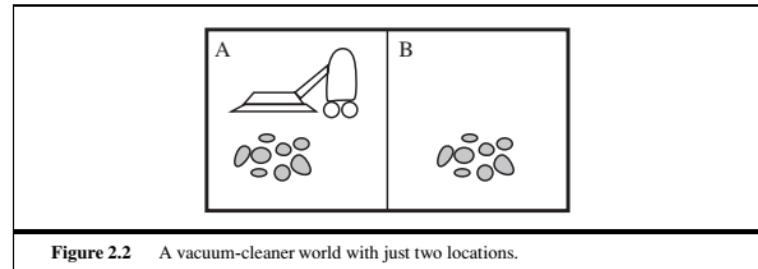


Figure 2.2 A vacuum-cleaner world with just two locations.

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
:	:
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
:	:

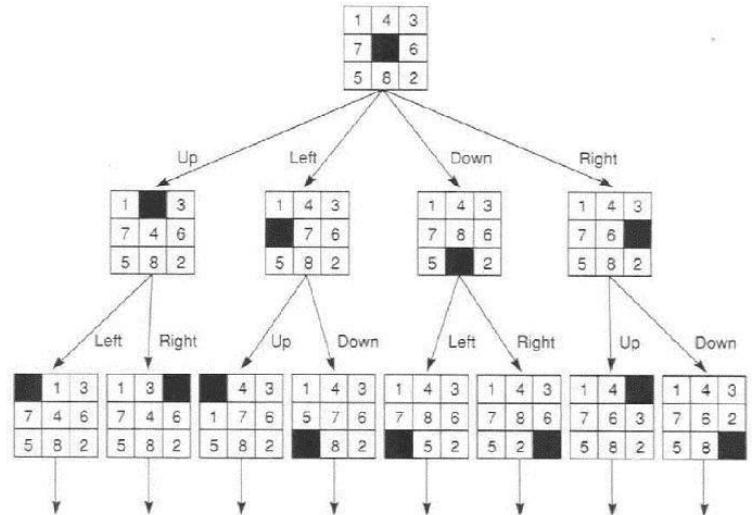
Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2.

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Como é que um agente pode agir, estabelecendo objetivos e considerando possíveis sequências de ações para atingir esses objetivos?
- A concretização de um simples agente que resolve problemas necessita primeiramente de **formular um objetivo e um problema**, procurar a sequência de ações que resolvam o problema, executando-as uma de cada vez. Quando completo, formulará um outro objetivo e executará de novo.
- Portanto, considera-se que neste contexto a resolução de problemas apresenta-se como a formulação de um problema como **um problema de procura**.



- “Problem Solving Agent”: Procura encontrar a sequência de ações que leva a um estado desejável!
- Formulação do Problema:
 - Quais as ações possíveis? (qual o seu efeito sobre o estado do mundo?)
 - Quais os estados possíveis? (como representá-los?)
 - Como avaliar os Estados
- Problema de procura:
 - Solução: sequência de ações
- Fase final é a execução!
- **Formular → procurar → Executar**



- Muitos dos problemas em ciências da computação podem ser formulados como:
 - Um conjunto S de ESTADOS (possivelmente infinito)
 - Um estado INICIAL $s \in S$
 - Uma relação de TRANSIÇÃO T ao longo deste espaço de estados
 - Um conjunto de estados FINAIS (objetivos): $O \subseteq S$
- Um problema pode ser definido formalmente em cinco componentes:
 1. Representação do Estado
 2. Estado Inicial (Atual)
 3. Estado Objetivo (define os estados desejados)
 4. Operadores (Nome, Pré-Condições, Efeitos, Custo)
 5. Custo da Solução

Resolução de problemas

- O problema pode ser resolvido através da procura e um caminho entre o estado inicial e um estado objetivo
- Em suma, a formulação do problema envolve decidir que ações e estados a considerar tendo em conta um objetivo

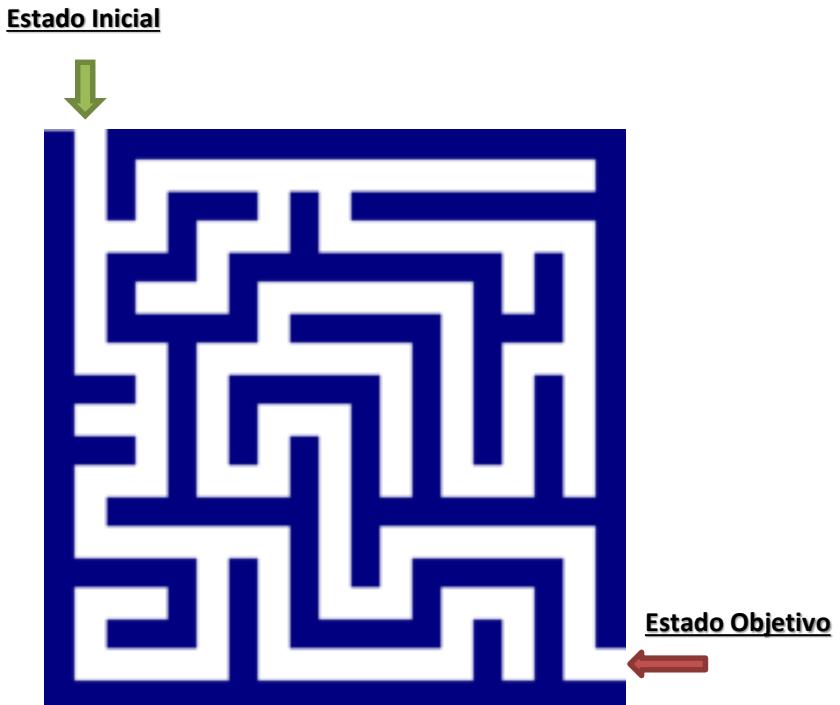
```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation

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

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Formulação de problemas Componentes de um problema de procura

- Estado inicial
- Ações
- Modelo de transição
 - Que estado resulta da execução de uma determinada ação em um determinado estado?
- Estado Objetivo
- Custo do caminho
 - Suponha que seja uma soma dos custos não negativos de cada etapa
- A solução ideal é a sequência de ações que fornece o menor custo de caminho para alcançar a meta



Formulação de problemas

Exemplo: Viajar de Arad para Bucharest

Estado Inicial: Arad;

Estado Objetivo: Bucharest

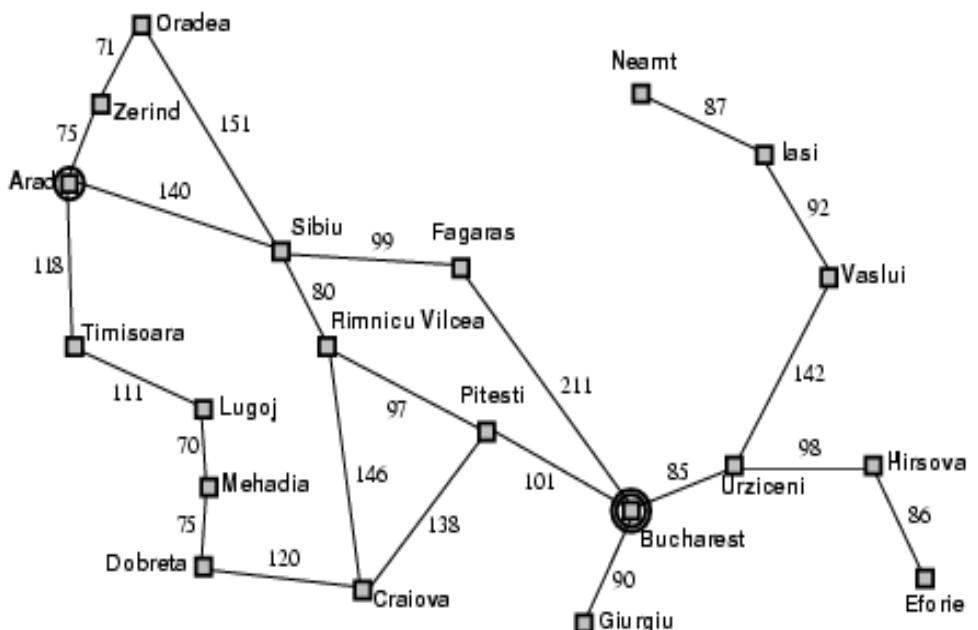
Estados: as várias cidades

Operações: conduzir entre as cidades

Solução:

- Um caminho*: sequência de cidades, e.g., Arad, Sibiu, Fagaras, Bucharest
 - O caminho mais curto
 - O caminho mais rápido
 - O caminho mais “verde”

* O custo da solução é determinado pelo custo de cada caminho, que reflete uma medida de desempenho da solução (neste caso o custo será a distância entre cidades).



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Uma solução para um problema é o caminho do estado inicial para o estado do objetivo;
- A qualidade da solução é medida pela função de custo do caminho e uma solução “ideal” tem o menor custo do caminho entre todas as possíveis soluções;
- O mundo “real” é obviamente mais complexo:
 - o espaço de estado é um abstração para a solução de problemas;
- Estado (abstrato) = conjunto de estados reais;
- Ação (abstrata) = combinação complexa de ações reais;
 - Eg., "Arad \Rightarrow Zerind" representa (uma abstração) um conjunto mais complexo de rotas possíveis, desvios, paragens, etc.
- Para garantia de realização, qualquer estado “em Arad” deve chegar a algum estado “em Zerind”;
- Solução (abstrata) = conjunto de caminhos reais que são soluções no mundo real;
- Cada ação abstrata deve ser “mais fácil” do que o problema original.

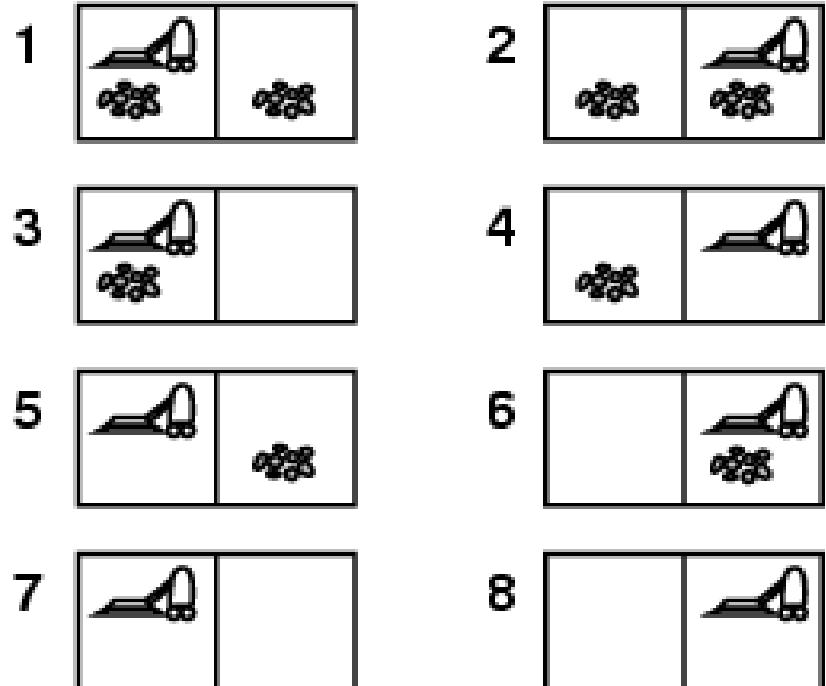
- Ambiente determinístico, totalmente observável → **problema do estado único**
 - O agente “**sabe**” exatamente o estado em que estará; a solução é uma sequência.
- Ambiente determinístico, não acessível → **problema de múltiplos estados**
 - O agente “**não sabe**” onde está; a solução é uma sequência
- Ambiente não determinístico e/ou parcialmente acessível → **problema de contingência**
 - Perceções fornecem novas informações sobre o estado atual
 - Frequentemente intercalam procura e execução
- Espaço de estados desconhecido → **problema de exploração**

Tipos de problemas

Exemplo: o problema do aspirador

- 2 localizações (lixo e não lixo)
- 3 ações (left, right, suck)
- Objetivo: limpar o lixo
- Custo: uma unidade por ação

- Problema de:
 - Estado Único se...
 - Múltiplos Estados se...
 - Contingência se...
 - Exploração se...



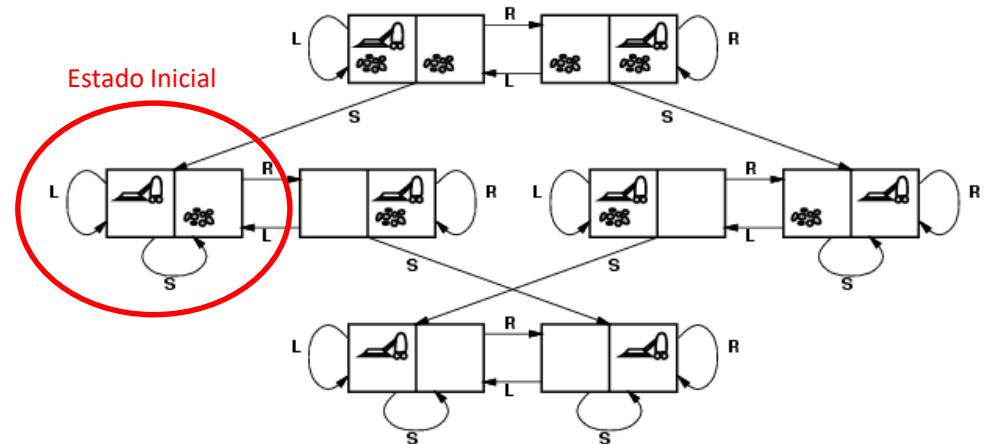
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Exemplo: o problema do aspirador (problema de estado único)

Se o ambiente é completamente observável, o aspirador saberá sempre onde está e onde está o lixo.

A solução é então reduzida à procura de um caminho do estado inicial até ao estado objetivo.

Solução? [Right, Suck]

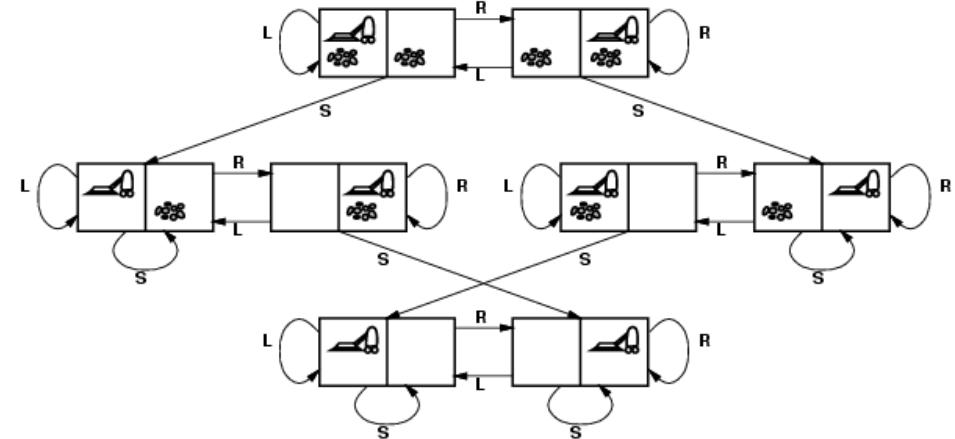


Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Exemplo: o problema do aspirador (problema de estado único)

Formulando o problema:

- Estado: 8 estados representados (definidos pela posição do robô e lixo)
- Estado inicial: Qualquer um
- Operadores: esquerda, direita, aspirar
- Teste Objetivo: Não há lixo em nenhum dos quadrados
- Custo da Solução: Cada ação custa 1 (custo total = número de passos da solução)



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

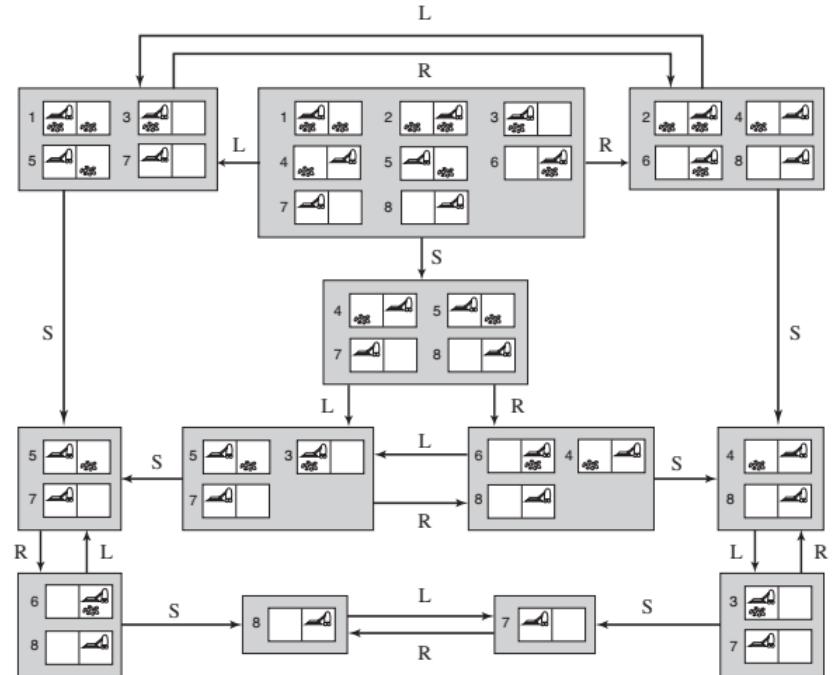
Tipos de problemas

Exemplo: o problema do aspirador (problema de estados múltiplos)

Se o aspirador não tem sensores, então não saberá onde está nem onde está o lixo. Mesmo assim, conseguirá resolver o problema.

Solução? {1,2,3,4,5,6,7,8}

- Right – {2,4,6,8}
- Suck – {4,8}
- Left – {3,7}
- Suck – {7}



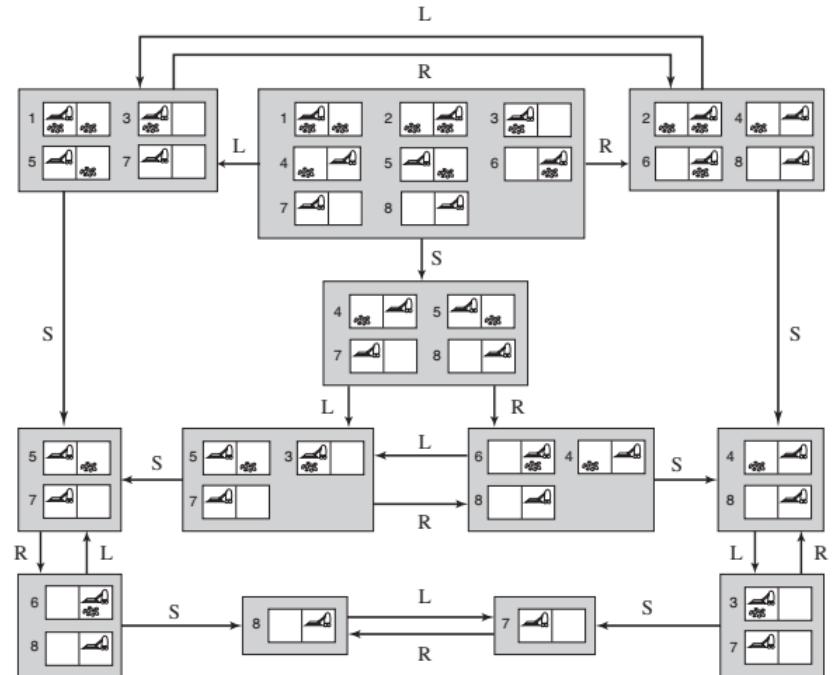
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Tipos de problemas

Exemplo: o problema do aspirador
 (problema de estados múltiplos)

Formulando o problema:

- Conjunto de Estados: subconjunto dos estados representados
- Operadores: esquerda, direita e aspirar
- Teste Objetivo: Todos os estados do conjunto não podem ter lixo
- Custo da Solução: Cada ação custa 1



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Exemplo: o problema do aspirador

(problema de contingência)

O agente não sabe qual o efeito que terá suas ações, isto devido ao ambiente ser parcialmente observável.

Portanto, a sequência de ações deve ser planeada mediante cada percepção do ambiente, ou seja, por cada ação realizada o agente recolhe informação através do seu sensor e só depois decide a próxima ação a executar.

A solução será uma lista de ações condicionais que intercalará procura e execução.

Exemplo: Iniciar em {5} ou {7} – [Right, se tiver lixo então Suck]

Exemplo: o problema do aspirador (problema de contingência)

O agente não sabe qual o efeito que terá suas ações, isto devido ao ambiente ser parcialmente observável. Portanto, a sequência de ações deve ser planeada mediante cada percepção do ambiente, ou seja, por cada ação realizada o agente recolhe informação através do seu sensor e só depois decide a próxima ação a executar.

A solução será uma lista de ações condicionais que intercalará procura e execução.

Exemplo: Iniciar em {5} ou {7} – [Right, se tiver lixo então Suck]

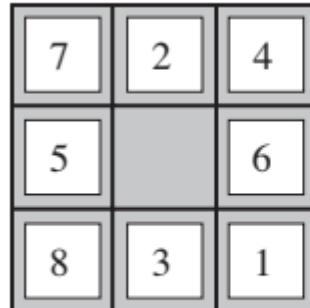
Exemplo: o problema do aspirador (problema de exploração)

É um caso extremo de um problema de contingência que é resolvido considerando informação adicional para uma solução que intercale procura e execução.

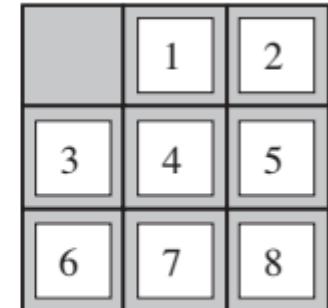
Exemplo de Problemas do Mundo Real

Problema puzzle de 8 peças

- Estados: descrição da localização das oito peças e quadrado em branco
- Estado Inicial: a configuração inicial do puzzle
- Ações: Movimentar o quadrado branco para esquerda, direita, cima e para baixo
- Estado Objetivo: estado que corresponde à configuração do puzzle à direita
- Custo: cada passo custa 1 unidade, o custo da solução é o número de passos para resolver o problema



Start State



Goal State

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Exemplo de Problemas do Mundo Real

Problema de Rotas/Caminhos

- Encontrar o melhor caminho de um ponto a outro (aplicações: google maps, redes de computadores, planeamento militar, viagens aéreas)
- Visitar cada ponto pelo menos uma vez num dado espaço (Ex: Caixeiro viajante visitar cada cidade exatamente uma vez, encontrar o caminho mais curto)

Outros:

- Navegação autónoma (com alguns graus de liberdade)
 - A dificuldade é incrementada rapidamente com o número de graus de liberdade. Possíveis complicações incluem: erros de percepção do ambiente, ambientes desconhecidos, etc.
- Sequênciação da montagem automática
 - Planeamento da montagem de objectos complexos (por robôs)
 - etc.

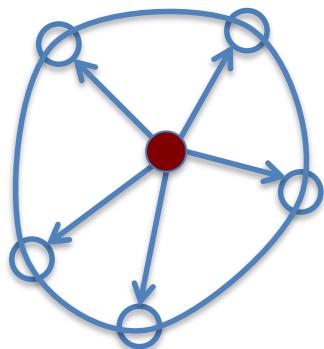
- Uma solução para um dado problema é definida como a sequência de ações desde o estado inicial até ao estado objetivo. A “qualidade” da solução é medida através da função do custo do caminho e pode ser:
 - Uma solução **satisfatória** se é uma qualquer solução;
 - uma solução **semi-óptima** é aquela que tem aproximadamente o menor custo entre todas as soluções;
 - uma solução **óptima** é aquela que tem o menor custo entre todas as soluções.

- Começando no estado inicial (nodo) e expandi-lo, fazendo uma lista de todos os possíveis estados sucessores;
- Manter uma lista de estados (nodos) não expandidos;
- Em cada etapa, escolha um estado da lista de estados não expandidos para expandir;
- Continue até atingir o estado objetivo;
- Tente expandir o menor número possível de estados.

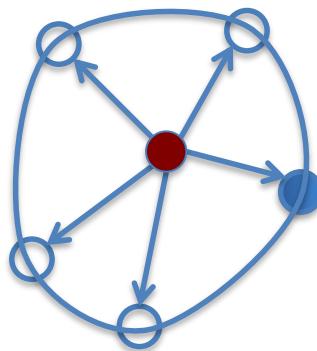


start

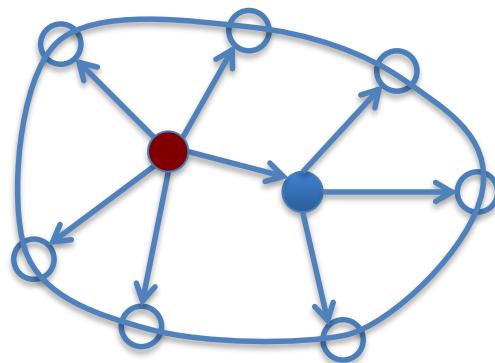
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



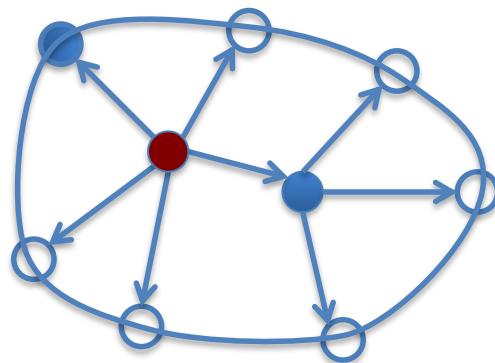
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



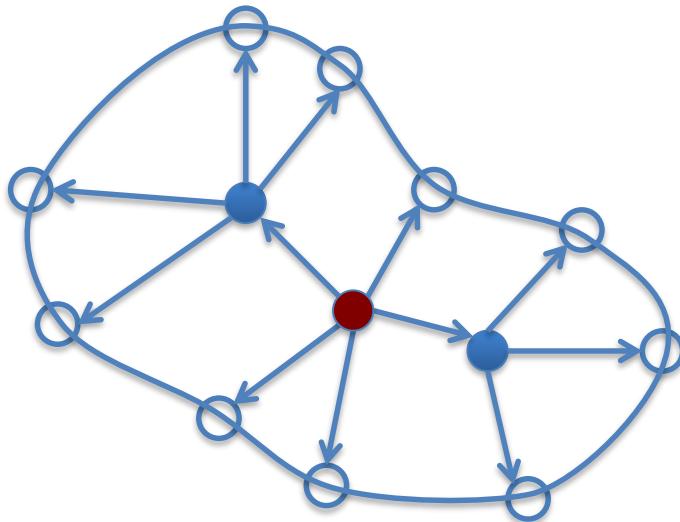
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



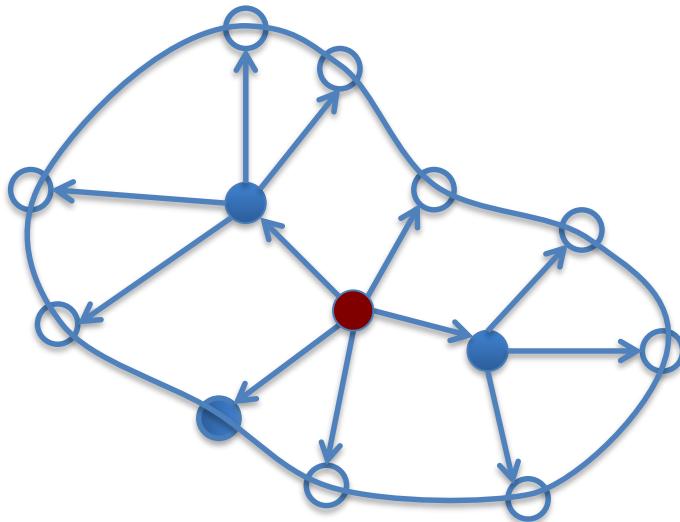
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



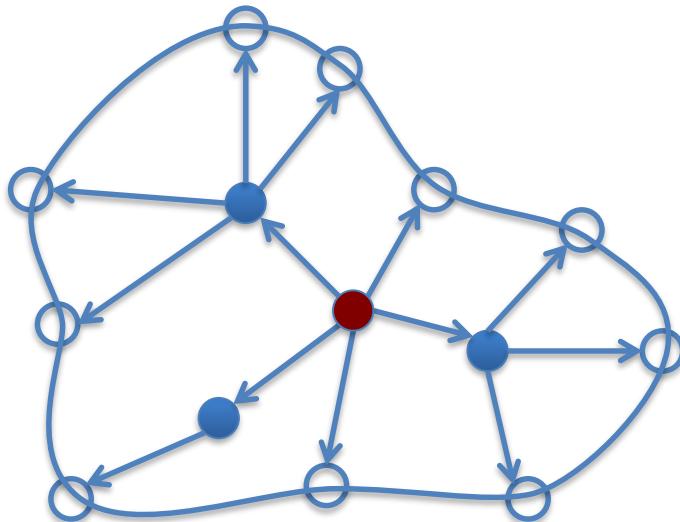
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



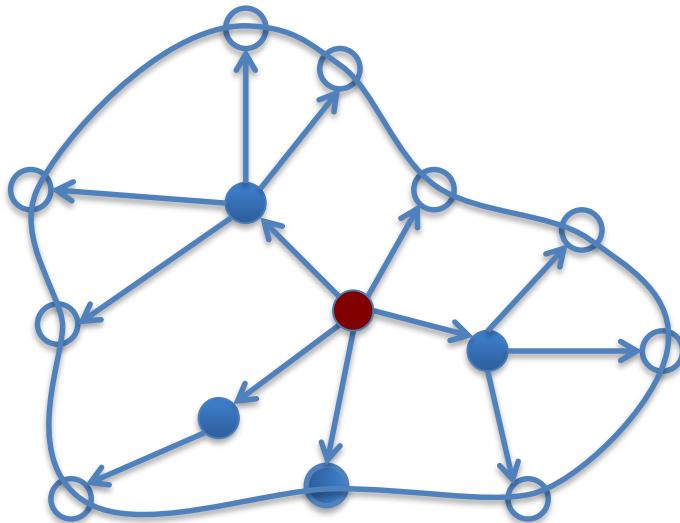
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



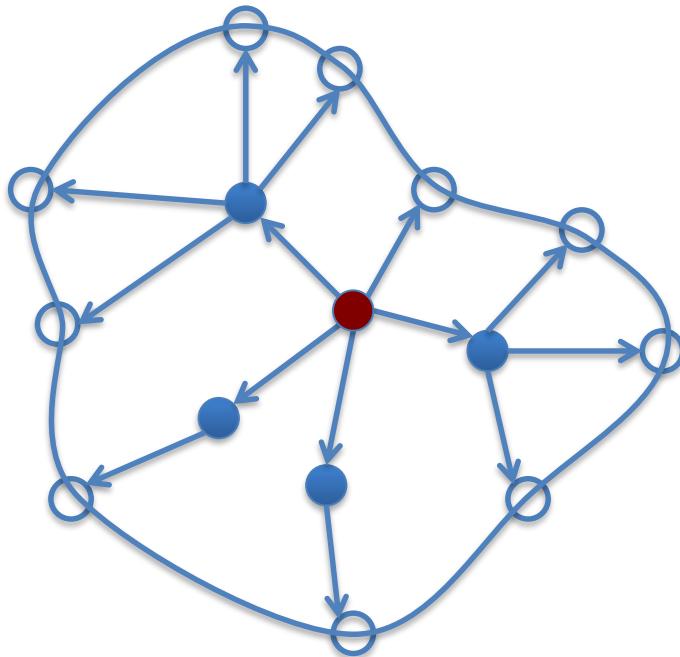
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



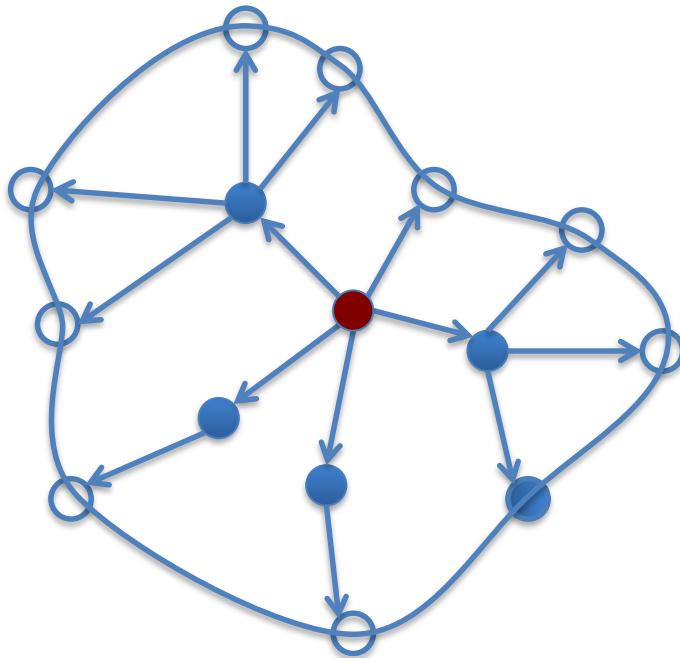
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



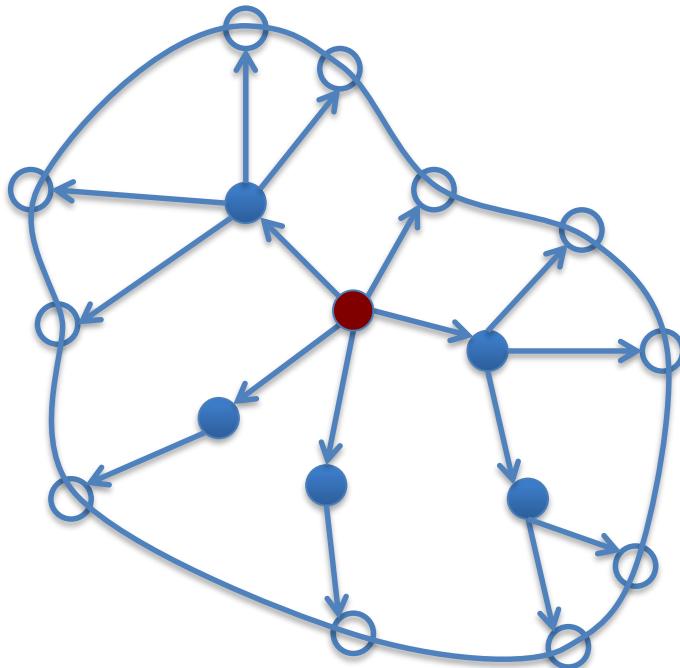
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.



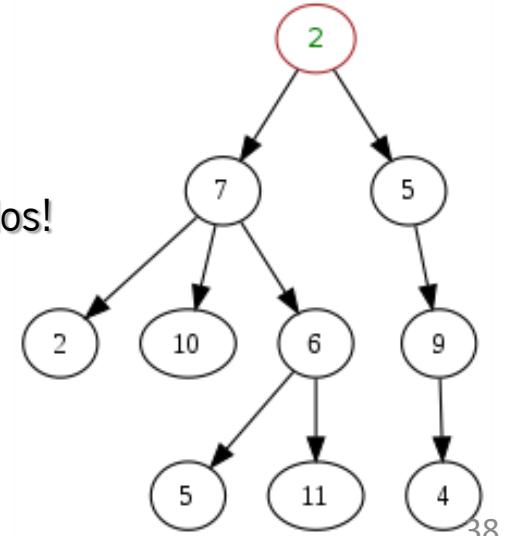
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Metodologia para realizar a Procura da Solução:

1. Começar com o estado inicial
2. Executar o teste do objetivo
3. Se não foi encontrada a solução, usar os operadores para expandir o estado atual gerando novos estados-sucessores (expansão)
4. Executar o teste do objetivo
5. Se não tivermos encontrado a solução, escolher qual o estado a expandir a seguir (estratégia de procura) e realizar essa expansão
6. Voltar a 4.

Procura da Solução Árvore de Procura

- Através do **espaço de estados** do problema podemos formar uma **árvore de procura** que nos auxilie a encontrar a solução.
- O **estado inicial** determina o **nó raiz** da árvore e os **ramos** de cada nó são as **ações possíveis** a partir do nó (estado) para as folhas (próximos estados).
- Portanto, é uma árvore de procura composta por nós.
Nós folha ou não têm sucessores ou ainda não foram expandidos.
- Importante distinguir entre a árvore de procura e o espaço de estados!



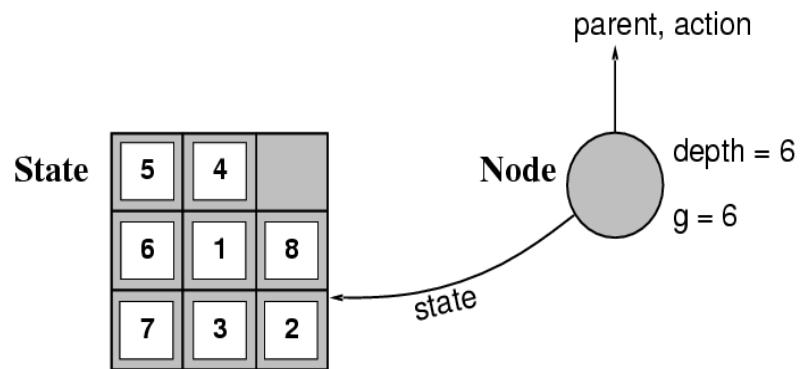
Estados versus Nós

- Um **estado** é a representação de uma configuração física
- Um **nó** é uma estrutura de dados constituído por parte da árvore de procura que inclui **estado**, **nó pai***, **ação****, **custo do caminho $g(x)$ *****, **profundidade**.

*nó que lhe deu origem

** operador aplicado para o gerar

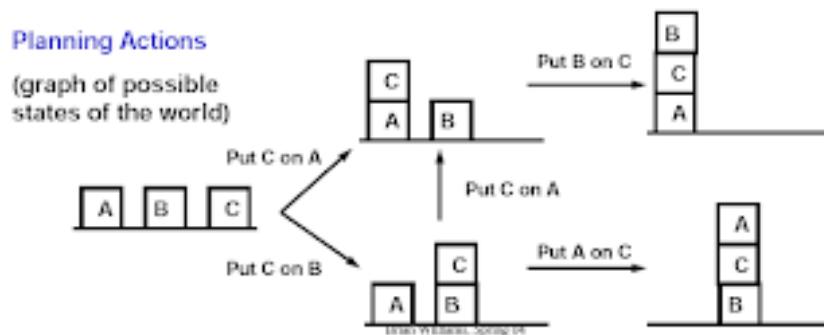
*** custo do caminho desde o nó inicial



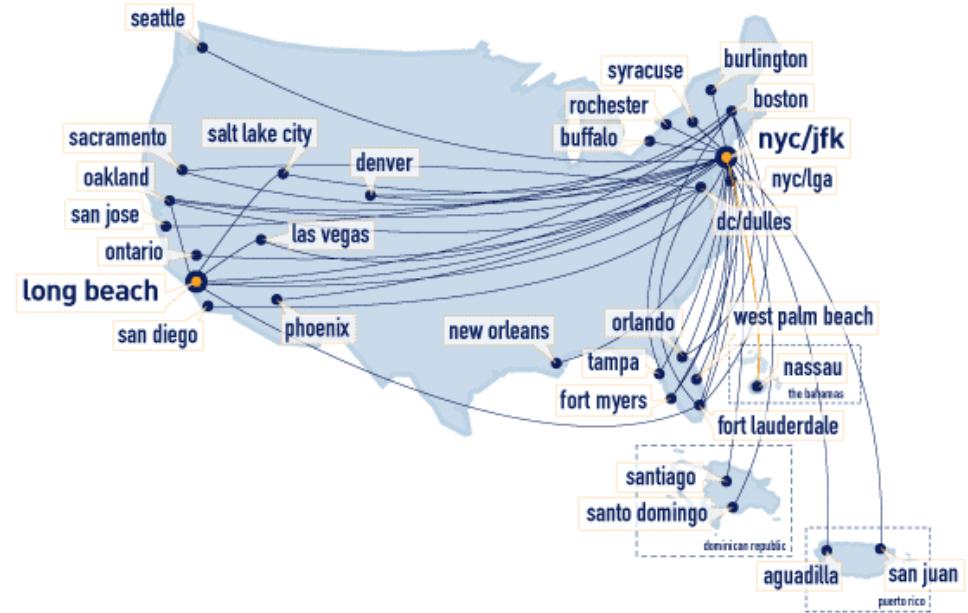
Ao expandir a árvore criam-se novos nós, preenchendo os vários campos e utilizando os “sucessores” do problema para criar os estados correspondentes

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

Exemplos de Grafos



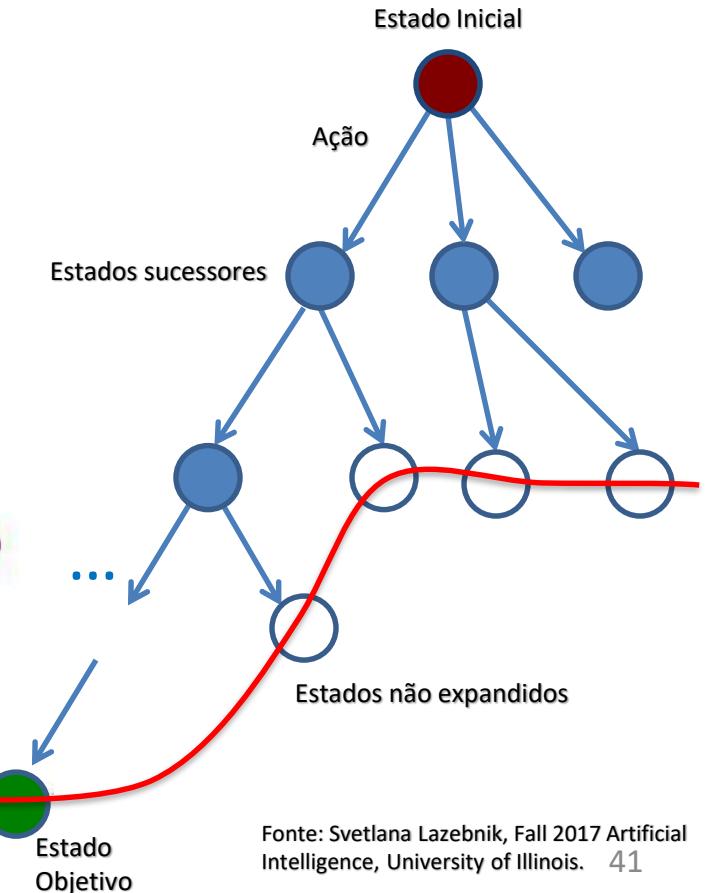
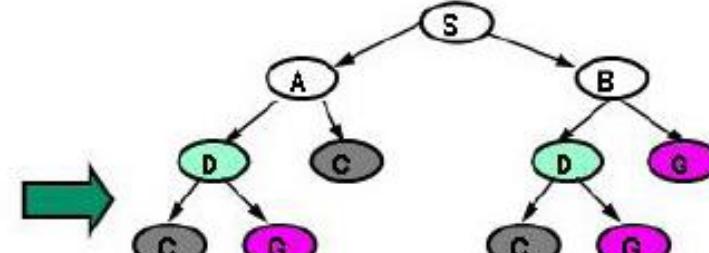
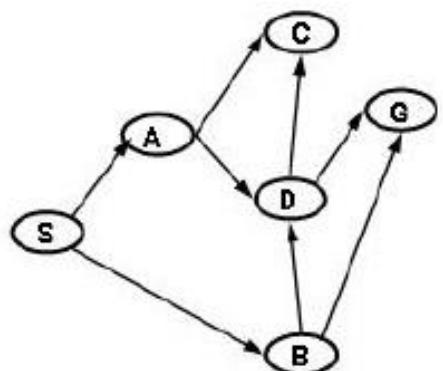
Fonte: Brian Williams, 2014



Fonte: Jet Blue airlines: <http://www.jetblue.com/travelinfo/routemap.html>

Grafos de procura como Árvores de procura

- Árvores são grafos sem ciclos e em que os nodos têm ≤ 1 pai;
- Podemos transformar problemas de procura em grafos (de S a G) em problemas de procura em árvore:
 - substituindo links não direcionados por 2 links direcionados;
 - evitando loops no caminho (ou monitorizando os nós visitados globalmente).



Fonte: https://ocw.mit.edu/courses/health-sciences-and-technology/hst-947-medical-artificial-intelligence-spring-2005/lecture-notes/ch2_search1.pdf

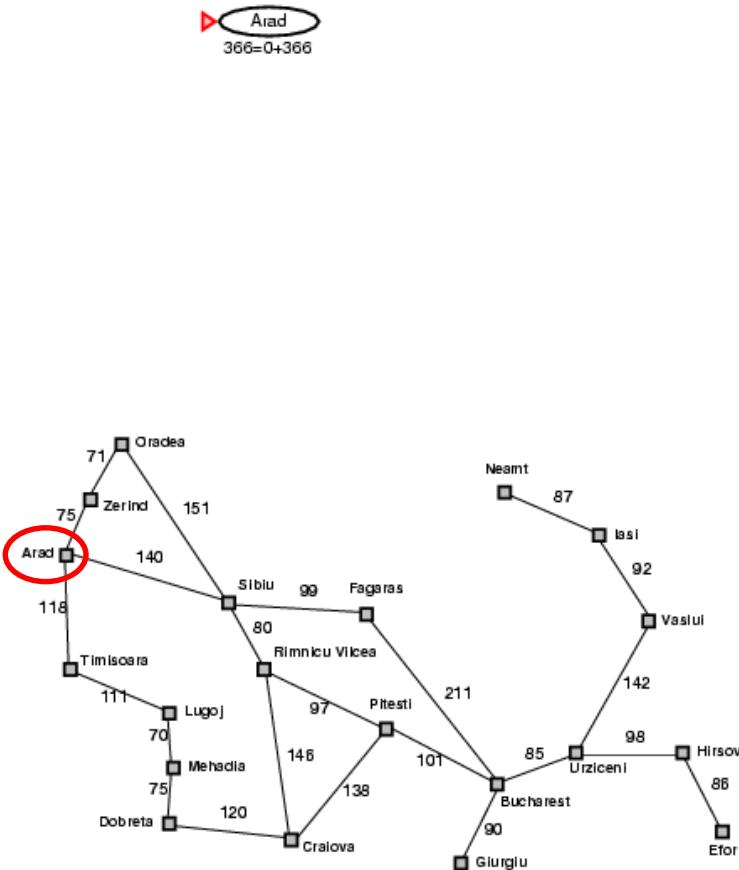
Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois. 41

Algoritmo de procura em árvore

- Inicialize a lista de estados não expandidos usando o estado inicial
- Enquanto a lista de estados não expandidos não estiver vazia
 - Escolha um nó de lista de estados não expandidos de acordo com a estratégia de procura e remova-o da lista
 - Se o nó contiver o estado do objetivo, devolva a solução
 - Senão, expanda o nó e inclua seus filhos na lista de estados não expandidos

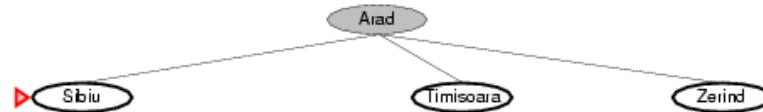
Exemplo de procura em árvore

Inicial: Arad
Objetivo: Bucharest

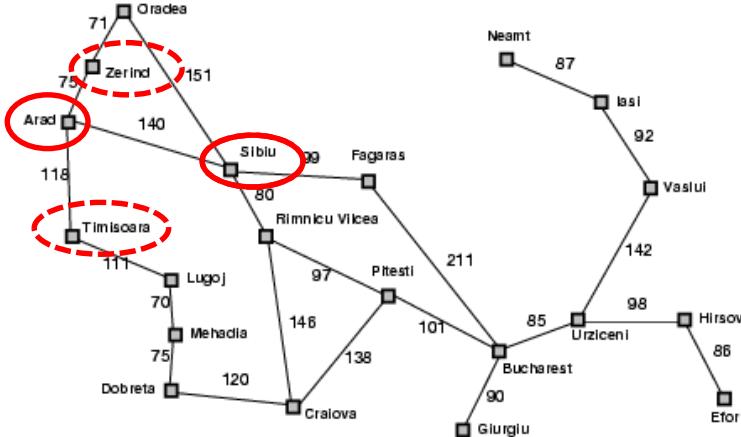


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois. 43

Exemplo de procura em árvore

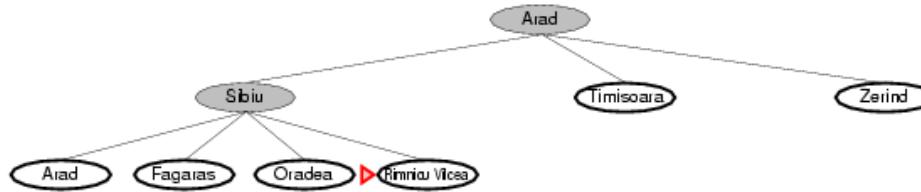


Inicial: Arad
Objetivo: Bucharest

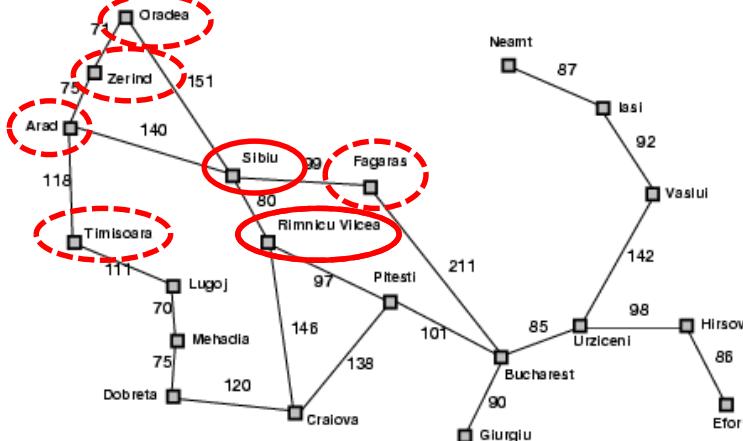


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois. 44

Exemplo de procura em árvore

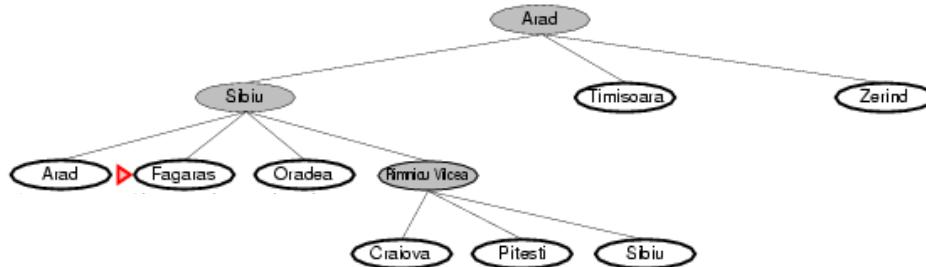


Inicial: Arad
 Objetivo: Bucharest

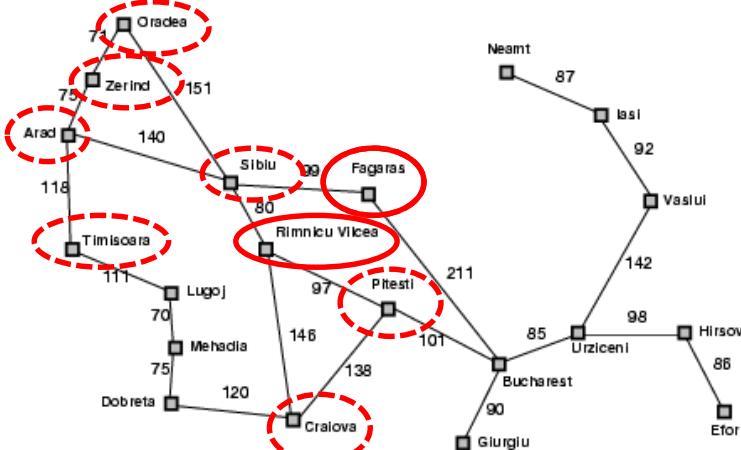


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois. 45

Exemplo de procura em árvore

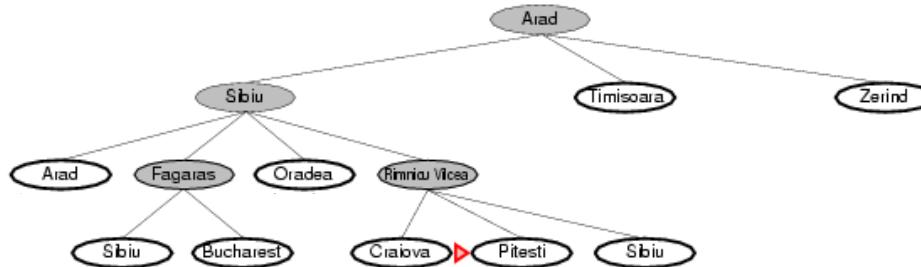


Inicial: Arad
Objetivo: Bucharest

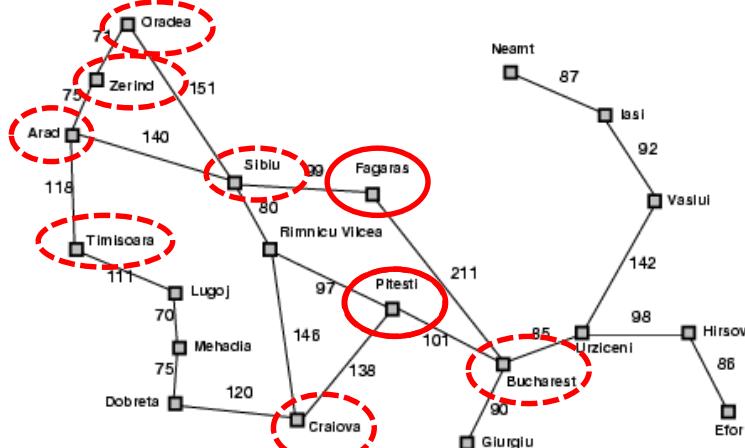


Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois. 46

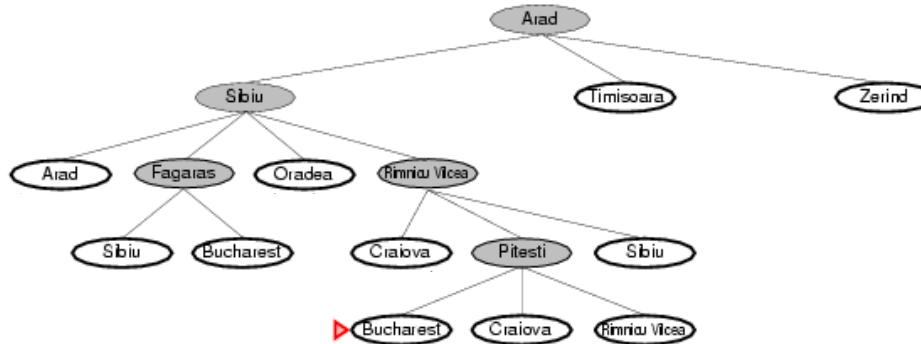
Exemplo de procura em árvore



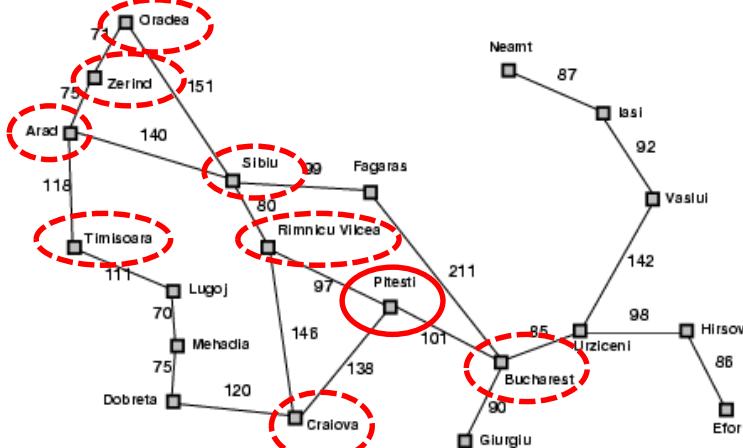
Inicial: Arad
 Objetivo: Bucharest



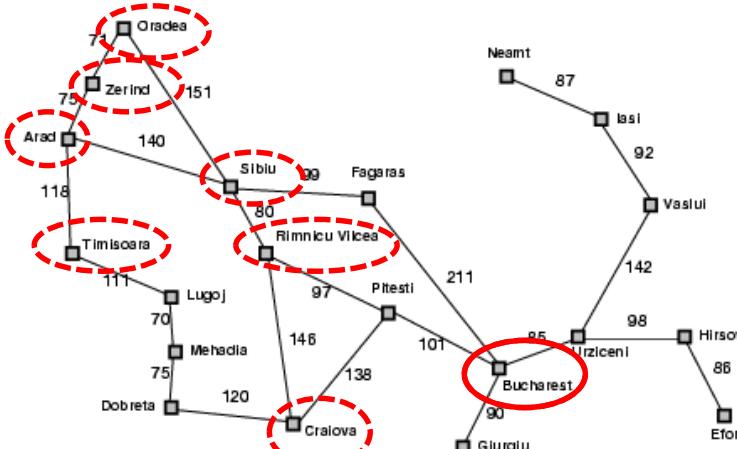
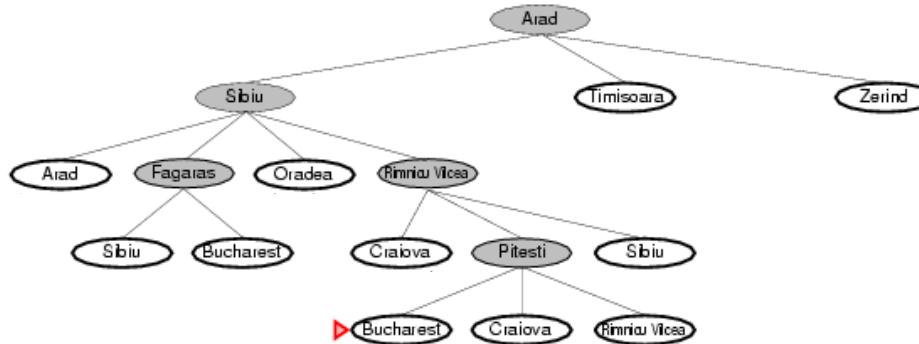
Exemplo de procura em árvore



Inicial: Arad
 Objetivo: Bucharest



Exemplo de procura em árvore

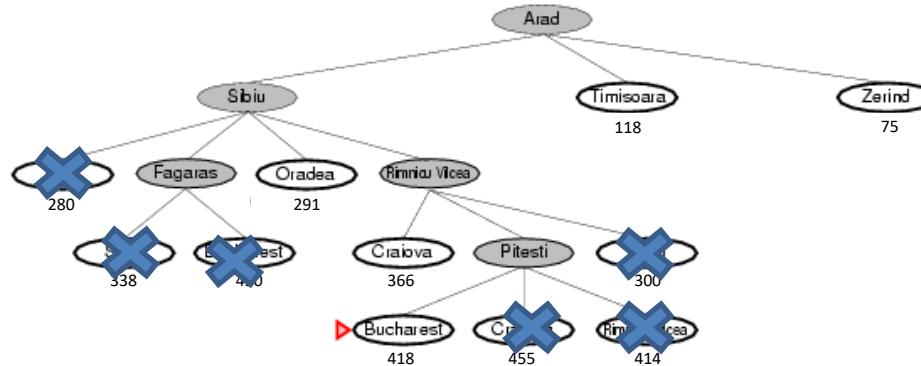


Inicial: Arad
 Objetivo: Bucharest

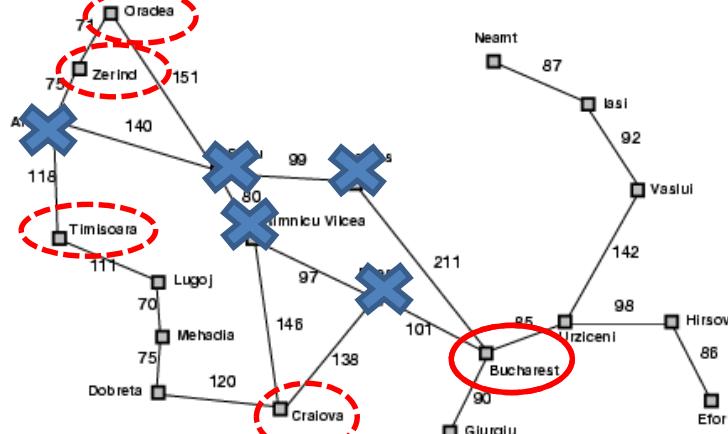
Algoritmo de procura em árvore Manipulação de estados repetidos

- Inicialize a lista de estados não expandidos usando o estado inicial
- Enquanto a lista de estados não expandidos não estiver vazia
 - Escolha um nó da lista de estados não expandidos de acordo com a estratégia de procura e remova-o da lista
 - Se o nó contiver o estado do objetivo, devolva a solução
 - Senão, expanda o nó e inclua os seus filhos na lista de estados não expandidos
- Para lidar com estados repetidos:
 - De cada vez que expandir um nó, adicione esse estado ao conjunto explorado; não coloque estados explorados na lista de estados não expandidos novamente
 - Sempre que adicionar um nó à lista de estados não expandidos, verifique se ele já existe na lista de estados não expandidos com um custo de caminho mais alto e, se sim, substitua esse nó pelo novo.

Procura sem estados repetidos



Inicial: Arad
Objetivo: Bucharest



Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois. 51

Podemos avaliar o desempenho do algoritmo de procura através dos seguintes critérios:

- Completude: Está garantido que encontra a solução?
- Complexidade no Tempo: Quanto tempo demora a encontrar a solução?
- Complexidade no Espaço: Quanta memória necessita para fazer a procura?
- Otimização: Encontra a melhor solução?

O tempo e a complexidade do espaço são sempre considerados tendo em conta a medição da dificuldade do problema (e.g. o tamanho do grafo e do espaço de estados).

- A estratégia: a ordem da expansão do nó
- Critérios de avaliação:
 - Completude: Está garantido que encontra a solução?
 - Otimização: Encontra a melhor solução?
 - Complexidade no Tempo: Quanto tempo demora a encontrar a solução?
 - Complexidade no Espaço: Quanta memória necessita para fazer a procura?
- O tempo e a complexidade do espaço são medidos em termos de:
 - b : o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de procura
 - d : a profundidade da melhor solução
 - m : a máxima profundidade do espaço de estados

Estratégias de Procura

- **Tipos de Estratégias de Procura:**

- Procura Não-Informada (cega)

- Primeiro em Largura (BFS)
 - Primeiro em Profundidade (DFS)
 - Custo Uniforme
 - Iterativa
 - Bidirecional

- Procura Informada (heurística)

- Gulosa
 - Algoritmo A*

- Uma **estratégia de procura** é definida escolhendo a ordem da expansão do nó;
- **Estratégias de procura não informadas** usam apenas as **informações disponíveis na definição do problema**;
- Nas **estratégias de procura informadas** dá-se ao algoritmo “**dicas**” sobre a adequação de diferentes estados.

■ Procura Primeiro em Largura (Breadth-first search)

- Estratégia: Todos os nós de menor profundidade são expandidos primeiro
- Bom: Procura muito sistemática
- Mau: Normalmente demora muito tempo e sobretudo ocupa muito espaço

■ Propriedades:

- Completa: Sim, se b (fator de ramificação) for finito
- Tempo: supondo fator de ramificação b então $n=1+b+b^2+b^3+\dots+b^m = O(b^d)$ é exponencial em d
- Espaço: Guarda cada nó em memória $O(b^d)$
- Ótima: Sim, se o custo de cada passo for 1

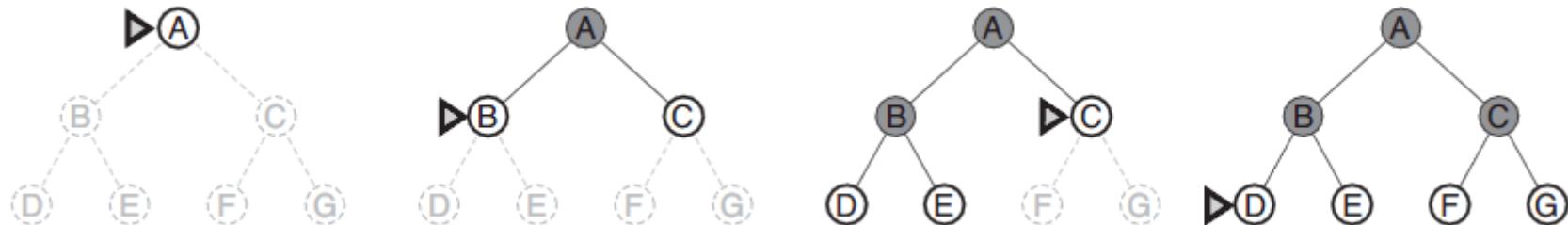
■ Em geral só pequenos problemas podem ser resolvidos assim!

- b: o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de procura
- d: a profundidade da melhor solução
- m: a máxima profundidade do espaço de estados

- Procura Primeiro em Largura (Breadth-first search) (2)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
    node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    frontier  $\leftarrow$  a FIFO queue with node as the only element
    explored  $\leftarrow$  an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node  $\leftarrow$  POP(frontier) /* chooses the shallowest node in frontier */
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child  $\leftarrow$  CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier  $\leftarrow$  INSERT(child, frontier)
```

- Procura Primeiro em Largura (Breadth-first search) (3)



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

■ Procura Primeiro em Profundidade (Depth-First Search)

- Estratégia: Expandir sempre um dos nós mais profundos da árvore +++
- Bom: Muito pouca memória necessária, bom para problemas com muita soluções
- Mau: Não pode ser usada em árvores com profundidade infinita, pode ficar presa em ramos errados

■ Propriedades:

- Completa: Não, falha em espaços de profundidade infinita, com repetições (loops)
 - Modifique para evitar estados repetidos ao longo do caminho
- Tempo: $O(b^m)$, mau se $m > d$
- Espaço: $O(bm)$, espaço linear
- Ótima: Não (em princípio devolve a 1ª solução que encontra)

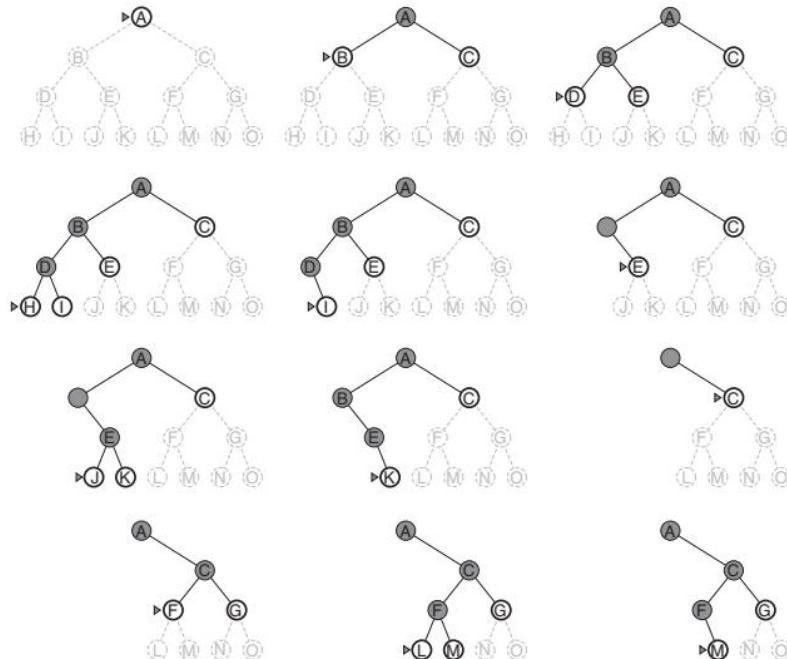
- b: o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de procura
- d: a profundidade da melhor solução
- m: a máxima profundidade do espaço de estados

■ Por vezes é definida uma profundidade limite (l) e transforma-se em Procura com Profundidade Limitada

Procura Primeiro em Profundidade

- Procura Primeiro em Profundidade (Depth-First Search) (2)

* Sempre
nos da esq.



Para problemas com várias soluções, esta estratégia pode ser bem mais rápida do que procura em largura;

No entanto, esta estratégia deve ser evitada quando as árvores geradas são muito profundas ou geram caminhos infinitos.

Procura Primeiro em Profundidade (Depth-First Search) Depth-limited search

= procura em profundidade com limite de profundidade l , ie, nós em profundidade l não têm sucessores

```

function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)
function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred?  $\leftarrow$  false
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result  $\leftarrow$  RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred?  $\leftarrow$  true
        else if result  $\neq$  failure then return result
    if cutoff-occurred? then return cutoff else return failure

```

Um dos problemas da procura Primeiro em Profundidade prende-se com a **incapacidade desta lidar com caminhos infinitos**;

A Procura em Profundidade **com limite de profundidade** procura evitar este problema, fixando o nível máximo de procura.

- **Procura Primeiro em Profundidade (Depth-First Search)**

- escolher o primeiro elemento da lista de estados não expandidos
- adicionar extensões de caminho à frente da lista de estados não expandidos

- **Procura Primeiro em Largura (Breadth-first search)**

- escolher o primeiro elemento de lista de estados não expandidos
- adicionar extensões de caminho no final da lista de estados não expandidos

$g(N) = \text{custo}$

Procura de Custo Uniforme

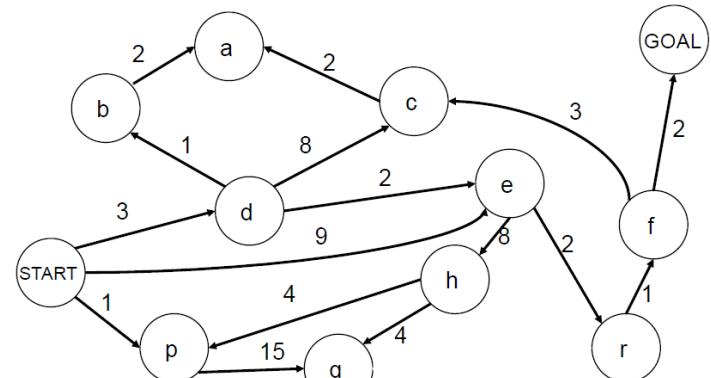
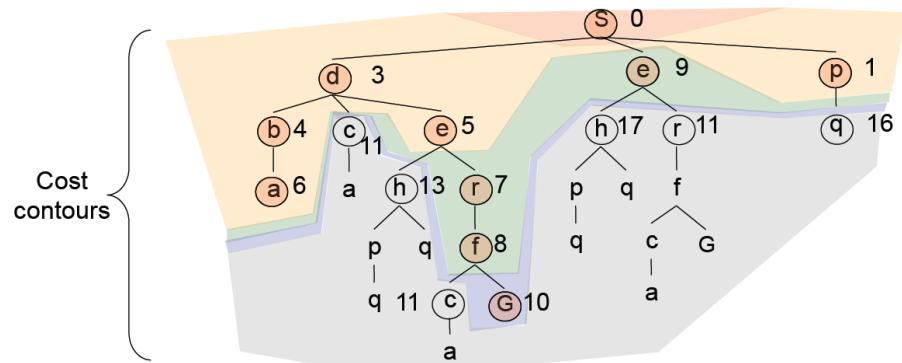
- Estratégia:
 - Para cada nó da lista de estados não expandidos, guardar o custo total do caminho do estado inicial para esse nó
 - Expandir sempre o nó com menor custo da lista de estados não expandidos (medido pela função de custo da solução)
- Procura Primeiro em Largura é igual a Procura de Custo Uniforme se $g(N)=\text{Depth}(N)$ ou seja;
- Equivalente a Procura Primeiro em Largura (Breadth-first search), se os custos forem todos iguais
- Implementação: lista de estados não expandidos é uma lista prioritária ordenada pelo custo do caminho
- Temos de garantir que $g(\text{sucessor}) \geq g(N)$
- Equivalente ao algoritmo de Dijkstra em geral

Em todos os nós N, $g(N)$ é o custo conhecido de ir da raiz até ao nó N.

O algoritmo de Dijkstra (Edsger Dijkstra, 1956), resolve o problema do caminho mais curto num grafo dirigido ou não dirigido com arestas de peso não negativo.

Procura de Custo Uniforme (2)

- Ordem de expansão:
(S,p,d,b,e,a,r,f,e,G)



Fonte: Svetlana Lazebnik, Fall 2017 Artificial Intelligence, University of Illinois.

Procura de Custo Uniforme (3)



Fonte https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Practical_optimizations_and_infinite_graphs

- Propriedades:

- Completa: Sim, se o custo da etapa for maior que alguma constante positiva ε (não queremos sequências infinitas de etapas com um custo total finito)
- Tempo:
 - *Número de nós com custo de caminho \leq custo da solução ideal (C^*)*, $O(b^{C^*/\varepsilon})$
 - *Ipode ser maior que O(bd): a procura pode explorar caminhos longos que consistem em pequenos passos antes de explorar caminhos mais curtos que consistem em passos maiores*
- Espaço: $O(b^{C^*/\varepsilon})$
- Ótima: Sim
 - b: o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de procura
 - d: a profundidade da melhor solução
 - m: a máxima profundidade do espaço de estados

Procura Iterativa Aprofundamento Progressivo

Se não conhecemos o valor limite máximo, estaremos condenados a uma estratégia de procura em profundidade primeiro e temos que lidar com o problema de eventuais caminhos infinitos. A resposta passa pela alteração do princípio da procura limitada fazendo variar esse limite entre zero e infinito.

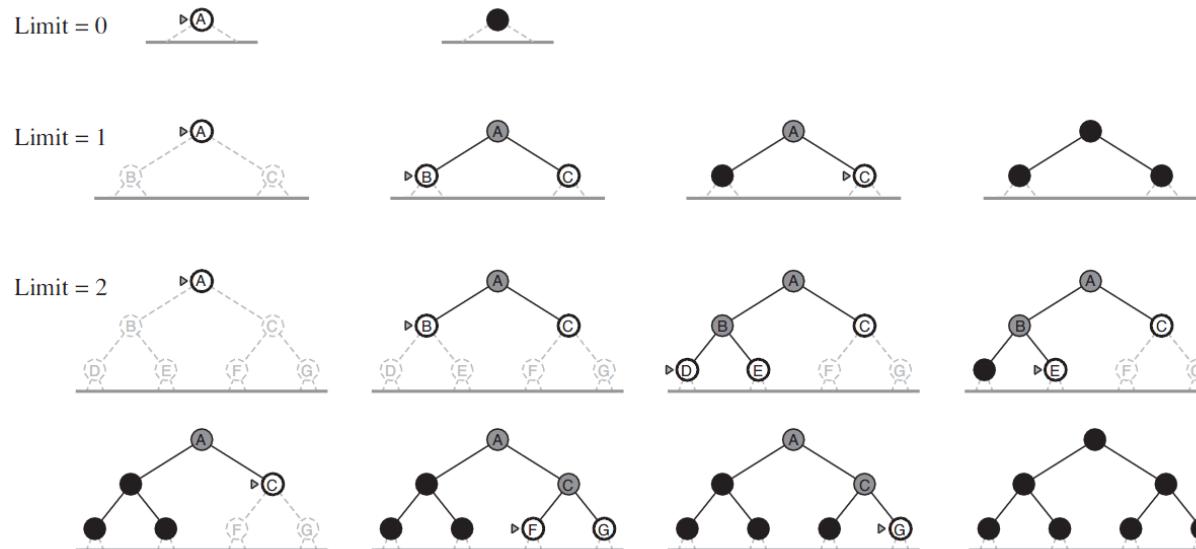
Usar Procura Primeiro em Profundidade (Depth-First Search) como uma sub-rotina

- Verificar a raiz
- Desenvolver um DFS procurando um caminho de comprimento 1
- Se não houver um caminho de comprimento 1, desenvolver um DFS procurando um caminho de comprimento 2
- Se não houver um caminho de comprimento 2, desenvolver um DFS procurando um caminho de comprimento 3...

!!!

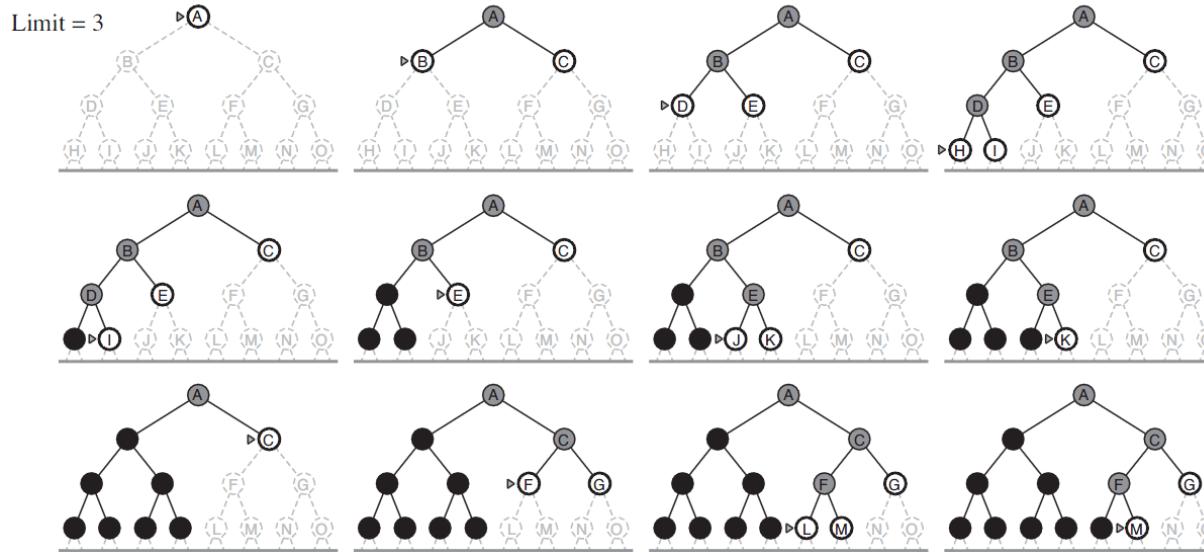
Procura Iterativa Aprofundamento Progressivo

- Procura em Profundidade Iterativa (2)



Procura Iterativa Aprofundamento Progressivo

- Procura em Profundidade Iterativa (3)



O algoritmo de procura iterativa é uma **boa opção para problemas em que somos obrigados a recorrer a um método cego.**

O **espaço de procura é grande, mas não sabemos qual é o nível máximo em que pode estar uma solução.**

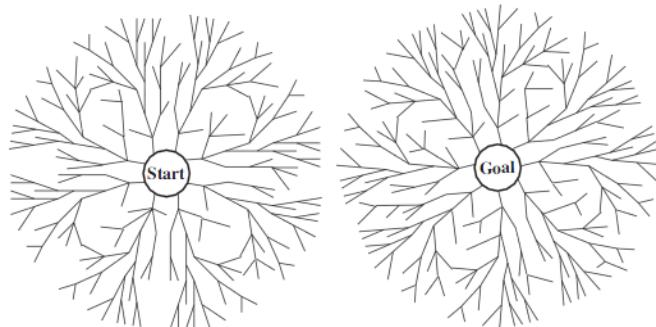
Procura Iterativa Aprofundamento Progressivo

- Estratégia: Executar procura em profundidade limitada, iterativamente, aumentando sempre o limite da profundidade
 - Propriedades:
 - Completa: Yes
 - Tempo: $(d+1)b^0 + d b^1 + (d-1)b^2 + \dots + b^d = O(b^d)$
 - Espaço: $O(b^d)$
 - Ótima: Sim, se o custo for 1.
 - Em geral é a melhor estratégia (não-informada ou cega) para problemas com um grande espaço de procura e em que a profundidade da solução não é conhecida
- \bullet b: o máximo fator de ramificação (o número máximo de sucessores de um nó) da árvore de procura
 \bullet d: a profundidade da melhor solução
 \bullet m: a máxima profundidade do espaço de estados



Procura Bidirecional

- Estratégia: Executar uma procura para a frente desde o estado inicial e para trás desde o objetivo, simultaneamente.
 - Bom: Pode reduzir enormemente a complexidade no tempo $O(b^{d/2})$
 - Problemas: Será possível gerar os predecessores? E se existirem muitos estados objetivo? Como fazer o “matching” entre as duas proezas? Que tipo de proezas fazer nas duas metades?
- Eg., Para encontrar uma rota na Romênia, existe apenas um estado objetivo, portanto, a proezura para trás é muito parecida com a proezura para a frente; Mas se o objetivo é uma descrição abstrata, como o de que “nenhuma rainha ataca outra rainha” no problema das rainhas n, a proezura bidirecional é de difícil uso.



Comparação entre Estratégias de Procura

- Avaliação das estratégias de procura:

- B é o fator de ramificação
- d é a profundidade da solução
- m é a máxima profundidade da árvore
- l é a profundidade limite da procura

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

- **Procura informada (Heurística)**

- Utiliza **informação sobre o problema** para evitar que o algoritmo de procura fique “perdido vagueando no escuro”

- **Estratégia de Procura:**

- Definida através da escolha da ordem de expansão dos nós

- **Procura do Melhor Primeiro (Best-First Search)**

- Utiliza **uma função de avaliação** que devolve um número indicando o **interesse de expandir um nó**
 - Procura Gulosa (Greedy-Search) – $f(n) = h(n)$ (função que estima a distância à solução)
 - Algoritmo A* - $f(n) = g(n) + h(n)$ (estima o custo da melhor solução que passa por n)

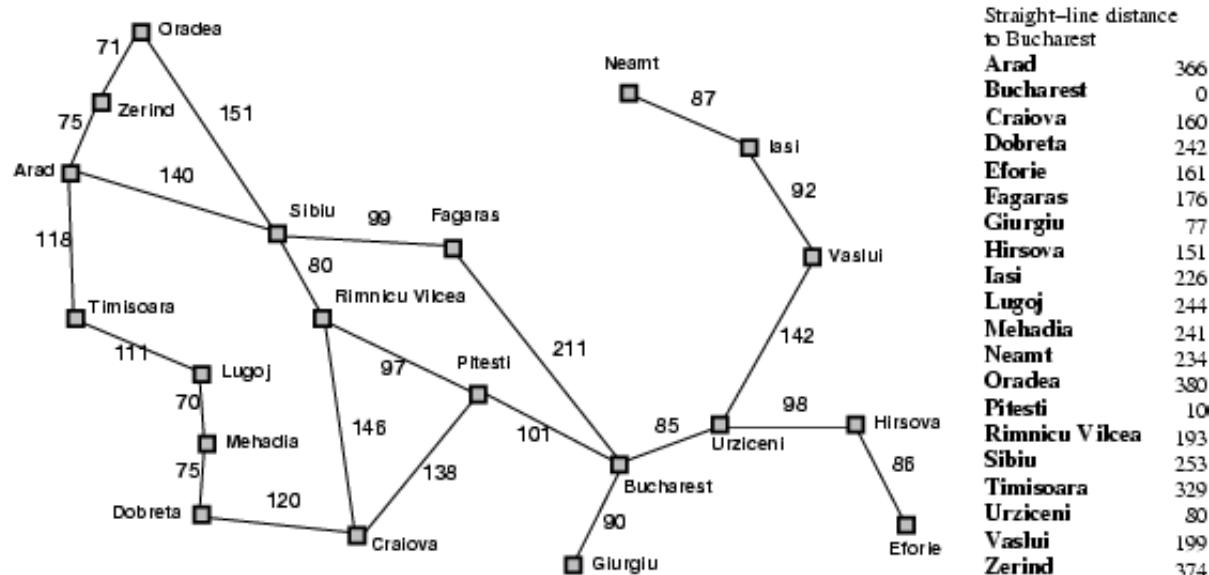
- Como técnica de procura para a obtenção de metas em problemas não algorítmicos que geram “explosões” combinatórias;
- Como um método de aproximação da resolução de problemas utilizando funções de avaliação de tipo heurística (dica);
- Como um método de poda (corte) para estratégias de programas de jogos.

- Numa procura, podemos aplicar dois tipos de heurísticas genéricas, qual o nó em que será feita a expansão e quais os nós que devem ser descartados.
 - Se o universo é totalmente conhecido, a heurística será desenvolvida através da atribuição de números;
 - Se o universo não é totalmente conhecido, a heurística será desenvolvida através da aplicação de regras.

- As heurísticas são específicas para cada problema. Estas funções podem ser pouco exactas ou até nem serem capazes de encontrarem a melhor resposta, no entanto a sua utilização permite o não uso de análises combinatórias.
- A implementação de uma heurística é algo difícil! Na medida em que, é difícil medir com precisão o valor de uma determinada solução e é, ainda, difícil medir determinados conhecimentos de forma a permitir que seja efetuada uma análise matemática do seu efeito no processo de procura.

Heurística para o problema da Romênia

- Estado Inicial: Arad; Objetivo: Bucharest; $h(n)$ = distância em linha reta



- **Informada (Procura Gulosa – Greedy-Search)**

- **Estratégia:**

- Expandir o nó que parece estar mais perto da solução

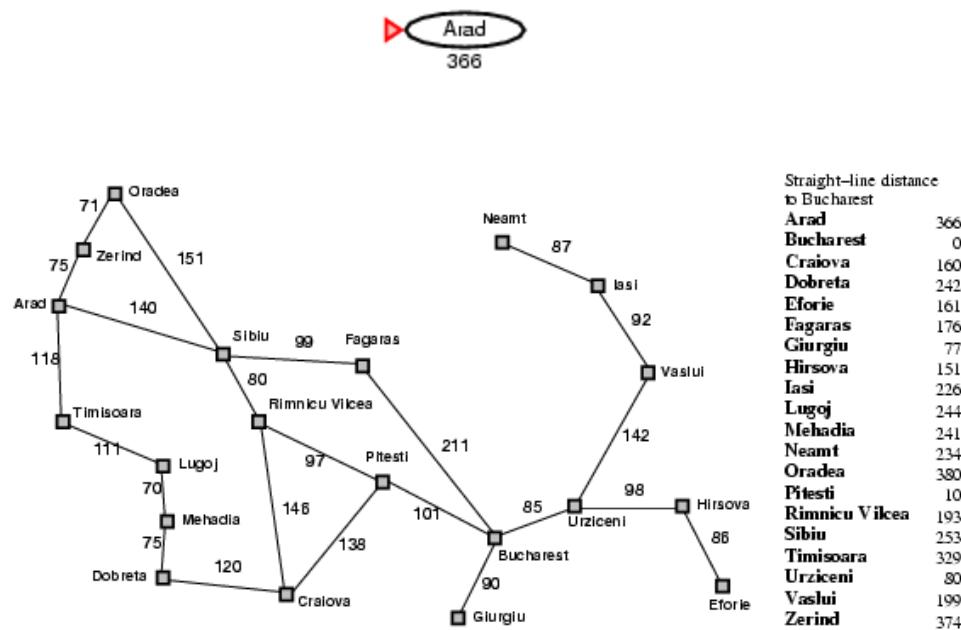
- $h(n)$ = custo estimado do caminho mais curto do estado n para o objetivo (função heurística)

- function GREEDY-SEARCH(problem) returns a solution or failure
 - return BEST-FIRST-SEARCH(problem,h)

- **Exemplo:**

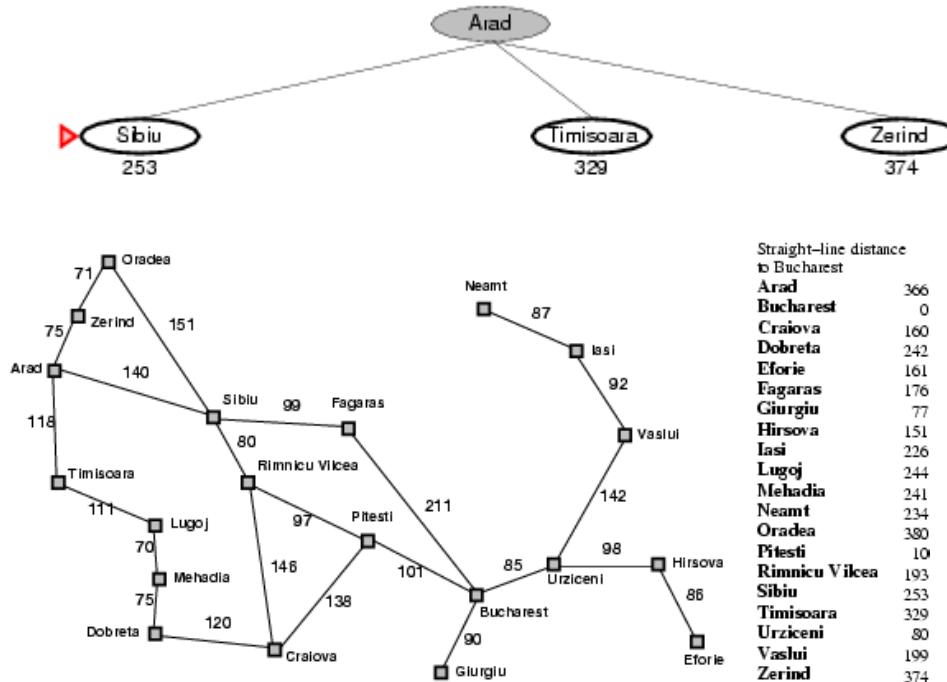
- $h(n)$ = distância em linha reta entre n e o objetivo

- Estado Inicial: Arad; Objetivo: Bucharest; $h(n)$ = distância em linha reta



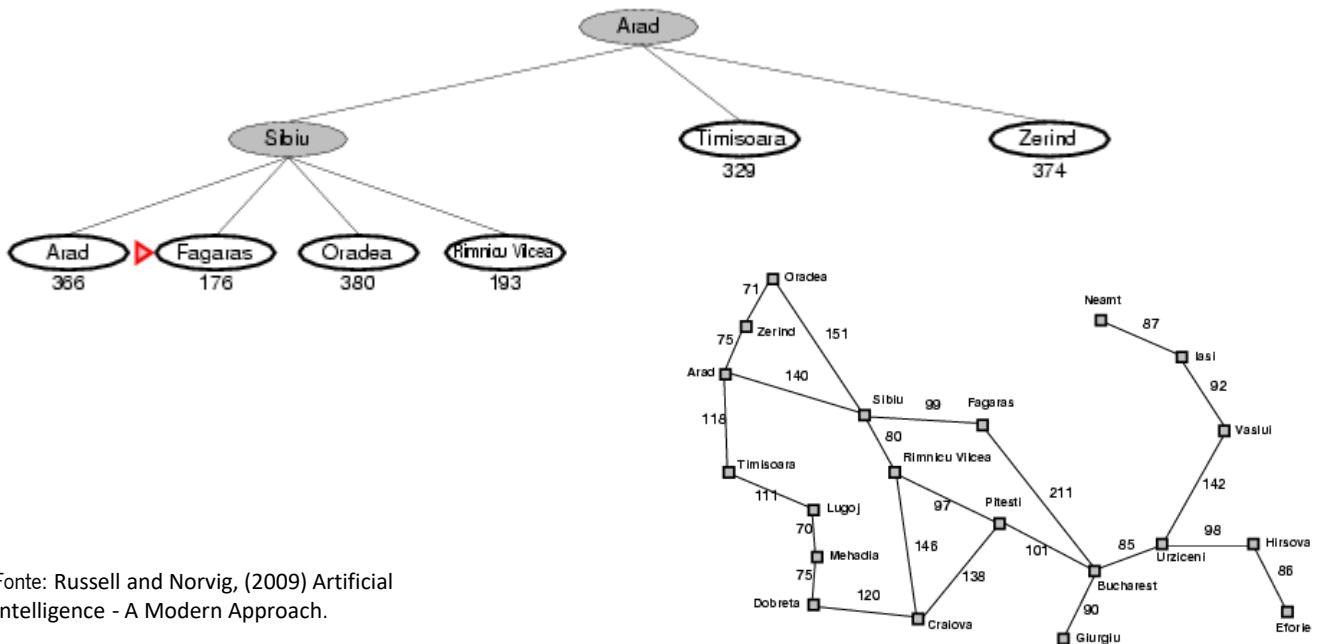
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Estado Inicial: Arad; Objetivo: Bucharest; $h(n)$ = distância em linha reta



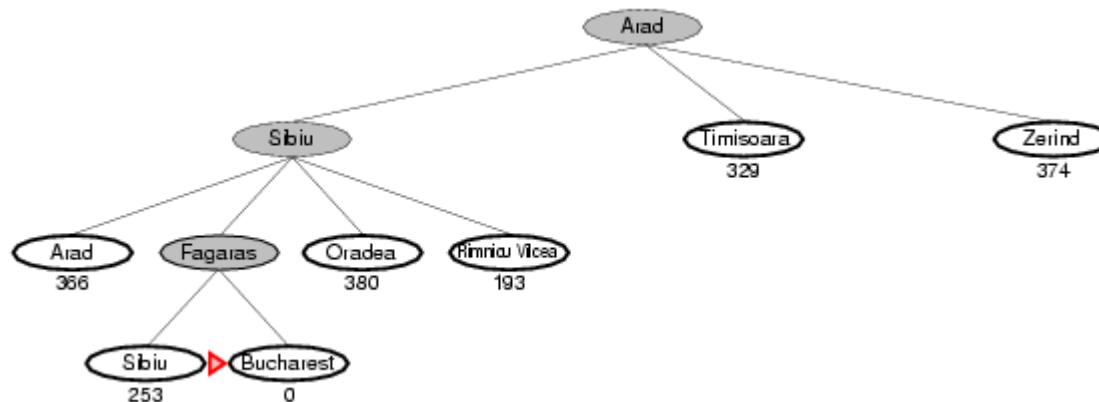
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Estado Inicial: Arad; Objetivo: Bucharest; $h(n)$ = distância em linha reta



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Estado Inicial: Arad; Objetivo: Bucharest; $h(n)$ = distância em linha reta



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

■ Propriedades

- Completa? Não, pode entrar em ciclo.
 - Suscetível a falsos arranques
- Complexidade no tempo? $O(b^m)$
 - mas com uma boa função heurística pode diminuir considerably
- Complexidade no espaço? $O(b^m)$
 - Mantém todos os nós em memória
- Ótima? Não, porque não encontra sempre a solução ótima.
- Necessário detetar estados repetidos.

- **Informada (Procura A*)**

- **Estratégia:**

- evitar expandir caminhos que são dispendiosos
- O algoritmo A* combina a procura gulosa com a uniforme, minimizando a soma do caminho já efetuado com o mínimo previsto do que falta até a solução. Usa a função:

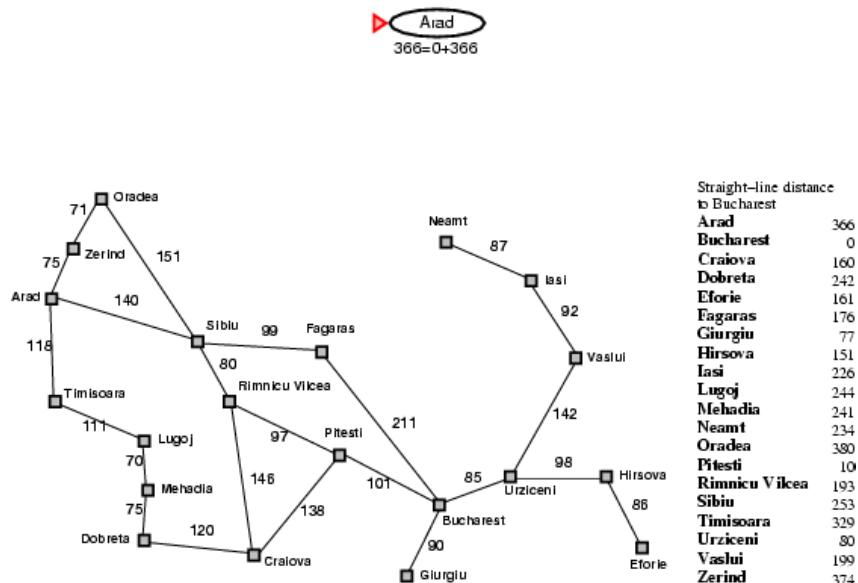
$$f(n) = g(n) + h(n)$$

- $g(n)$ = custo total, até agora, para chegar ao estado n (custo do percurso)
- $h(n)$ = custo estimado para chegar ao objetivo (não deve sobreestimar o custo para chegar à solução (heurística))

$f(n)$ = custo estimado da solução menos dispendiosa que passa pelo nó n

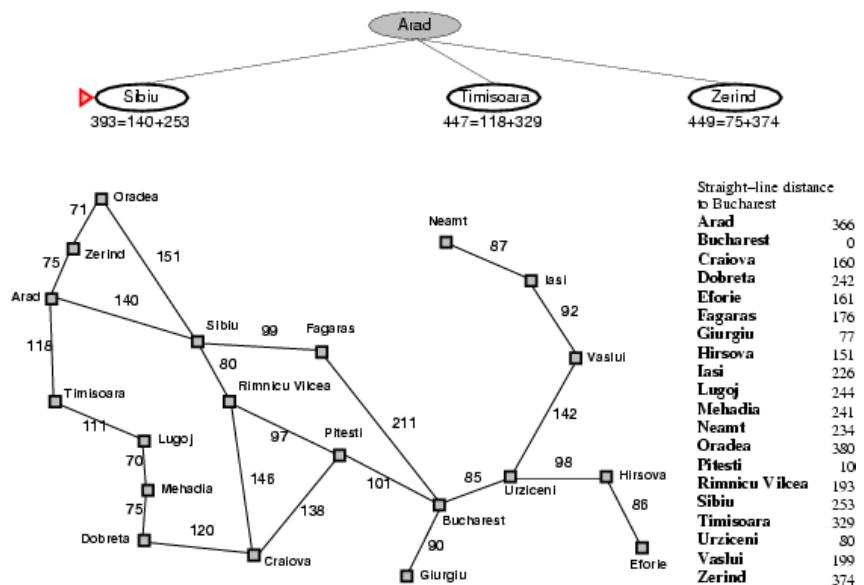
- **function** A*-SEARCH(problem) **returns** a solution or failure
 - return** BEST-FIRST-SEARCH(problem,g+h)
 - Algoritmo A* é ótimo e completo.
 - Complexidade no tempo exponencial em (erro relativo de h^* comprimento da solução)
 - Complexidade no espaço: Mantém todos os nós em memória.

- Estado Inicial: Arad; Objetivo: Bucharest; $f(n) = g(n) + h(n)$;
- $g(n)$ = custo até n; $h(n)$ = distância em linha reta ao objetivo



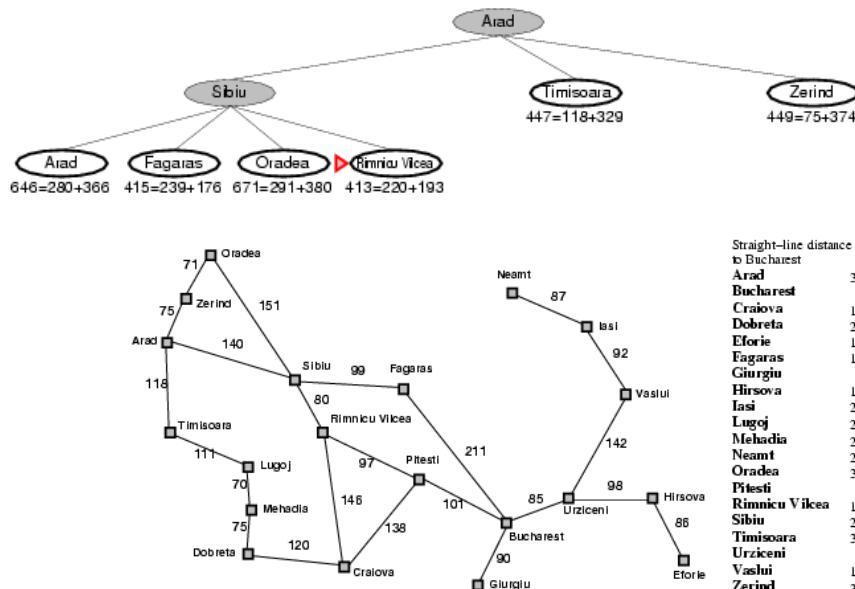
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Estado Inicial: Arad; Objetivo: Bucharest; $f(n) = g(n) + h(n)$;
- $g(n)$ = custo até n; $h(n)$ = distância em linha reta ao objetivo



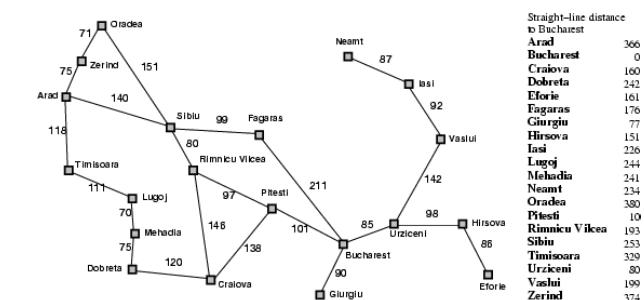
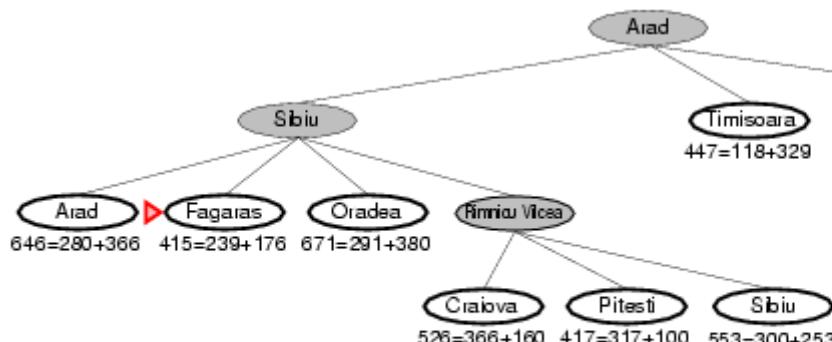
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Estado Inicial: Arad; Objetivo: Bucharest; $f(n) = g(n) + h(n)$;
- $g(n)$ = custo até n; $h(n)$ = distância em linha reta ao objetivo



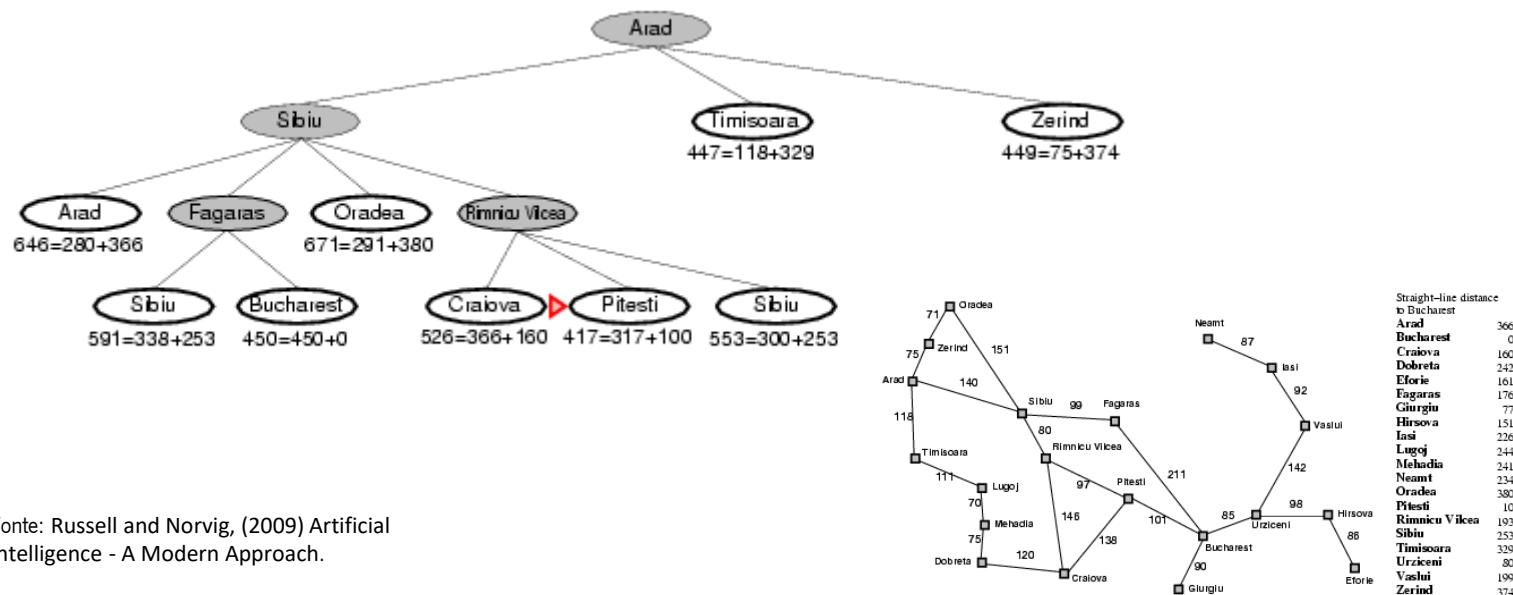
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Estado Inicial: Arad; Objetivo: Bucharest; $f(n) = g(n) + h(n)$;
- $g(n)$ = custo até n; $h(n)$ = distância em linha reta ao objetivo



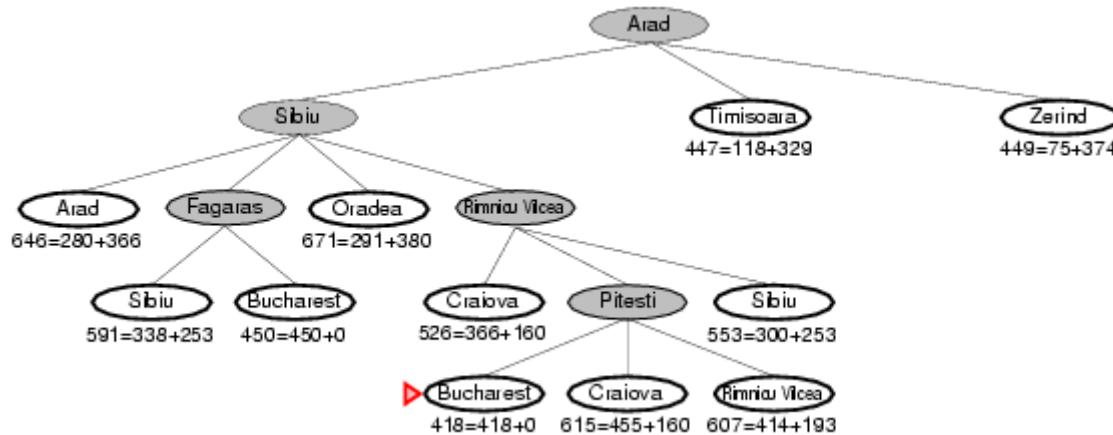
Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Estado Inicial: Arad; Objetivo: Bucharest; $f(n) = g(n) + h(n)$;
- $g(n) = \text{custo até } n$; $h(n) = \text{distância em linha reta ao objetivo}$



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

- Estado Inicial: Arad; Objetivo: Bucharest; $f(n) = g(n) + h(n)$;
- $g(n)$ = custo até n; $h(n)$ = distância em linha reta ao objetivo



Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.



Fonte:
https://en.wikipedia.org/wiki/A*_search_algorithm

■ Heurísticas – 8 Puzzle

- Solução típica em 20 passos com um fator de ramificação médio: 3
- Número de estados: $320 = 3.5 \times 10^9$
- N° Estados (sem estados repetidos) = $9! = 362880$
- Heurísticas:
 - $H_1(n)$ = N° de peças fora do lugar
 - $H_2(n)$ = Soma das distâncias das peças até às suas posições corretas

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

■ Heurísticas – 8 Puzzle (2)

o Relaxamento de Problemas como forma de inventar heurísticas:

- Peça pode-se mover de A para B se A é adjacente a B e B está vazio
- a) Peça pode-se mover de A para B se A é adjacente a B
- b) Peça pode-se mover de A para B se B está vazio
- c) Peça pode-se mover de A para B

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- **Procura com Memória Limitada - IDA*/SMA***
- **IDA* - Procura com Profundidade Iterativa (Iterative Deepening Search)**
 - Estratégia: Utilização de um custo limite em cada iteração e realização de procura em profundidade iterativa
 - Problemas em alguns problemas reais com funções de custo com muitos valores
- **SMA* - Procura Simplificada com Memória Limitada (Simplified Memory Bounded A*)**
 - IDA* de uma iteração para a seguinte só se recorda de um valor (o custo limite)
 - SMA* utiliza toda a memória disponível, evitando estados repetidos
 - Estratégia: Quando é necessário gerar um sucessor e não tem memória, esquece o nó da fila que aparente ser pouco prometedor (com um custo alto).

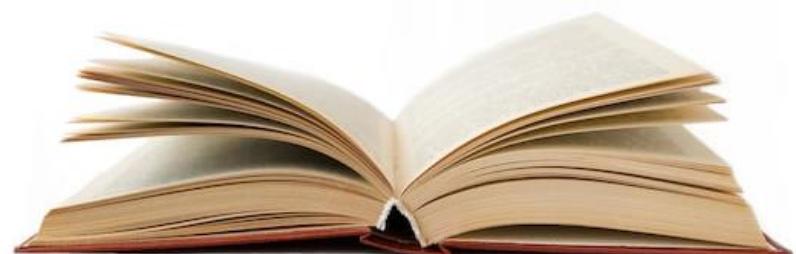
Algoritmo	Completo	Otimal	Tempo complexidade	Espaço complexidade
Primeiro em Largura (BFS)	Sim	Se todos os custos escalonados forem iguais	$O(b^d)$	$O(b^d)$
Primeiro em Profundidade (DFS)	Não	Não	$O(b^m)$	$O(bm)$
Iterativa	Sim	Se todos os custos escalonados forem iguais	$O(b^d)$	$O(bd)$
Custo Uniforme	Sim	Sim	$\text{Nº de nodos com } g(n) \leq C^*$	
Gulosa	Não	Não	no pior caso : $O(b^m)$ No melhor caso: $O(bd)$	
A*	Sim	Sim (se a heurística for admissível)	$\text{Nº de nodos com } g(n)+h(n) \leq C^*$	

Bibliografia Recomendada

- Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach, 3rd edition, ISBN-13: 9780136042594.
- E.Costa, A.Simões (2008), Inteligência Artificial-Fundamentos e Aplicações, FCA, ISBN: 978-972-722-340-4, 2008.

Outro material

- Svetlana Lazebnik, Lecture notes Fall 2017 Artificial Intelligence, University of Illinois.
- Luís Paulo Reis, Lecture notes Artificial Intelligence (2019), Universidade do Porto





Universidade do Minho
Escola de Engenharia
Departamento de Informática

MÉTODOS DE RESOLUÇÃO DE PROBLEMAS E DE PROCURA

LICENCIATURA EM ENGENHARIA INFORMÁTICA
MESTRADO integrado EM ENGENHARIA INFORMÁTICA

Inteligência Artificial
2022/23