Inteligência Artificial

CTC15 Aula 2b

Sumário

- Agentes que resolvem problemas
- Tipos de problemas
- Formulação de problemas
- Exemplos de problemas
- Algoritmos de busca básicos

Agentes que resolvem problemas

Um agente simples:

function SIMPLE-PROBLEM-SOLVING-AGENT(p) **returns** an action
  inputs: p, a percept
  static: s, an action sequence, initially empty
    $state$, some description of the current world state
    g, a goal, initially null
    $problem$, a problem formulation

  $state \leftarrow$ UPDATE-STATE($state, p$)
  if s is empty **then**
    $g \leftarrow$ FORMULATE-GOAL($state$)
    $problem \leftarrow$ FORMULATE-PROBLEM($state, g$)
    $s \leftarrow$ SEARCH($problem$)
  $action \leftarrow$ RECOMMENDATION($s, state$)
  $s \leftarrow$ REMAINDER($s, state$)
  return $action$

Isto é solução offline. Solução online envolve decisões sem conhecimento completo do problema e solução.

Exemplo: Férias na Romênia

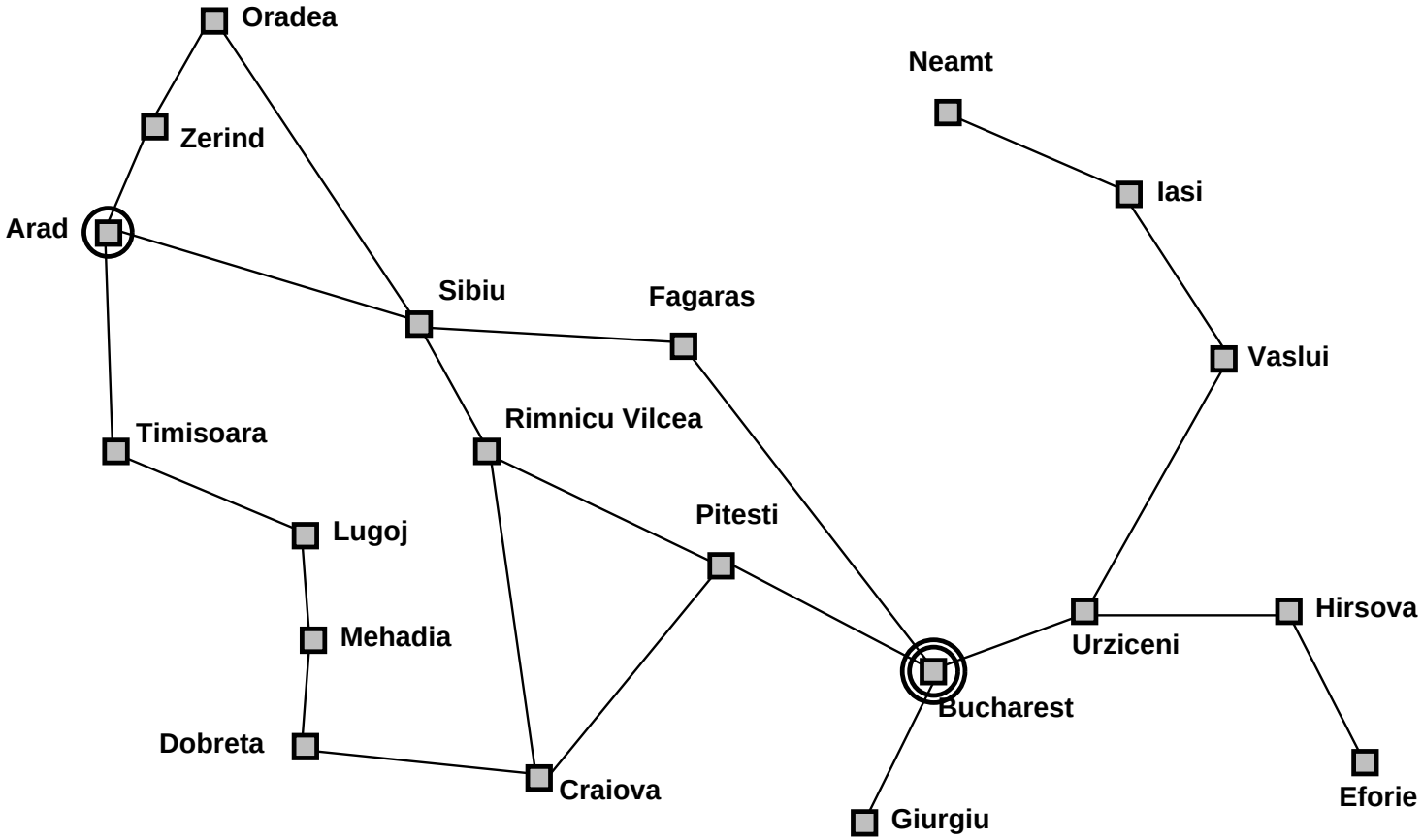
Em férias na Romênia; atualmente em Arad.
Vôo para Bucareste sai amanhã

Formular objetivo:
estar em Bucareste

Formular problema:
estados: várias cidades
operadores: dirigir entre cidades

Achar solução:
sequência de cidades, e.g., Arad, Sibiu, Fagaras, Bucareste

Exemplo: Romênia



Tipos de problemas

Determinístico, accessível $\Longrightarrow$ *problema de estados simples*
Determinístico, inaccessível $\Longrightarrow$ *problema de estados múltiplos*

Não-determinístico, inaccessível $\Longrightarrow$ *problema de contingência* (talvez)
sensores devem ser usados durante execução
solução é uma *árvore* ou *política*
muitas vezes *alterna* entre busca e execução

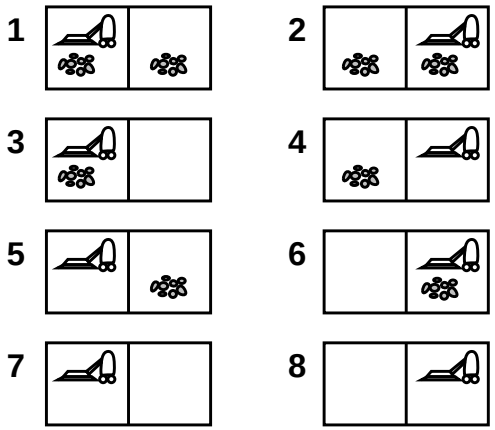
Espaço de estados desconhecido $\Longrightarrow$ *problema de exploração online*

Exemplo: problema do aspirador

Est. simples, início #5. Solução??

Est. múltiplos, início $\{1, 2, 3, 4, 5, 6, 7, 8\}$
e.g., $Direita$ para $\{2, 4, 6, 8\}$. Solução??

Contingência, início {1,3}
Murphy: Aspirador pode sujar tapete limpo
Sensor local: sujeira, localização.
Solução??



Formulação do problema de estados simples

Um $problema$ é definido por:

estado inicial — e.g., "em Arad"

teste de objetivo, que pode ser
$explícito$, e.g., $x =$ "em Bucareste"
$implícito$, e.g., $NoDirt(x)$

custo da trajetória (aditiva)
e.g., soma das distâncias, número de operadores executados, etc.

Uma solução é uma sequência de operadores
que leva do estado inicial ao estado-objetivo

Selecionando um espaço de estados

O mundo real é extremamente complexo
$\Rightarrow$ espaço de estados deve ser abstraído do processo de solução

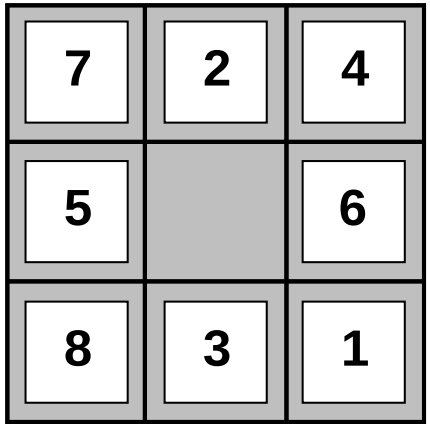
estado abstrato = conjunto de estados reais

operador abstrato = combinação de ações reais
e.g., "Arad $\rightarrow$ Zerind" representa um conjunto complexo
de possíveis rotas, retornos, paradas para descanso, etc.
Para realizabilidade garantida, qualquer estado real "em Arad"
deve levar a $algum$ estado real "em Zerind"

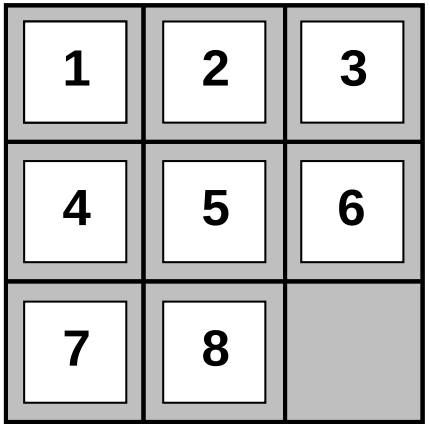
Solução (abstrata) =
conjunto de trajetória reais que são soluções no mundo real

Cada ação abstrata deve ser mais "fácil" do que no problema original!

Exemplo: Um quebra-cabeças



Start State



Goal State

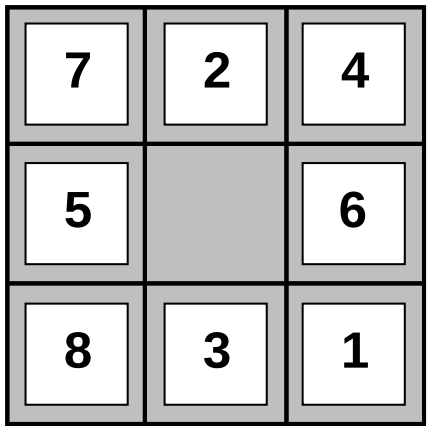
estados??

operadores??

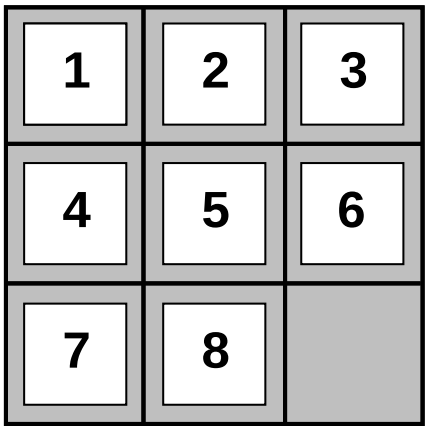
teste do objetivo??

custo da trajetória??

Exemplo: Um quebra-cabeças



Start State

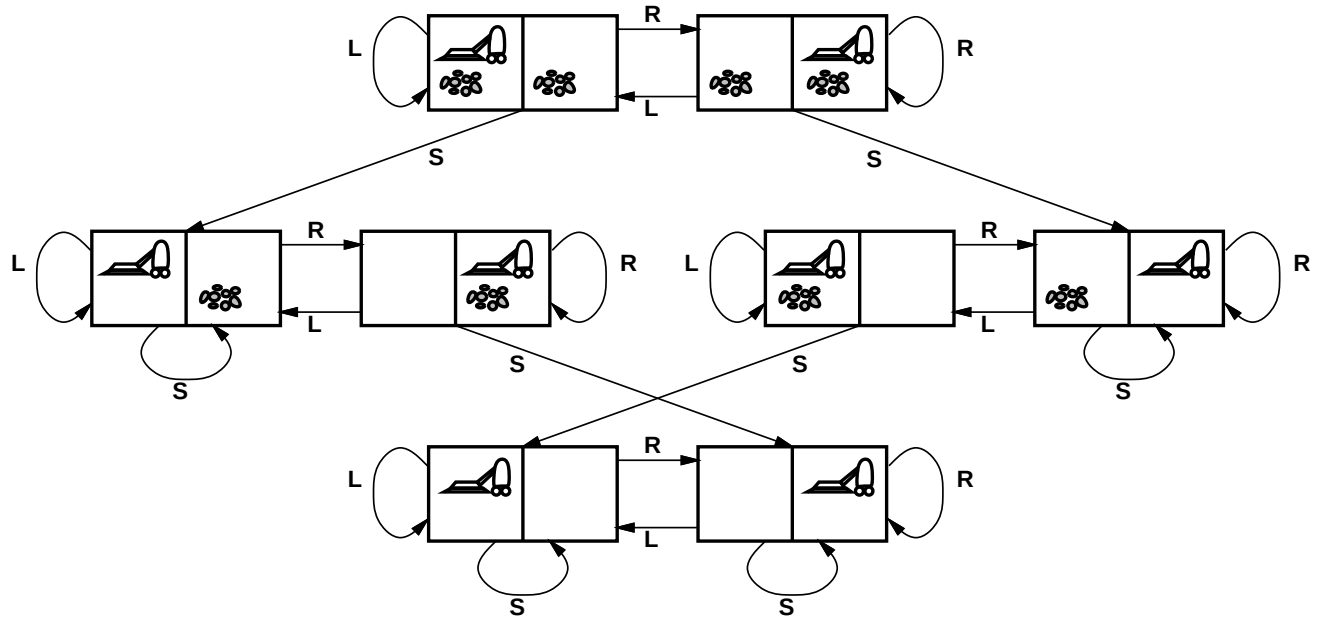


Goal State

estados??: localização dos blocos móveis
operadores??: 'mover espaço' pra esquerda,pra direita,pra cima, pra baixo
teste do objetivo??: $=$ atingir estado-objetivo (dado)
custo da trajetória??: 1 por movimento

[Obs: achar solução ótima para n casas é NP-completo]

Exemplo: grafo do problema do aspirador



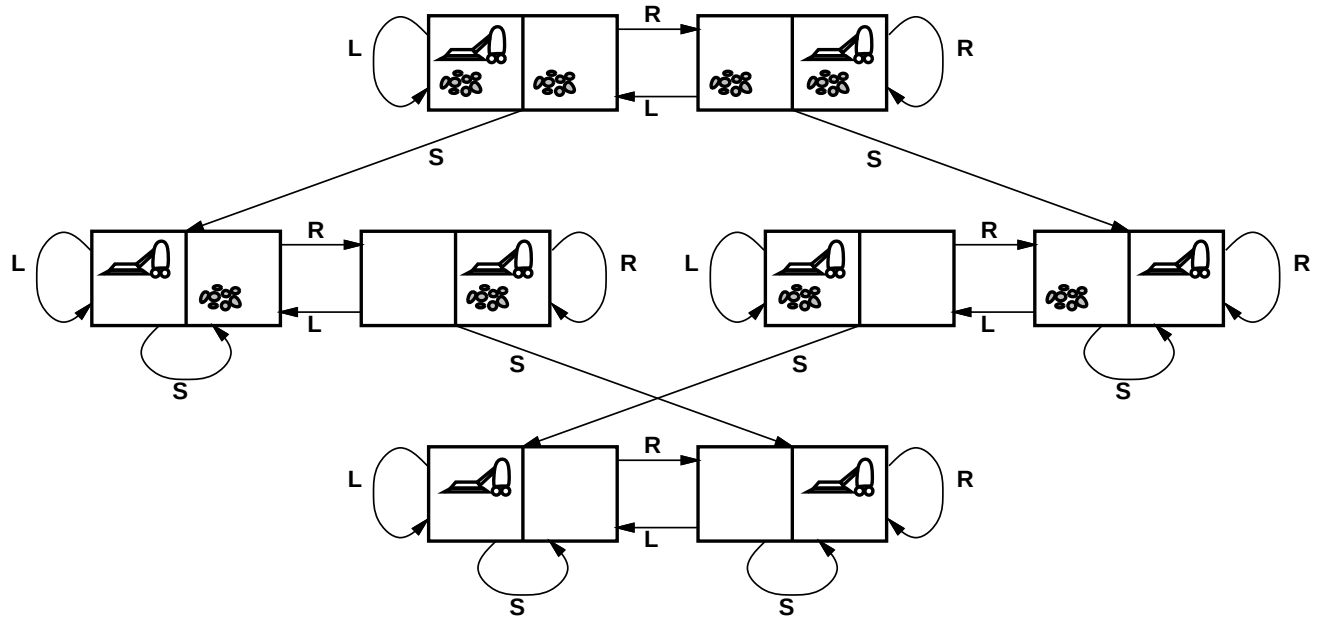
estados??

operadores??

teste do objetivo??

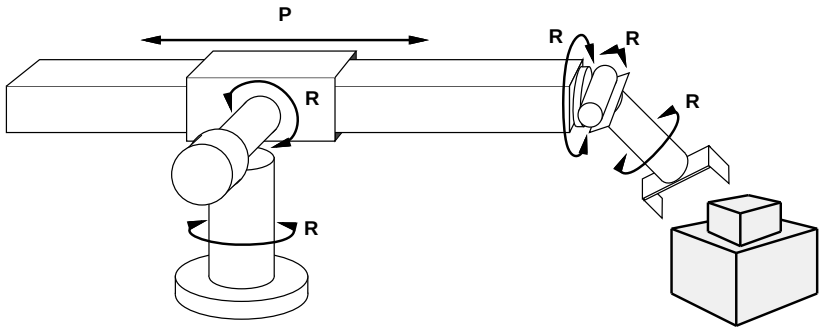
custo da trajetória??

Exemplo: grafo do problema do aspirador



estados??: localização do aspirador e da sujeira (sujeira é variável binária)
operadores??: $Esquerda$, $Direita$, $Aspira$
teste do objetivo??: limpeza total
custo da trajetória??: 1 por operador

Exemplo: Linha de montagem automatizada



estados??: coordenadas reais dos
ângulos das juntas do robô
partes do objeto por montar

operadores??: movimentos contínuos das juntas do robô

teste do objetivo??: montagem completa do objeto

custo da trajetória??: tempo para execução

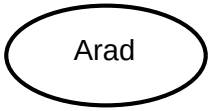
Algorimos de busca

Idéia:

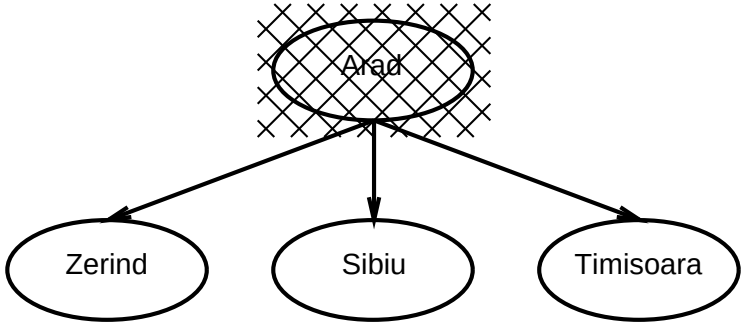
exploração simulada do espaço de estados
via geração dos sucessores dos estados já explorados
(estados *expansores*)

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

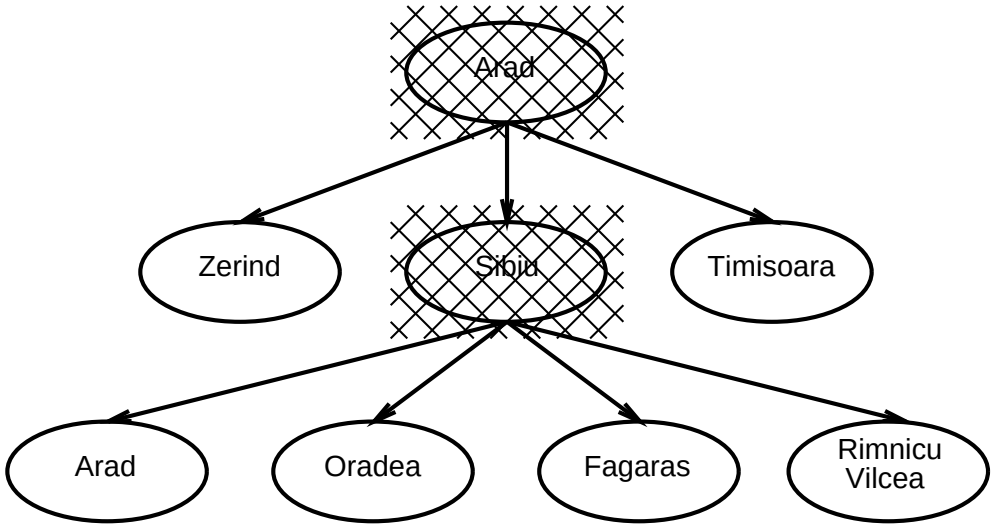
Busca genérica: exemplo



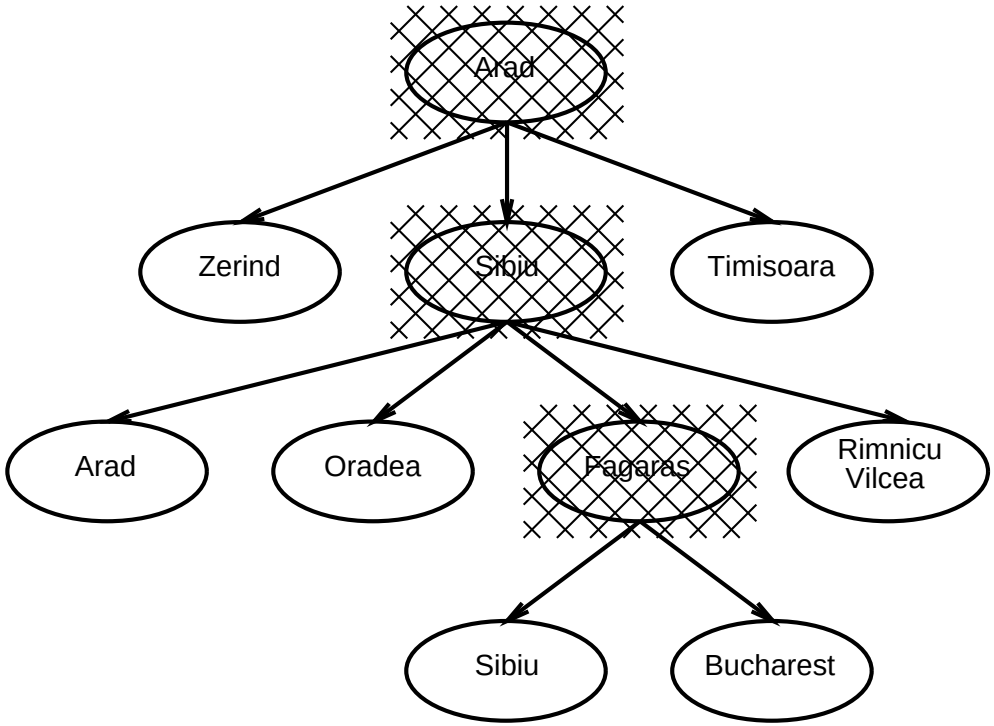
Busca genérica: exemplo



Busca genérica: exemplo



Busca genérica: exemplo



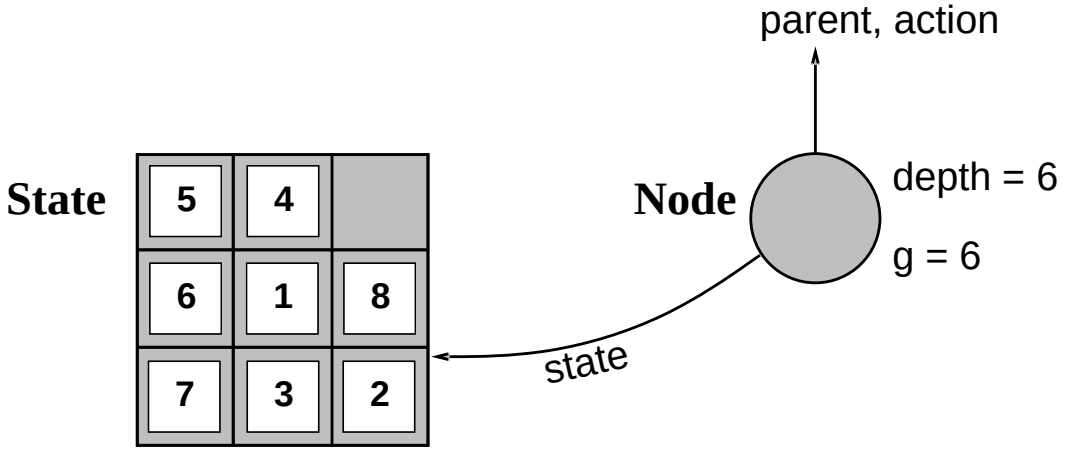
Implementação de algoritmos de busca

```
function General-Search( problem, Queuing-Fn) returns a solution, or failure

    nodes ← Make-Queue(Make-Node(Initial-State[problem]))
    loop do
        if nodes is empty then return failure
        node ← Remove-Front(nodes)
        if Goal-Test[problem] applied to State(node) succeeds then return node
        nodes ← Queuing-Fn(nodes, Expand(node, Operators[problem]))
    end
```

Estados vs. nós

Um *estado* é uma (representação de) uma configuração física
Um *nó* é uma estrutura de dado constituinte de uma árvore de busca
e inclui *pai*, *filhos*, *profundidade*, *custo da trajetória* $g(x)$
Estados não têm *pai*, *filhos*, *profundidade*, ou *custo da trajetória*!



A função EXPAND cria novos nós, preenchendo os respectivos campos e usando os OPERADORES (ou SUCCESSORFN) do problema para criar os estados correspondentes.

Estratégias de busca

Uma estratégia de busca é uma escolha da *ordem de expansão dos nós*

Estratégias são avaliadas pelos seguintes parâmetros:
completeza—a solução (quando existe) é sempre encontrada?
complexidade no tempo—número de nós gerados/expandidos
complexidade no espaço—número máximo de nós na memória
otimalidade—a solução encontrada é de custo mínimo?

Complexidades no tempo e espaço são medidas em termos de
b—fator de ramificação máximo da árvore de busca
d—profundidade da solução de custo mínimo
m—profundidade máxima do espaço de busca (pode ser ∞)

Estratégias de busca desinformada

Estratégias *desinformadas* usam apenas a informação disponibilizada pela definição do problema

Busca em largura

Busca de custo uniforme

Busca em profundidade

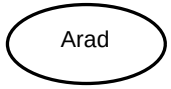
Busca em profundidade limitada

Busca de aprofundamento iterativo

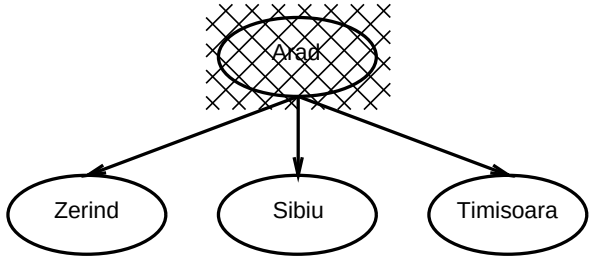
Busca em largura

Expande o nó menos profundo

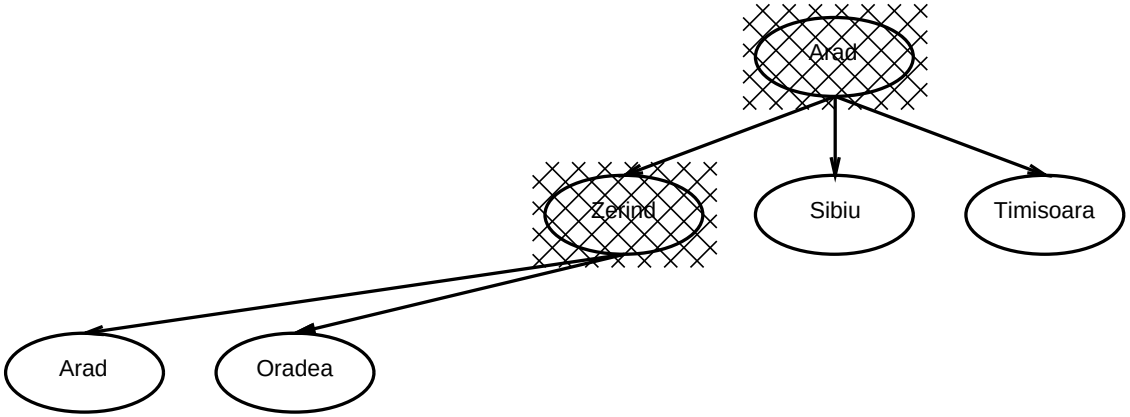
Implementação:
QueueingFn = coloca sucessores no final da fila



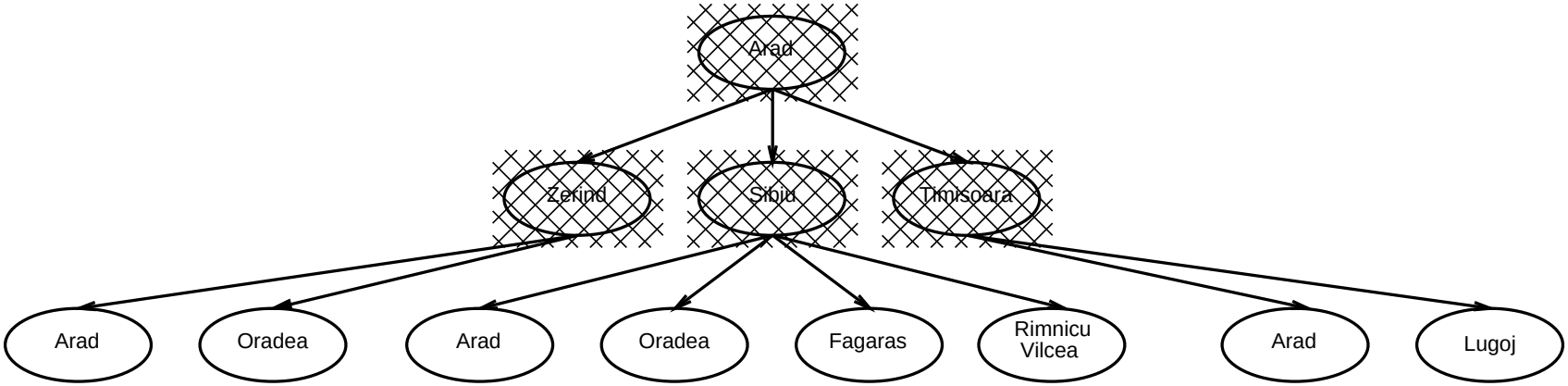
Busca em largura



Busca em largura



Busca em largura



Propriedades da busca em largura

Completa?? Sim (se b for finito)

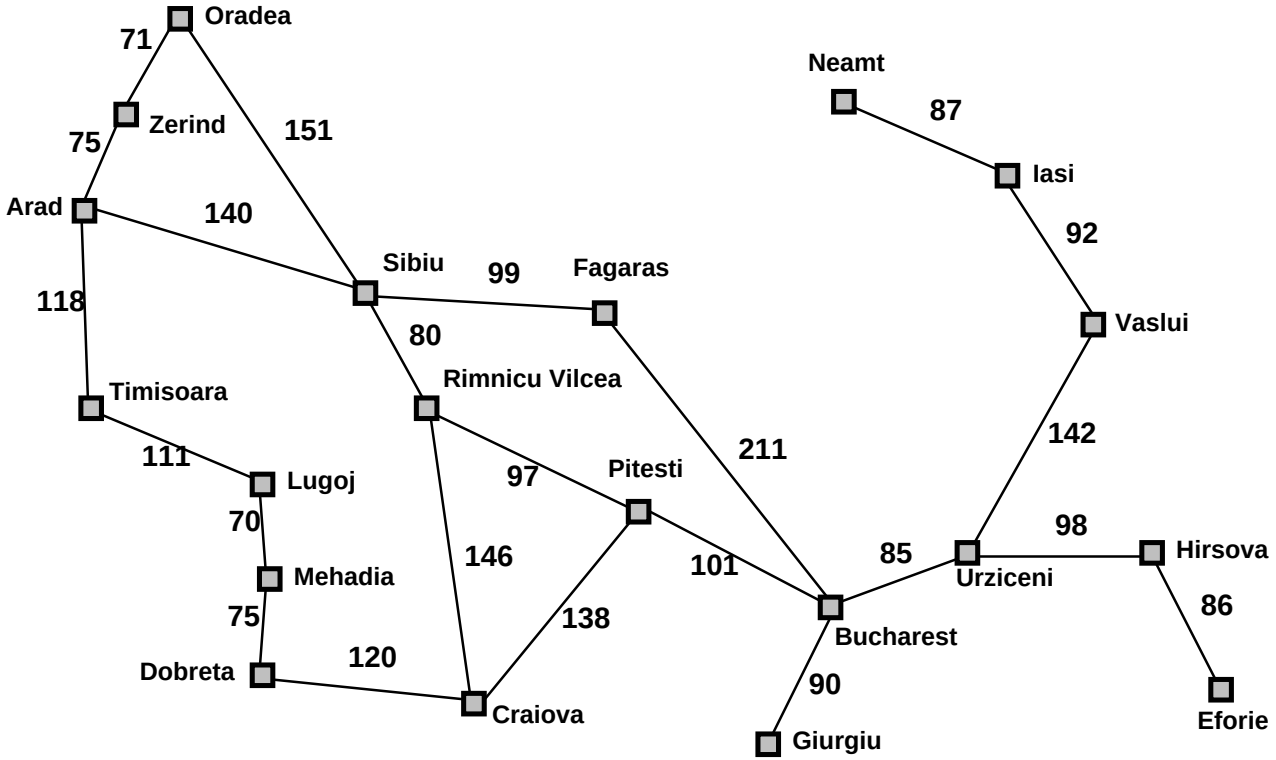
Tempo?? $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$, i.e., exponencial em d

Espaço?? $O(b^d)$ (mantém todos os nós na memória)

Ótima?? Sim (se custo por passo $= 1$); em geral não-ótima

Espaço é o grande problema; pode facilmente gerar nós à razão de 1MB/seg (24hrs = 86GB !!!).

Romênia com custos por passo em km

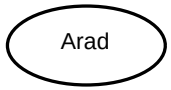


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

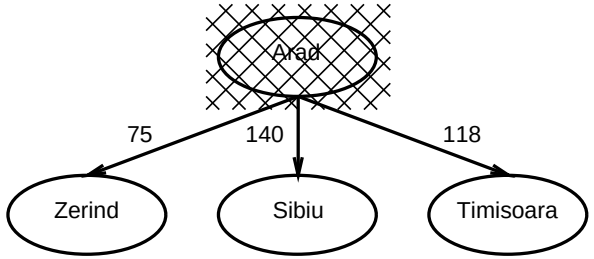
Busca de custo uniforme

Expandir nó de custo mínimo

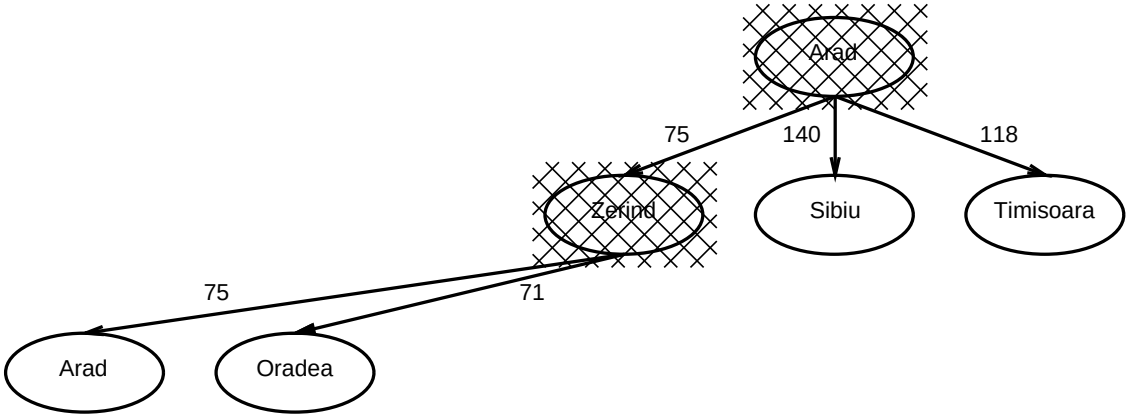
Implementação:
$\textsc{QueueingFn}$ = inserir em ordem de custo crescente



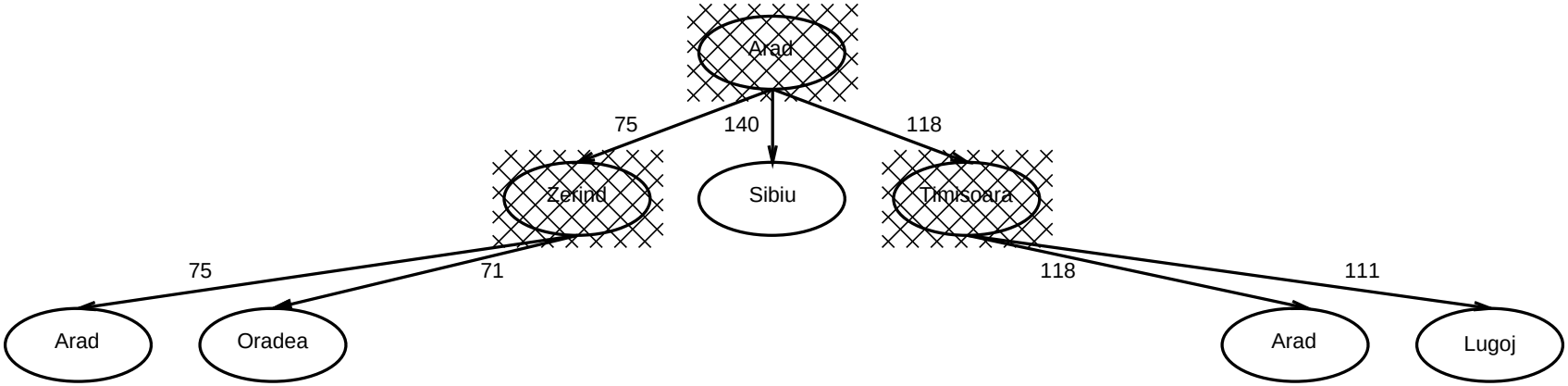
Busca de custo uniforme



Busca de custo uniforme



Busca de custo uniforme



Propriedades da busca de custo uniforme

Completa?? Sim

Tempo?? # de nós com $g \leq$ custo da solução ótima

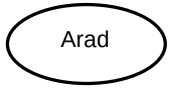
Espaço?? # de nós com $g \leq$ custo da solução ótima

Ótima?? Sim, se o custo por passo ≥ 0

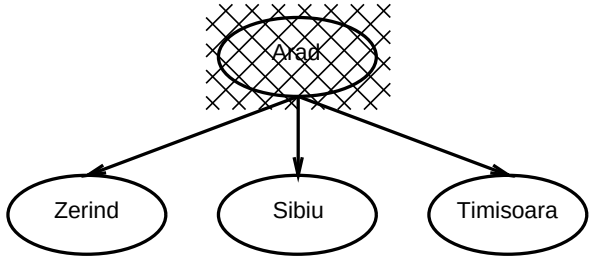
Busca em profundidade

Expandir o nó mais profundo

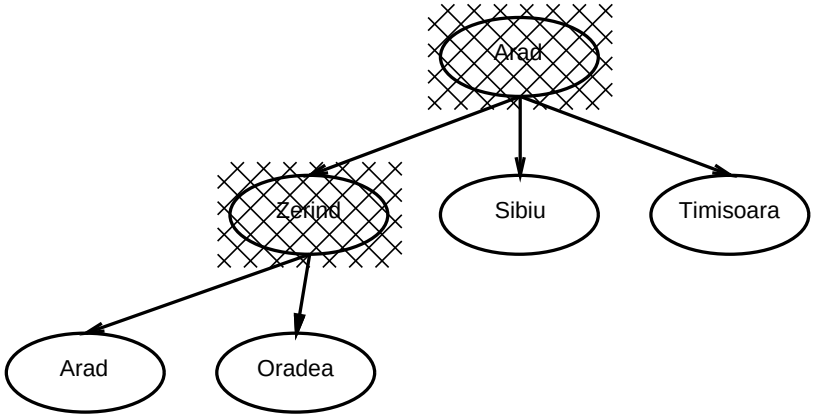
Implementação:
$\textsc{QueueingFn}$ = insere sucessores na frente da fila



Busca em profundidade



Busca em profundidade

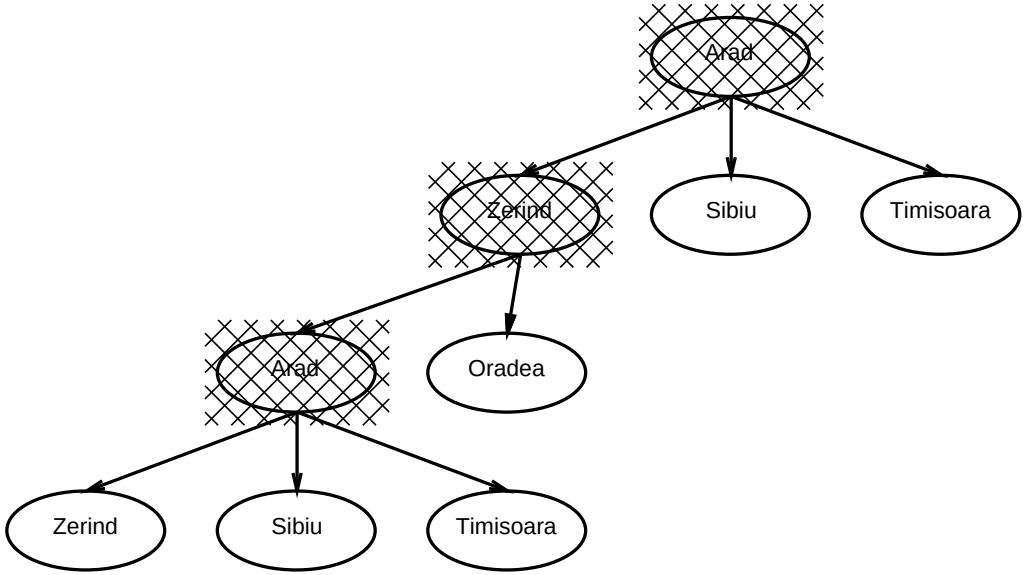


Busca em profundidade

.

.

.

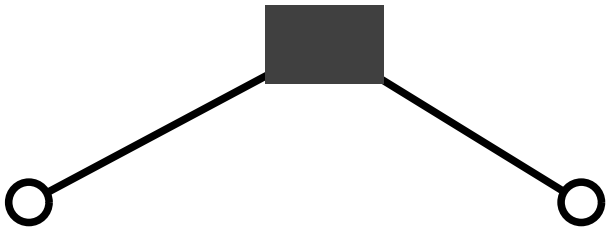


I.e., busca em profundidade pode executar ciclos infinitos

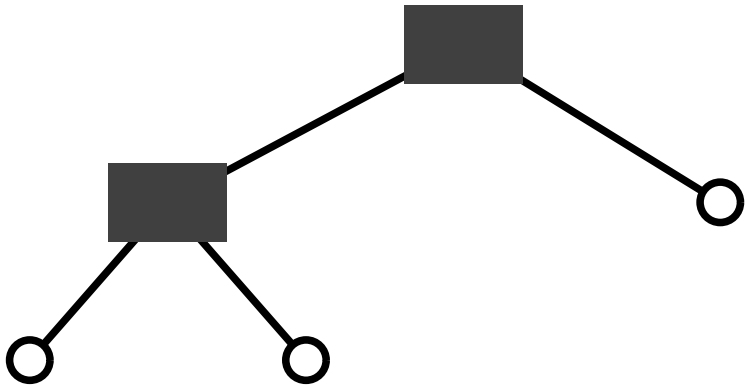
O espaço de busca deve ser finito e acíclico (ou deve haver checagem de estados repetidos)

BeP em uma árvore binária de profundidade 3

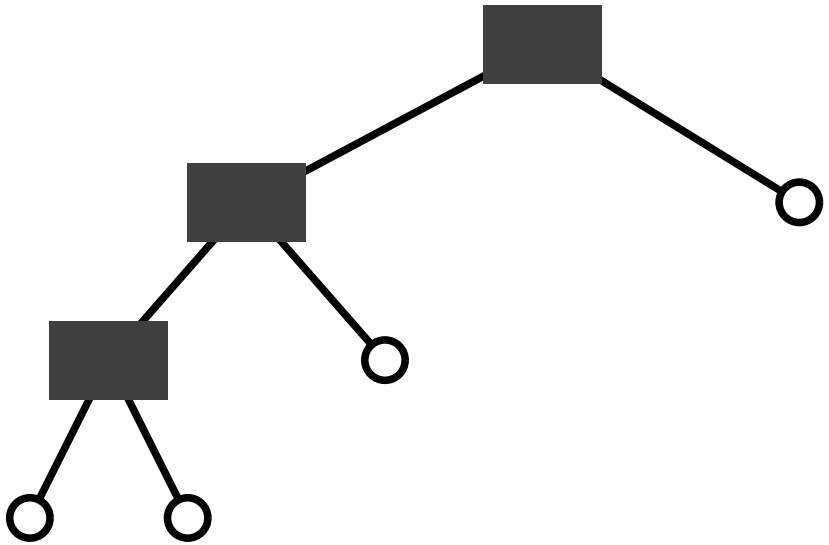
BeP em uma árvore binária de profundidade 3



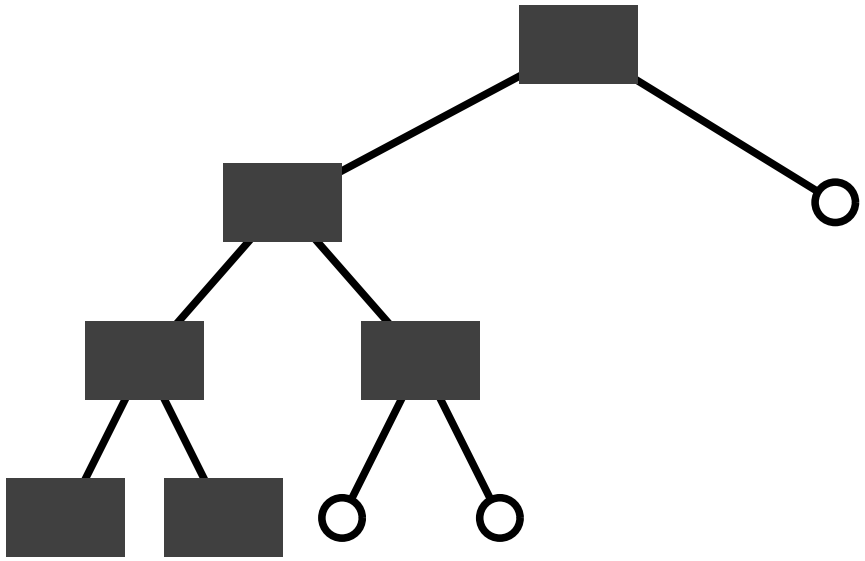
BeP em uma árvore binária de profundidade 3



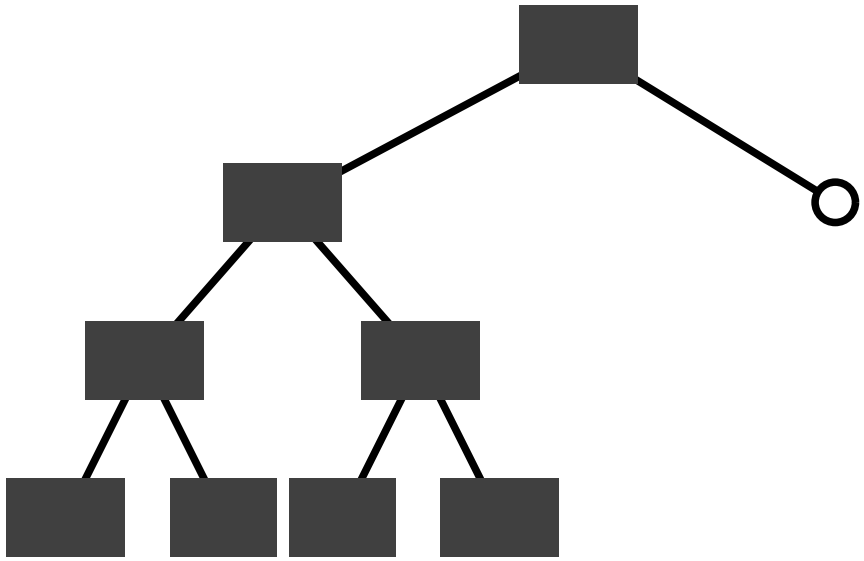
BeP em uma árvore binária de profundidade 3



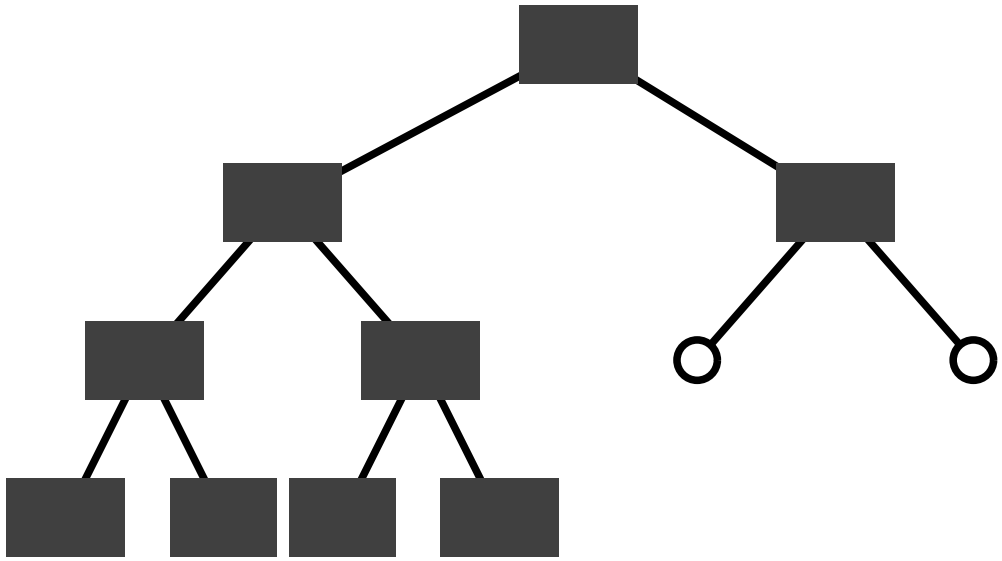
BeP em uma árvore binária de profundidade 3



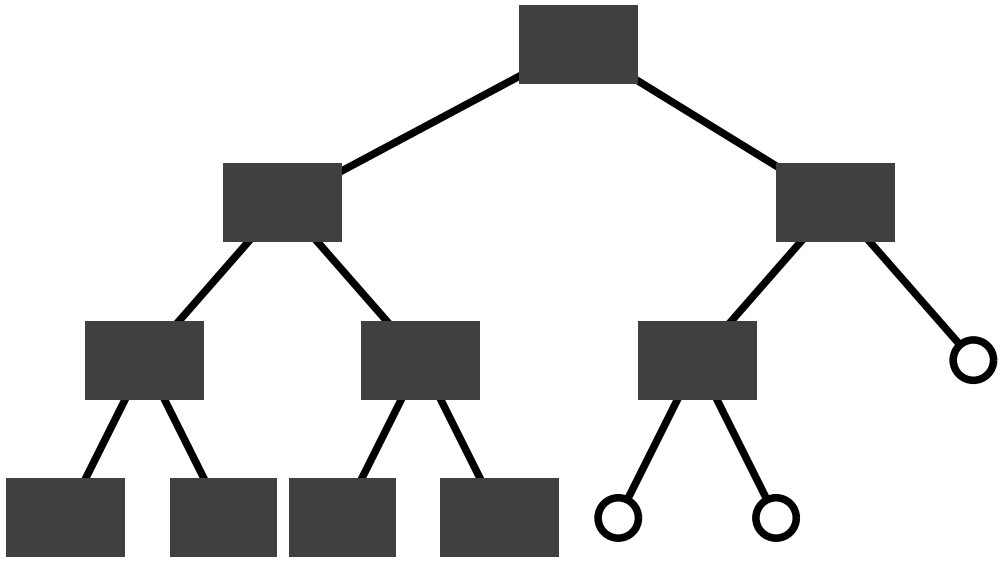
BeP em uma árvore binária de profundidade 3



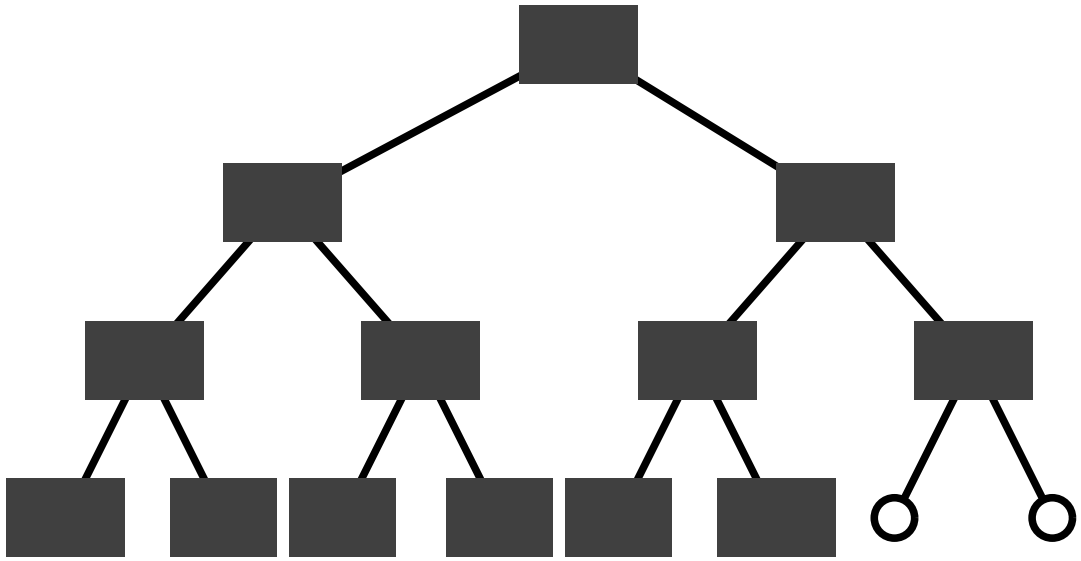
BeP em uma árvore binária de profundidade 3



BeP em uma árvore binária de profundidade 3



BeP em uma árvore binária de profundidade 3



Propriedades da busca em profundidade

Completa?? Não: deficiente em espaços de profundidade infinita ou cíclicos
Modificado para evitar esrados repetidos na trajetória
$\Rightarrow$ completo em espaços finitos

Tempo?? $O(b^m)$: péssimo se m for muito maior do que d
mas se soluções são densas, pode ser muito mais rápido do que em largura

Espaço?? $O(bm)$, i.e., linear no espaço!

Ótima?? Não

Busca em profundidade limitada

= busca em profundidade com limite de profundidade l

Implementação:
Nós à profundidade l não têm sucessores

Problema: não sei a profundidade da solução!

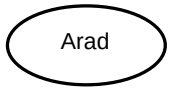
Busca de aprofundamento iterativo

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution sequence

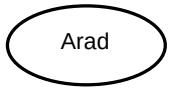
inputs: *problem*, a problem

for *depth* $\leftarrow$ 0 **to** ∞ **do**
  result $\leftarrow$ DEPTH-LIMITED-SEARCH(*problem*, *depth*)
  if *result* $\neq$ cutoff **then return** *result*
end

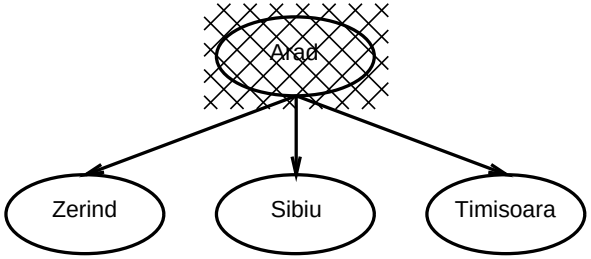
Busca de aprofundamento iterativo $l = 0$



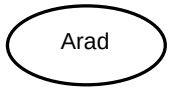
Busca de aprofundamento iterativo $l = 1$



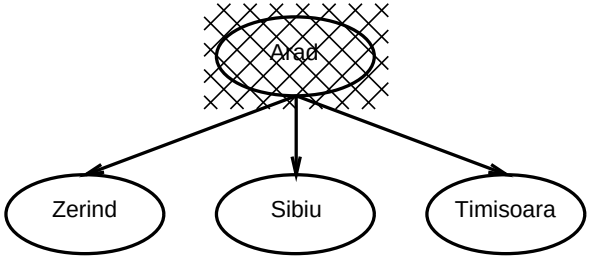
Busca de aprofundamento iterativo $l = 1$



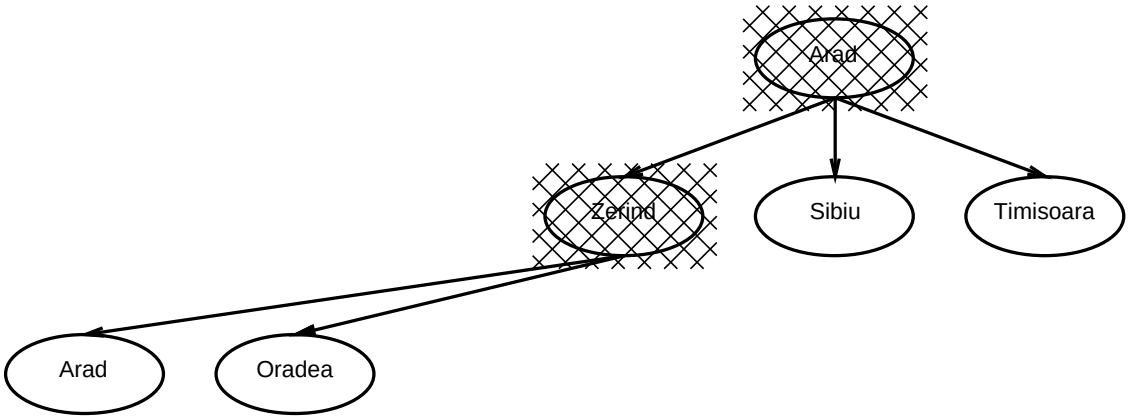
Busca de aprofundamento iterativo $l = 2$



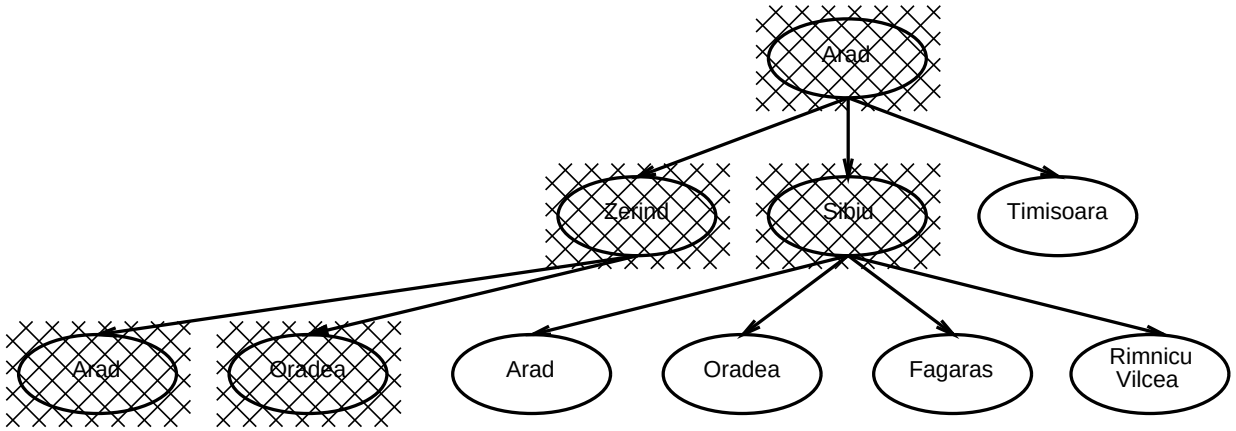
Busca de aprofundamento iterativo $l = 2$



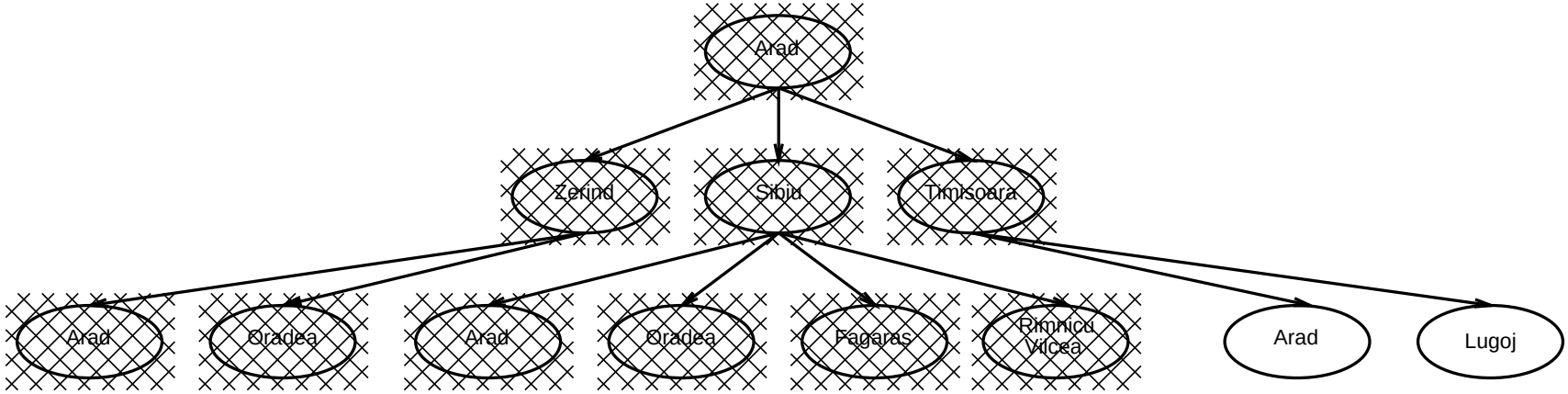
Busca de aprofundamento iterativo $l = 2$



Busca de aprofundamento iterativo $l = 2$



Busca de aprofundamento iterativo $l = 2$



Propriedades da busca de aprofund. iterativo

Completa?? Sim

Tempo?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Espaço?? $O(bd)$

Ótima?? Sim, se custo por passo $= 1$

Resumo

- A formulação de um problema requer abstração para evitar detalhes irrelevantes do mundo real, de modo a se definir um espaço de estados que possa ser explorado.
- Existem várias estratégias de busca não-informada.
- Busca de aprofundamento iterativo é linear no espaço
  e não requer muito mais tempo do que os outros algoritmos.