Agentes baseados em metas

Capítulo 3
Parte I

Leliane Nunes de Barros
leliane@ime.usp.br

Agente reativo simples

- também chamado de *agente situado* ou *agente estímulo-resposta*
- reage a estímulos do ambiente
- não possue estado interno
- não é capaz de planejar com antecedência
- o projetista escolhe e codifica (*hard-wire)* a meta dentro do agente (em oposição a construir um agente com capacidade de tomada de decisões)

Agente reativo simples

- não funciona bem em ambientes para os quais o mapeamento entre percepções ações é muito grande para ser armazenado ou levaria muito tempo para aprender
- não sabe o que suas ações podem realizar (quando elas podem ser executadas e quais são os seus efeitos): essa informação está implícita nas regras que mapeiam percepções em ações.

Agente que resolve problemas

- Um tipo de agente baseado em metas
- Decide o que fazer (possui uma especificação de sua própria meta)
- Conhece os efeitos de suas ações e em que situações elas podem ser executadas
- Usando algoritmos de busca de propósito geral, o agente encontra uma sequência de ações que o leva a um estado do mundo desejado (estado meta)
- Como definir problemas para o agente que resolve problemas?

Agente que resolve problemas

- *Meta:* conjunto de estados do mundo onde a meta do agente é satisfeita
- *Ações:* causam transições de um estado para outro

Questões que o agente deve responder:
- Qual é o estado meta que devo atingir?
- Que tipo de ações eu posso usar?
- Que mudanças no mundo eu provoco?
- Quais ações me levam ao estado meta?

Agente que resolve problemas

FORMULAÇÃO
BUSCA
EXECUÇÃO

- *Formulação da meta*: determina o(s) estado(s) meta
- *Formulação do problema:* escolha de um conjunto relevante de estados e ações
- *Busca:* "imaginar" ou "deliberar" as sequências de ações que atinjam um estado meta
- *Solução*: conjunto de passos que atinge a meta
- *Execução*: realização dos passos/ações da solução

Agente que resolve problemas

FORMULAÇÃO
BUSCA
EXECUÇÃO

função Agente-Resolve-Problemas (*p*) **devolve** *ação*

***entradas*:** *p,* uma percepção

estático: *seq*, uma sequência de ações, inicialmente vazia
estado, descrição do estado atual
g, uma meta, inicialmente nula
problema, uma formulação de problema

estado ← ATUALIZA-ESTADO (*estado, p*)

se *seq* vazia ***então faça***

g ← FORMULA-META (*estado)*
problema ← FORMULA-PROBLEMA (*estado, g*)
seq ← BUSCA (*problema)*

ação ← PRIMEIRA (*seq*)
seq ← RESTO (*seq)*
devolve *ação*

Agente que resolve problemas fazendo uma busca no espaço de estados: suposições sobre o ambiente

- estático
- totalmente observável
- discreto
- determinístico

Formulação de Problema

- Envolve o conhecimento que um agente deve usar para resolver o problema:
  - estado inicial (que o próprio agente sabe estar)
  - descrição das ações do agente, dada por:
    - função sucessor de estado (Sucessor-FN(s)): devolve um conjunto de pares ordenados (ação, estado-sucessor) ou
    - conjunto de operadores: que podem ser aplicados aos estados, gerando os estados sucessores
  - teste de meta: checa por um conjunto explícito de estados meta ou por uma propriedade do estado (ex.: cheque-mate do xadrez)
  - função custo de caminho (p. ex: a soma dos custos das ações individuais em um caminho)

Espaços de problemas

- **Espaço de estados** do problema: conjunto de todos os estados alcançáveis a partir do estado inicial (uma medida do tamanho de complexidade)
- **Caminho** no espaço de estados é uma seqüência de estados conectados por uma seqüência de ações
- Uma **solução** para um problema é um caminho do estado inicial a um dos estados meta
- **Solução ótima:** solução de menor custo
- **Árvore de busca:** árvore construída, a partir do espaço de busca, para um problema específico; os estados do mundo representam os nós da árvore
  - Estado inicial: raiz da árvore
  - Estado Meta: folha(s) da árvore

Exemplo: Romênia

Férias na Romênia. No momento em Arad. O vôo sai no dia seguinte para Bucharest.

Formular meta:

- chegar em Bucharest

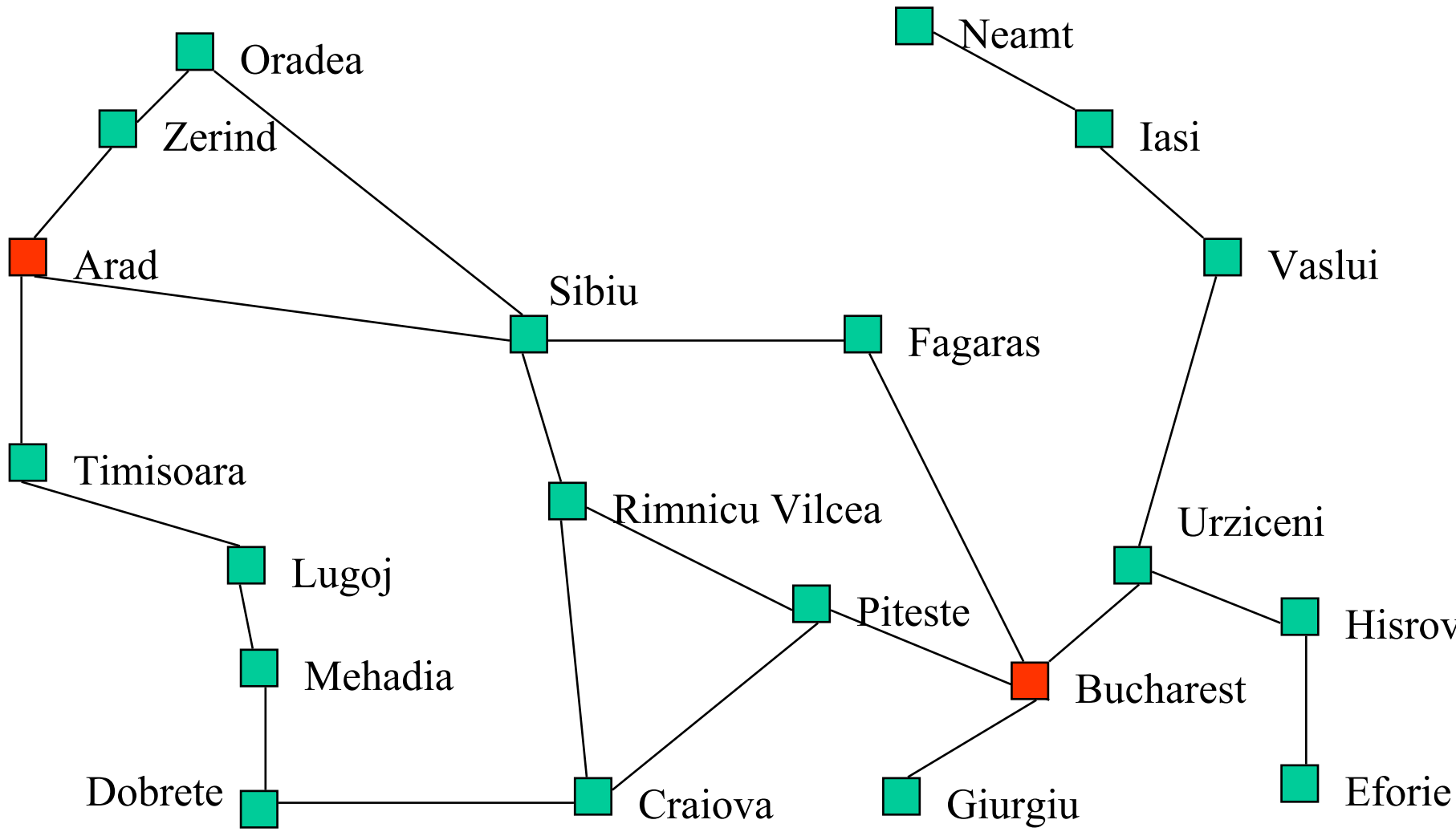
Formular problema:

- *estado inicial*: Arad
- *ações*: dirigir entre cidades
- *função custo*: kilometragem total ou tempo de viajem
- *teste de meta:* estou em Bucharest?

Encontrar solução:

- sequência de cidades, por exemplo: Arad, Sibiu, Fagaras Bucharest

Mapa de estradas

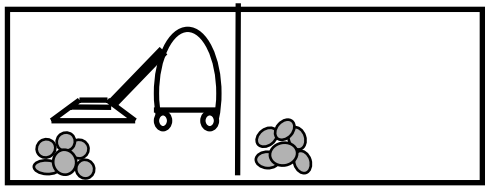


O mundo do aspirador

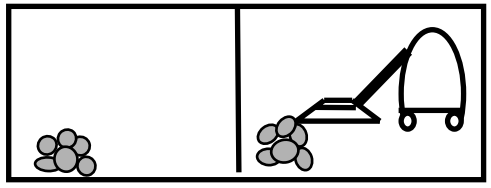
- **Estado do mundo**: localização do agente e localizações com sujeira
- **Conjunto de Estados**: o agente está em um dos 2 cômodos e cada um pode estar sujo ou limpo: 8 estados do mundo possíveis
- **Estado Inicial**: qualquer um dos 8 estados
- **Função sucessor**: gera os estados legais resultantes da execução das 3 ações: *Esquerda, Direita e Aspira*
- **Teste de meta**: teste se os 2 cômodos estão limpos (sensores: sensor de posição; sensor local de sujeira)
- **Custo**: cada passo tem custo 1, o custo do caminho é o número de passos do caminho

Estados do mundo do aspirador

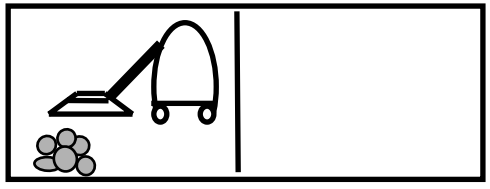
1



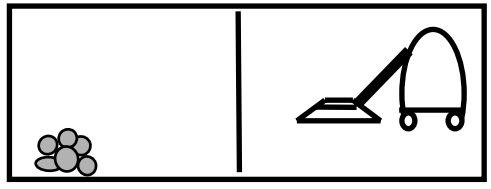
2



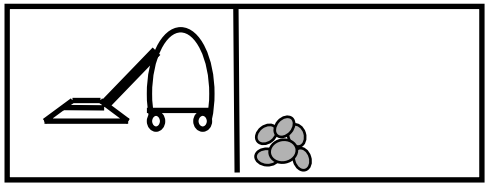
3



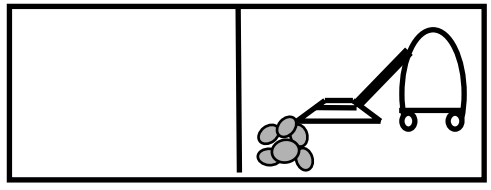
4



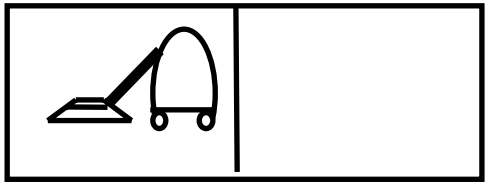
5



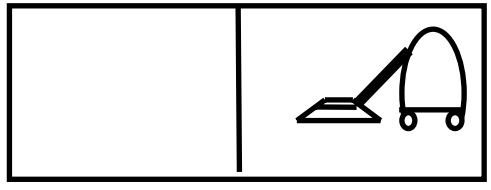
6



7 **estado meta**

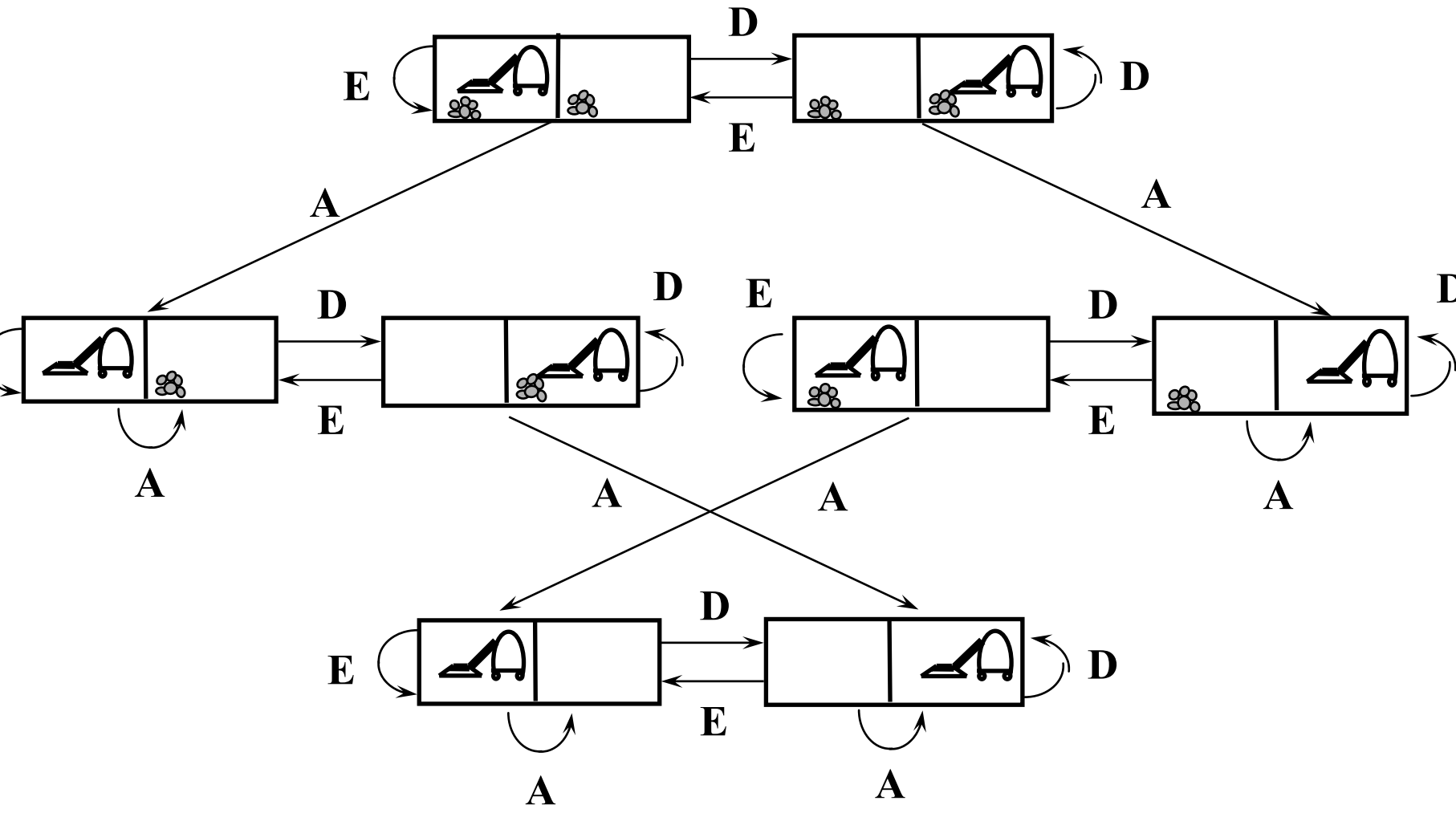


8



estado meta

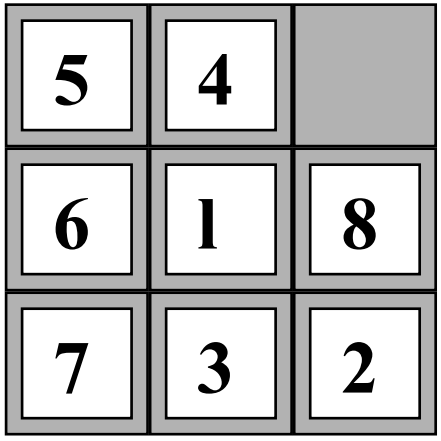
Espaço de estados do mundo do aspirador



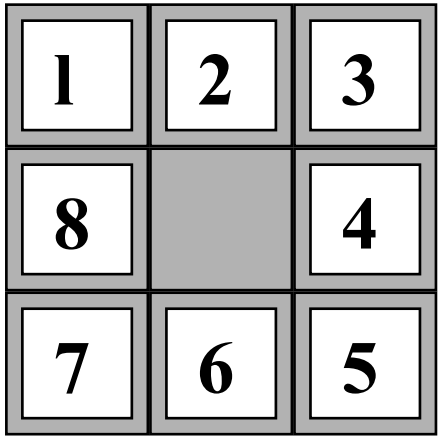
O mundo do aspirador

- *A toy problem*
  - localizações discretas
  - sujeira discreta
  - ações determinísticas (limpeza prefeita)
  - ambiente estático (cômodos permanecem limpos)
- Um ambiente maior, com n localizações possui ($n \times 2^n$) estados.

8-puzzle



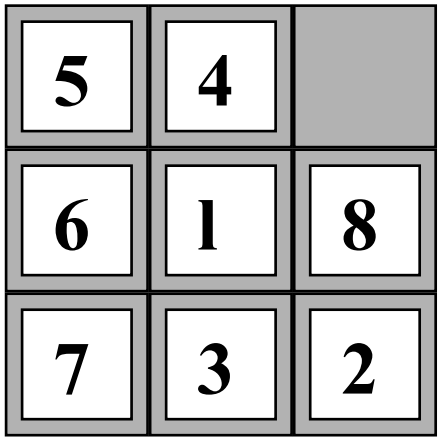
Estado Inicial



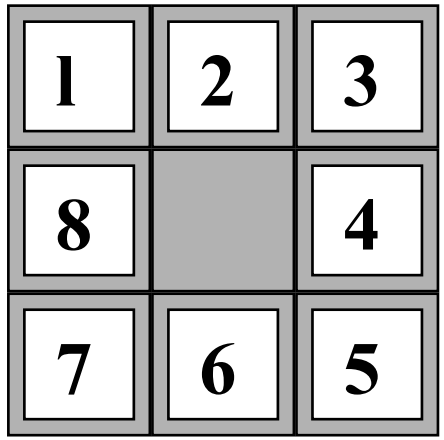
Estado Meta

Estados: ??
Operadores: ??
Teste de meta: ??
Função custo: ??

8-puzzle



Estado Inicial

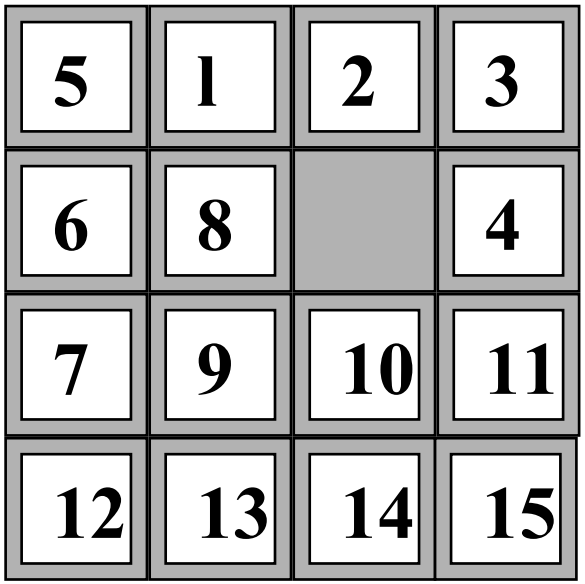


Estado Meta

Estados: posição de cada uma das 8 peças em um dos 9 quadrados
Operadores: movimentação do quadrado vazio para a esquerda, direita, para cima ou para baixo, que resulte em estados legais
Teste de meta: estado deve casar com a configuração do Estado Meta
Função custo: cada passo custa 1 ⟶ comprimento do caminho.

9!/2=181.440 estados alcançáveis

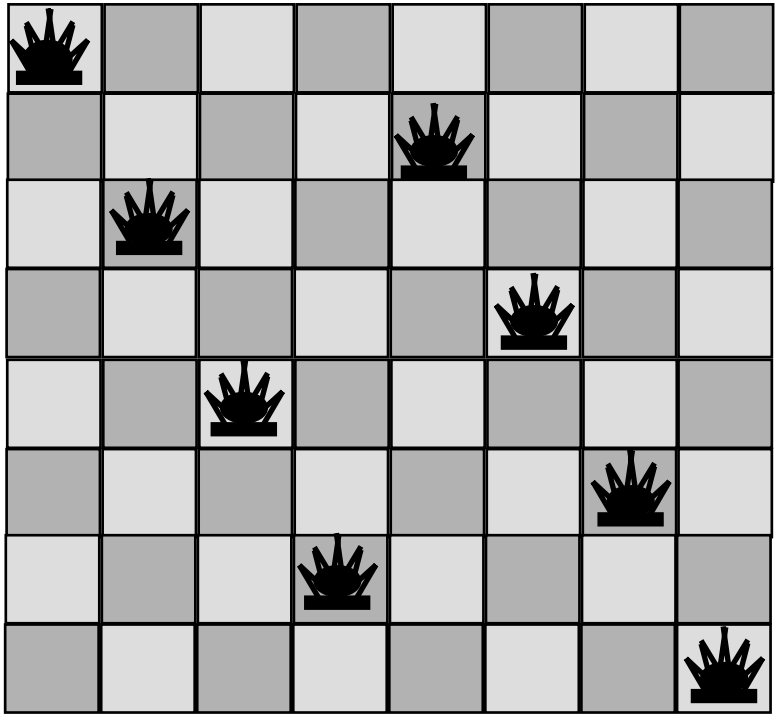
15-puzzle (tabuleiro 4 x 4)



1.3 trilhões de estados alcançáveis

- Instâncias aleatórias são resolvidas otimamente em alguns mili-segundos pelos melhores algoritmos de busca
- 24-puzzle (5 x 5): possui 10^{25} estados.

Problema das 8-rainhas



•Teste de meta: 8 rainhas no tabuleiro sem perigo de ataque

•A formulação correta faz uma grande diferença no tamanho do espaço de busca:

• 8-rainhas: de 3 x 10^{14} para 2057 sequências possíveis para serem investigadas)

• 100-rainhas: redução de 10^{400} para 10^{52} (ainda é um problema muito grande para ser resolvido por um agente de busca)

Problemas do mundo real (pag 67)

- Problemas de roteamento (redes de computadores, planejamento de operações militares, sistemas de planejamento de viagens aéreas)
- Navegação de robôs
- Linha de montagem automática
- Busca na Internet

Busca por soluções

Objetivo:

- Procurar por uma sequência de ações que transfore o estado inicial do mundo no estado meta

Possibilidades:

- Não existir solução
- Encontrar uma solução (qualquer solução)
- Encontrar a solução ótima (custo de caminho baixo)

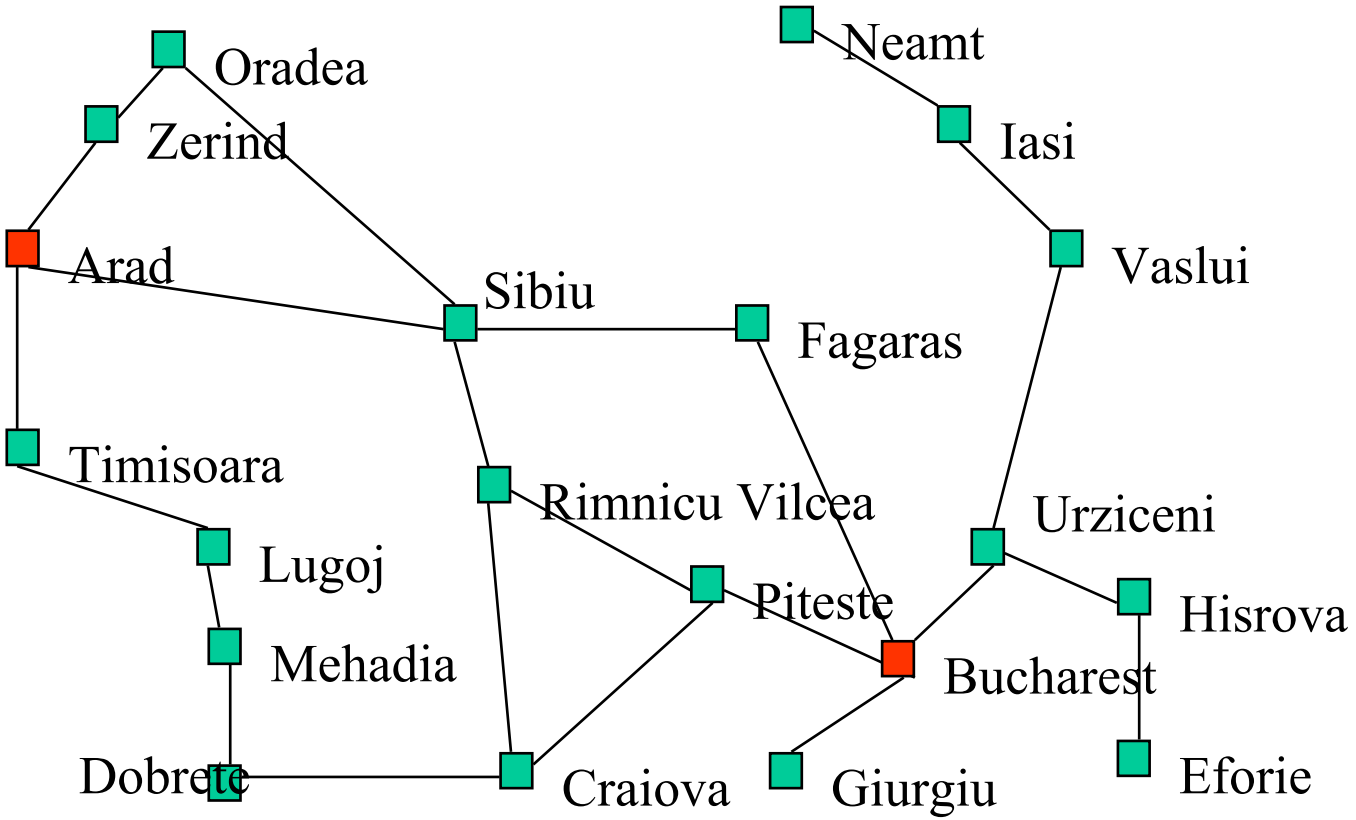
Custo do caminho x Custo da busca

- compromisso entre encontrar a melhor solução e o tempo gasto na busca

Algoritmo de busca

- analisa múltiplas escolhas de caminho para gerar uma sequência de ações
- faz essencialmente: (1) *escolher uma opção e deixar outras para analisar mais tarde ou* (2) *avaliar todas as seqüências em paralelo*
- usa uma estrutura em árvore para visitar as opções ➔ *árvore de busca*
- a decisão de quem escolher primeiro define uma estratégia de busca diferente

Espaço de estado: 20 nós
Árvore de busca: infinitos nós



Estrutura de dados para árvores de busca

- Um nó é formado por 5 componentes:
  - estado
  - nó pai
  - operador (usado para gerá-lo)
  - profundidade
  - custo do caminho

  função *EXPANDA*: gera os 5 componentes de um nó

Algoritmo geral de busca

função Busca-Geral(*problema*, *Queueing-FN*) devolve uma solução, ou falha

/* inicializa a árvore de busca com o estado inicial do *problema* */

nós ← CRIA-LISTA(Estado-Inicial[problema])

laço faça

se a lista de *nós* estiver vazia então devolva falha

nó ← Remove-Primeiro (*nós*)

se Testa-Meta(*nó*) então devolva *nó* (e o caminho da raiz até o nó)

nós ← *Queueing-FN* (nós, EXPANDA(nó, Ações[problema]))

fim

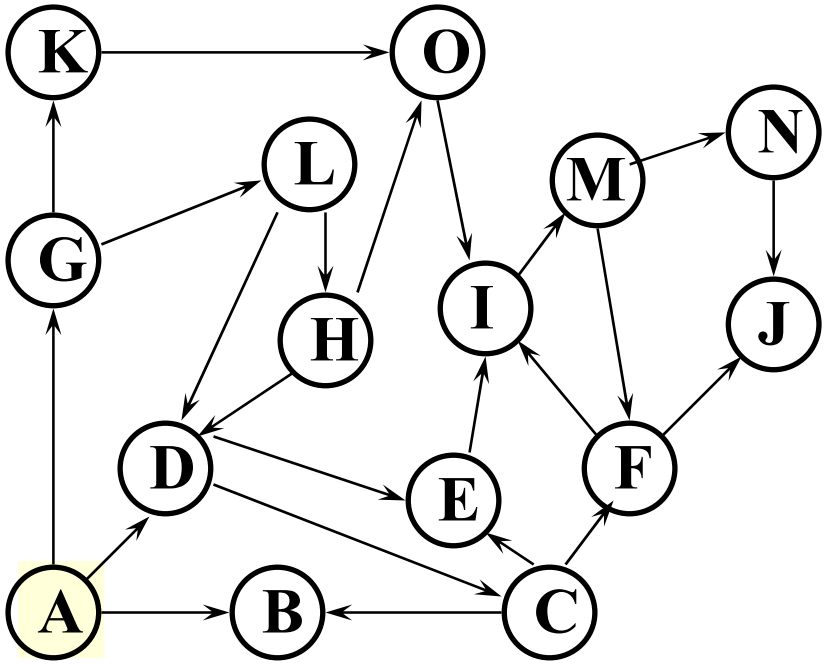
Estratégias em *Queueing-FN*:

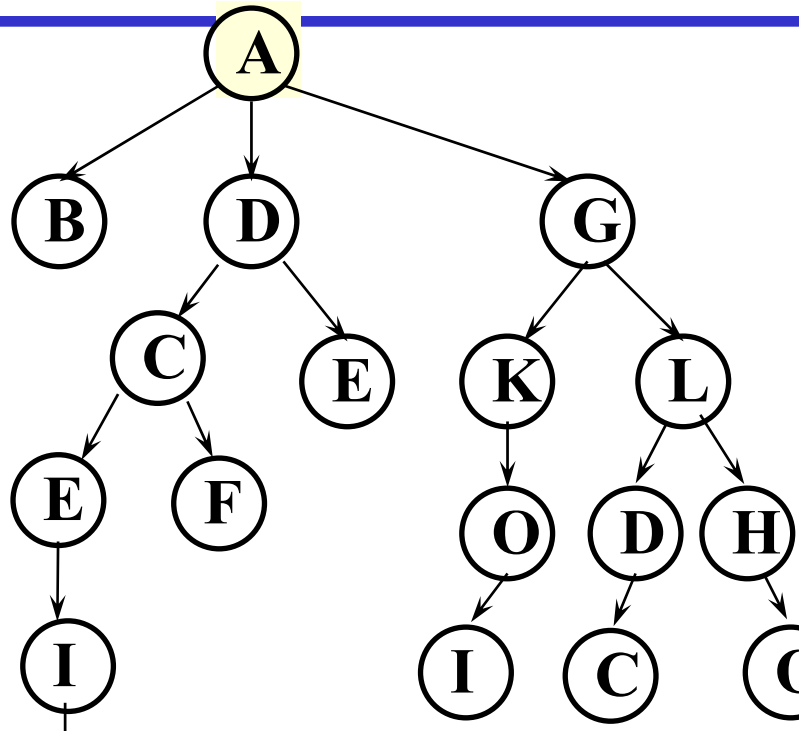
inserção numa fila: busca em largura

inserção numa pilha: busca em profundidade

fila de prioridade: A* (ordenar a fila de acordo uma função de avaliação)

espaço de estados X árvore de busca



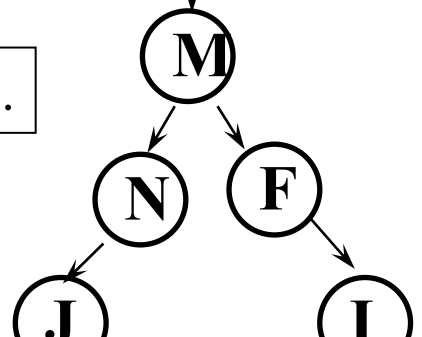


Busca em Largura (BFS)

A BDG CEKL EFODH IICO M NF JI ...

Busca em Profundidade (DFS)

AB DCEIMNJ FI F E GKOI ...



Árvore de busca gerada sem nós repetidos adotando uma estratégia de bus do tipo DFS.

Estratégias de Busca: avaliação

- Completude:
  - a estratégia garante encontrar uma solução, se existir uma?
- Qualidade da solução:
  - a estratégia encontra a solução ótima?
- Complexidade de tempo:
  - quanto tempo gasta para encontrar uma solução?
- Complexidade de memória:
  - quanto de memória é preciso?
- Que características deve ter um espaço de busca para que a estratégia seja adequada ?

Estratégias de busca não-informada

- BUSCA CEGA: não usa informações adicionais sobre a solução procurada. Variam quanto a ordem em que nós são expandidos:
  - Busca em largura - *Breadth first*
  - Busca em profundidade - *Depth first*
  - Busca de custo uniforme
  - Busca em profundidade limitada
  - Busca em profundidade iterativa
  - Busca bidirecional

Não possuem informação sobre o número de passos, ou o custo, do caminho entre um nó e a meta.