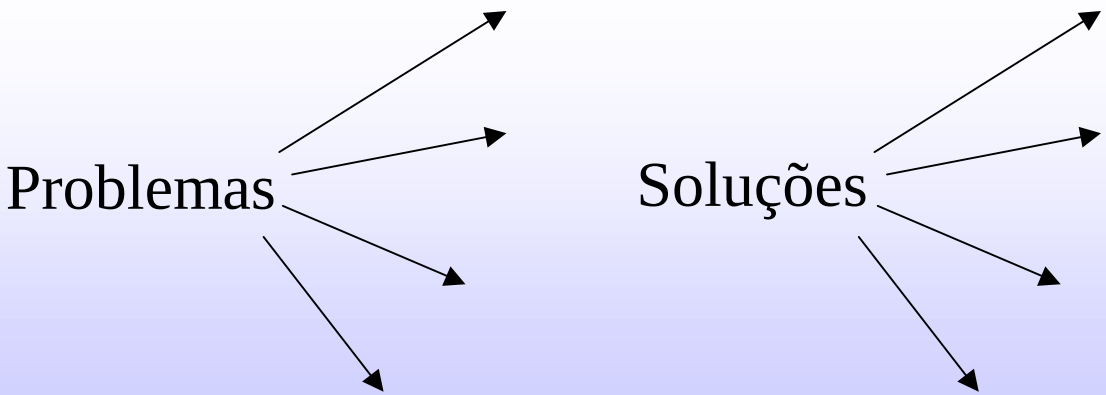
Agentes Resolvedores de Problemas

Agentes inteligentes que agem de tal modo que o *ambiente* passe por uma *seqüência de estados* que maximiza a *medida de desempenho*



Agentes Resolvedores de Problemas

- *Passos:*
  - *Formulação do OBJETIVO*
    - *nível de detalhe das ações*
  - *Formulação do PROBLEMA*
    - *decidir quais ações e estados considerar*
  - *Busca*
    - *dada uma seqüências de ações, qual a melhor?*
  - *Execução*

  - *Formulação – Busca – Execução*

Agentes Resolvedores de Problemas

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*

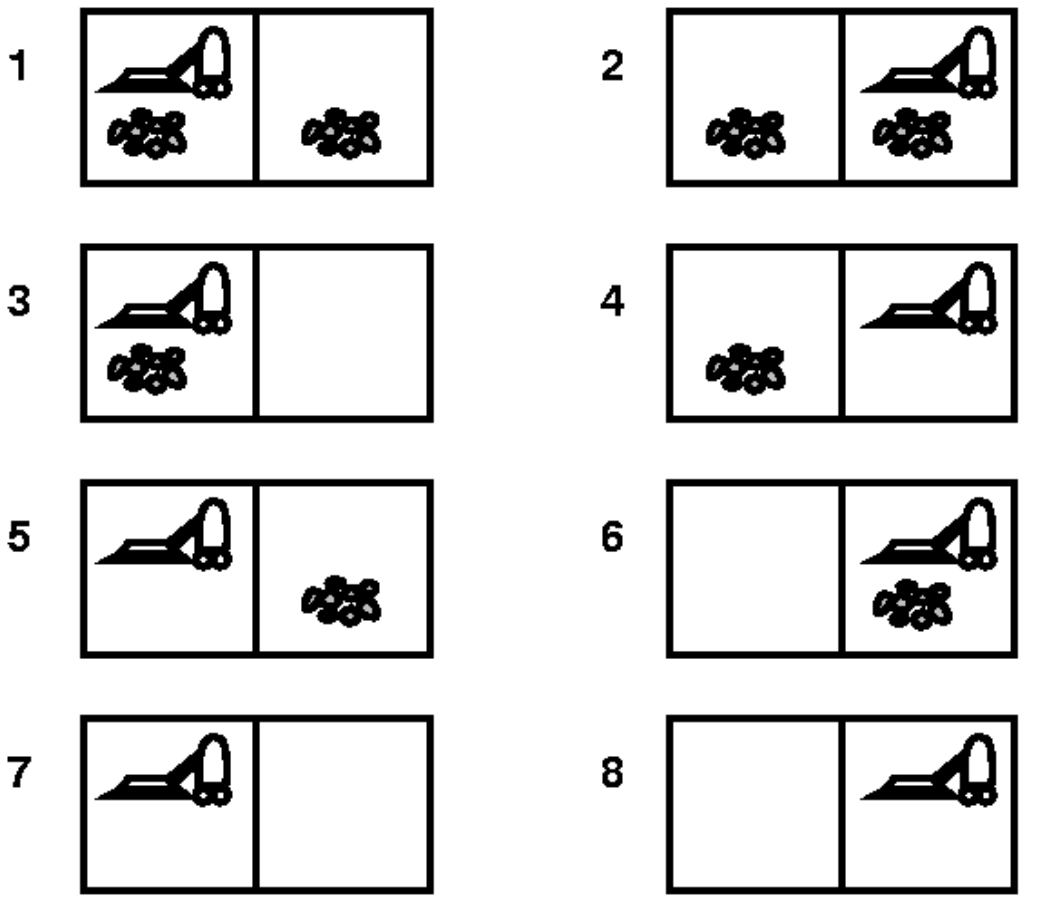
Formulando Problemas

- quantidade de conhecimento concernente às ações e aos estados
- depende de como o agente está conectado ao ambiente
- problemas
  - DE ESTADO ÚNICO
  - DE MÚLTIPLOS ESTADOS
  - DE CONTINGÊNCIA
  - DE EXPLORAÇÃO

Tipos de Problemas

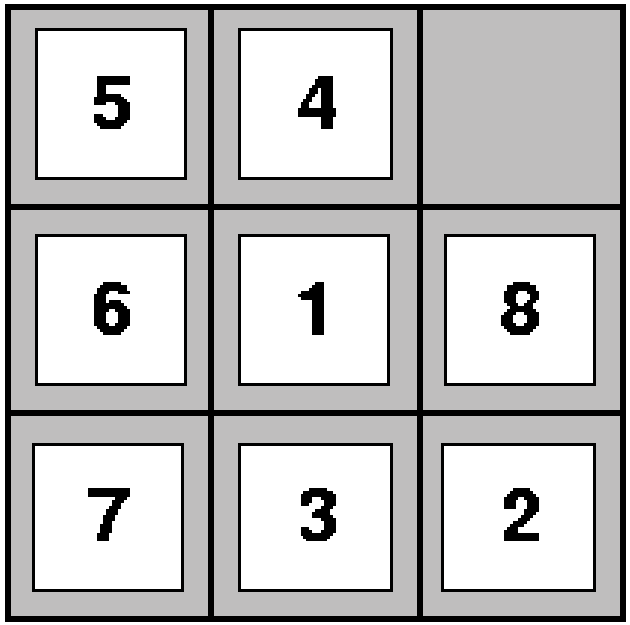
- estado único
  - o agente sabe em que estado está
  - o agente sabe o que cada uma das ações fazem
  - o agente calculará exatamente o que fazer
- múltiplos estados
  - o agente sabe o que cada uma das ações fazem
- problemas de contingência
  - deve usar os sensores enquanto age
  - a solução é uma árvore de ações ou uma política
  - às vezes pode ser viável o entrelaçamento (agir antes de buscar)
- exploração
  - espaço de estados desconhecido

Exemplo: aspirador de pó

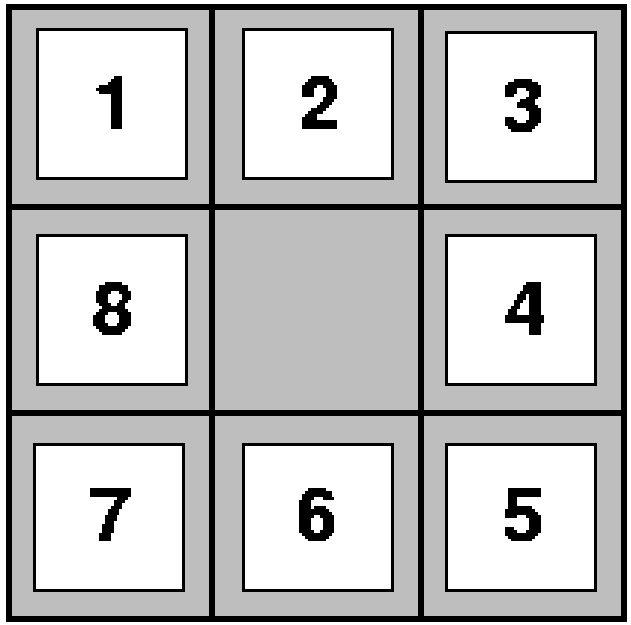


- estado único
  - início no estado 5
- múltiplos estados
  - início em
    - {1, 2, 3, 4, 5, 6, 7, 8)
  - ação Dir
    - {2, 4, 6, 8}
- contingência
  - ação Asp pode sujar um carpete limpo
  - sensoriamento
    - local
    - sujo ou não

Exemplo: 8-puzzle



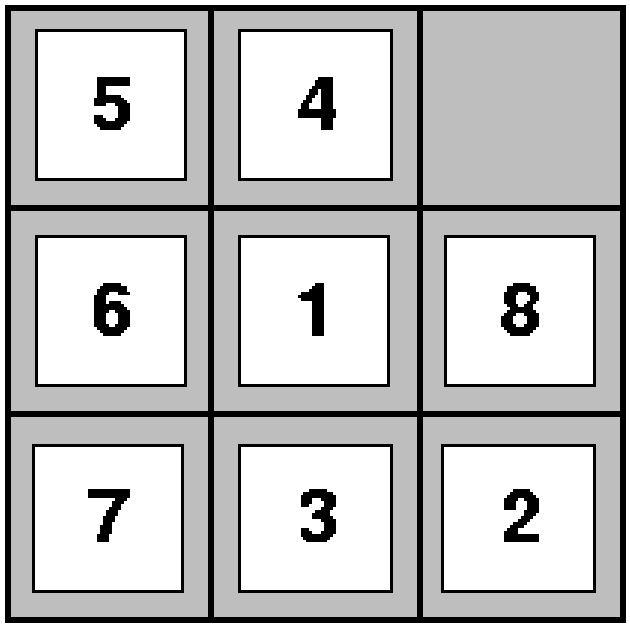
estado inicial



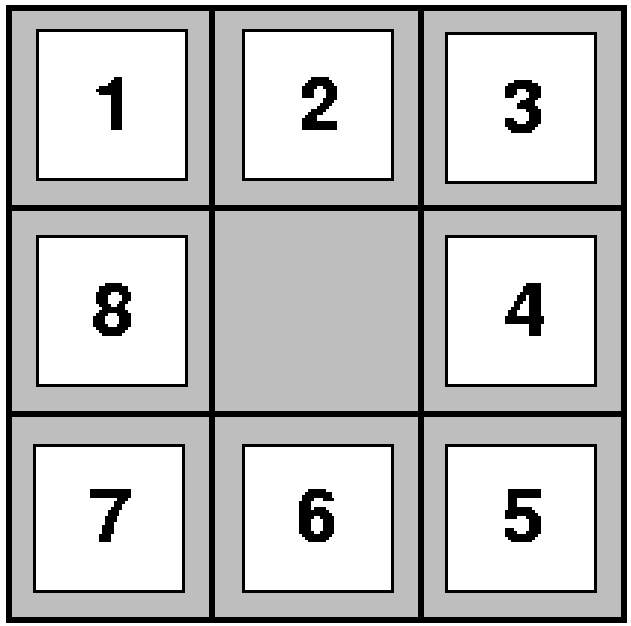
estado final

- ESTADOS
- OPERADORES
- OBJETIVOS
- CUSTO DO CAMINHO

Exemplo: 8-puzzle



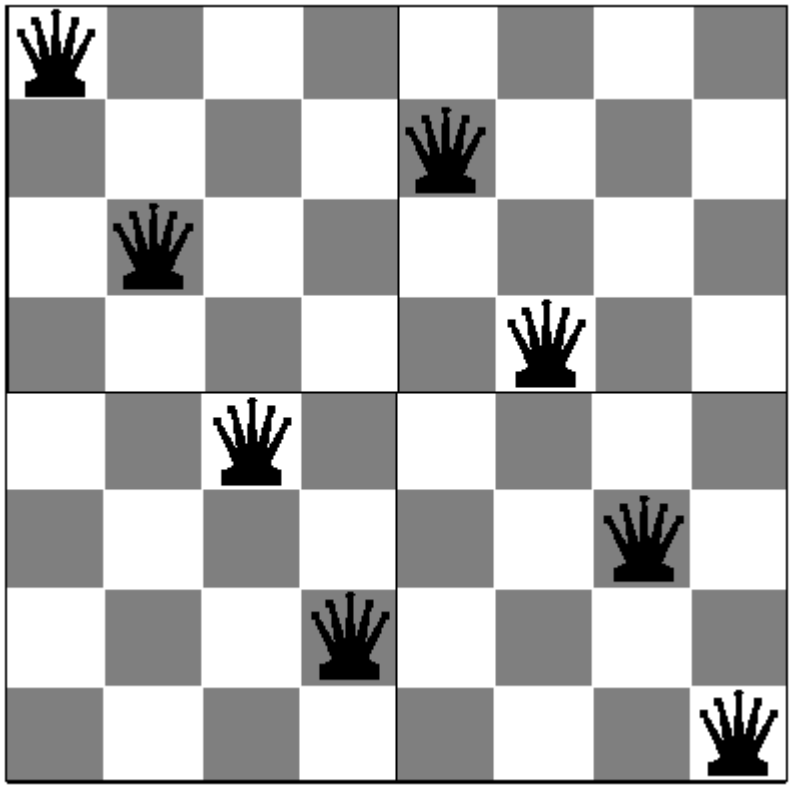
estado inicial



estado final

- ESTADOS
  - localizações dos quadrados (todas)
- OPERADORES
  - mover o espaço vazio para Dir, Esq, Cima, Baixo
- OBJETIVOS
  - chegar ao estado objetivo
- CUSTO DO CAMINHO
  - 1por movimento

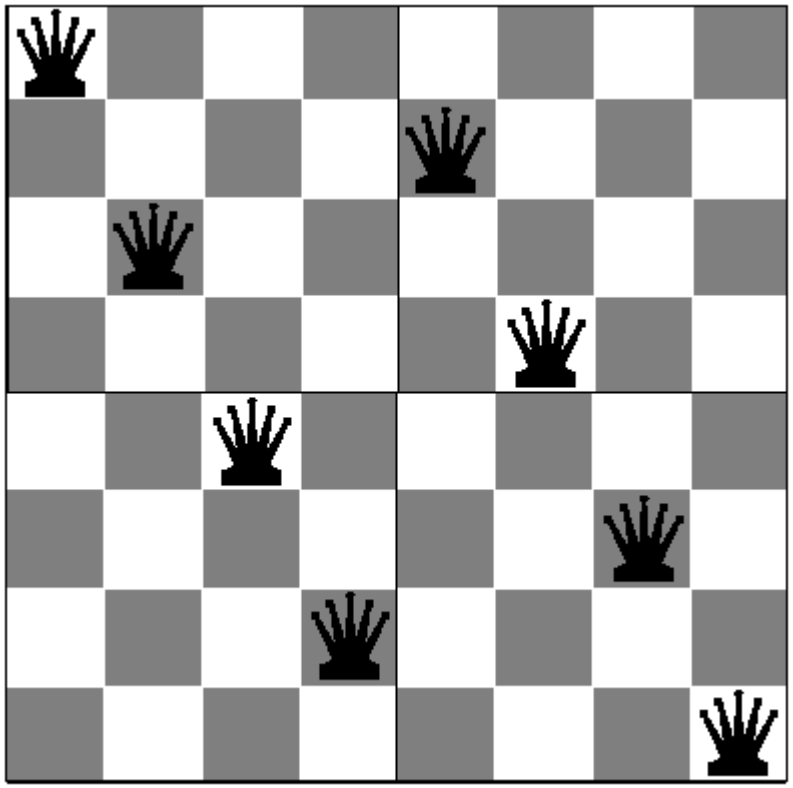
O problema das 8 rainhas



- Estados
  - Qualquer arranjo de 0 a 8 rainhas no tabuleiro
- Operadores
  - Colocar uma rainha em uma casa
- Teste de Objetivo
  - 8 rainhas estão no tabuleiro. Nenhuma está sofrendo algum ataque
- Custo do Caminho
  - Zero

64^8 possíveis seqüências a serem investigadas

O problema das 8 rainhas



2057 *possíveis seqüências a serem investigadas*

- Estados
  - Qualquer arranjo de 0 a 8 rainhas no tabuleiro, nenhuma sendo ameaçada
- Operadores
  - Colocar uma rainha em uma casa cuja coluna seja a primeira à esquerda que não esteja ameaçada por outra rainha

FORMULAÇÃO DO PROBLEMA é fundamental!!!

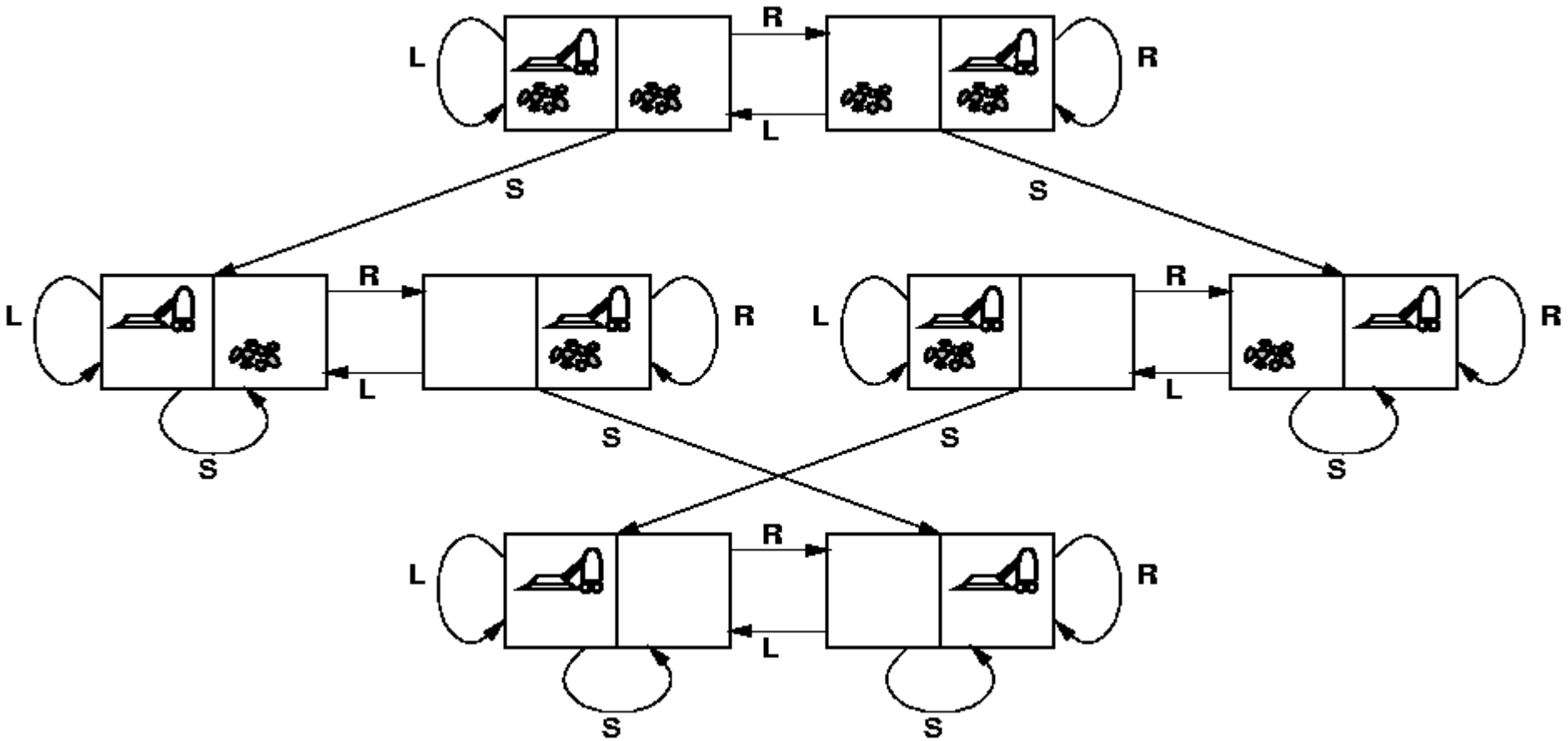
Criptoaritmética

```
  FORTY      29786
+ TEN      + 850
+ TEN      + 850
-----      -----
SIXTY      31486
```

F = 2, O = 9, R = 7, ...

- Estados
  - Algumas letras trocadas por dígitos
- Operadores
  - Trocar todas as ocorrências de uma letra por um dígito que ainda não esteja sendo usado
- Teste de Objetivo
  - Somente dígitos e a soma está correta
- Custo do Caminho
  - Zero. Todas as soluções são igualmente válidas

Espaço estado do aspirador de pó



estados: um dos 8 acima
operadores: L, R, S

objetivo: todos os estados que não têm sujeira
custo do caminho: 1 por ação

Espaço estado do aspirador

estados:

subconjuntos dos estados {1,..., 8}

operadores:

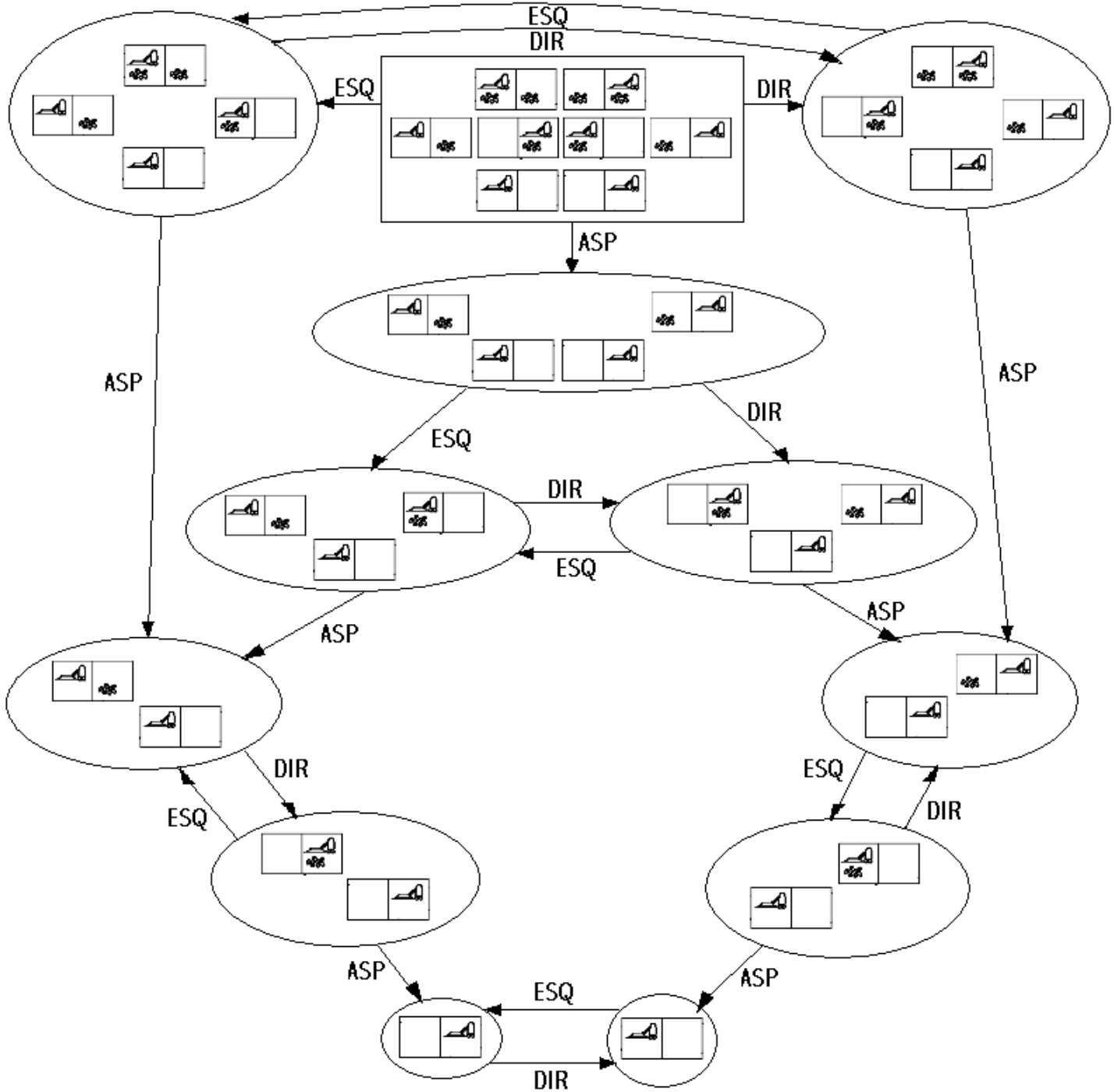
L, R, S

objetivo:

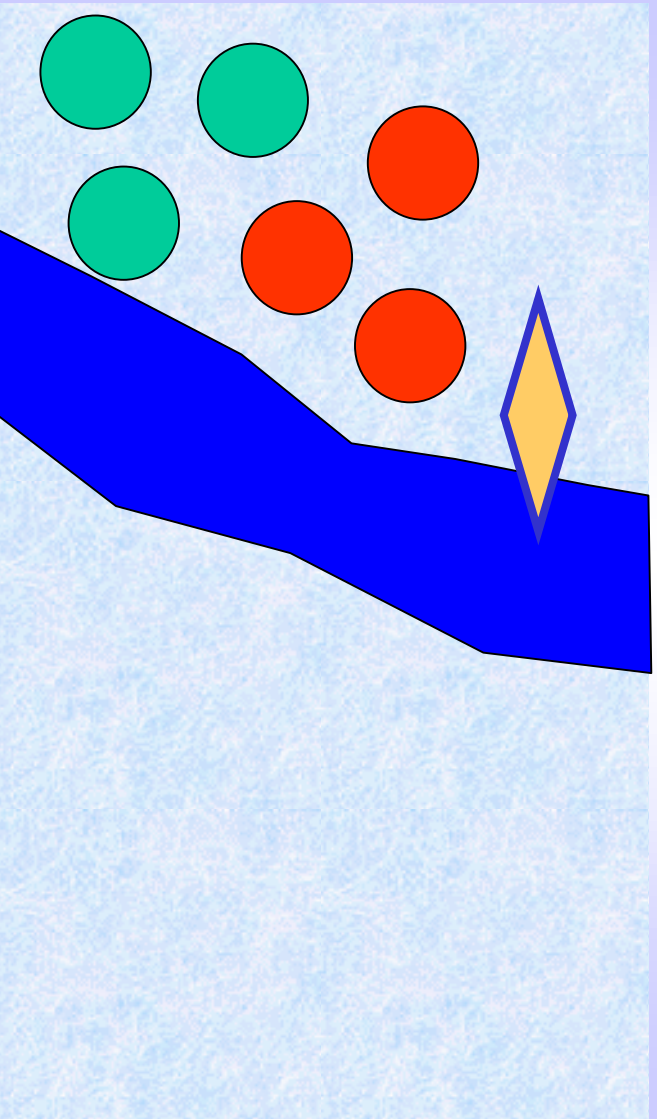
todos os estados que não têm sujeira

custo do caminho

1 por ação



Missionários e Canibais

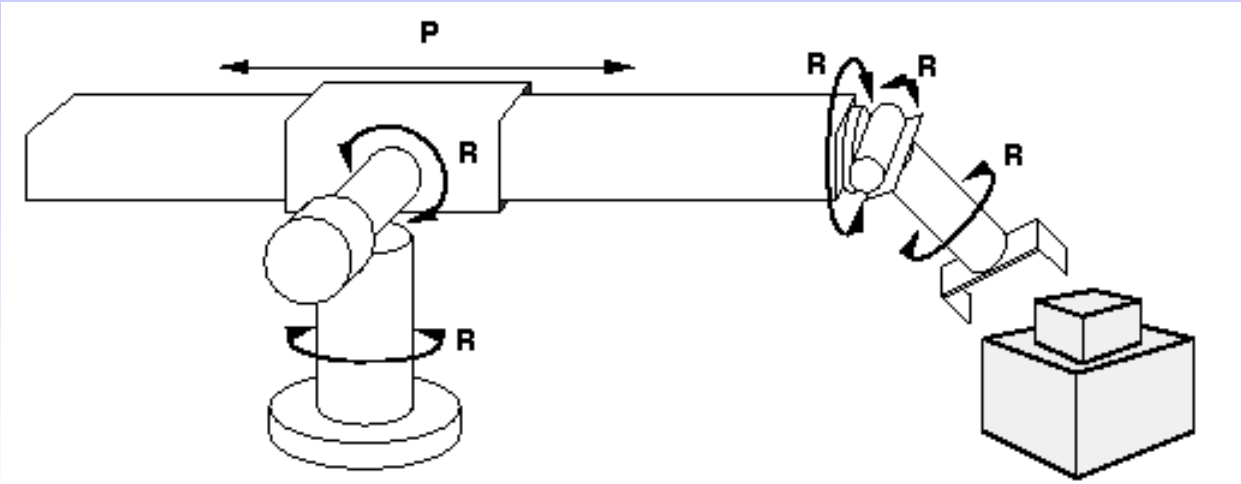


- Estados
  - uma seqüência de três números representando
    - <missionários, canibais, barco>
- Operadores
  - 1 missionário e 1 canibal,
  - 2 missionários,
  - 2 canibais,
  - 1 missionário,
  - 1 canibal
- Teste de Objetivo
  - Estado final deve ser <0, 0, 0>
- Custo do Caminho
  - número de passagens pelo rio

Problemas do mundo real

- Roteamentos
- Problemas do Caixeiro Viajante
- Layout de VLSI
- Navegação de Robôs
- Sequenciamento de Montagem

Robô na linha de montagem



- Estados
  - coordenadas de valor real para os ângulos de junção
  - partes do objeto a ser montado
- Operadores
  - movimentos contínuos das juntas dos robôs
- Teste de Objetivo
  - montagem finalizada
- Custo do Caminho
  - tempo de montagem

Algoritmos de Busca

- Idéia básica:
  - Off-line
  - Exploração simulada do espaço de estados gerando sucessores dos estados já gerados
  - EXPANDIR ESTADOS

function BUSCA_GERAL(*problema*, *estratégia*) **returns** uma solução, ou falha

inicializa a árvore de busca usando o estado inicial do *problema*

loop

if não houver candidatos para expansão

then return falha

escolha um nodo folha para expandir conforme uma *estratégia*

if o estado contiver o estado objetivo

then return a solução correspondente

else expanda o nodo e adicione os nodos resultantes à árvore de busca

end

Implementação de Algoritmos de Busca

function BUSCA_GERAL(*problema*, FILA_FN) **returns** uma solução, ou falha

nodos ←FAZ_FILA(FAZ_NODO(ESTADO_INICIAL[*problema*]))

loop

if *nodos* = vazio

then return falha

nodo ← REMOVE_DA_FRENTE(*nodos*)

if TESTE_OBJETIVO[*problema*] aplicado ao ESTADO(*nodo*) obtiver sucesso

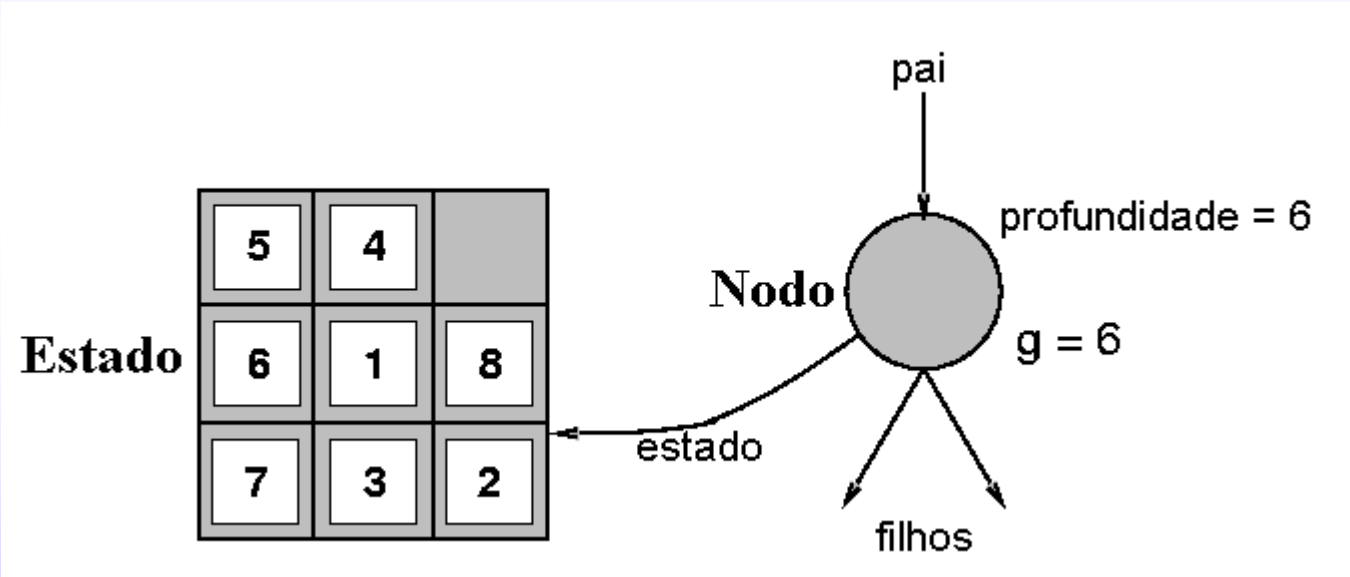
then return *nodo*

nodos ←POE_NA_FILA(*nodos*, EXPANDE(*nodo*, OPERADORES[*problema*]))

end

Estados versus Nodos

- O estado é uma representação de uma configuração física
- O nodo é uma estrutura de dados que faz parte de uma árvore de busca e abrange:
  - pai, filho, profundidade, custo do caminho g(x)
- Estados não têm pai, nem filho, nem profundidade e nem custo do caminho



- A função EXPANDE cria novos nodos, preenchendo os vários campos usando os OPERADORES ou (SUCESSOR_FN) do problema para criar os estados correspondentes

Estratégias de Busca

- Uma estratégia é definida pela escolha da ordem de expansão do nodo
- Uma estratégia é ao longo das seguintes dimensões:
  - COMPLETEZA
    - sempre encontra uma solução se ela existe?
  - COMPLEXIDADE DE TEMPO
    - número de nodos gerados/expandidos
  - COMPLEXIDADE DE ESPAÇO
    - número máximo de nodos na memória
  - OPTIMALIDADE
    - sempre encontra uma solução de custo mínimo?
- Complexidade de tempo e espaço são medidas em:
  - *b* - fator máximo de ramificação da árvore de busca
  - *d* - profundidade da solução de custo mínimo
  - *m* - profundidade máxima do espaço de estados (pode ser ∞)

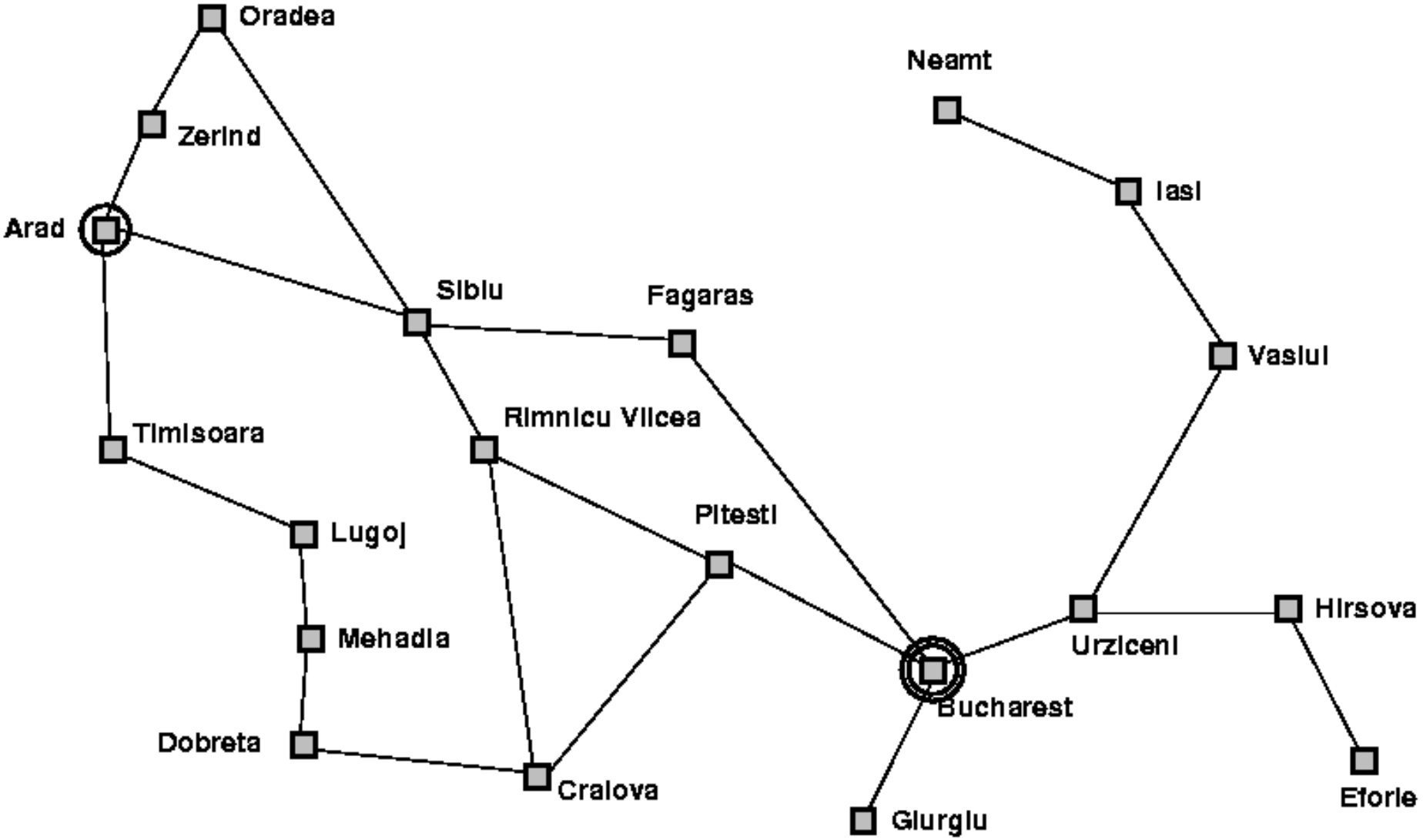
Estratégias de Buscas não Informadas

- O que não é informado?
  o número de passos ou o custo do caminho desde o estado inicial até o objetivo

- Busca Primeiro-na-Largura
  - Breadth-First

  Fila FIFO
- Busca de Custo Uniforme
- Busca Primeiro-na-Profundidade
- Busca de Profundidade Limitada
- Busca de Aprofundamento Iterativo

Problema: Roteamento



Problema: Roteamento

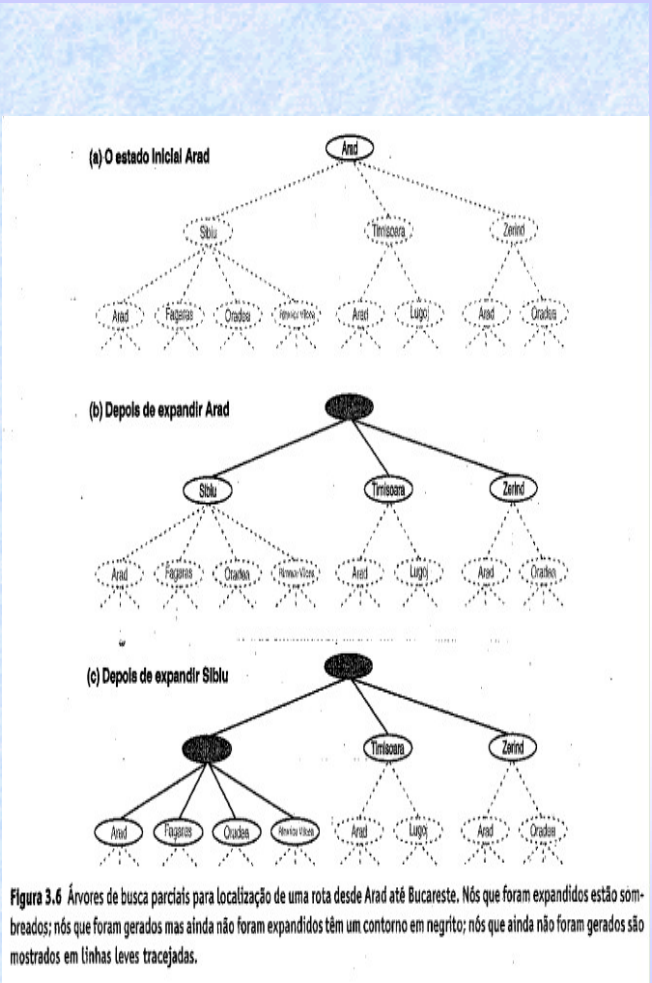
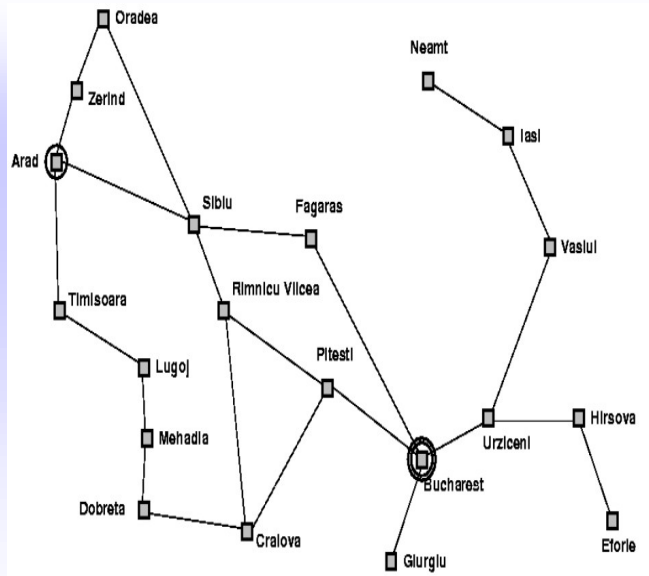


Figura 3.6 Árvores de busca parciais para localização de uma rota desde Arad até Bucareste. Nós que foram expandidos estão sombreados; nós que foram gerados mas ainda não foram expandidos têm um contorno em negrito; nós que ainda não foram gerados são mostrados em linhas leves tracejadas.



- O problema
  - Estado Inicial
    - ARAD
  - Operadores
    - Arad ← Zerind
    - Arad ← Sibiu
  - Teste de Objetivo
    - Explícito = Bucarest
  - Custo do Caminho
    - soma das distâncias
- Solução
  - seqüência de operadores saindo do estado inicial e chegando ao estado objetivo

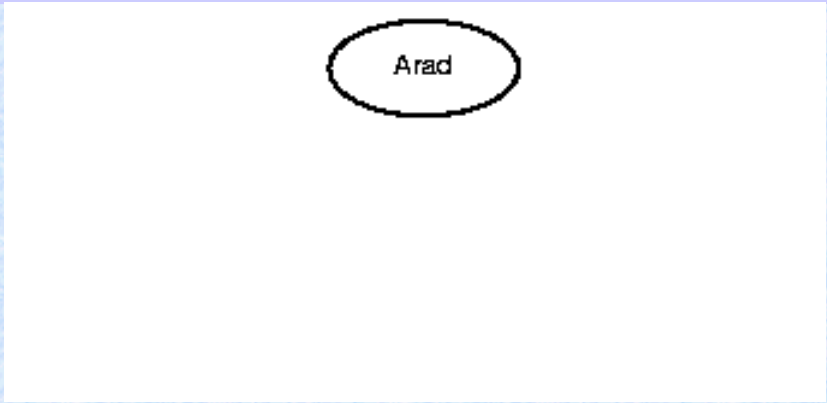
Busca Primeiro-na-Largura (Breadth-First)

BUSCA EM EXTENSÃO

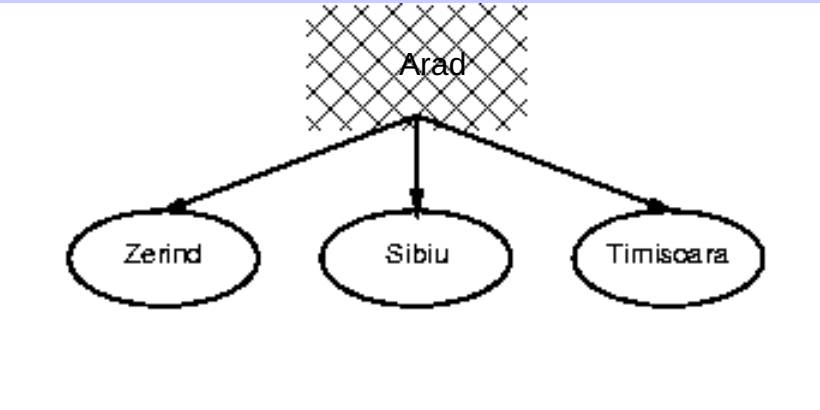
- Expande o nodo menos profundo ainda não expandido

- Implementação
  - POE_NA_FILA
    - põe sucessores no final da fila

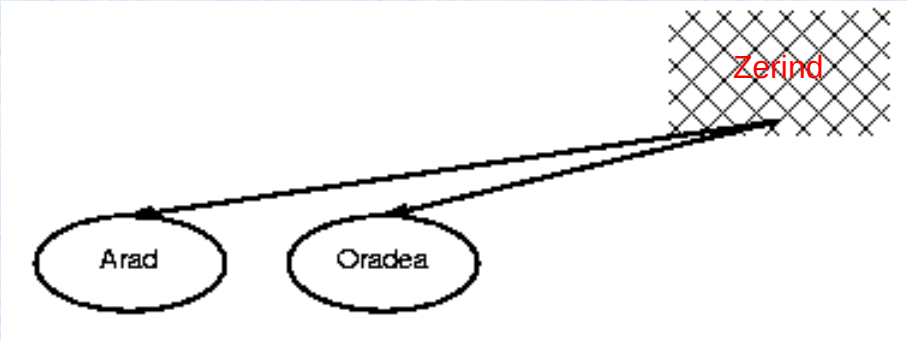
Busca Primeiro-na-Largura



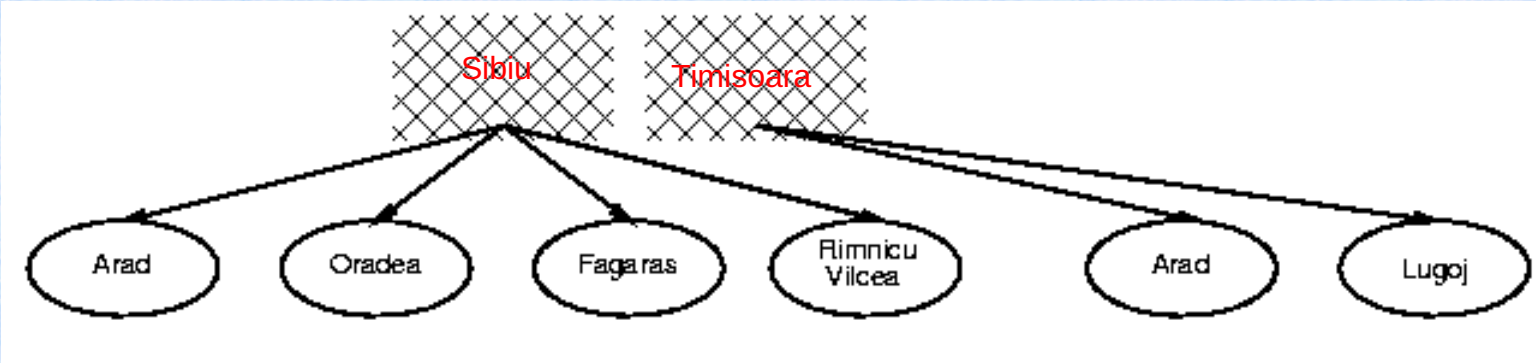
Busca Primeiro-na-Largura



Busca Primeiro-na-Largura



Busca Primeiro-na-Largura



Propriedades da Busca Primeiro-na-Largura

- Completo?
- Tempo?
- Espaço?
- Ótima?

Propriedades da Busca Primeiro-na-Largura

Profundidade	Nodos	Tempo	Memória
0	1	1 milisegundos	100 Bytes
2	111	0.1 segundo	11 KB
4	11,111	11 segundos	1 MB
6	10^6	18 minutos	111 MB
8	10^8	31 horas	11 GB
10	10^{10}	128 dias	1 TB
12	10^{12}	35 anos	111 TB
14	10^{14}	3500 anos	11,111 TB

Propriedades da Busca Primeiro-na-Largura

- Completo?
  - SIM. b é finito
- Tempo?
  - $1 + b + b^2 + b^3 + \ldots + b^d = \mathbf{O}(b^d)$ ou seja, d-exponencial
- Espaço?
  - $\mathbf{O}(b^d)$ – mantém todos os nodos na memória
- Ótima?
  - SIM, se o custo for 1 por passo. Em geral é não-ótima.

    Por exemplo, se as distâncias de um nó a outro forem 1. Sempre expande o nó mais raso ainda não expandido.
- ESPAÇO é o grande problema.
  - Pode facilmente gerar nodos em 1 MB/segundo
  - em 24 horas = 86 GBytes

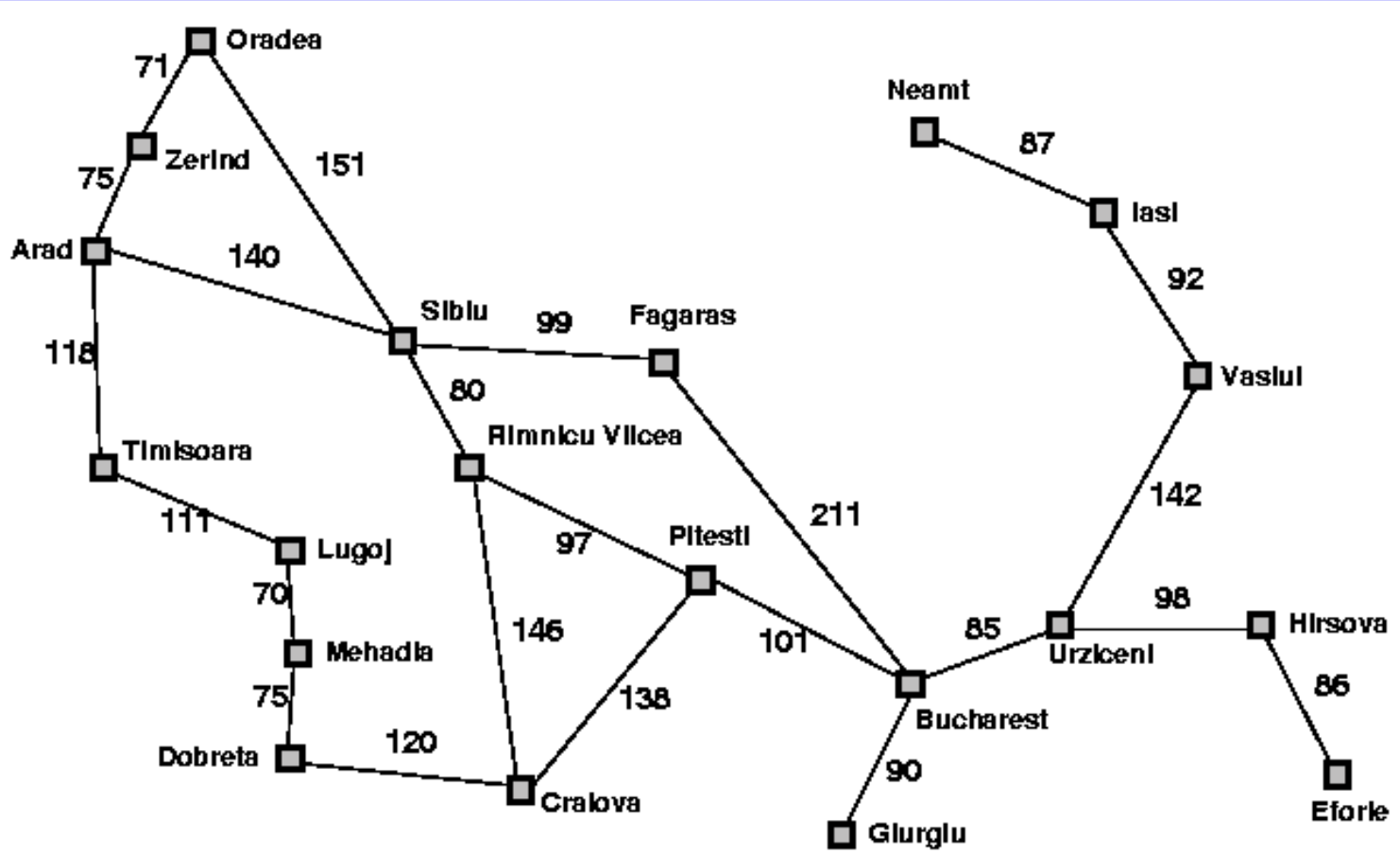
Busca de Custo Uniforme

Caminho mínimo ou Djkstra

- Expande o nodo com menor custo ainda não expandido

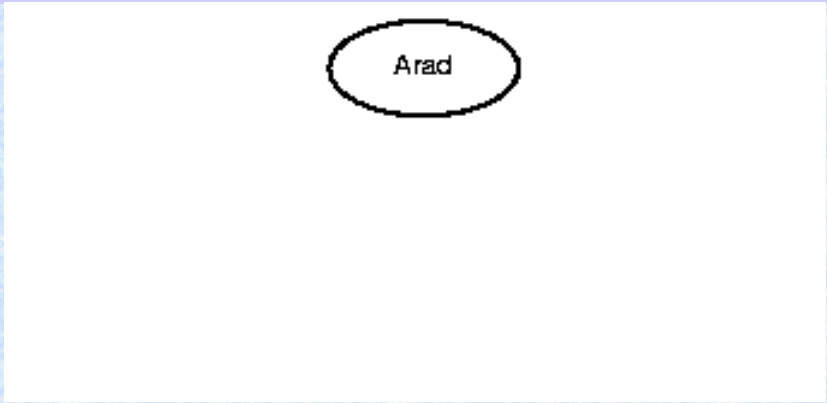
- Implementação
  - POE_NA_FILA
    - insere na ordem crescente de custo do caminho

Problema: Roteamento (com custo em Km)

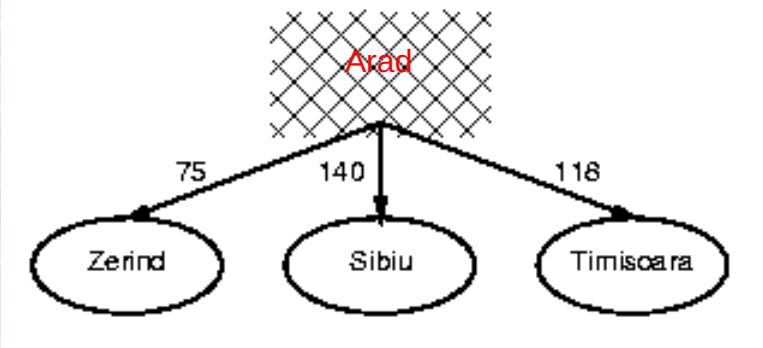


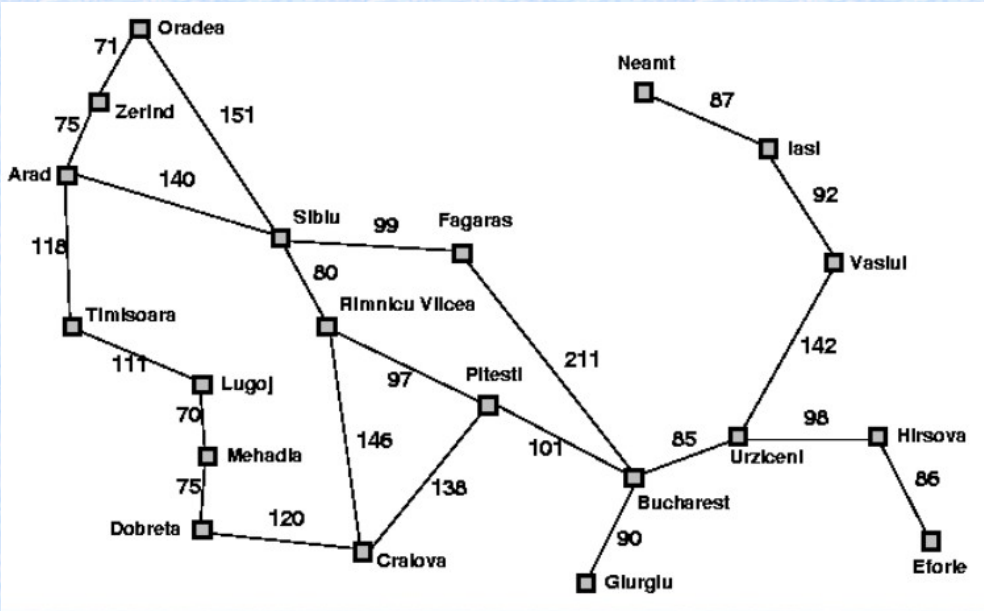
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Busca de Custo Uniforme

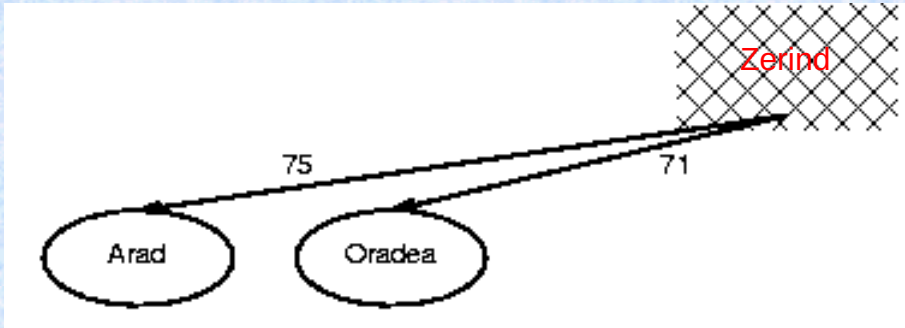


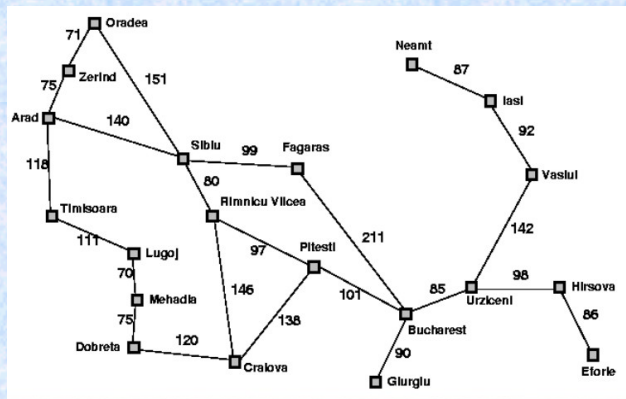
Busca de Custo Uniforme



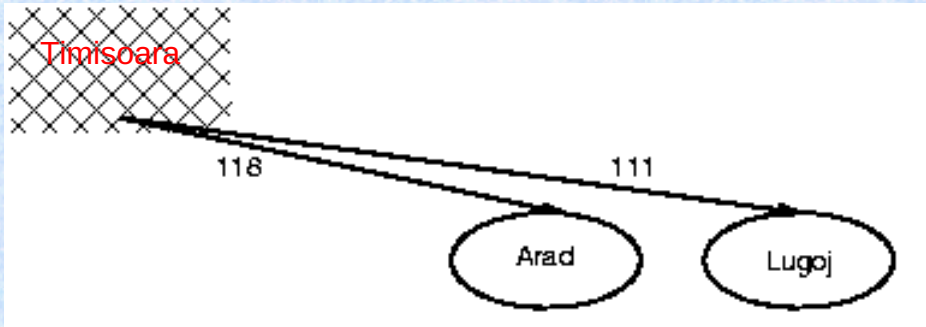


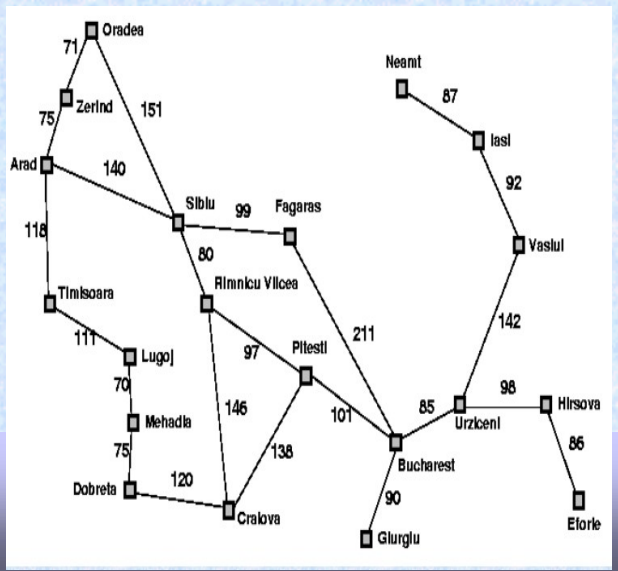
Busca de Custo Uniforme





Busca de Custo Uniforme





Propriedades da Busca de Custo Uniforme

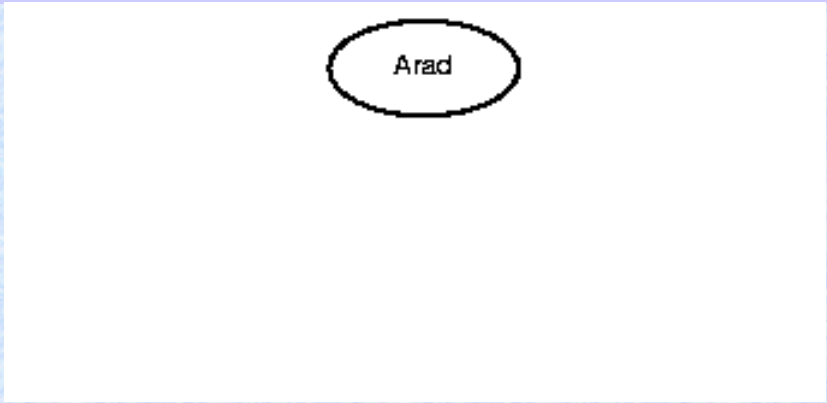
- Completo?
  - SIM. se o custo do passo $\geq \varepsilon$
- Tempo?
  - nro de nodos com custo g $\leq$ solução de custo ótimo
- Espaço?
  - nro de nodos com custo g $\leq$ solução de custo ótimo
- Ótima?
  - SIM

Busca Primeiro-na-Profundidade

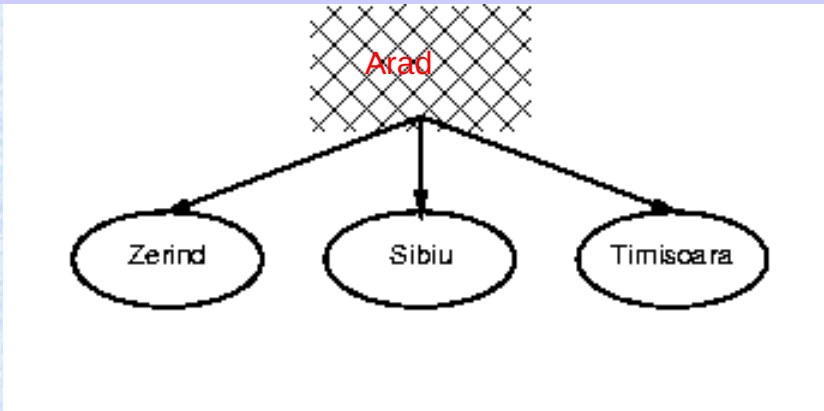
- Expande o nodo mais profundo ainda não expandido

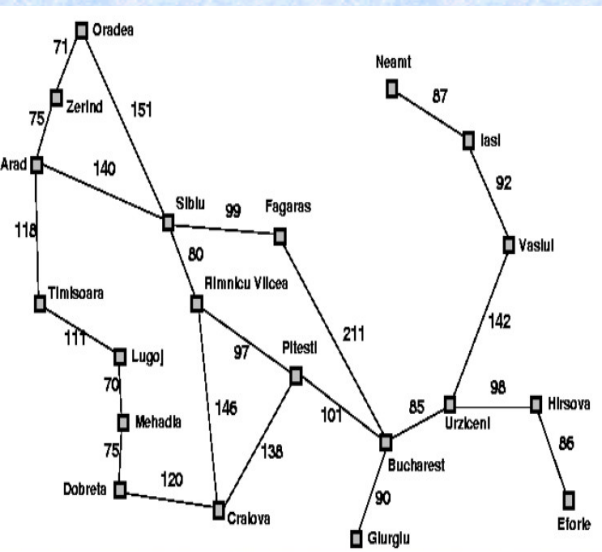
- Implementação
  - POE_NA_FILA
    - insere os sucessores no início da fila

Busca de Primeiro-na-Profundidade

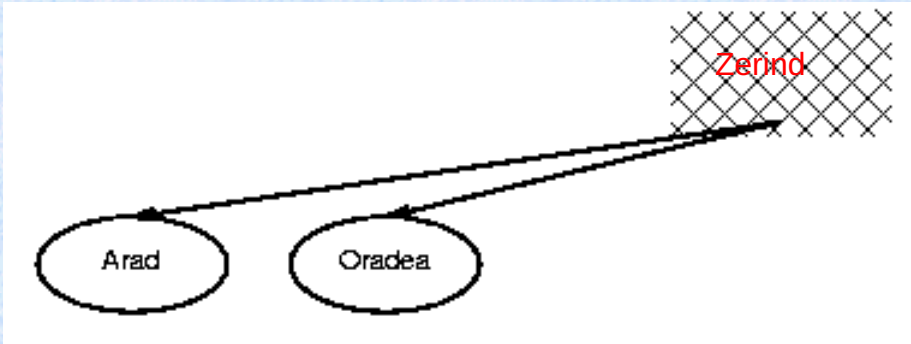


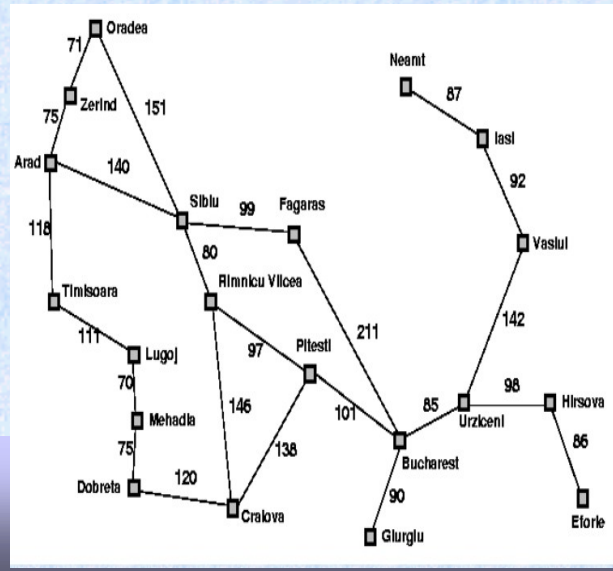
Busca Primeiro-na-Profundidade



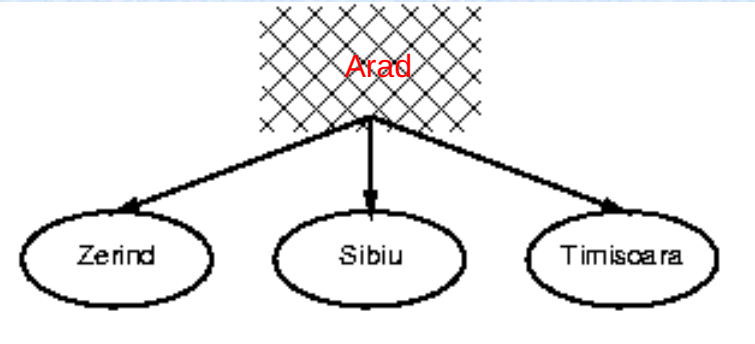


Busca Primeiro-na-Profundidade





Busca Primeiro-na-Profundidade



- Busca Primeiro-na-Profundidade pode realizar infinitas excursões cíclicas;
- É necessário um espaço de busca finito e não-cíclico (ou uma verificação de estados repetidos)

Propriedades da Busca Primeiro-na-Profundidade

- Completo?
  - NÃO. Falha nos espaços de estados de profundiade infinita, em espaços com loops.
  - Modifica-se para evitar estados repetidos ao longo do caminho
  - Completo para espaços de estados finitos.
- Tempo?
  - $O(b^m)$ – muito ruim se *m* for muito maior que *d*
  - Se as soluções forem densas, pode ser muito melhor do que a busca primeiro-na-largura.
- Espaço?
  - O(bm) – o espaço de estados é linear
- Ótima?
  - NÃO

Busca de Profundidade Limitada

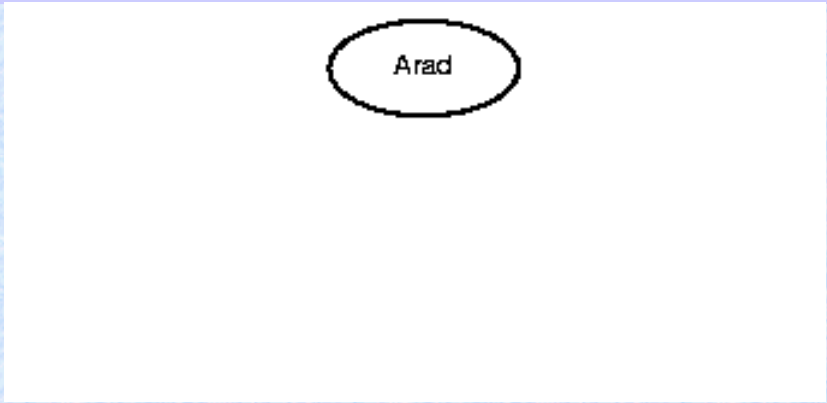
- Igual à busca primeiro-na-profundidade só que há um limite *l* para expandir

- Implementação
  - POE_NA_FILA
    - insere na ordem crescente de custo do caminho, só que nodos com profundidade maior que *l* não têm sucessores

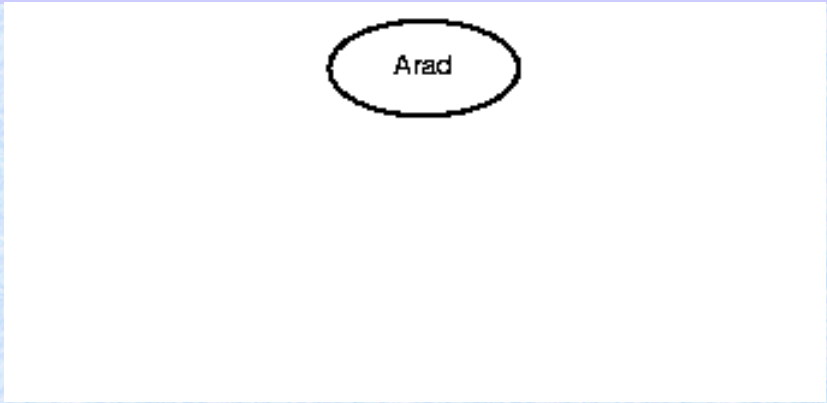
Busca de Aprofundamento Iterativo

function BUSCA_PROFUND_ITER(*problema*) **returns** uma sequência-solução
inputs: *problema*, um problema
for *profundidade* ← 0 **to** ∞
    resultado ← BUSCA_PROFUNDIDADE_LIMITADA(*problema*, *profundidade*)
    if *resultado* ≠ limiar
    then return *resultado*
end

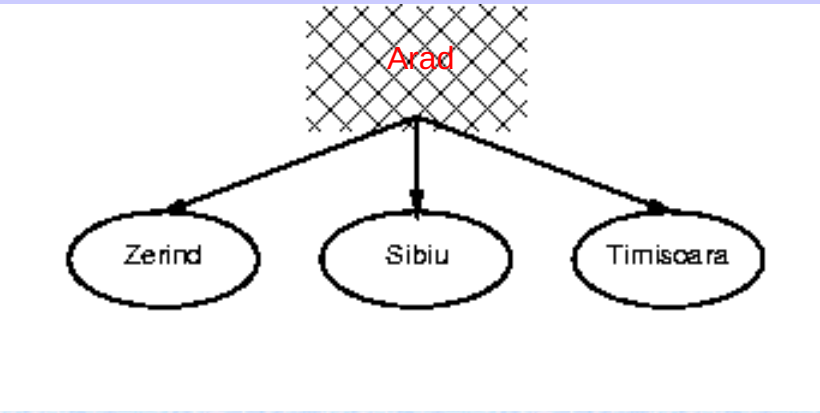
Busca de Aprofundamento Iterativo (l = 0)



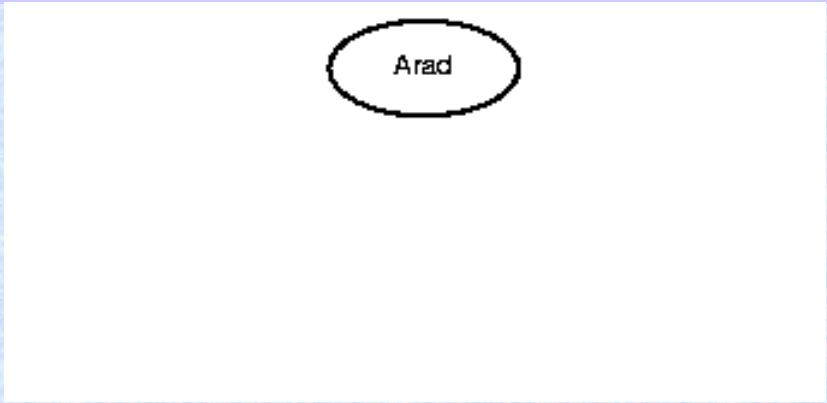
Busca de Aprofundamento Iterativo (l = 1)



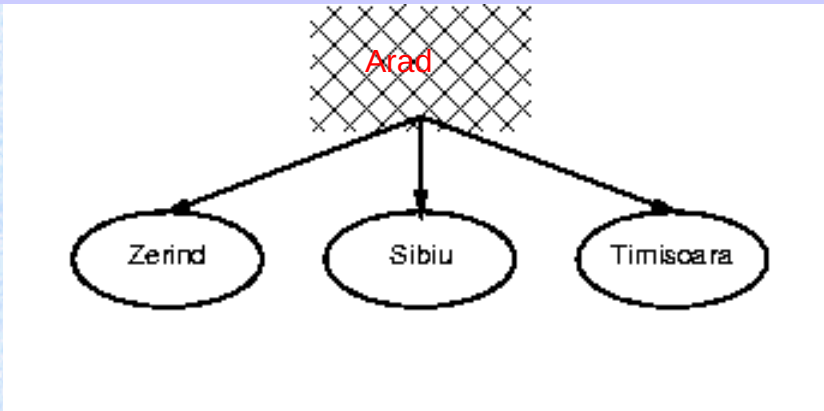
Busca de Aprofundamento Iterativo (l = 1)

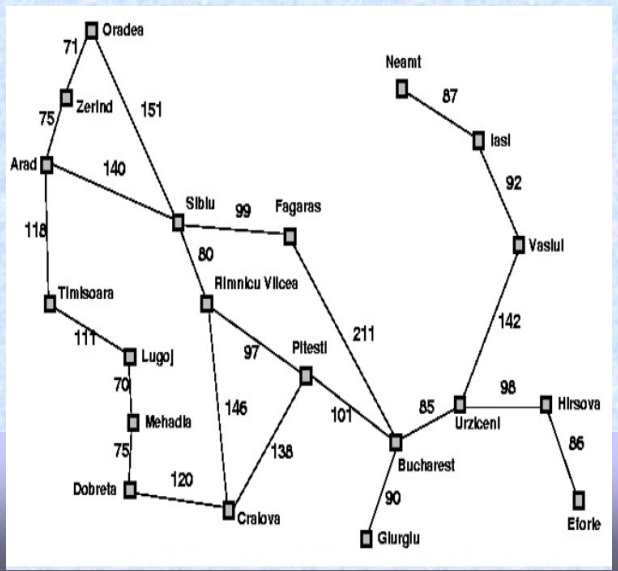


Busca de Aprofundamento Iterativo (l = 2)

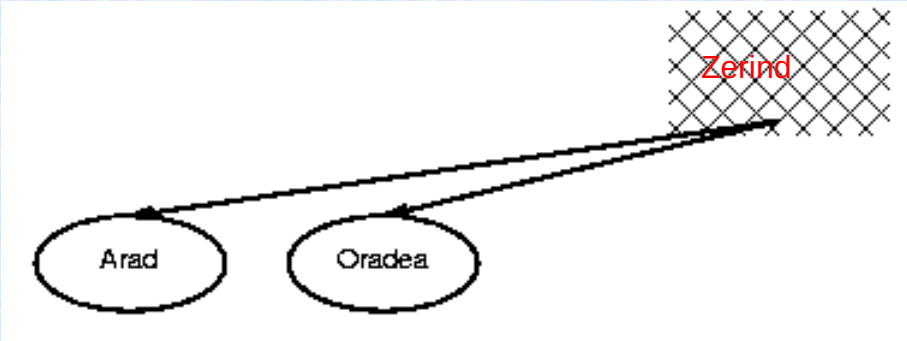


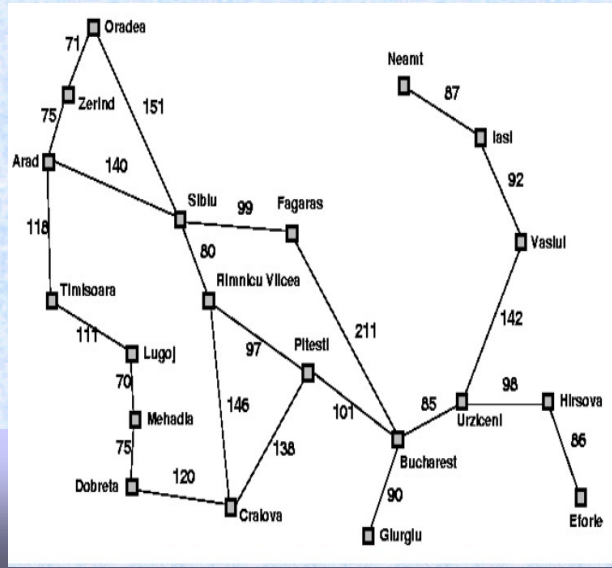
Busca de Aprofundamento Iterativo (l = 2)



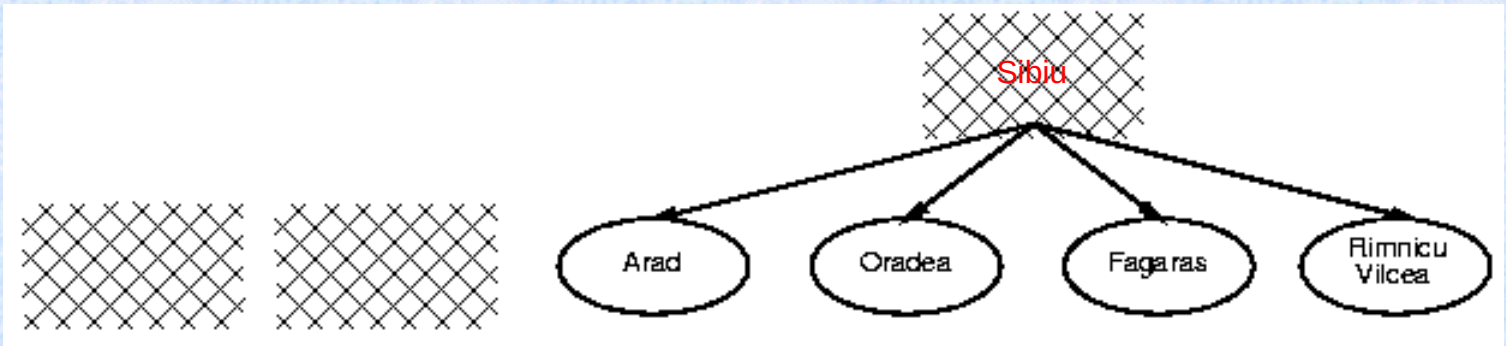


Busca de Aprofundamento Iterativo (l = 2)

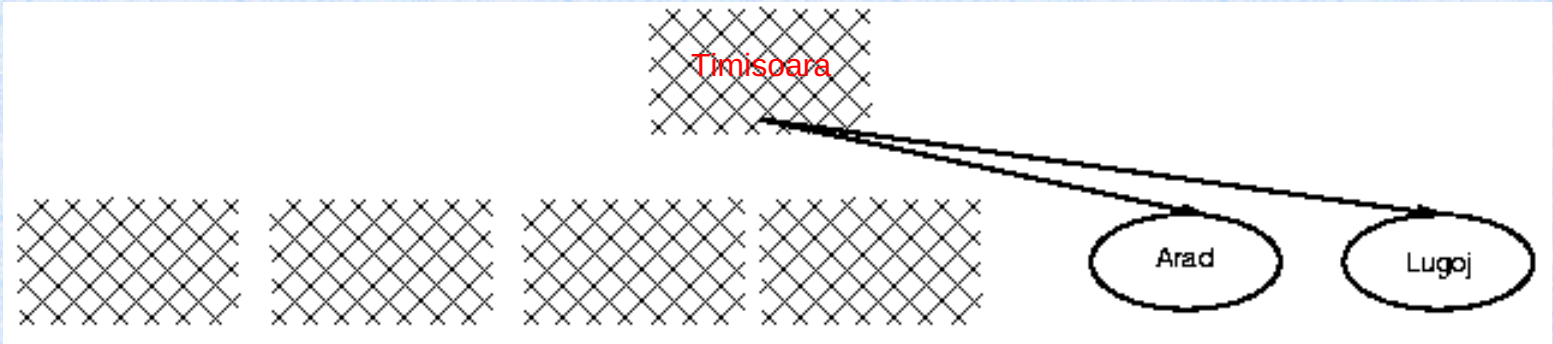


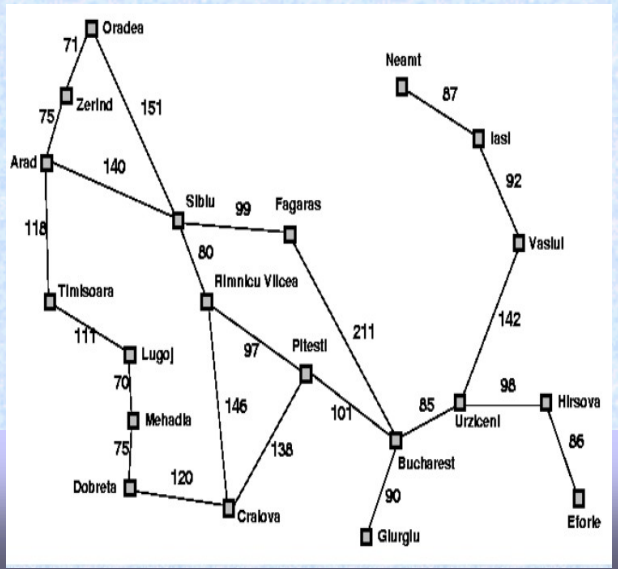


Busca de Aprofundamento Iterativo (*l* = 2)



Busca de Aprofundamento Iterativo (l = 2)





Propriedades da Busca por Aprofundamento Iterativo

- Completo?
  - SIM
- Tempo?
  - $(d+1)b^0 + db^1 + (d-1)b^2 + (d-2)b^3 + \ldots + b^n = \mathbf{O}\,(b^d)$
  - para $b = 10$ e $d = 5$
    - busca profundidade limitada em d
      - $1 + b + b^2 + \ldots + b^{d-2} + b^{d-1} + bd =$
      - $1 + 10 + 100 + 1000 + 10000 + 100000 = 111{,}111$
    - aprofundamento iterativo
      - $6 + 50 + 400 + 3000 + 20000 + 100000 = 123{,}456$
- Espaço?
  - $O(bd)$
- Ótima?
  - SIM, se o custo do passo for 1
  - Pode ser alterado para explorar árvores de custo uniforme

Comparação entre os métodos

Critério	1º-na-largura	Custo Uniforme	1º-na-profundidade	Profundidade Limitada	Aprofundamento Iterativo	Bidirecional
Tempo	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Espaço	b^d	b^d	bm	bl	bd	$b^{d/2}$
Ótimo?	SIM	SIM	NÃO	NÃO	SIM	SIM
Completo?	SIM	SIM	NÃO	SIM, se $l \geq d$	SIM	SIM

Comentários

- A formulação do problema requer, normalmente, uma abstração do mundo real, retirando detalhes para definir um espaço de estados que possa confiavelmente ser explorado
- Variedade de estratégias de busca não informadas
- O aprofundamento interativo usa somente espaço linear e não muito mais tempo que outros algoritmos não informados