

# Inteligência Artificial

## Solução de Problemas e Busca

Prof. Dr. Tiago Araújo

# Agentes de Resolução de Problemas

- Forma restrita de agente geral:

```
function AGENTE-SIMPLES-RESOLUCAO-PROBLEMAS(p) returns acao
  inputs: p, percepcao
  static: s, uma sequencia de acoes, comeca vazio
           estado, alguma descricao do estado do mundo
           g, objetivo, comeca vazio
           problema, a formulacao do problema

  estado ← ATUALIZA-ESTADO(estado, p)
  se s esta vazio entao
    g ← FORMULA-OBJETIVO(estado)
    problema ← FORMULA-PROBLEMA(estado, g)
    s ← BUSCA(problema)
  acao ← RECOMMENDACAO(s, estado)
  s ← RESTO(s, estado)
return acao
```

Nota: isto é em solução de problemas offline.

A solução de problemas on-line envolve agir sem conhecimento do problema e da solução.

## Exemplo: Romênia

De férias na Romênia; atualmente em Arad.

O voo parte amanhã de Bucareste

### Formular meta:

estar em Bucareste

### Formular o problema:

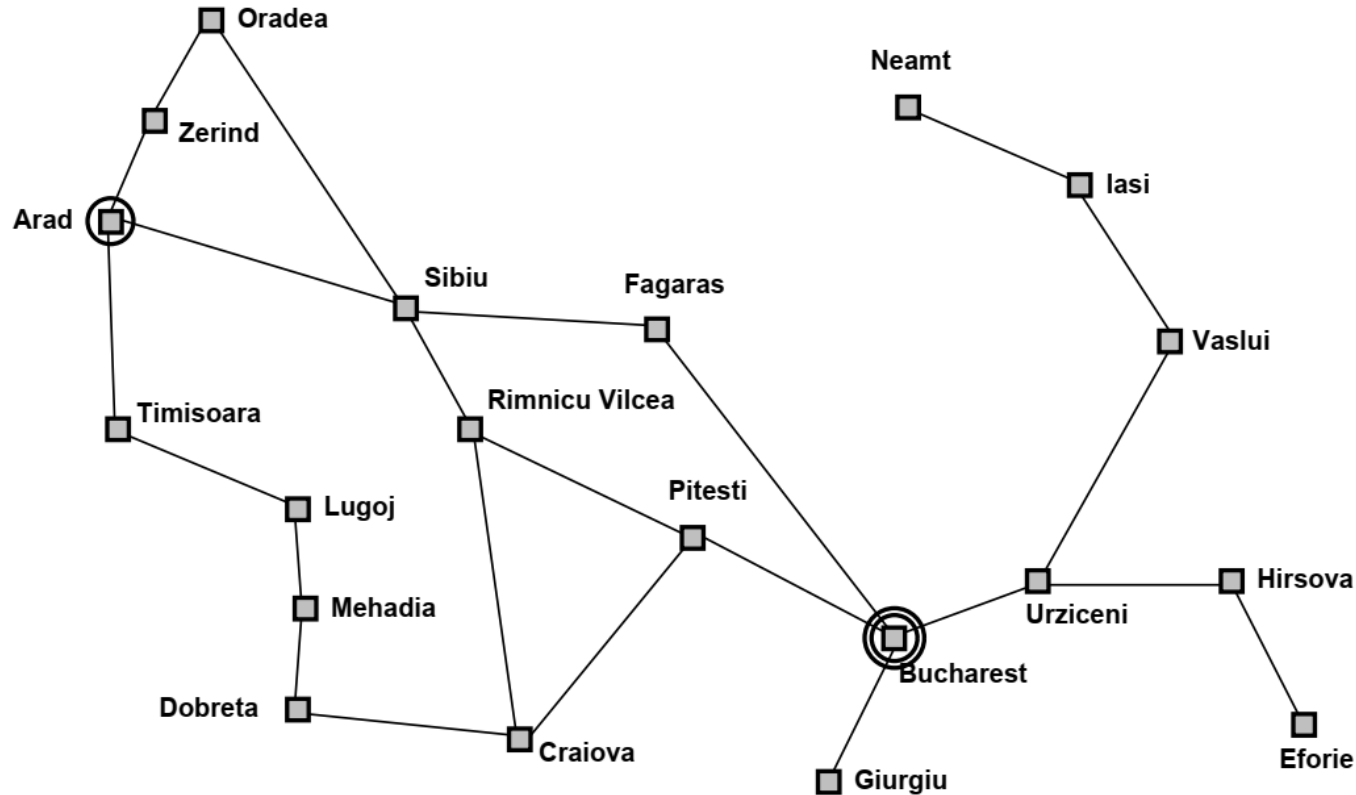
*estados*: várias cidades

*operadores*: dirigir entre cidades

### Encontrar solução:

sequência de cidades, por exemplo, Arad, Sibiu, Fagaras, Bucareste

# Exemplo: Romênia



# Tipos de Problemas

Determinístico, acessível ➡ problema do estado único

Determinístico, inacessível ➡ problema dos múltiplos estados

Não determinístico, inacessível ➡ problema de contingência

devem usar sensores durante a execução

solução é uma *árvore* ou **política** em

muitas vezes se **entrelaça** busca, execução

Espaço de estado não reconhecido ➡ problemas de exploração (“online”)

# Exemplo: Mundo do Aspirador

Estado único, começa em #5. Solução?

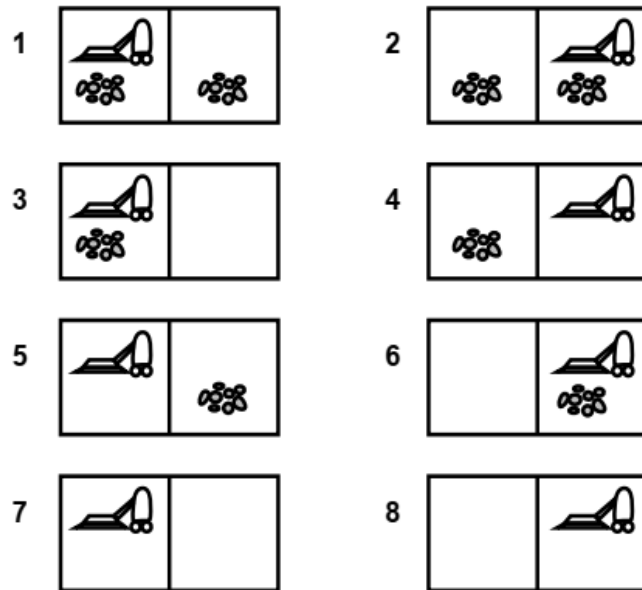
Estado múltiplo, começa em {1, 2, 3, 4, 5, 6, 7, 8}  
por exemplo.; *Direita* vai para {2, 4, 6, 8}. Solução?

Contingência, começa em #5

Lei de Murphy: *Sugar* (ação)  
*pode sujar seu carpete limpo*

Sensoriamento local: sujeira, apenas localização.

Solução??



# Formulação de problemas de Estado Único

- Um *problema* é definido por quatro itens:

estado inicial por exemplo “em Arad”

operadores (ou função sucessora  $S(x)$ )

por exemplo., Arad  Zerind      Arad  Sibiu      etc...

teste de objetivo, pode ser

*explícito* por exemplo.,  $x = \text{“Em Bucareste”}$

*implícito* por exemplo.,  $NoDirt(x)$

custo do caminho (aditivo)

por exemplo., sois de distância, número de operadores executados, etc.

Uma *solução* é a sequência de operadores levando dos estados iniciais a um estado de objetivo


# Selecione um Estado de espaço

O mundo real é absurdamente complexo

 o espaço de estado deve ser *abstraído* para a solução de problemas

(Abstração) estado = conjunto de estados reais

(Abstração) operador = combinação complexa de ações reais,

por exemplo  "Arad → Zerind" representa um

conjunto complexo de rotas possíveis, desvios, paradas de descanso, etc.

Para uma realização garantida, qualquer estado real "em Arad"

deve chegar a *algum* estado real "em Zerind".

(Abstração) solução =

conjunto de caminhos reais que são soluções em três mundos reais

Cada ação abstrata deve ser "mais fácil" do que o problema original!



## Exemplo: O quebra cabeça de 8

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

estados??

operadores??

teste de objetivo??

custo do caminho??

## Exemplo: O enigma de 8

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

estados?? localizações inteiras das telhas(ignorar posições intermediárias)

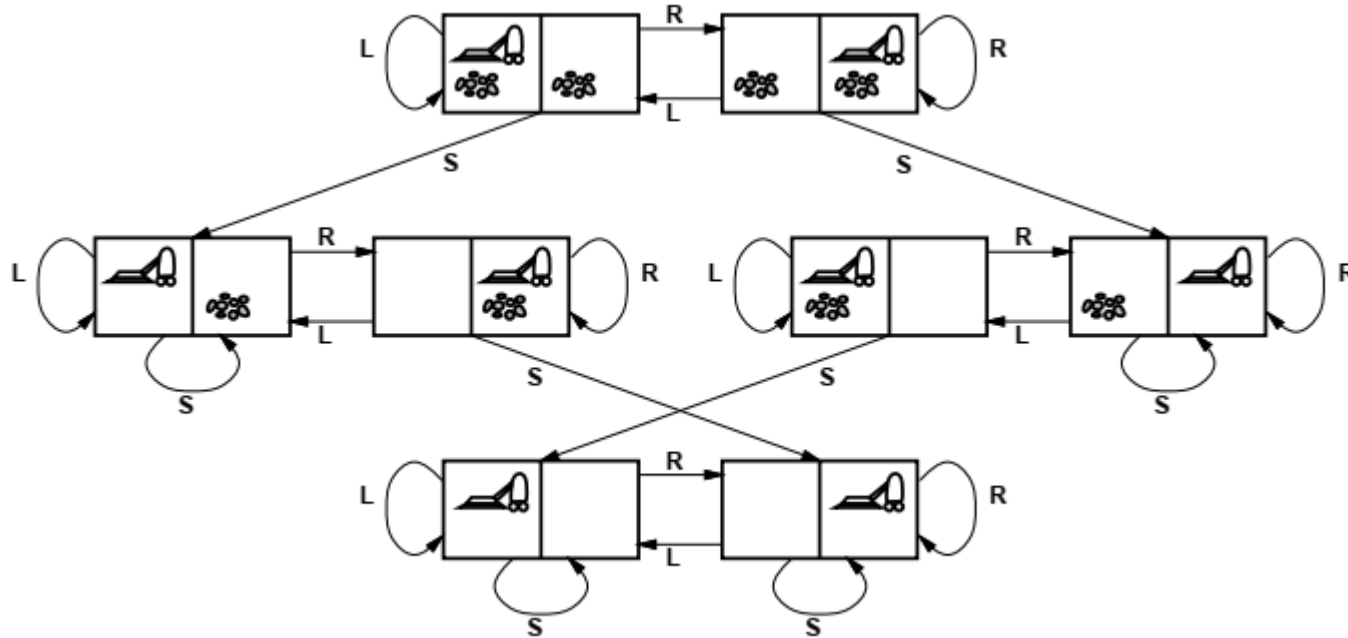
operadores?? mover-se para a esquerda, direita, para cima, para baixo (ignorar os limites, etc.)

teste de objetivo?? Estado objetivo (dado)

custo do caminho?? 1 por movimento

[Nota: a solução ótima da família  $n$ - Enigma é NP-difícil]

# Exemplo: Gráfico dos estados do mundo do aspirador



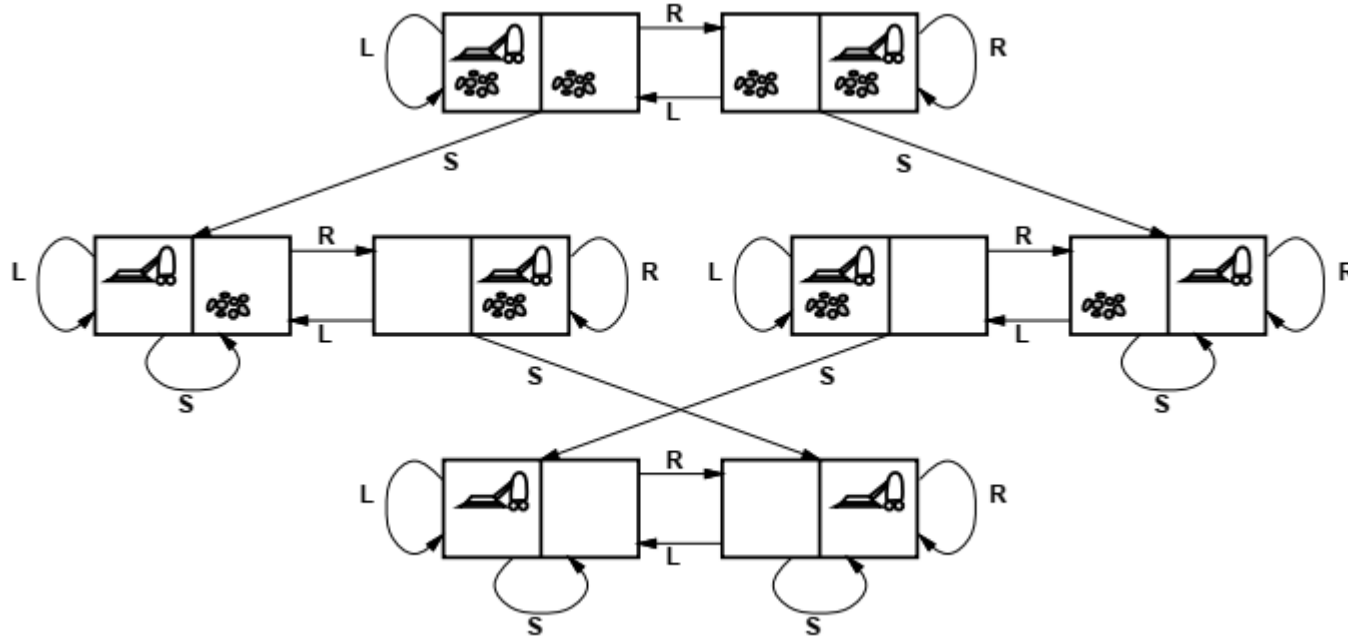
estados??

operadores??

teste de objetivo??

custo do caminho??

## Exemplo: Gráfico dos estados do mundo do aspirador



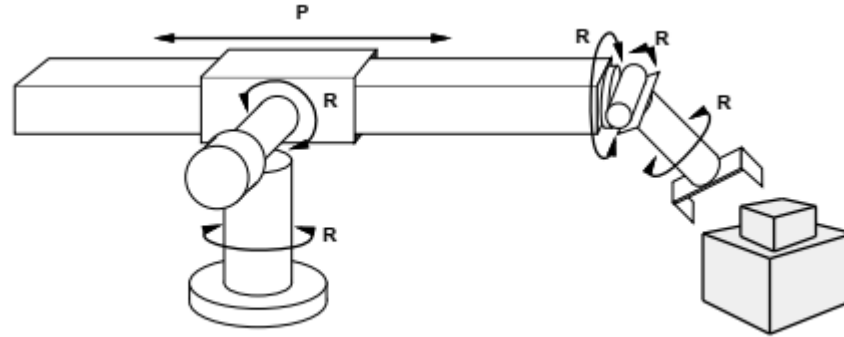
estados?? sujeira inteira e localização de robôs (ignorar *quantidades* de sujeira)

operadores?? *Direita, Esquerda, Sugar*

teste de objetivo?? sem sujeira

custo do caminho?? 1 por operação

## Exemplo: Montagem de robô



estados?? coordenadas de valor real do

robô unem ângulos partes do objeto a ser montado

operadores?? movimentos contínuos de juntas robotizadas

teste de objetivo?? montagem completa, *sem robô incluído!*

custo do caminho?? Tempo para executar

# Algoritmos de Busca

## ❖ Ideia básica:

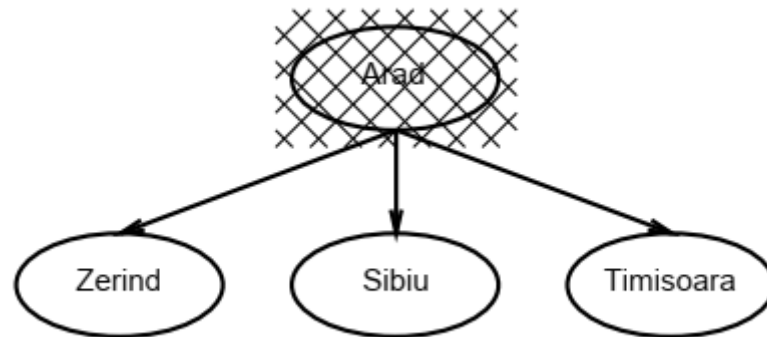
exploração simulada e offline do espaço dos estados gerando sucessores de estados já explorados (também conhecido como estados *em expansão*)

```
function BUSCA-GERAL( estrategia, problema) returns solucao, ou falha
  inicializa a busca da arvore usando o estado inicial do problema
  laco faca
    se nao tem candidatos para expansao entao return falha
    escolha um no folha para expansao de acordo com a estrategia
    se o no tem um estado de objetivo entao
      return a solucao correspondente
    senao expanda o no e adiciona os nos resultantes na arvore de busca
  end
```

# Exemplo de Busca Geral

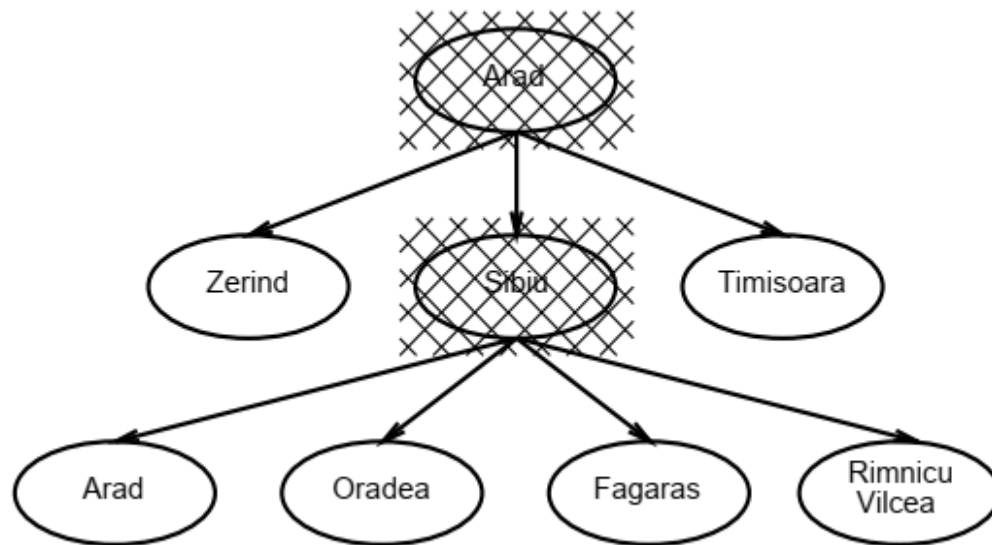


# Exemplo de Busca Geral

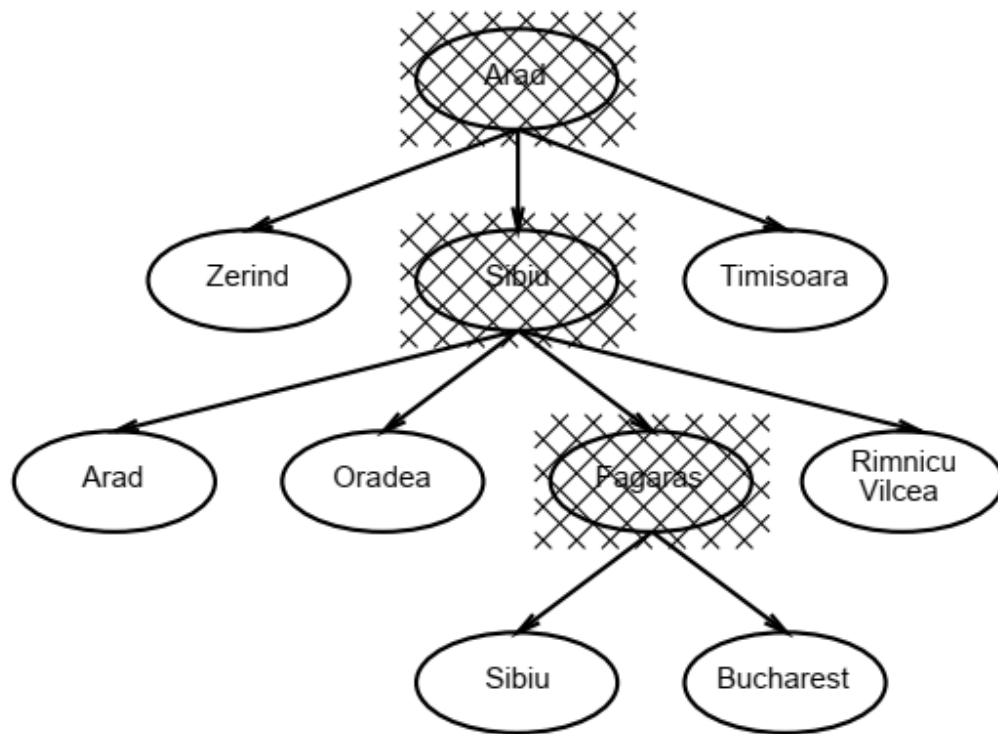




# Exemplo de Busca Geral



# Exemplo de Busca Geral



# Implementação do Algoritmo de Busca Geral

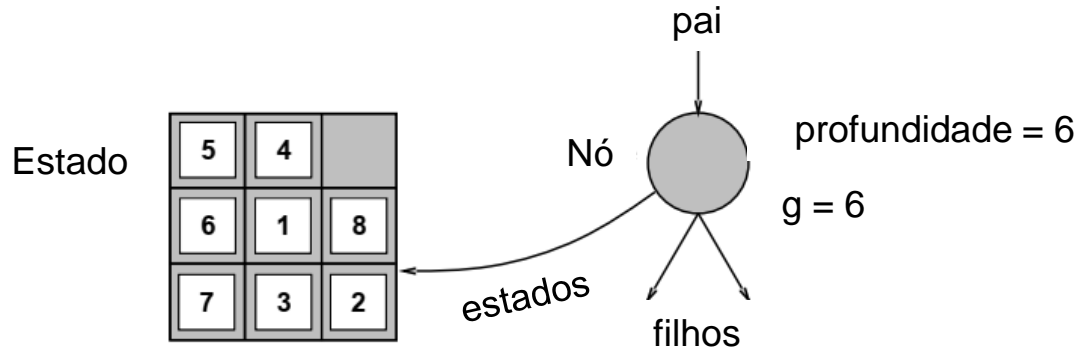
```
function BUSCA-GERAL(problema, FN-FILA) returns a solucao, ou falha  
  nos ← CRIAR-FILA(CRIAR-NO(ESTADO-INICIAL[problema]))  
  loop do  
    se nos esta vazio entao return falha  
    node ← REMOVER-FRENTE(nos)  
    se TESTE-OBJETIVO[problema] aplicado ao ESTADO(no) esta certo  
      entao return node  
    nos ← FN-FILA(nos, EXPANDE(no, OPERADORES[problema]))  
end
```

# Implementação cont: Estados versus. Nós

Um *estado* é uma (representação de) uma configuração física.

Um *nó* é uma estrutura de dados que constitui parte de uma árvore de busca que inclui *pais*, *filhos*, *profundidade*, *custo  $g(x)$  caminho*.

Os estados não têm pais, filhos, profundidade, ou custo do caminho!



A função EXPANDIR cria novos nós, preenchendo os diversos campos e utilizando os OPERADORES (ou SUCESSORFn) do problema para criar os estados correspondentes.

# Estratégias de Busca

Uma estratégia é definida pela escolha da *ordem de expansão do nó*

As estratégias são avaliadas nas seguintes dimensões:

completude - sempre encontra uma solução se ela existe?

complexidade de tempo - número de nós gerados/expandidos\

complexidade de espaço - número máximo de nós na memória

otimalidade - sempre encontra uma solução de menor custo?

A complexidade de tempo e 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 estado (pode ser  $\infty$ )

# Estratégias de Busca Desinformada

Estratégias *desinformadas* utilizam apenas as informações disponíveis na definição do problema

Busca em Largura

Busca de custo uniforme

Busca em profundidade

Pesquisa de profundidade limitada

Busca intencional de aprofundamento

# Busca em Largura

- Expandir o nó não expandido mais superficial

Implementação:

QUEUEINGFN= colocar os sucessores no fim da fila

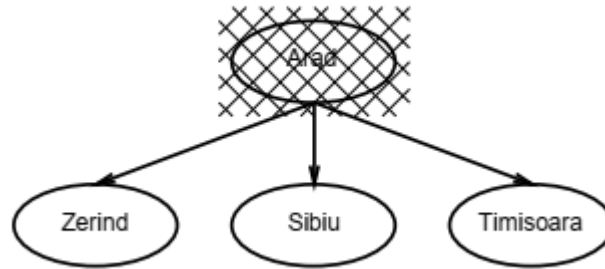


# Busca em Largura

- Expandir o nó não expandido mais superficial

Implementação:

QUEUEINGFN= colocar os sucessores no fim da fila



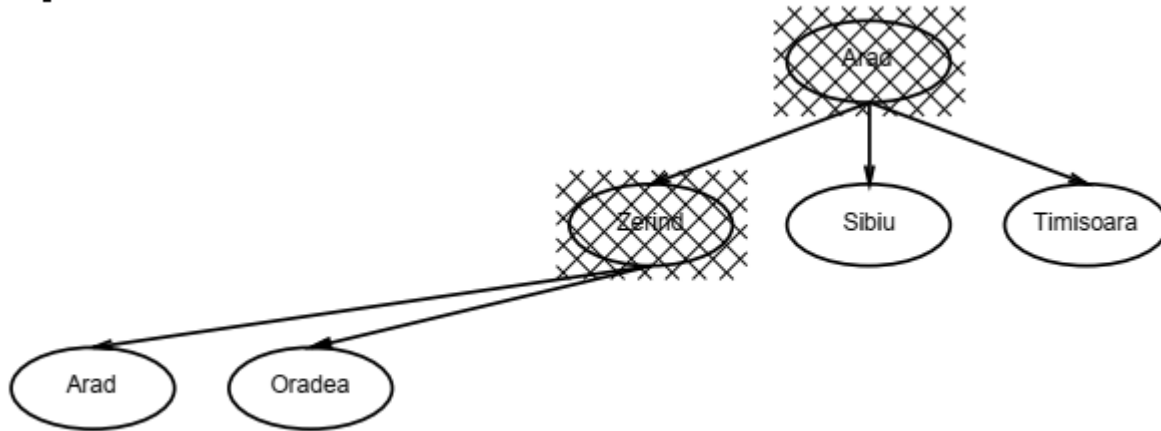


# Busca em Largura

- Expandir o nó não expandido mais superficial

Implementação:

QUEUEINGFN= colocar os sucessores no fim da fila

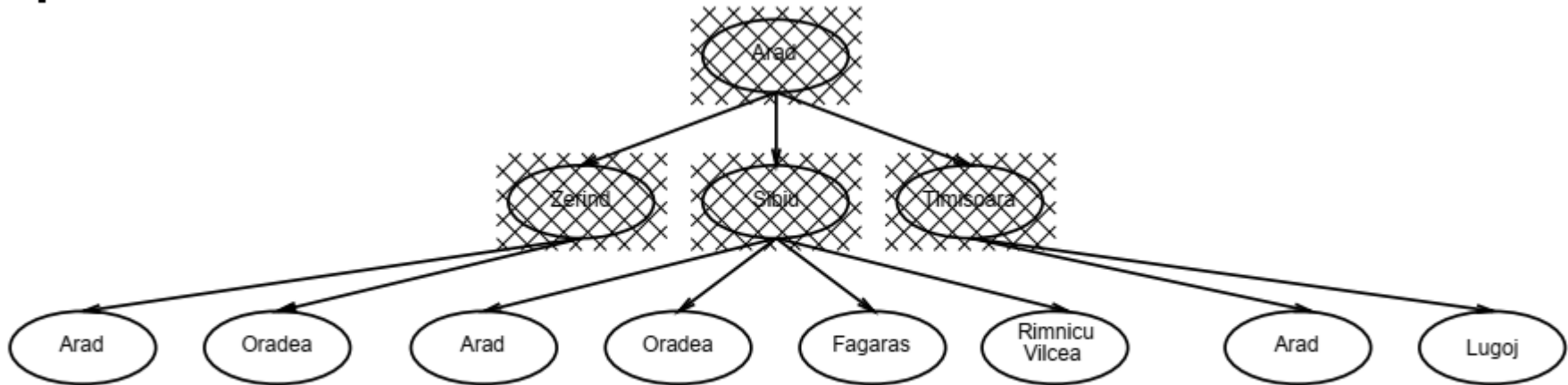


# Busca em Largura

- Expandir o nó não expandido mais superficial

Implementação:

QUEUEINGFN= colocar os sucessores no fim da fila



# Propriedades da Busca em Largura

Completa??

Tempo??

Espaço??

Otimizado??

# Propriedades da Busca em Largura

Completa?? Sim (se  $b$  for finito)

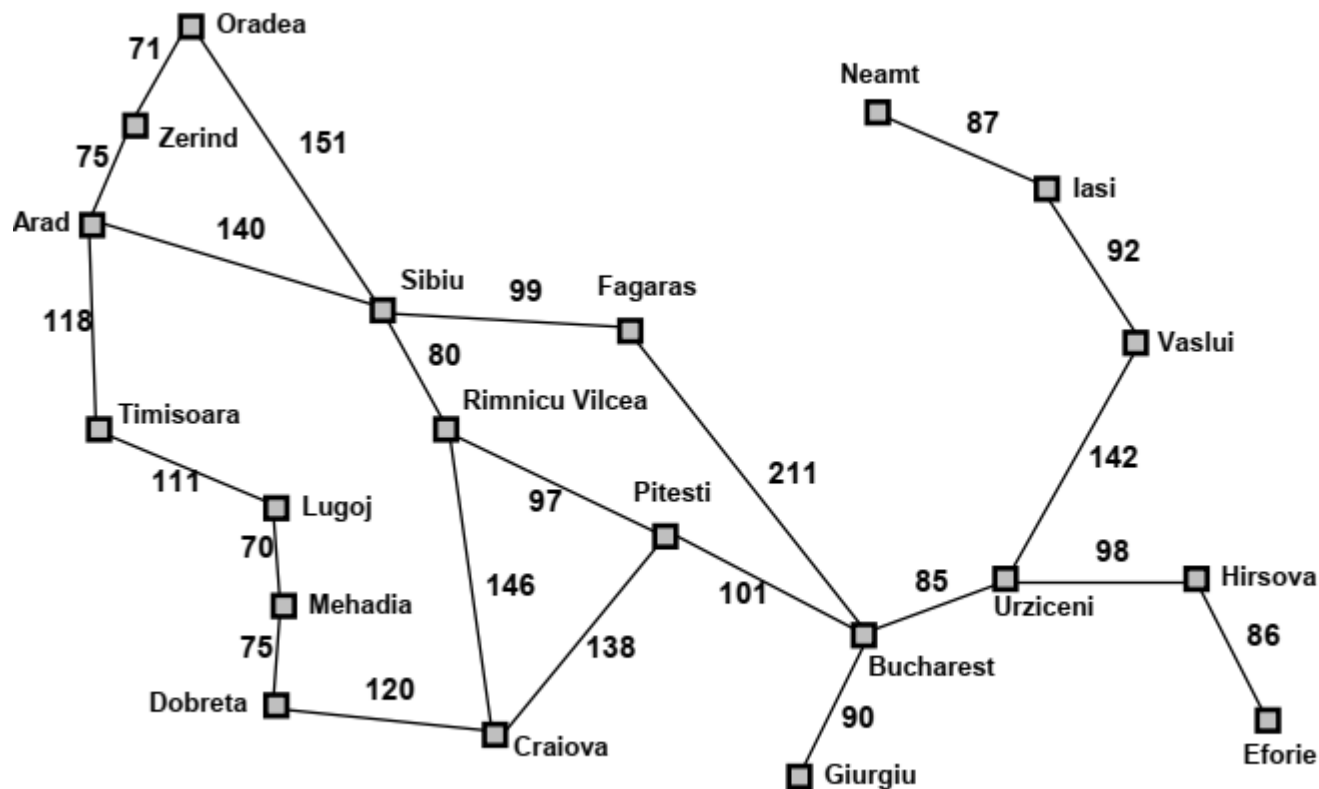
Tempo??  $1 + b + b^2 + b^3 + \dots + b^d = O(b^d)$ , ou seja exponencial em  $d$

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

Ótimo?? Sim (se custo = 1 por passo); não ótimo em geral

*Espaço* é um grande problema, pode facilmente gerar nós a 1MB/seg, por isso  
24hrs=86GB

# România com custo de passos 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 o nó não expandido de menor custo

Implementação:

QUEUEINGFN= inserir em ordem de aumentar o custo do caminho

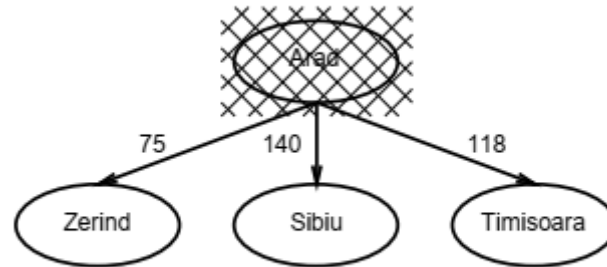


# Busca de Custo Uniforme

- Expandir o nó não expandido de menor custo

Implementação:

QUEUEINGFN= inserir em ordem de aumentar o custo do caminho

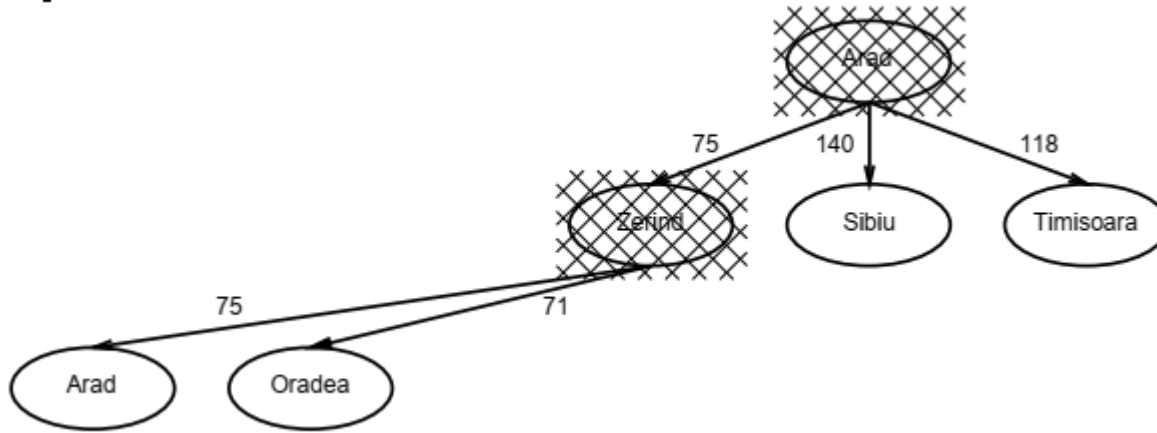


# Busca de Custo Uniforme

- Expandir o nó não expandido de menor custo

Implementação:

QUEUEINGFN= inserir em ordem de aumentar o custo do caminho



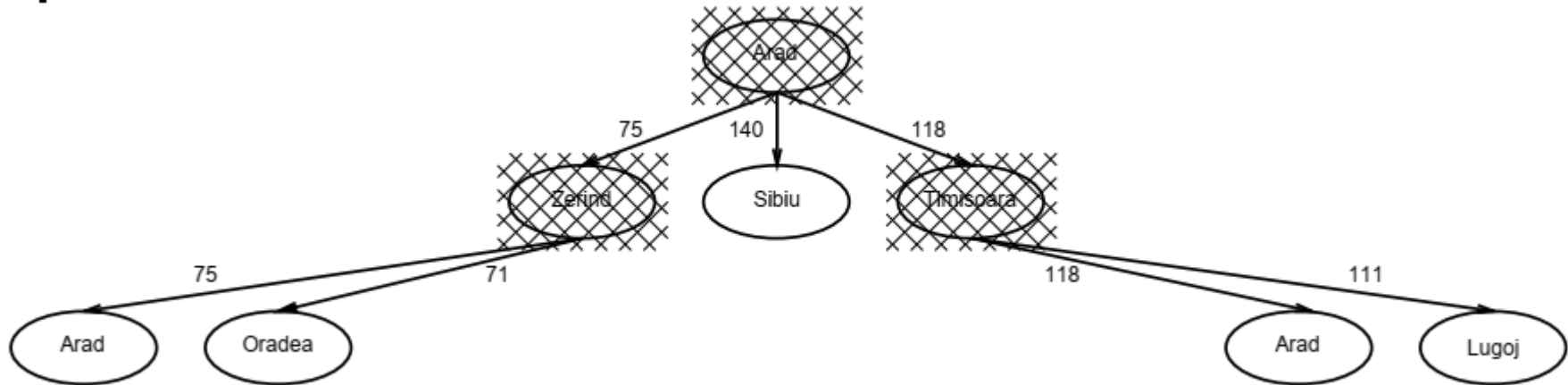


# Busca de Custo Uniforme

- Expandir o nó não expandido de menor custo

Implementação:

QUEUEINGFN= inserir em ordem de aumentar o custo do caminho



# Propriedades da Busca em Largura

Completa?? Sim, se o custo de cada passo for  $\geq E$

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

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

Otimizado?? Sim

# Busca em Profundidade

- Expandir o nó não expandido mais profundo

Implementação:

QUEUEINGFN= inserir sucessores na frente da fila

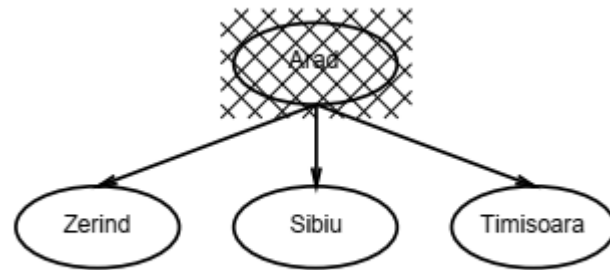


# Busca em Profundidade

- Expandir o nó não expandido mais profundo

Implementação:

QUEUEINGFN= inserir sucessores na frente da fila

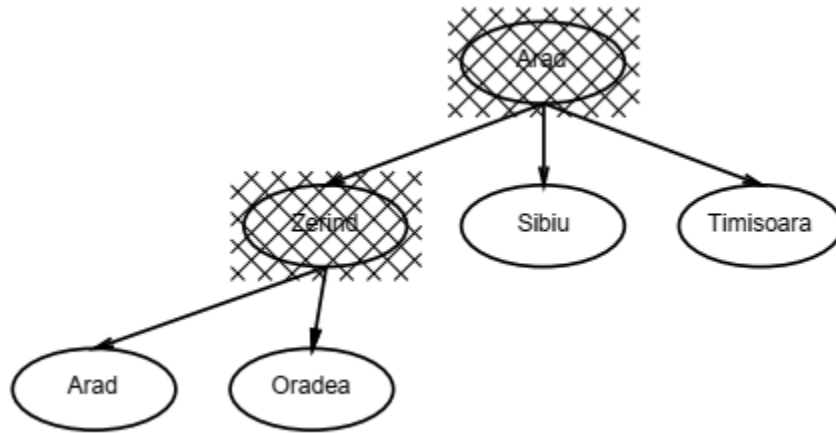


# Busca em Profundidade

- Expandir o nó não expandido mais profundo

Implementação:

QUEUEINGFN= inserir sucessores na frente da fila

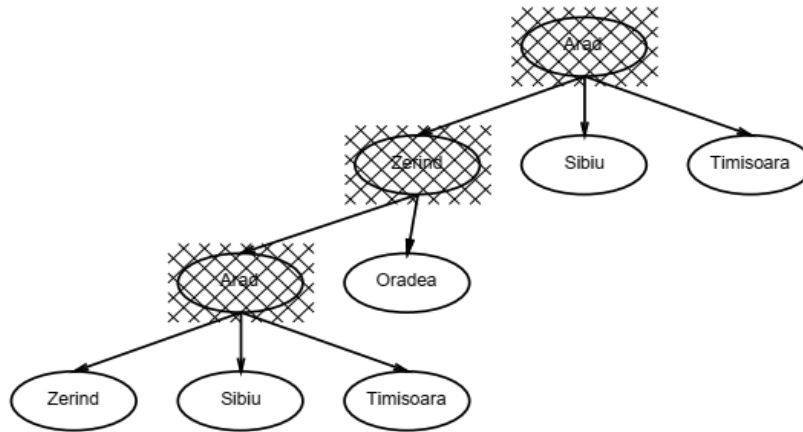


# Busca em Profundidade

- Expandir o nó não expandido mais profundo

## Implementação:

QUEUEINGFN= inserir sucessores na frente da fila



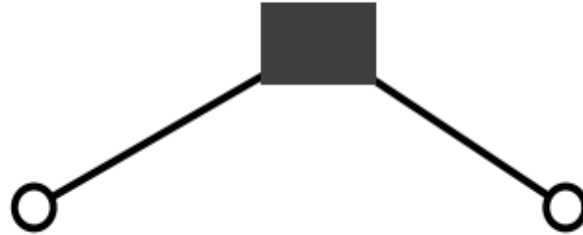
Isto é, a busca em profundidade pode realizar infinitas excursões cíclicas

Precisa de um espaço de busca finito, não cíclico (ou verificação de estado repetido)

# Busca em Profundidade em uma Árvore Binária de profundidade 3

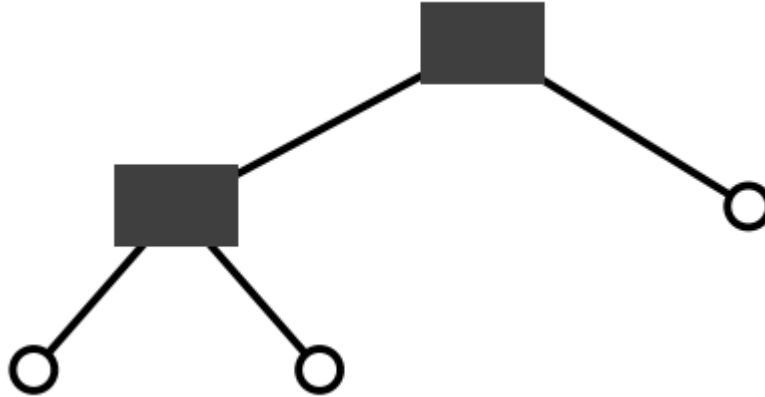


# Busca em Profundidade em uma Árvore Binária de profundidade 3

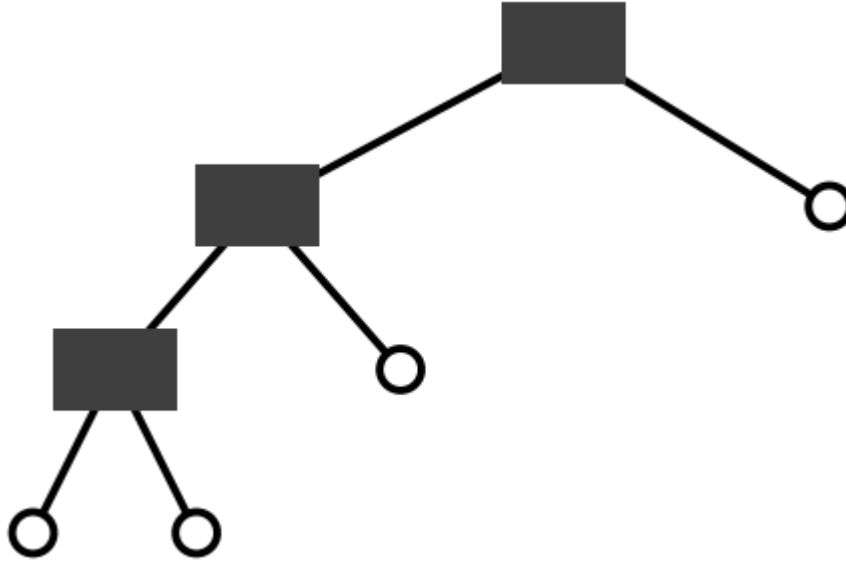




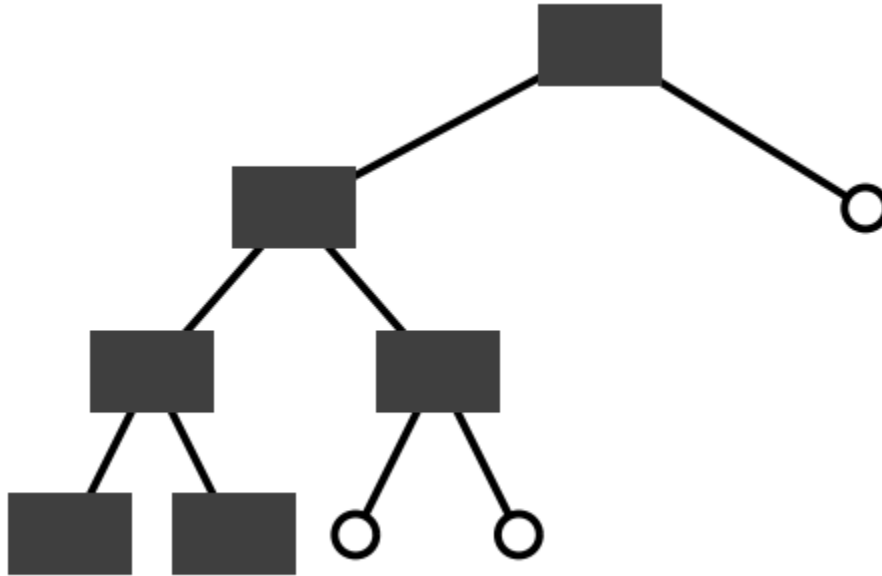
# Busca em Profundidade em uma Árvore Binária de profundidade 3



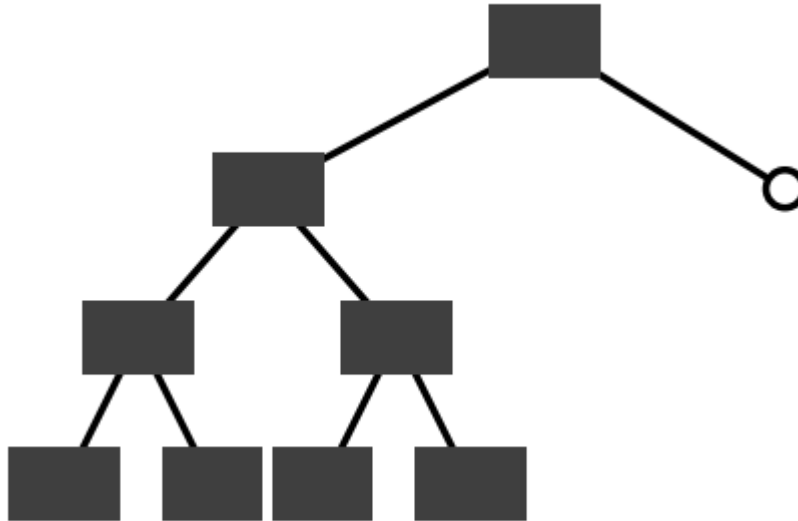
# Busca em Profundidade em uma Árvore Binária de profundidade 3



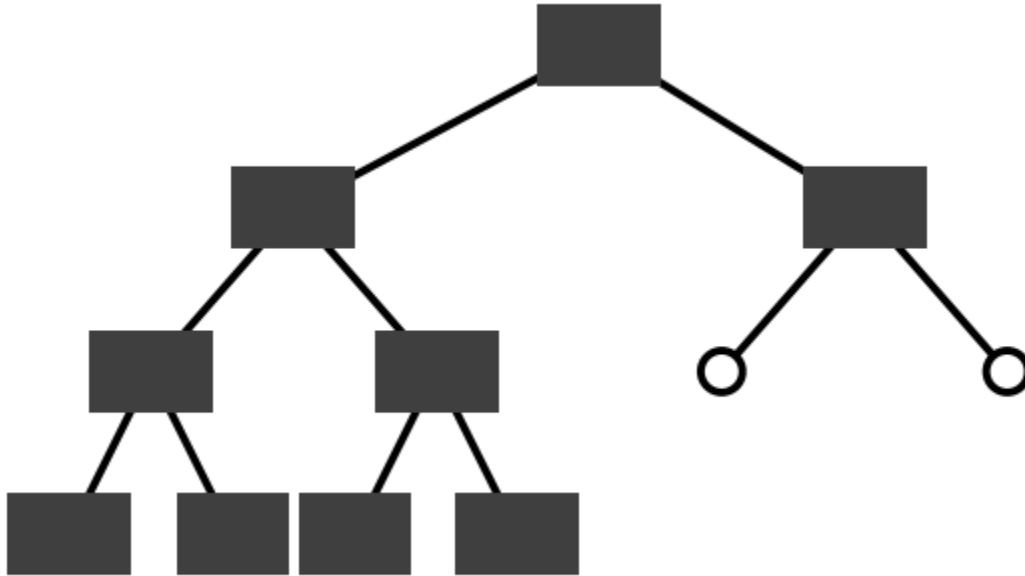
# Busca em Profundidade em uma Árvore Binária de profundidade 3



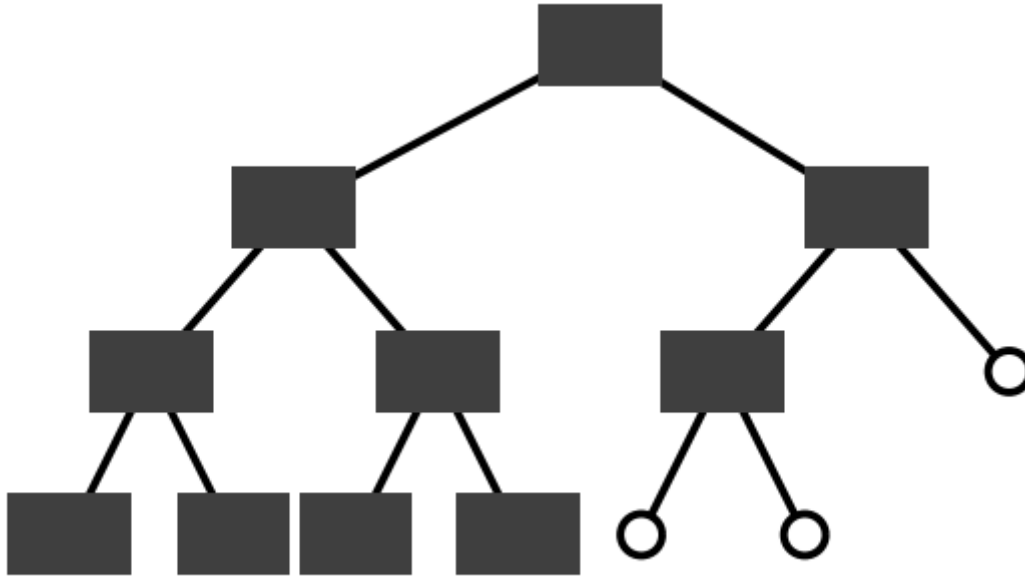
# Busca em Profundidade em uma Árvore Binária de profundidade 3



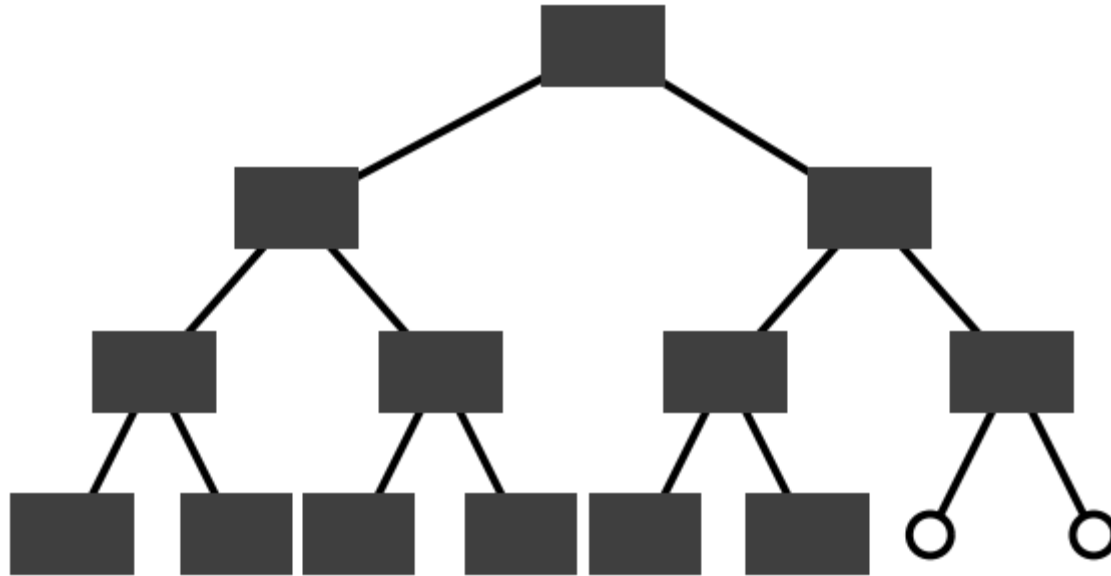
# Busca em Profundidade em uma Árvore Binária de profundidade 3



# Busca em Profundidade em uma Árvore Binária de profundidade 3



# Busca em Profundidade em uma Árvore Binária de profundidade 3



# Propriedades da Busca em Profundidade

Completa??

Tempo??

Espaço??

Otimizado??



# Propriedades da Busca em Profundidade

Completa?? Não: falha em espaços de profundidade infinita, espaços com loops

Modificar para evitar estados repetidos ao longo do caminho

➡ completa 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 a busca em largura.

Espaço??  $O(bm)$ , isto é, espaço linear!

Otimizado?? Não

# Busca em Profundidade limitada

= igual a busca em profundidade mas com uma profundidade limitada /

Implementação:

Os nós em profundidade  $l$  não têm sucessores

# Busca em Profundidade Iterativa

```
function BUSCA-APROFUNDAMENTO-ITERATIVO(problema) returns  
    uma sequencia de solucoes  
    inputs: problema, um problema  
  
    para profundidade  $\leftarrow$  0 ate  $\infty$  faca  
        resultado  $\leftarrow$  BUSCA-PROFUNDIDADE-LIMITADA(problema, profundidade)  
        se resultado  $\neq$  corte entao return resultado  
end
```

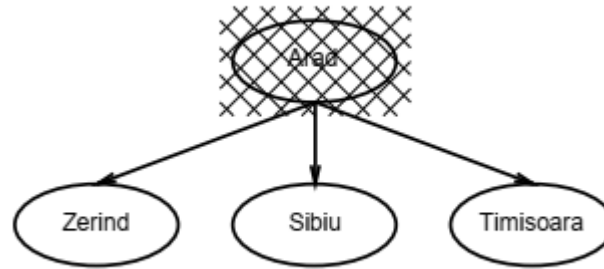
# Busca em Profundidade Iterativa $l=0$



# Busca em Profundidade Iterativa $l=1$



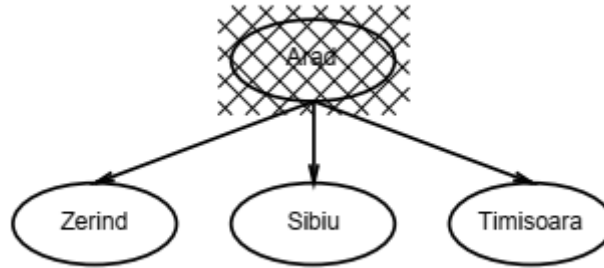
# Busca em Profundidade Iterativa $l=1$



# Busca em Profundidade Iterativa $l=2$

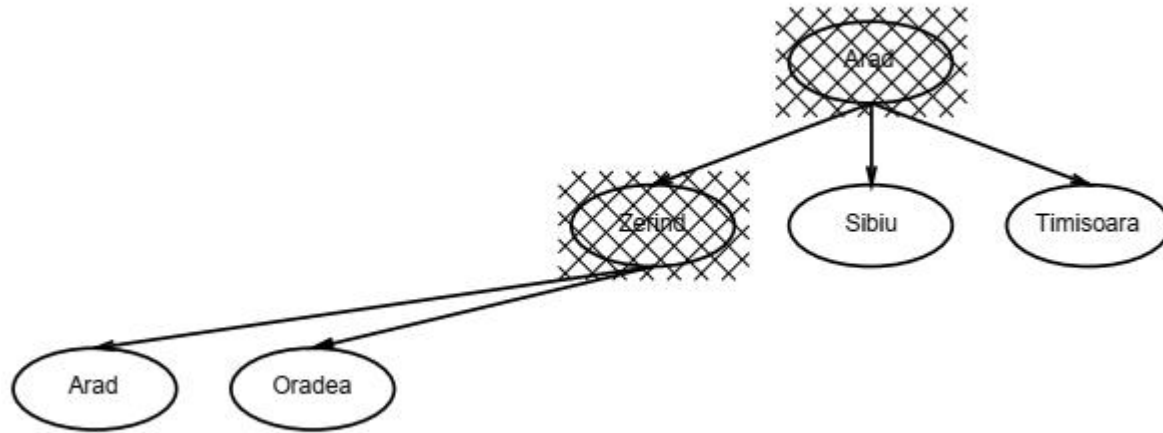


# Busca em Profundidade Iterativa $l=2$

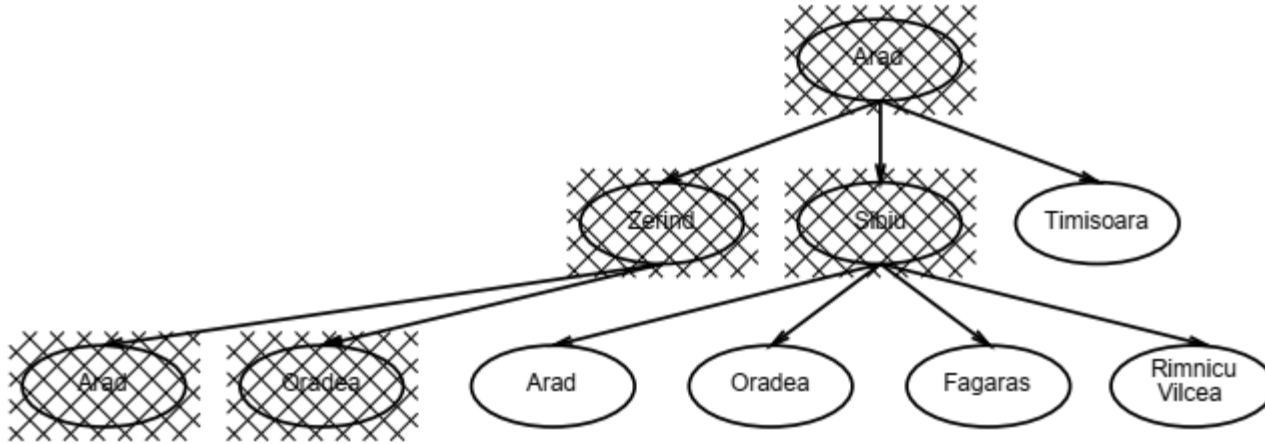




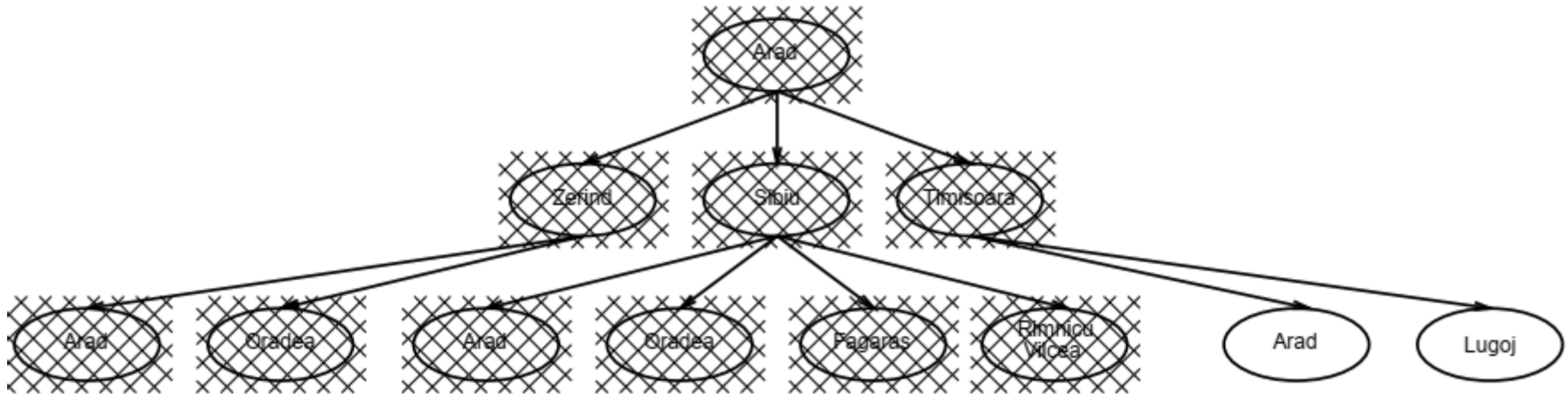
# Busca em Profundidade Iterativa $l=2$



# Busca em Profundidade Iterativa $l=2$



# Busca em Profundidade Iterativa $l=2$



# Propriedades da Busca em Profundidade Iterativa

Completa??

Tempo??

Espaço??

Otimizado??

# Propriedades da Busca em Profundidade Iterativa

Completa?? Sim.

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

Espaço??  $O(bd)$

Otimizado?? Sim, se os passos custarem =1

Pode ser modificado para explorar a árvore de custo uniforme

# Síntese

- A formulação de problemas geralmente requer a abstração de detalhes do mundo real para definir um espaço de estado que possa ser explorado de forma viável
- Variedade de estratégias de busca uniformizadas
- A busca intensiva de aprofundamento utiliza apenas espaço linear e não muito mais tempo do que outros algoritmos desinformados