Agentes que resolvem problemas

- Forma geral de um agente que resolve problemas

```
function SIMPLE_PROBLEM_SOLVING_AGENT( percept) returns action
inputs:      percept, a percept
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 ← UPDATE_STATE( state, percept)
if seq is empty then
     goal ← FORMULATE_GOAL( state)
     problem ← FORMULATE_PROBLEM( state, goal)
     seq ← SEARCH( problem)
action ← RECOMMENDATION( seq, state)
seq ← REMAINDER( seq, state)
return action
```

Exemplo: Roménia

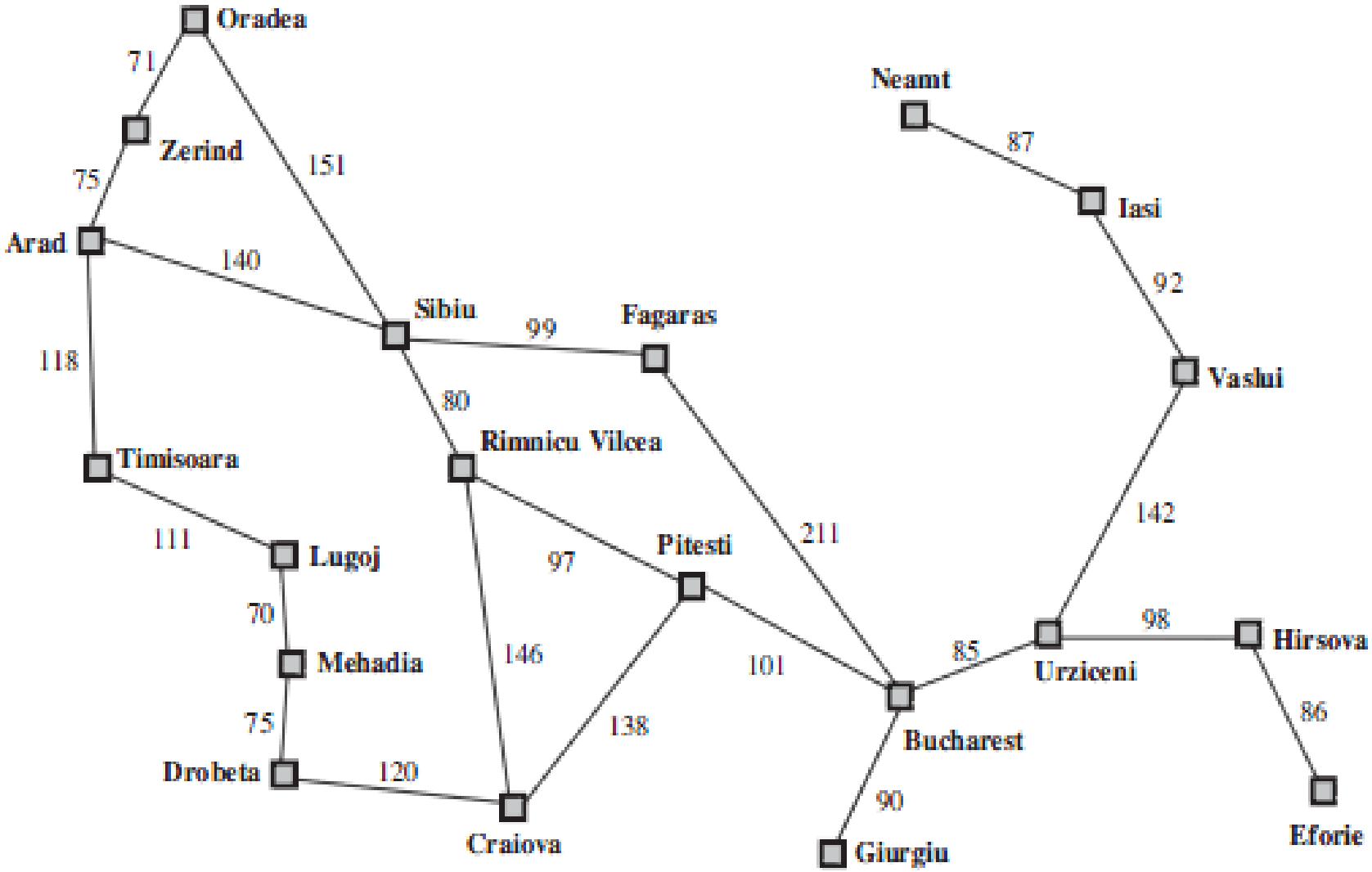
- De férias na Roménia; atualmente em Arad
- O voo parte amanhã de Bucareste

- Objetivo:
  - estar em Bucareste

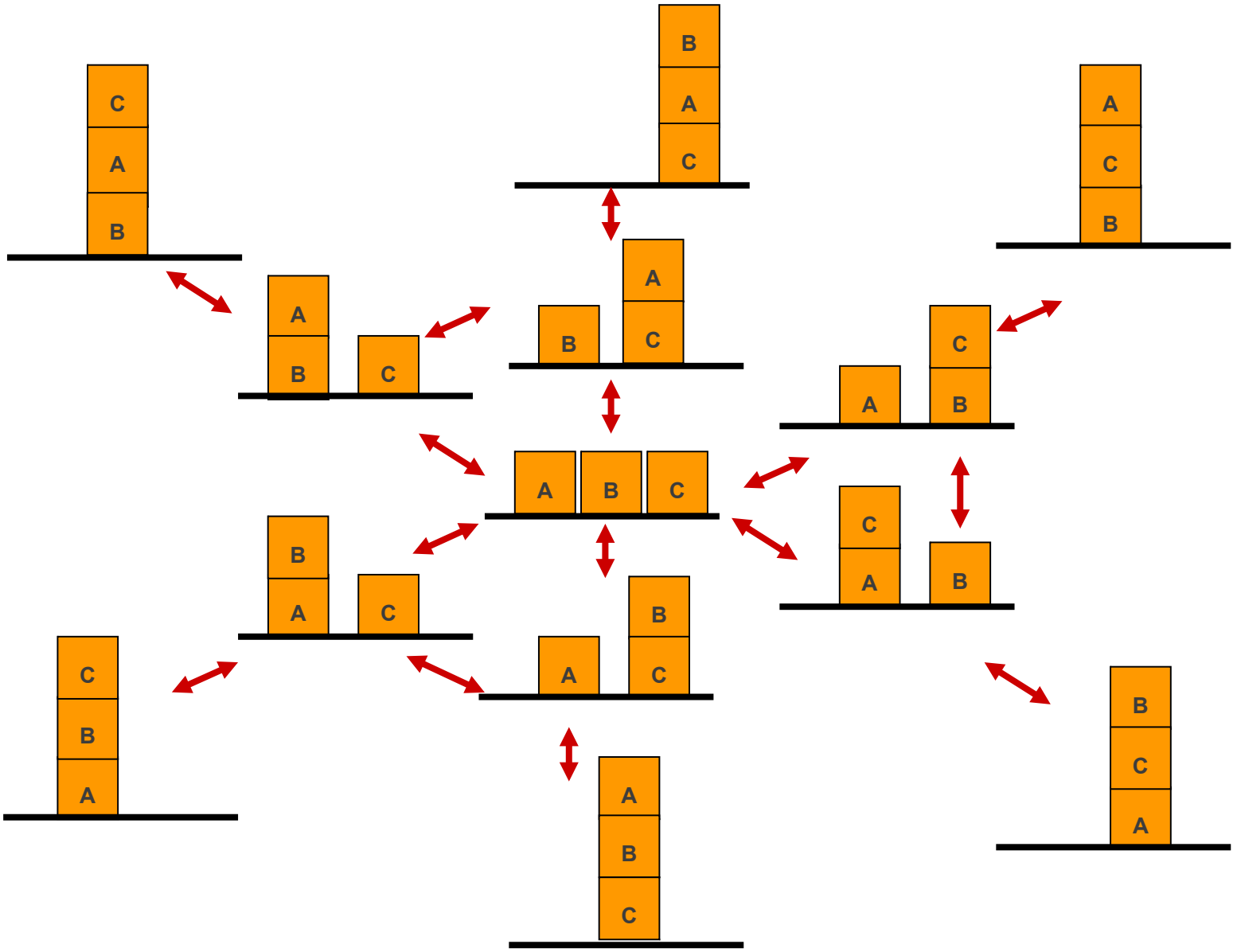
- Formular o problema
  - identificar os estados e as ações relevantes para a resolução do problema
    - estados: diversas cidades
    - ações: guiar de uma cidade para outra

- Encontrar uma solução:
  - sequência de cidades, por exemplo, Arad, Sibiu, Fagaras, Bucareste

Exemplo: Roménia (cont.)



Exemplo: o mundo dos blocos



Tipos de problemas

- Problema de estados simples (“single-state problem”)
  - o mundo é acessível; o agente pode calcular exactamente o efeito de cada ação.
- Problema de estados múltiplos (“multiple-state problem”)
  - o agente pode calcular exatamente o efeito de cada ação, mas tem acesso limitado ao estado do mundo (faltam sensores). O agente tem que raciocinar acerca de conjuntos de estados a que pode chegar, em vez de estados simples.
- Problema de contingência
  - não há garantia sobre o efeito das ações, são necessários sensores na fase de execução. A solução é uma árvore de ações, em vez de uma sequência de ações. Cada ramo da árvore corresponde a uma contingência possível.
- Problema de exploração
  - o agente não tem informação sobre o efeito das ações, tem que experimentar, e gradualmente ir descobrindo o efeito das ações e os tipos de estados existentes.

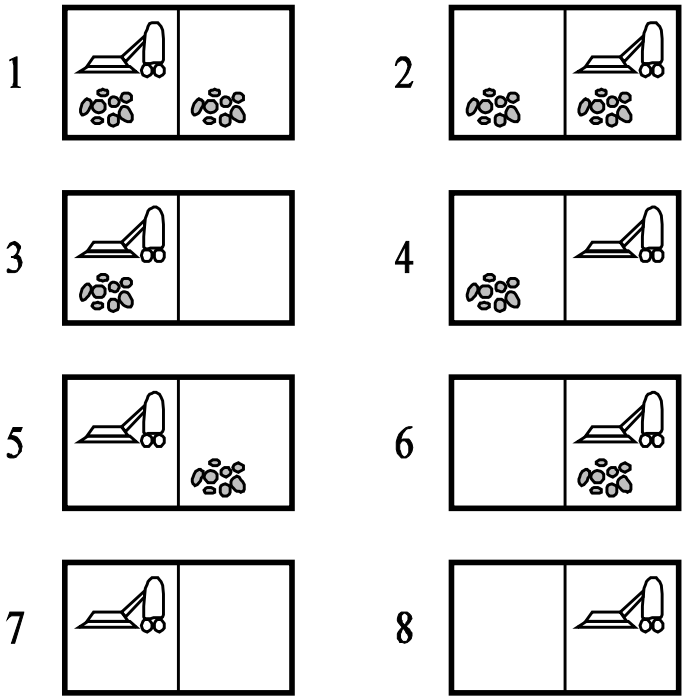
Exemplo: o mundo do aspirador (cont.)

- **Estados simples**
  - **Estado inicial é o 5.**
  - **Solução: [Direita, Aspira]**
- **Estados múltiplos**
  - **Estado inicial é {1,2,3,4,5,6,7,8}**

    Exemplo, Direita vai para {2,4,6,8}
  - **Solução: [Direita, Aspira, Esquerda, Aspira]**
- **Contingência**
  - **Estado inicial é o 5.**

    Ao aspirar pode não ficar limpo

    Sensores: lixo e localização
  - **Solução: [Direita, se Sujo então Aspira]**



Formulação do problema de estados simples

- Formalizando os elementos básicos da definição de um problema:
  - o *estado inicial*, onde o agente sabe estar
  - o conjunto de ações possíveis — *operadores*
    - *função sucessor S*: dado um estado x, S(x) retorna o conjunto de estados atingíveis a partir de x por efeito de uma ação
    - *espaço de estados* (“state space”): conjunto de todos os estados atingíveis a partir do estado inicial por efeito de uma qualquer sequência de acções
    - *caminho* (“path”): qualquer sequência de ações que levam de um estado a outro
  - *teste de objectivo* (“goal test”): aplicado pelo agente a qualquer estado para saber se é um objetivo
  - *custo do caminho* (“path cost”): associa um custo a um caminho; função g
- Um algoritmo de pesquisa recebe como entrada um problema, e retorna uma solução, isto é, um caminho desde o estado inicial até um estado que satisfaz o teste de objetivo.

Escolha de um espaço de estados

- O mundo real é imensamente complexo

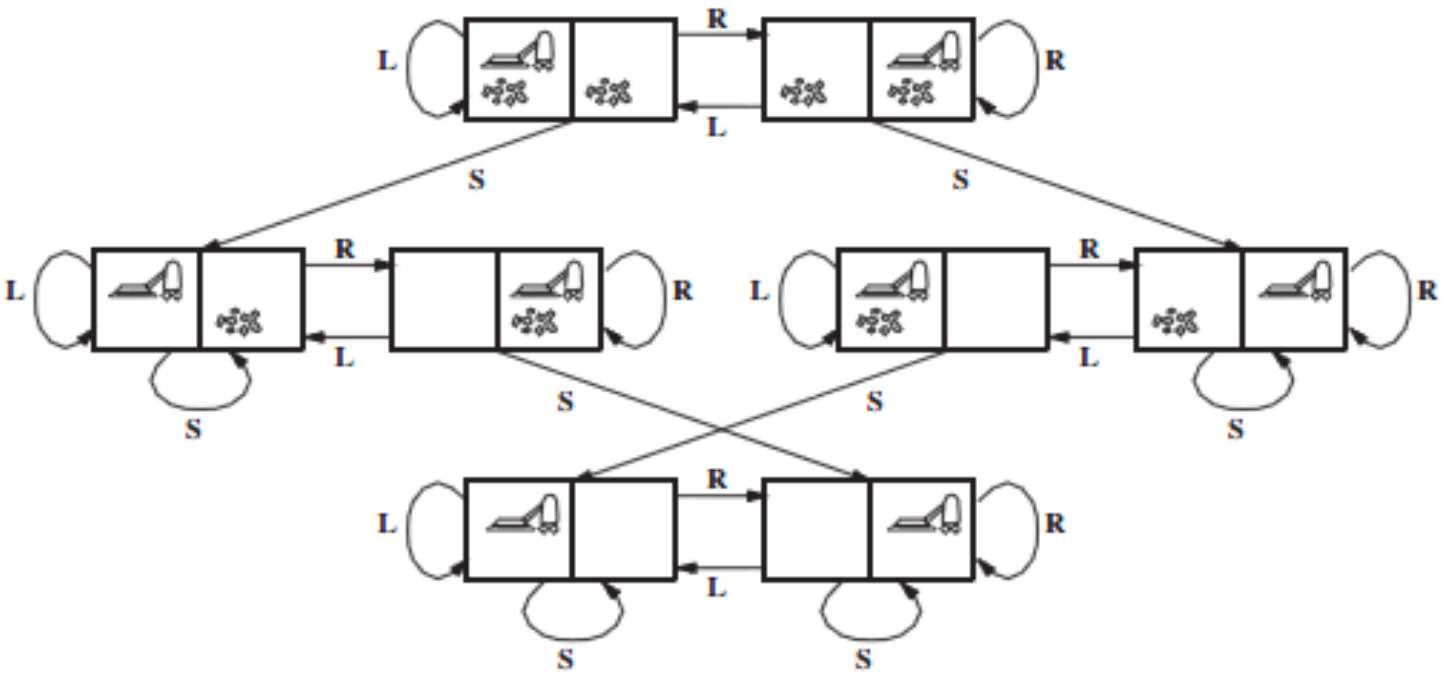
  ==> tem que se abstrair o espaço de estados

  - Um estado abstrato = um conjunto de estados reais
  - Uma ação abstrata é uma combinação complexa de ações reais
    - Por exemplo “Arad → Zerind” é uma conjunto complexo de paragens, desvios, abastecimento, ultrapassagens, etc.

- Solução abstrata
  - conjunto de caminhos reais que constituem uma solução no mundo real

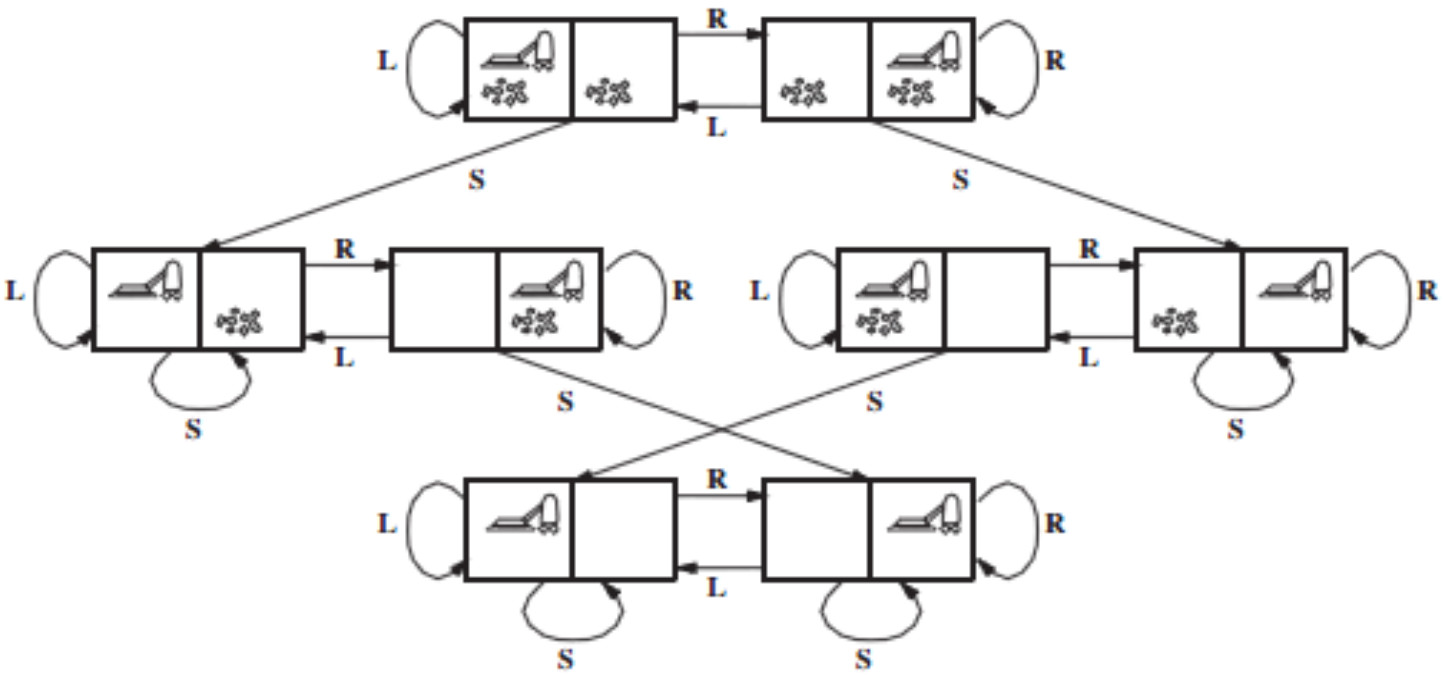
- Cada ação abstrata deve ser mais “simples” que a ação real

Exemplo: o mundo do aspirador



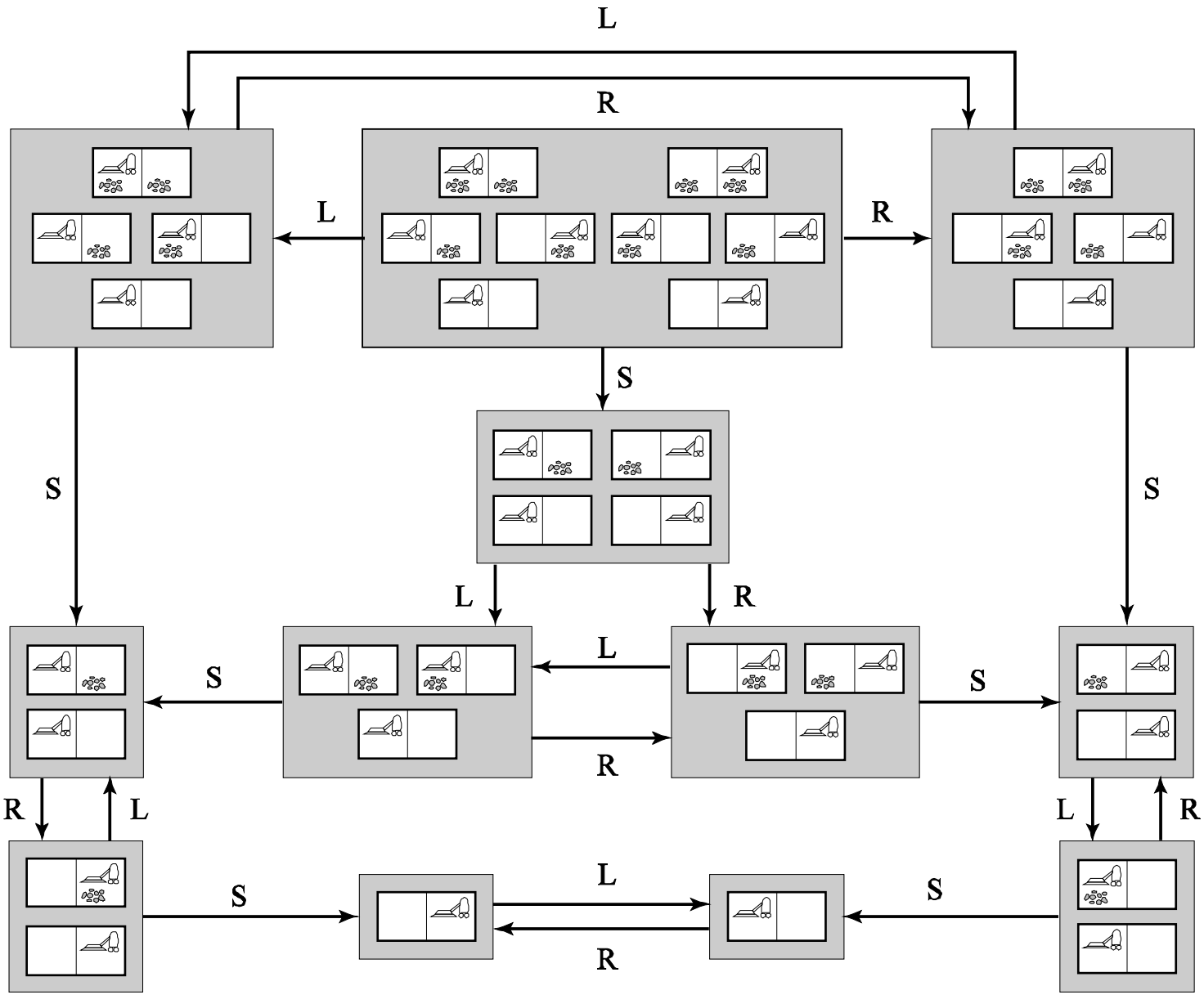
- **Estados ??**
- **Ações ??**
- **Teste de objetivo ??**
- **Custo do caminho ??**

Exemplo: o mundo do aspirador

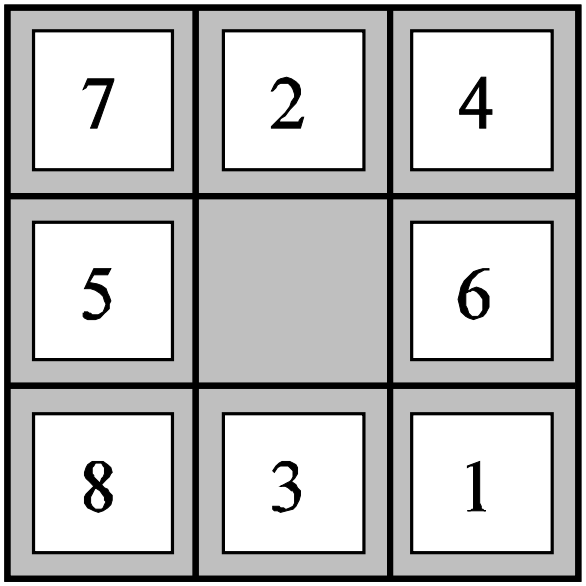


- **Estados ??** existência de lixo (ignoremos a quantidade) e localização
- **Ações ??** Esquerda, Direita, Aspirar, Nada
- **Teste de objetivo ??** Não há lixo
- **Custo do caminho ??** 1 por Ação (0 para Nada)

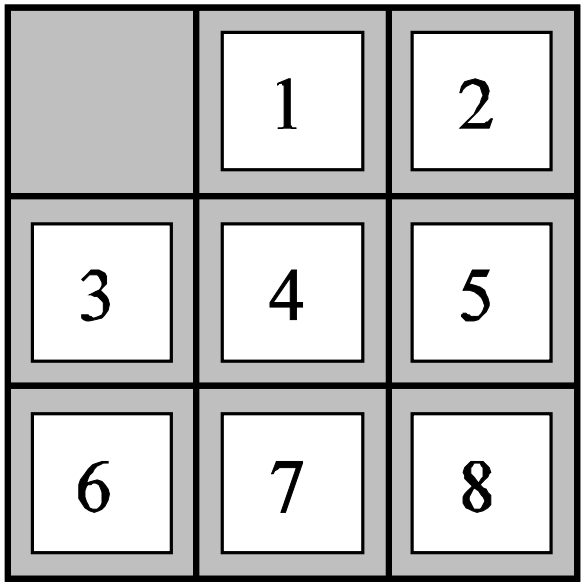
Exemplo: o mundo do aspirador (sem sensores)



Exemplo: o puzzle de oito



Start State



Goal State

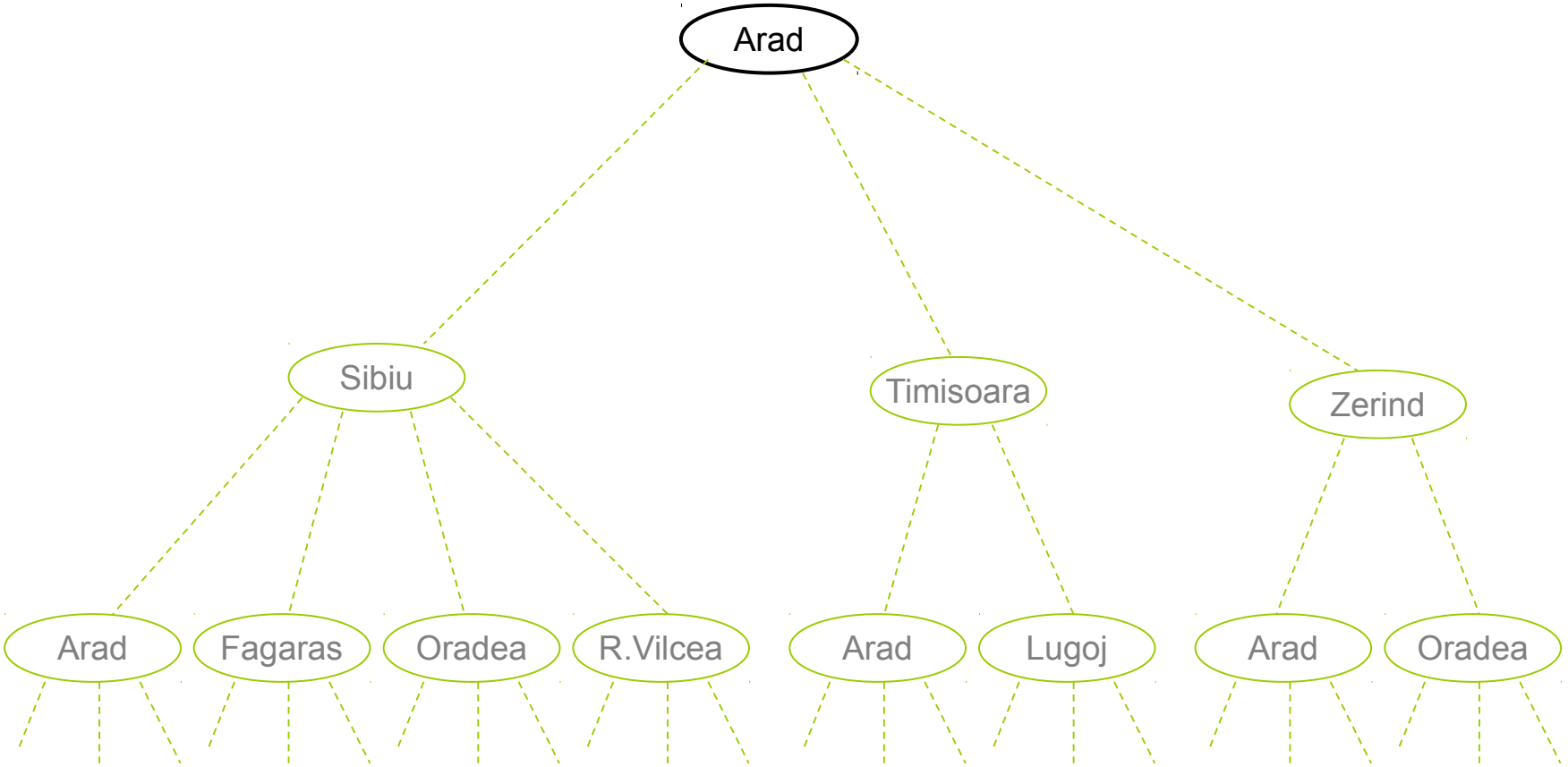
- **Estados ??**
- **Ações ??**
- **Teste de objetivo ??**
- **Custo do caminho ??**

Algoritmos de pesquisa em árvores

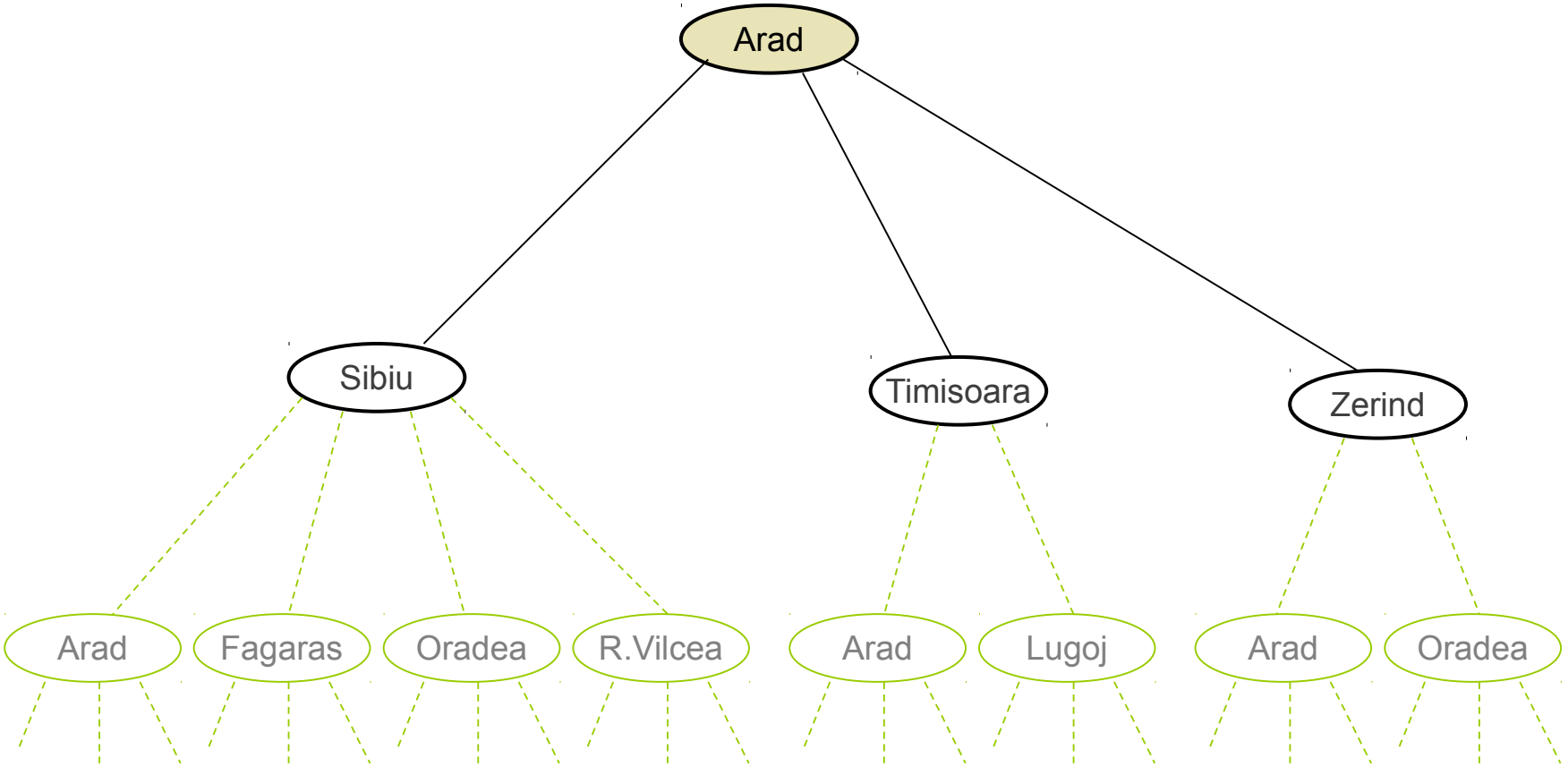
- Ideia base:
  - Exploração simulada do espaço de estados, pela geração de sucessores de estados já explorados

```
função PESQUISA ( problema, estratégia) retorna uma solução ou falhanço
    inicializa a árvore de pesquisa usando o estado inicial do problema
    repete
        se não há candidatos para expandir então
                retorna falhanço
        escolhe um nó terminal para expansão de acordo com a estratégia
        se o nó contém um estado final então
                retorna solução
        senão
            expande o nó e junta os nós resultantes à árvore de pesquisa
    fim
```

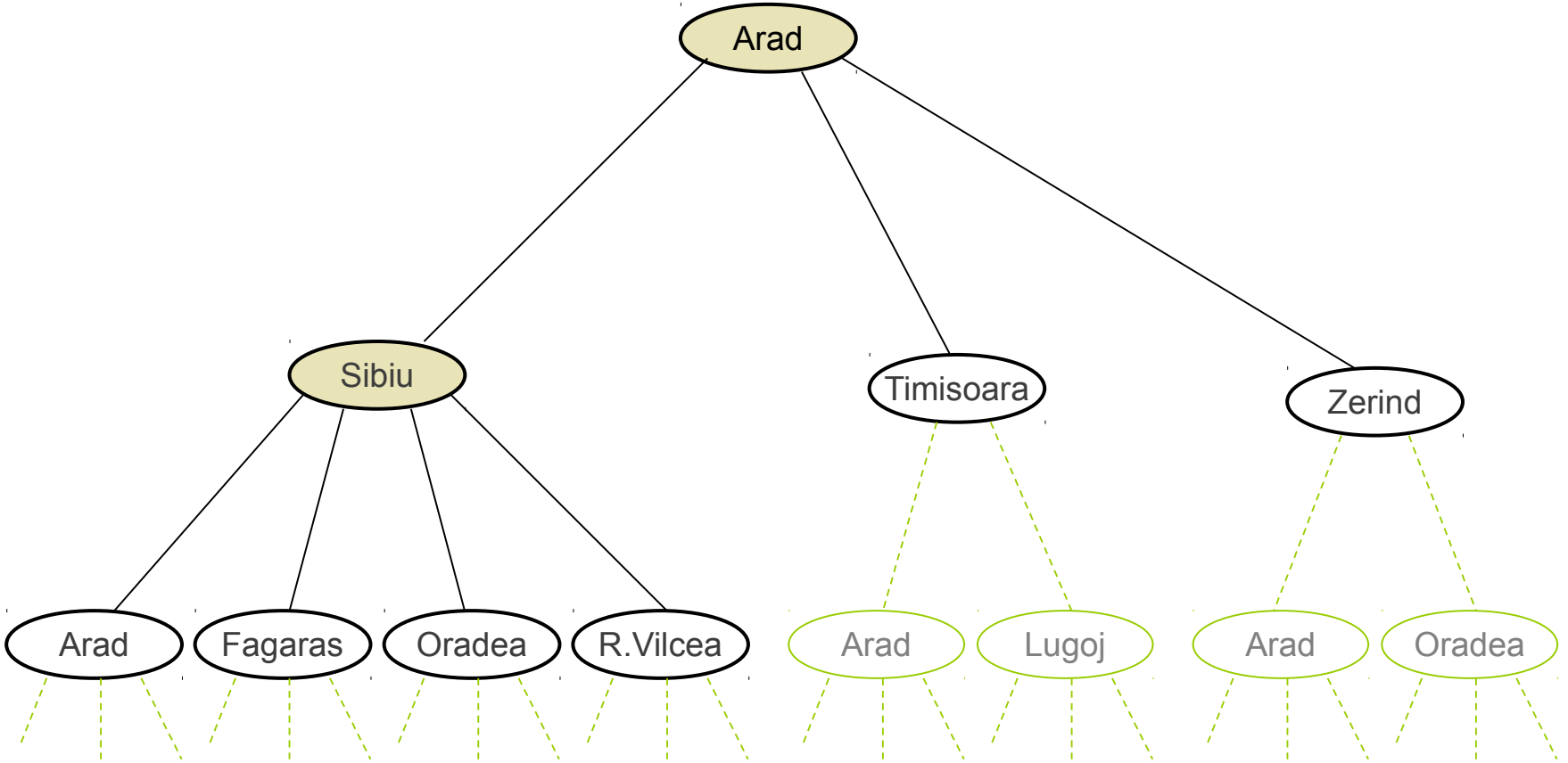
Exemplo: Roménia



Exemplo: Roménia

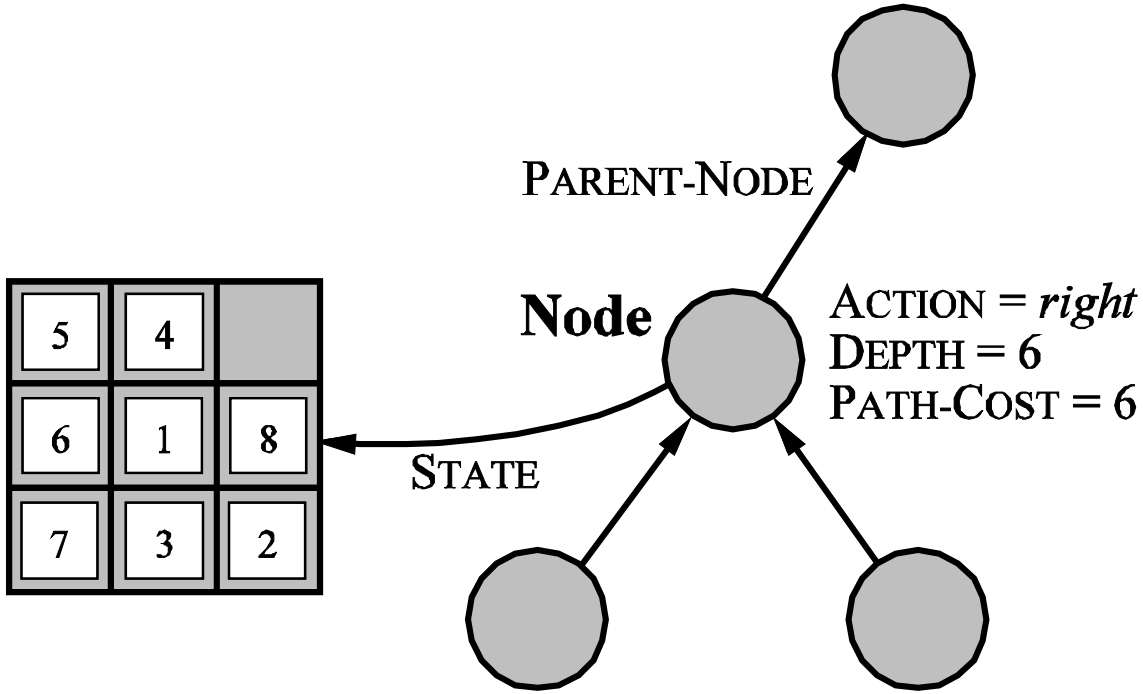


Exemplo: Roménia



Estados versus nós

- Estado
  - Representação de uma configuração física, uma representação do mundo
- Nó
  - Estrutura de dados que é parte de uma árvore de pesquisa
  - Inclui
    - Estado, pai, operador, profundidade, custo do caminho



Pesquisa em árvore

```
função PESQUISA ( problema, fronteira) retorna uma solução ou falhanço
    fronteira ← INSERE( CRIA-NÓ( ESTADO-INICIAL[PROBLEMA]))
    repete
       se VAZIA( fronteira) então
         retorna falhanço
       nó ← REMOVE-CABEÇA( fronteira)
       se TESTE-OBJECTIVO[ problema] aplicado a ESTADO( nó) sucede então
         retorna nó
       fronteira ← INSERE-TODOS( EXPANDE( nó, problema), fronteira)
    fim
```

Estratégias de pesquisa

- Uma estratégia de pesquisa é definida pela ordem pela qual os nós fronteira são expandidos
- Avaliação das estratégias de pesquisa:
  - Plenitude — garante encontrar uma solução se ela existir
  - Complexidade temporal — quanto tempo demora encontrar uma solução
  - Complexidade espacial — quanta memória é necessária para realizar a pesquisa
  - Optimização — quando há mais do que uma solução, é encontrada a de melhor qualidade (de mais baixo custo)

  - As complexidades temporal e espacial são medidas em função de:

    b — Branching factor (fator de ramificação) — um estado pode ser expandido em b sucessores.

    d — profundidade da solução de menor custo

    m — profundidade máxima do espaço de estados (pode ser ∞).