Inteligência Artificial

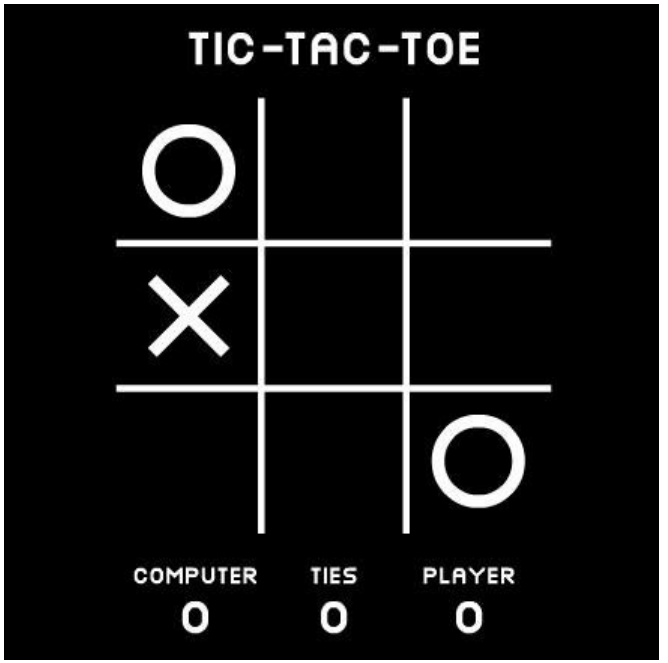
Resolução de problemas

Baseado no livro Inteligência Artificial de Russel e Norvig
Profs. Rodrigo Goulart e Alexandre Zamberlam

Resolução de problemas

- **IA** se detém da **busca de soluções** para problemas que não dispõem de solução algorítmica, ou que tem soluções algorítmicas mas sua complexidade as torna impraticáveis. Por exemplo:
  - Prova automática de teoremas
  - Quebra-cabeças
  - Jogos

Resolução de problemas

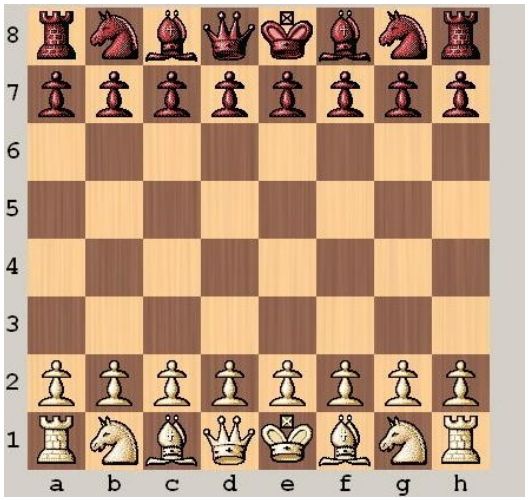


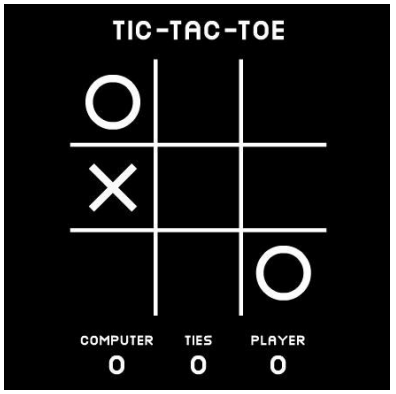
Características vs IA

- São solucionáveis por humanos (inteligência)

Características vs IA

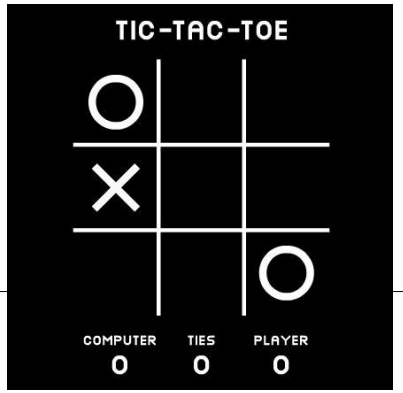
- São solucionáveis por humanos (inteligência)
- Classes de complexidade variável: jogo da velha vs. Xadrez





Características vs IA

- São solucionáveis por humanos (inteligência)
- Classes de complexidade variável: jogo da velha vs. Xadrez
- São problemas de conhecimento total (formalização)



Características vs IA

- São solucionáveis por humanos (inteligência)
- Classes de complexidade variável: jogo da velha vs. Xadrez
- São problemas de conhecimento total (formalização)
- Solução passa por uma sequência de situações legais, finita e conhecida

Solução

- Solução para problemas sem solução algorítmica
  - **Busca**

Solução

- Solução para problemas sem solução algorítmica
  - **Busca**
- Definida em termos de
  - **Objetivo**(s)
  - **Formulação do problema**, definido por:
    - Estado inicial, função sucessor, teste de objetivo (estados finais) e custo do caminho

Solução

- Solução para problemas sem solução algorítmica
  - **Busca**
- Definida em termos de
  - **Objetivo**(s)
  - **Formulação do problema**, definido por:
    - Estado inicial, função sucessor, teste de objetivo (estados finais) e custo do caminho

Problema do viajante

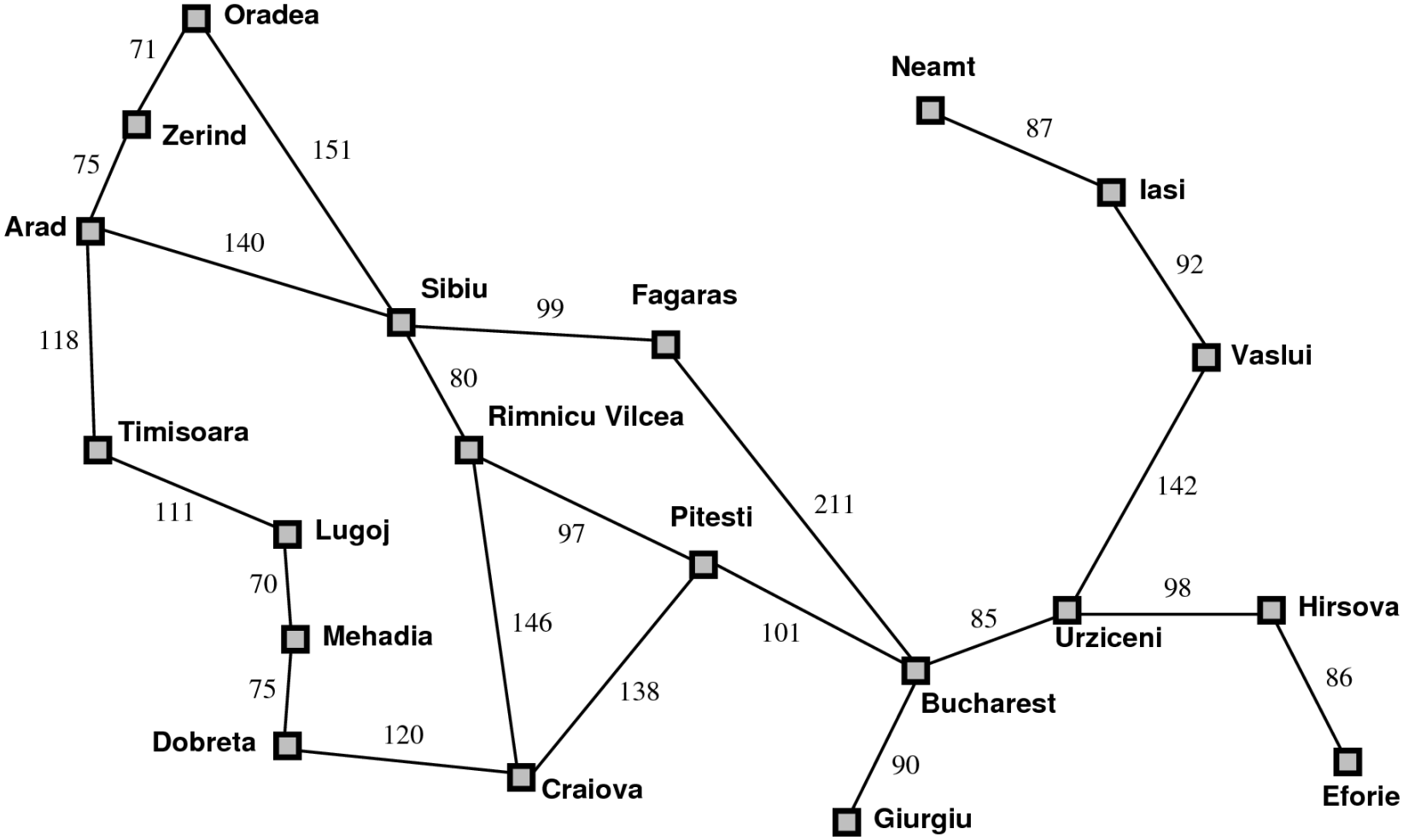
Problema do viajante

- Contexto
  - Um viajante se encontra na cidade de Arad, na Romênia, de férias.
  - Ele deseja **maximizar** alguns **fatores**: conhecimento da língua (romeno), bronzeado, paisagens, etc)

Problema do viajante

- Contexto
  - Um viajante se encontra na cidade de Arad, na Romênia, de férias.
  - Ele deseja **maximizar** alguns **fatores**: conhecimento da língua (romeno), bronzeado, paisagens, etc)
  - Ele ganhou uma passagem não reembolsável para Bucareste
  - **Objetivo:** chegar a Bucareste

Problema do viajante



Problema do viajante

- Estado inicial: em(Arad)
- Função sucessor: SUCESSOR(x) → <ação,sucessor>*
  Ex.:
  SUCESSOR(em(Arad)) →
  <ir(Sibiu),em(Sibiu)>, ...
- Teste de objetivo(estado final): {em(Bucareste)}
- Custo do caminho

Problema do viajante

Problema do viajante

- Custo do passo: c(x, a, y)
  Ex.:
  c(em(arad),ir(sibiu),em(sibiu))
  → 140

Problema do viajante

- Custo do passo: c(x, a, y)
  Ex.:
  c(em(arad),ir(sibiu),em(sibiu))
  → 140
- Espaço de estados

Problema do viajante

- Custo do passo: c(x, a, y)
  Ex.:
  c(em(arad),ir(sibiu),em(sibiu))
  → 140
- Espaço de estados
- Solução ótima

Problema do viajante

- Custo do passo: c(x, a, y)
  Ex.:
  c(em(arad),ir(sibiu),em(sibiu))
  → 140
- Espaço de estados
- Solução ótima
- Maximização de fatores

Outros exemplos de problemas

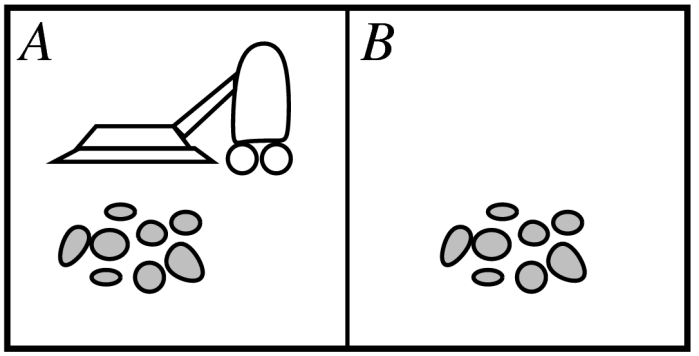
Outros exemplos de problemas

- Miniproblema vs. Problema do mundo real

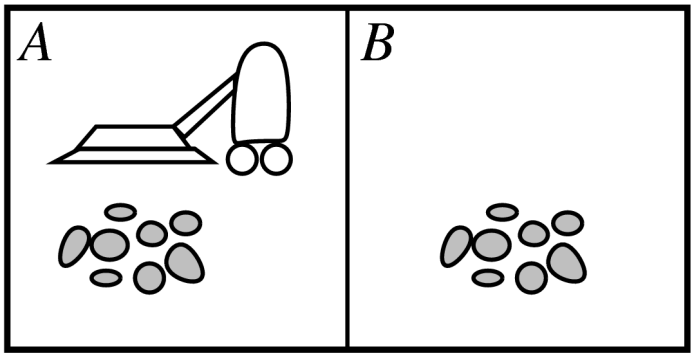
Outros exemplos de problemas

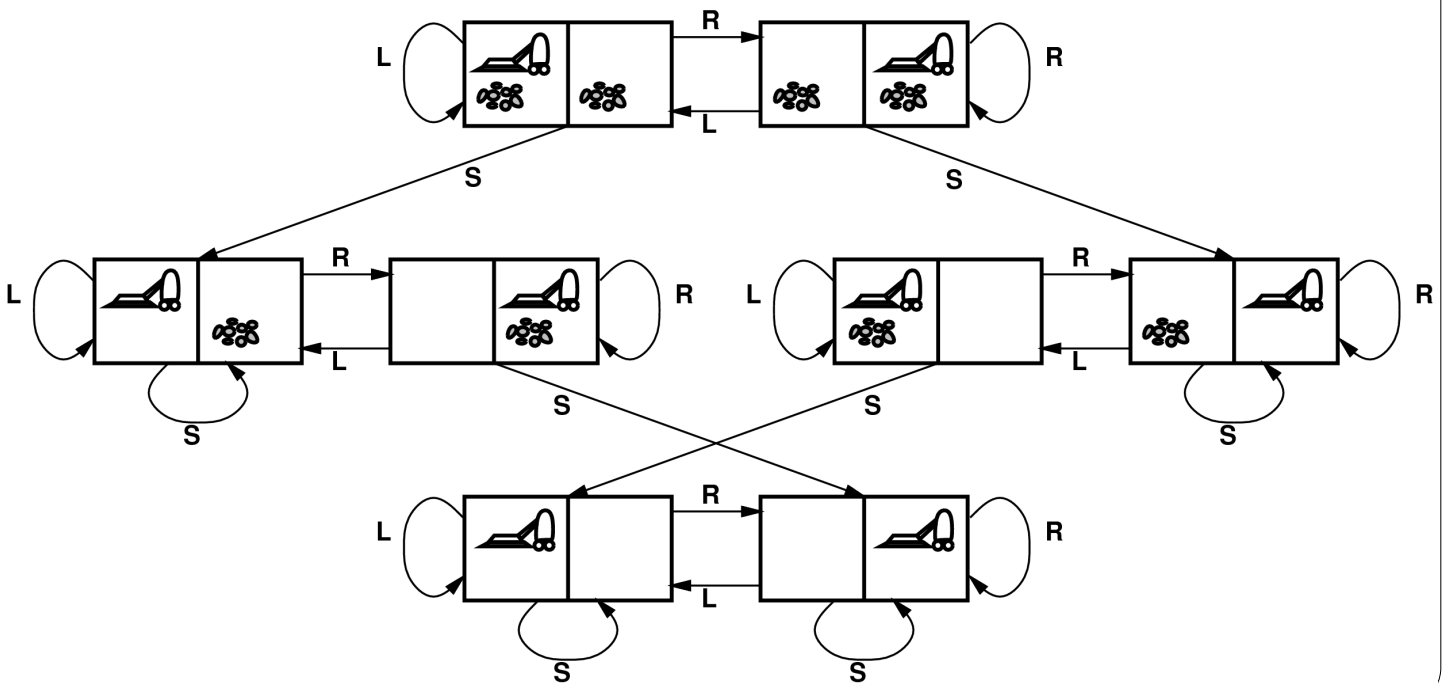
- Miniproblema vs. Problema do mundo real
- Miniproblemas
  - Aspirador de pó
  - Quebra-cabeça de 8 peças
  - Problema de 8 rainhas

Aspirador de pó



Aspirador de pó



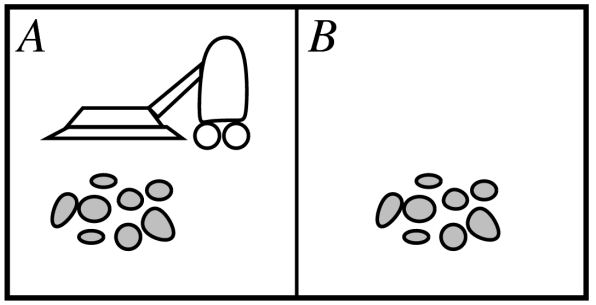


Aspirador de pó

- Estados: O aspirador robô pode se deslocar me 2 posições, que contém ou não sujeira

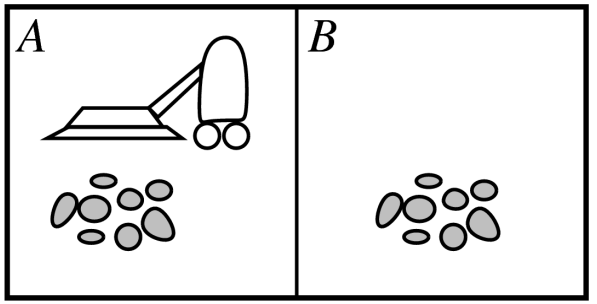
  (2 x 2^2 = 8 estados possíveis de mundo)

- Estado inicial: qualquer estado



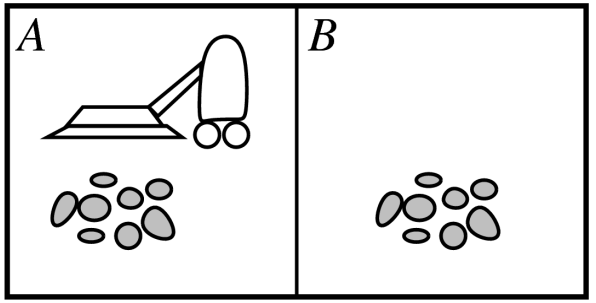
Aspirador de pó

- Função sucessor: gera os estados possíveis da tentativa de executar as 3 ações possíveis (direita, esquerda e aspirar)



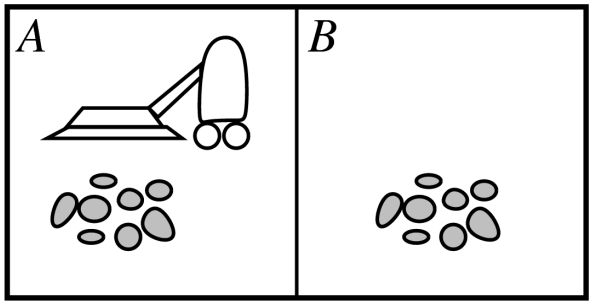
Aspirador de pó

- Teste de objetivo: verifica se todos os quadrados estão limpos
- Custo do caminho: custo 1, ou seja, o número de passos do caminho

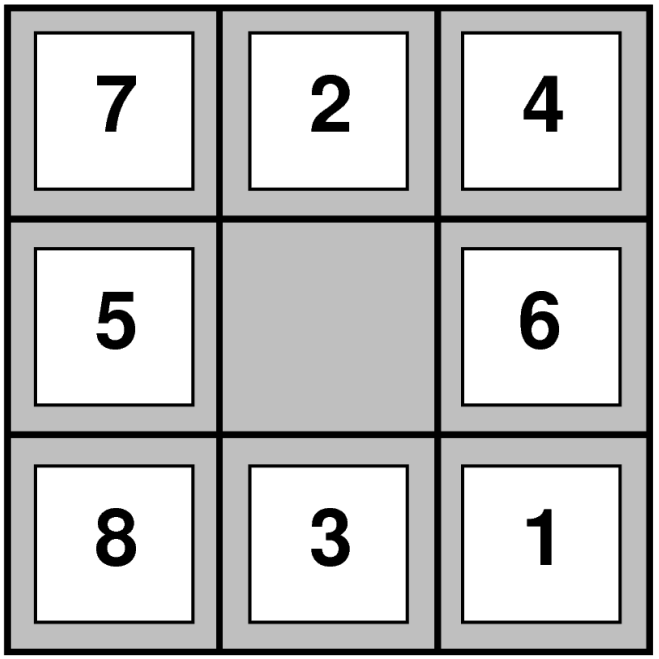


Aspirador de pó

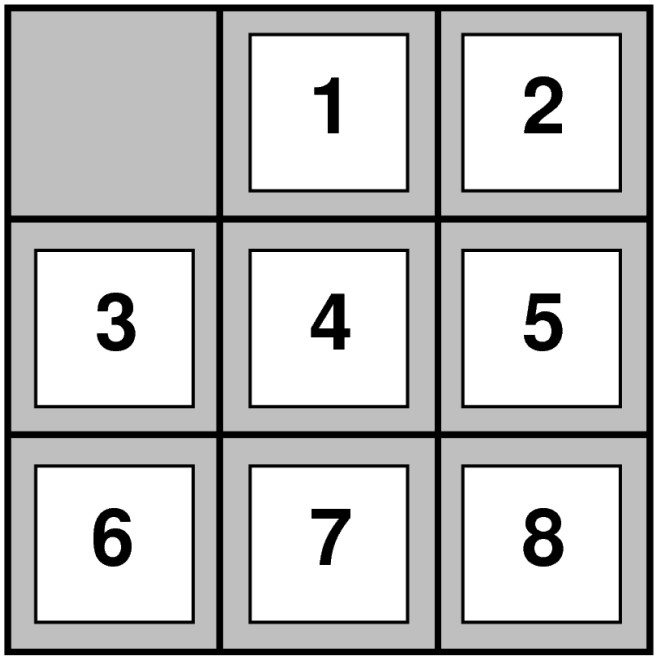
- Mundo real vs. Miniproblema
  - Posições discretas
  - Sujeira discreta
  - Limpeza confiável
  - Ambiente se mantém organizado
  - Ambientes maiores: n x 2^n



Quebra-cabeça



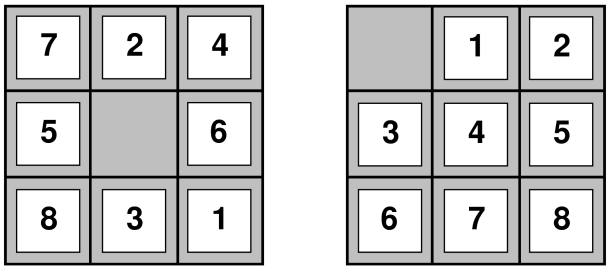
Start State



Goal State

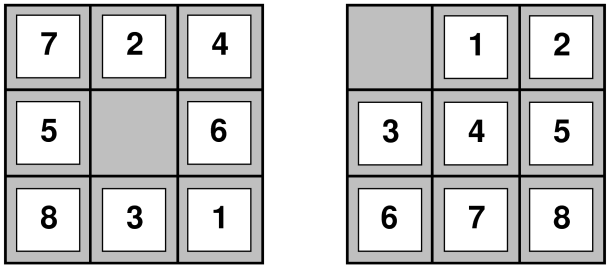
Quebra-cabeça

- Estados: 3x3, 8 peças, 1 posição vazia
- Estado inicial: qualquer estado é válido
- Função sucessor: estados possíveis do deslocamento do espaço vazio



Quebra-cabeça

- Teste de objetivo: estado objetivo (figura)
- Custo do caminho: custo 1, ou seja, o número de passos do caminho



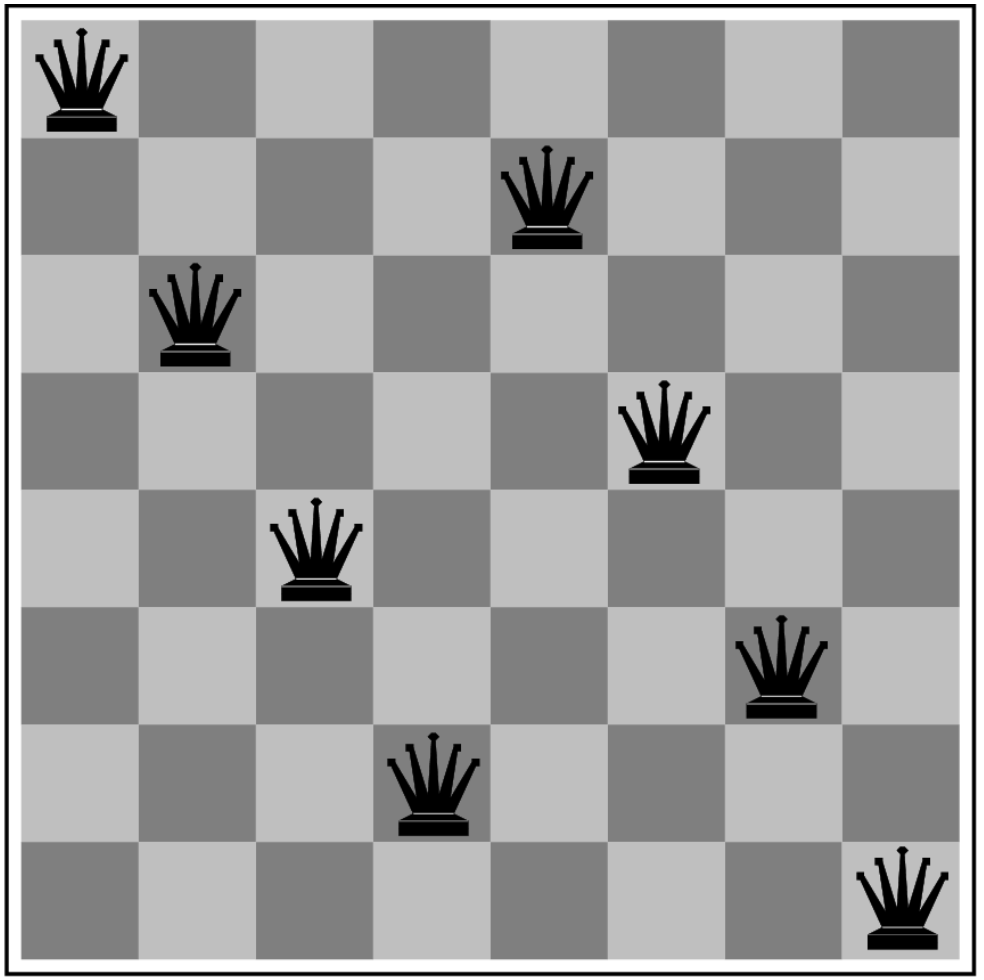
Quebra-cabeça

- Mundo real vs. Miniproblema
  - Classe de problema: quebra-cabeças de blocos deslizantes (NP-completo).
    - 8 peças: 181.440 estados possíveis
    - 15 peças: 1,3 trilhão estados possíveis

Quebra-cabeça

- Mundo real vs. Miniproblema
  - Questões físicas em geral são desconsideradas
    - Sacudir o tabuleiro
    - Extrair as peças

8 rainhas



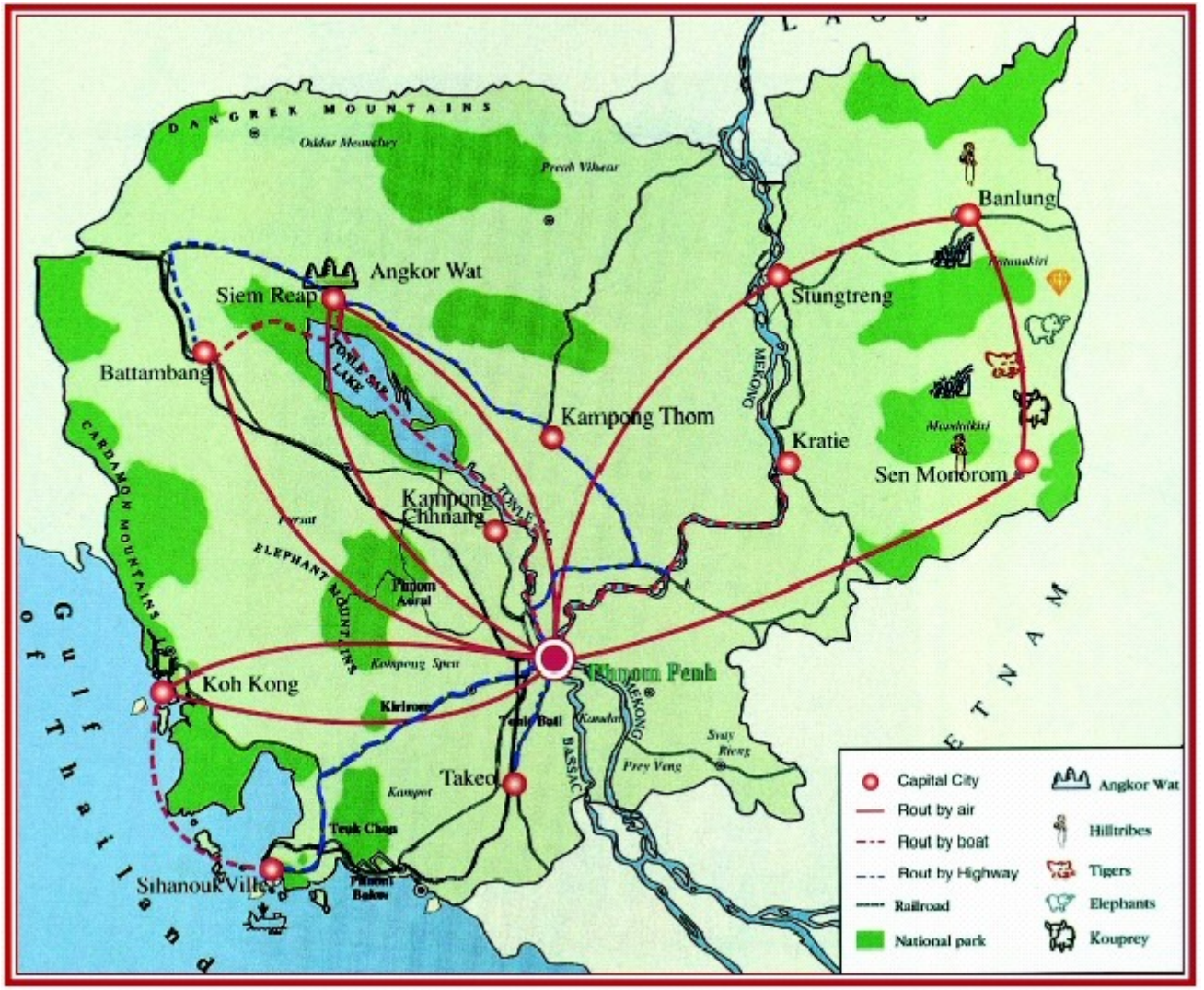
8 rainhas

- Estados: qualquer posição de 0 a 8 rainhas é um estado
- Estado inicial: nenhuma rainha no tabuleiro
- Função sucessor: colocar uma rainha em qualquer posição vazia
- Teste de objetivo: 8 rainhas estão no tabuleiro e nenhuma é atacada

Problemas de mundo real

- Roteamento
- Tour
- Caixeiro-viajante
- Layout VSLI
- Navegação de robôs
- Sequência automática de montagem

Roteamento

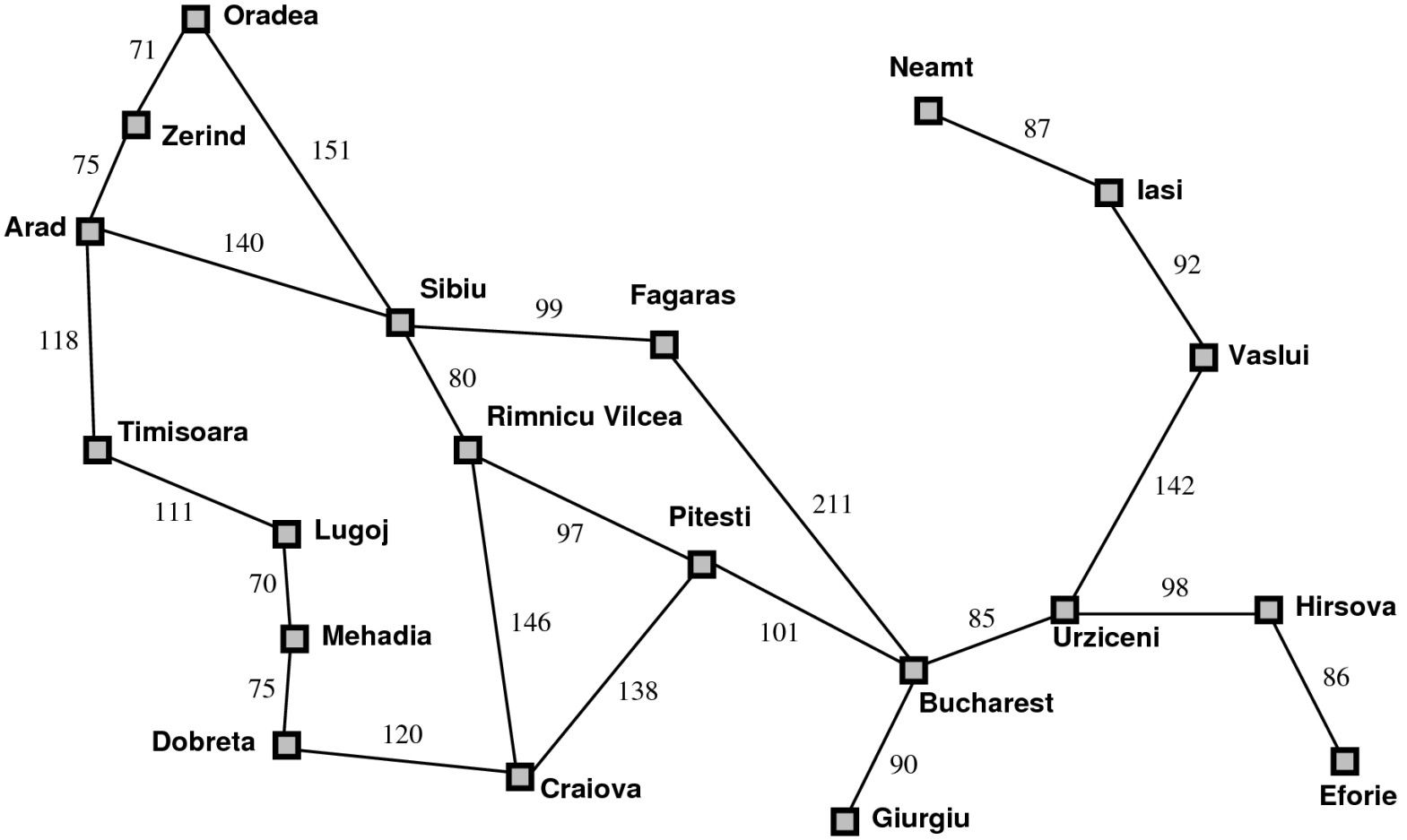


- Camboja
- tempo vs. local

Roteamento

- Custos
  - Montário
  - Espera
  - Tempo de voo
  - Procedimentos alfandegários
  - Qualidade de poltrona
  - Horário
  - Tipo de aeronave
  - etc.

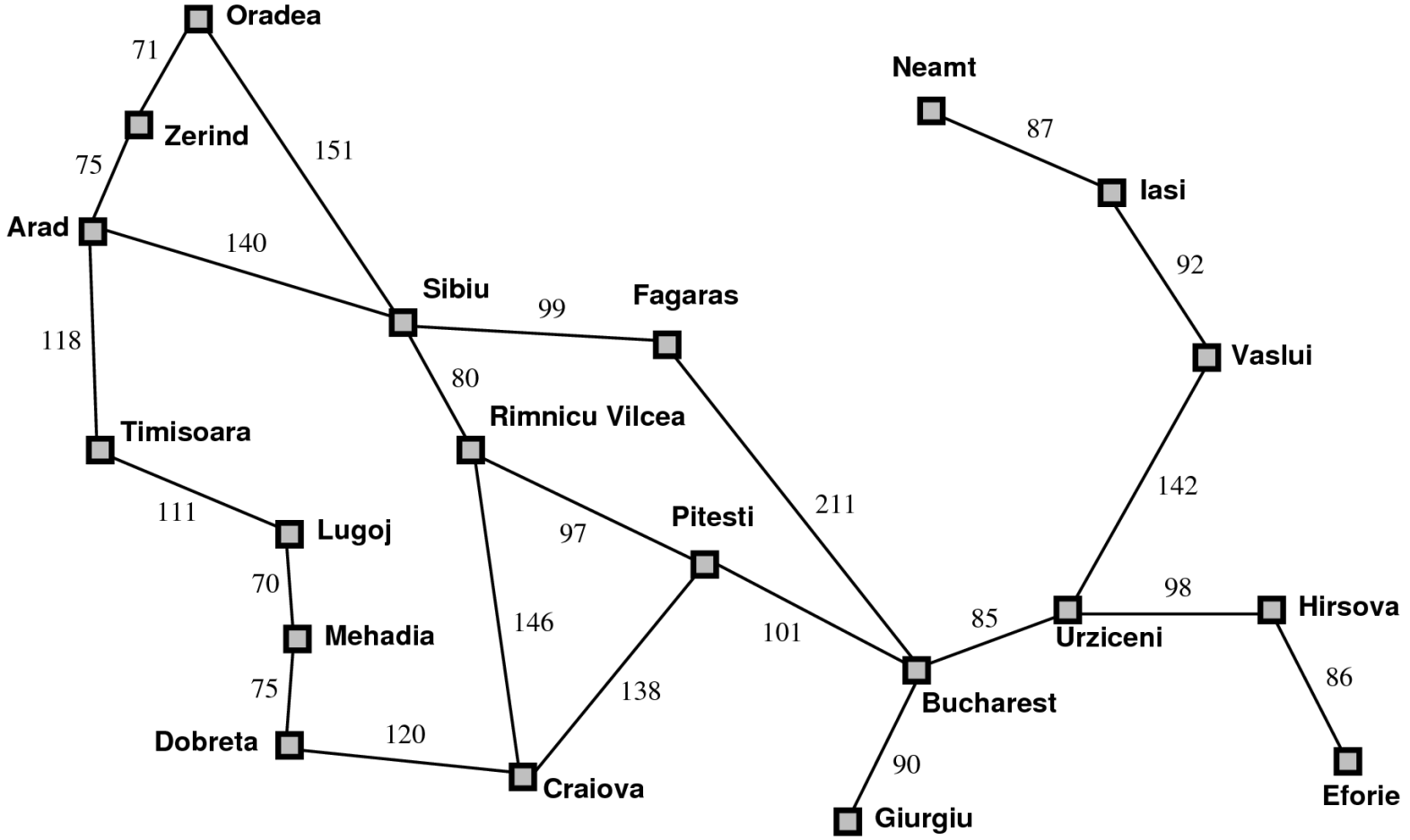
Tour



Visitar todas as cidades pelo menos 1 vez

Caixeiro-viajante

Caixeiro-viajante

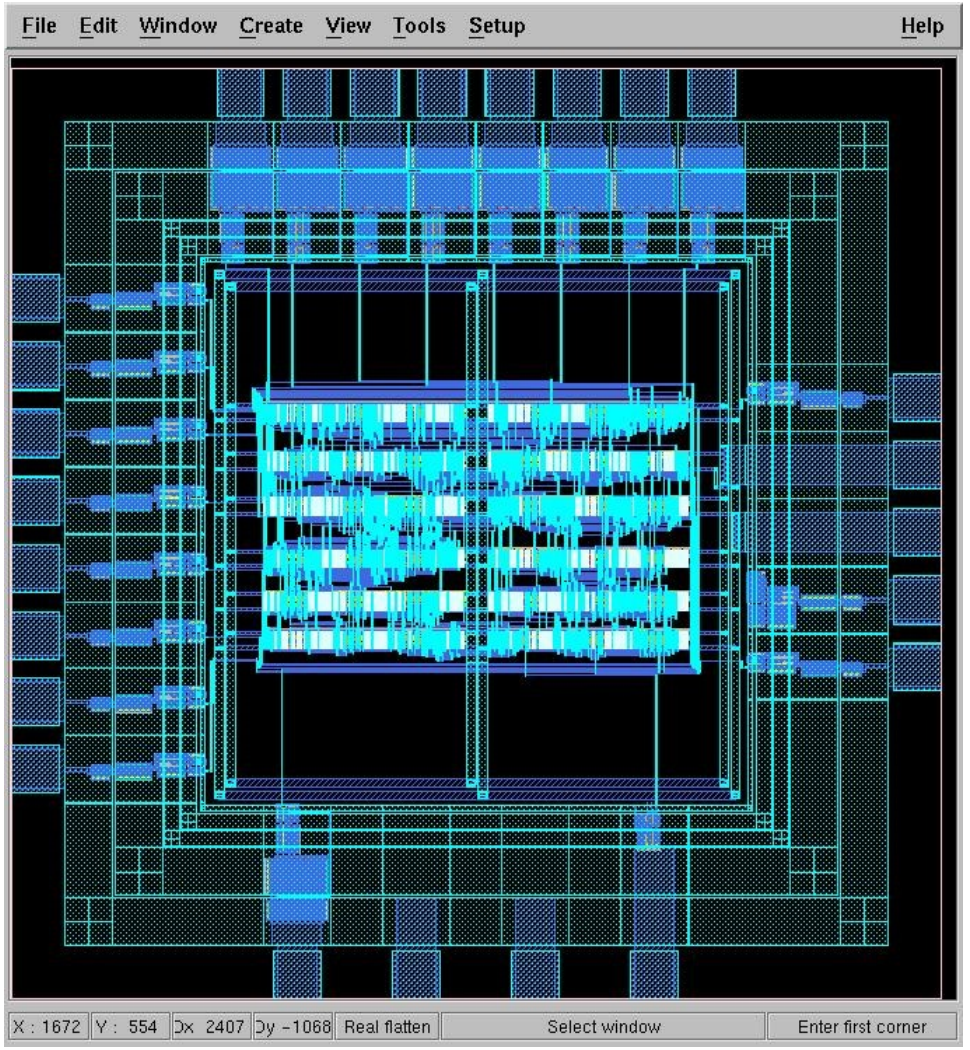


Visitar todas as cidades <u>apenas</u> 1 vez

Caixeiro-viajante

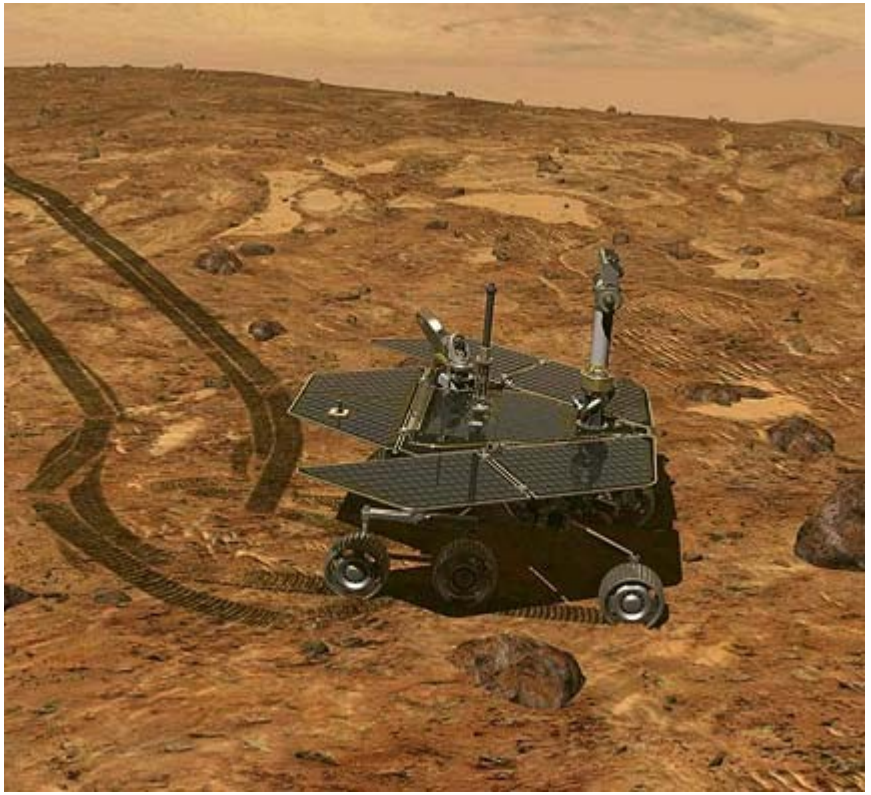
- NP-difícil
- Planejar a perfuração de placas de circuitos e de máquinas industriais em fábricas

Layout VSLI



Problema de posicionamento de células e roteamento de canais

Navegação de robôs



Espaço tridimensional

Como tornar finito o espaço de estados?

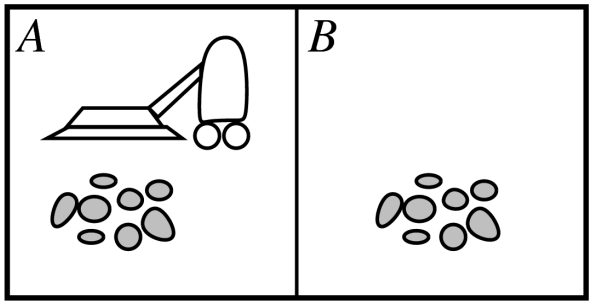
Sequência automática de montagem

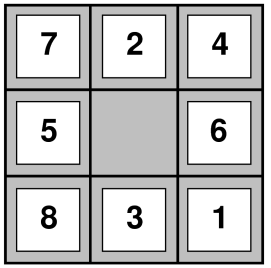
Ordem válida de montagem

Ordem ótima de montagem

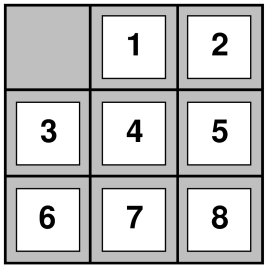
Exercício

- Modele para cada um dos miniproblemas o espaço de estados, defina use o estado inicial contido nas figuras destes slides e proponha (modele) uma solução para cada um deles.

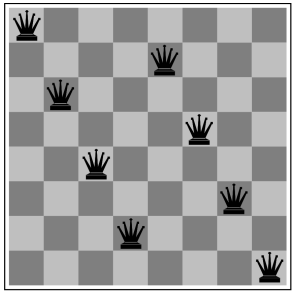




Start State

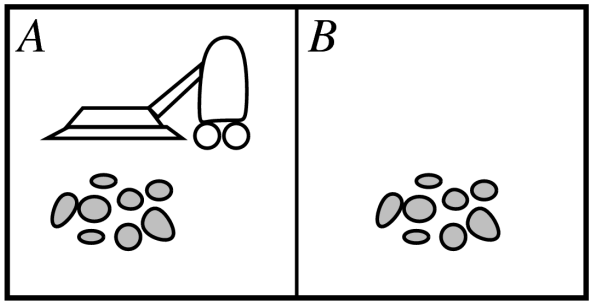


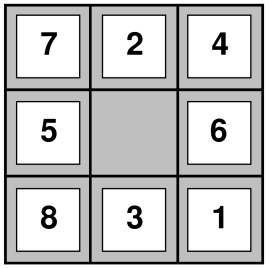
Goal State



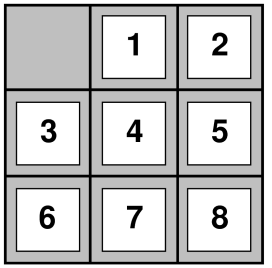
Exercício

- Considere a possibilidade de custos fixos e únicos ou fixos e diferentes, em diferentes casos

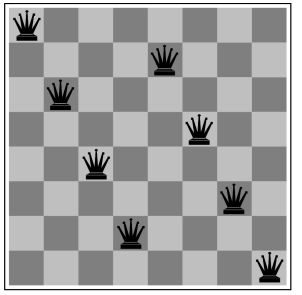




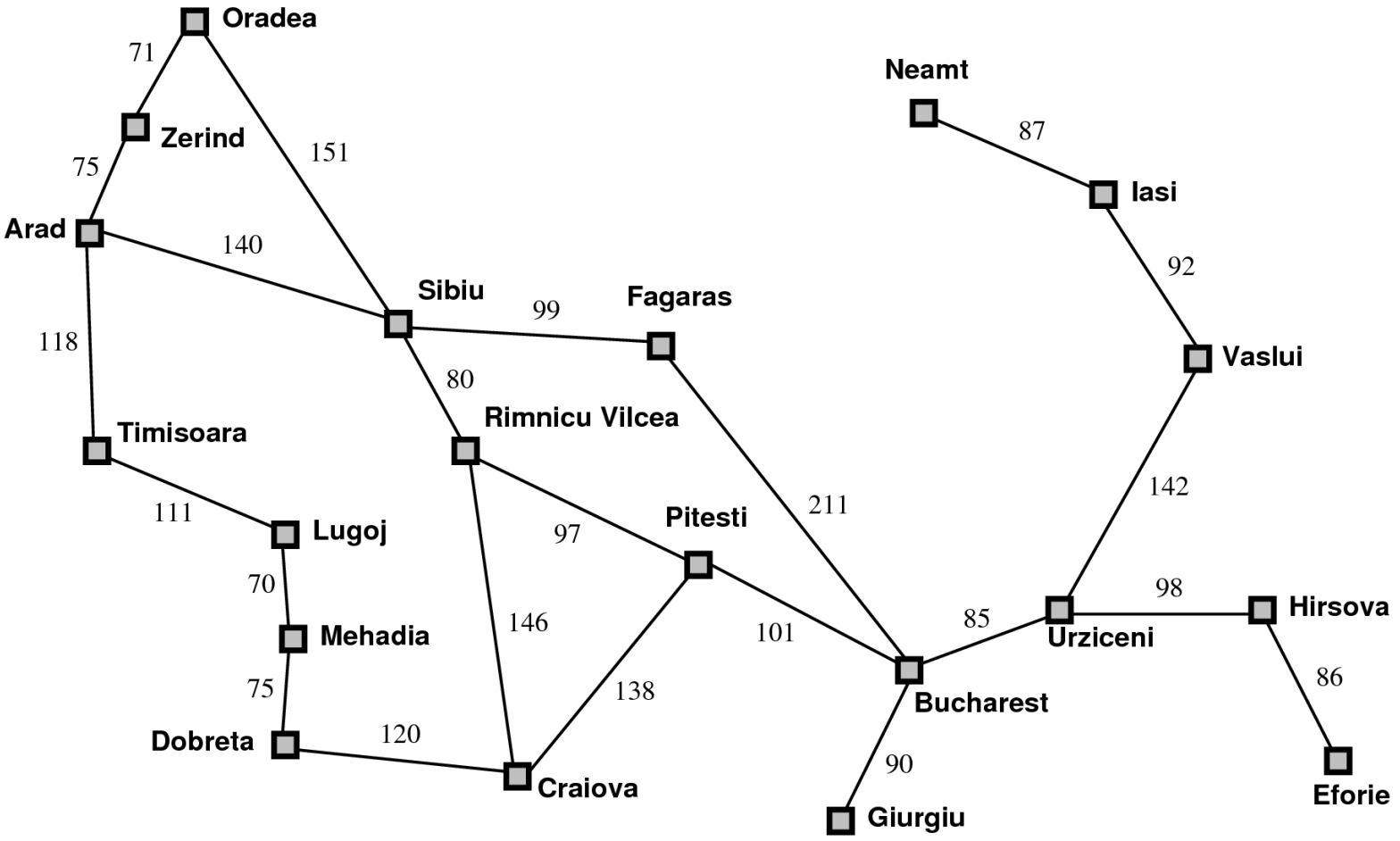
Start State



Goal State

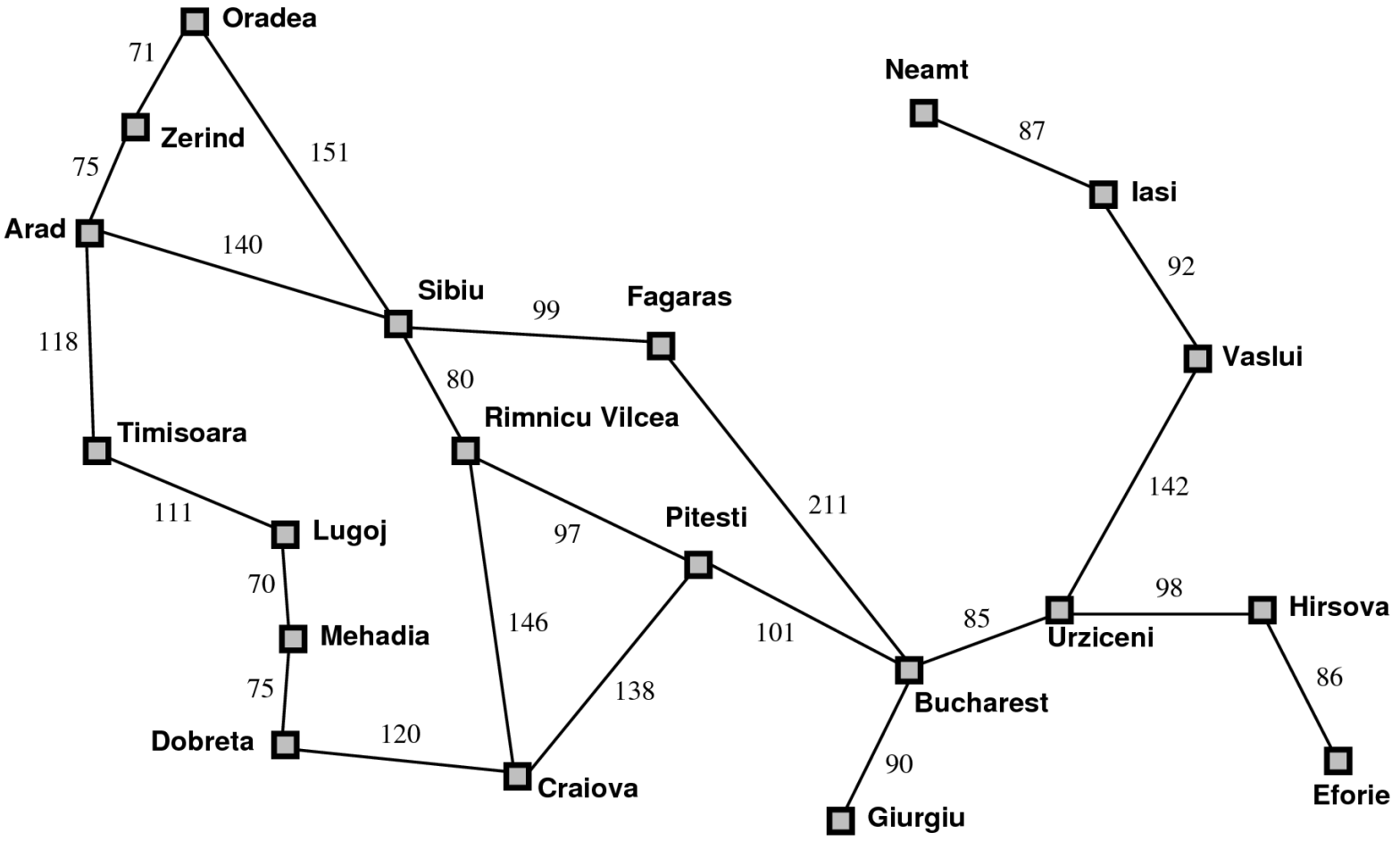


Em busca de soluções



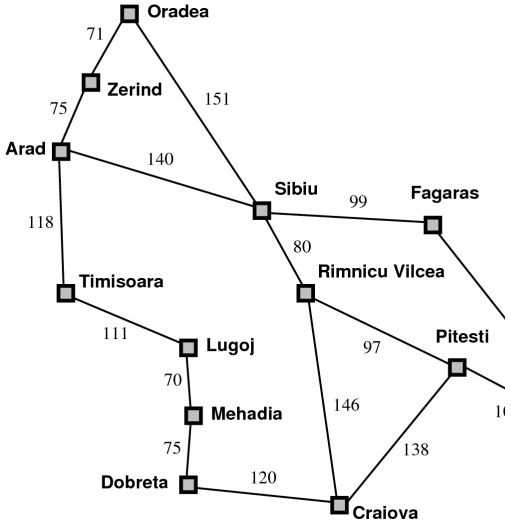
Em busca de soluções

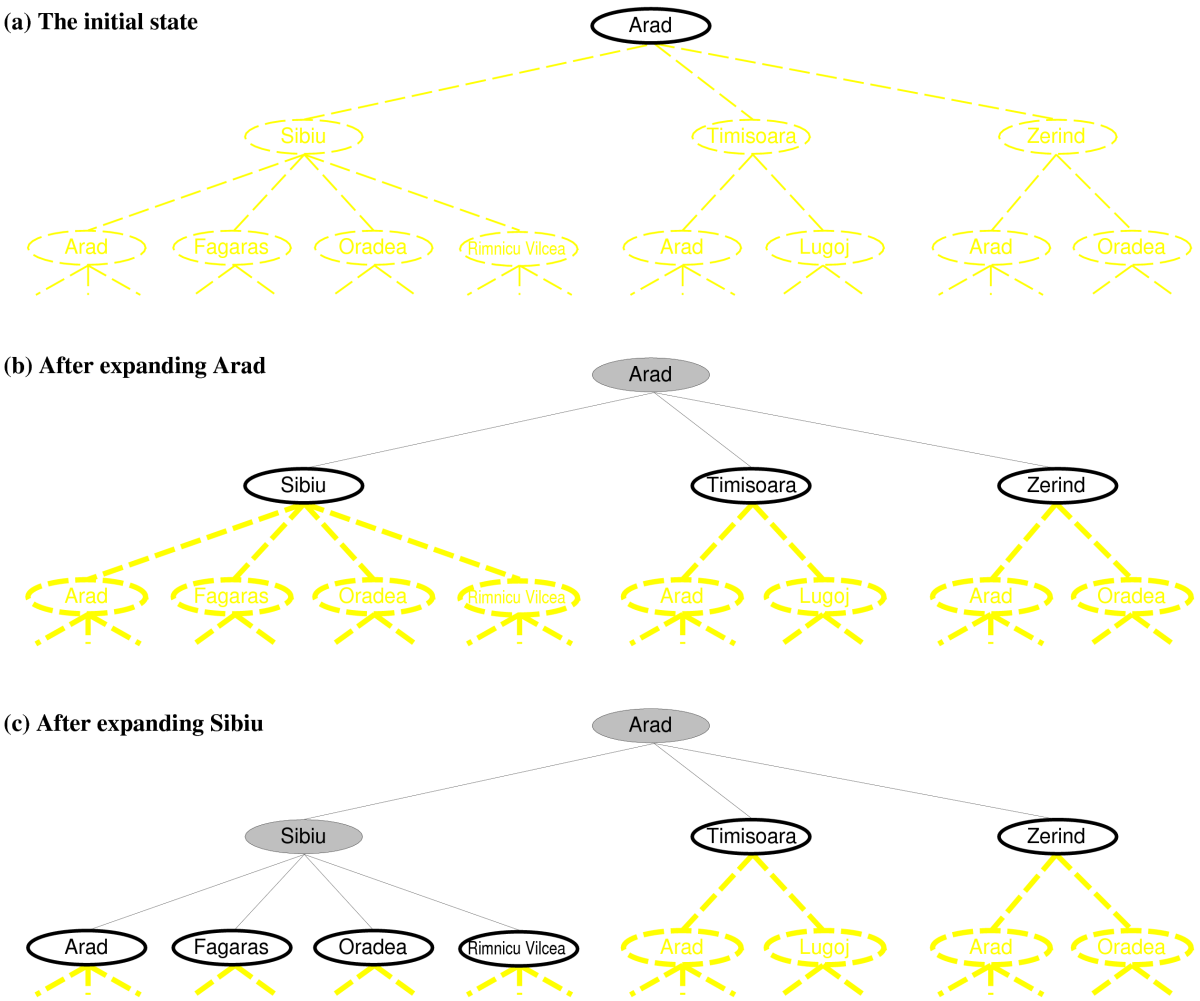
Estado inicial



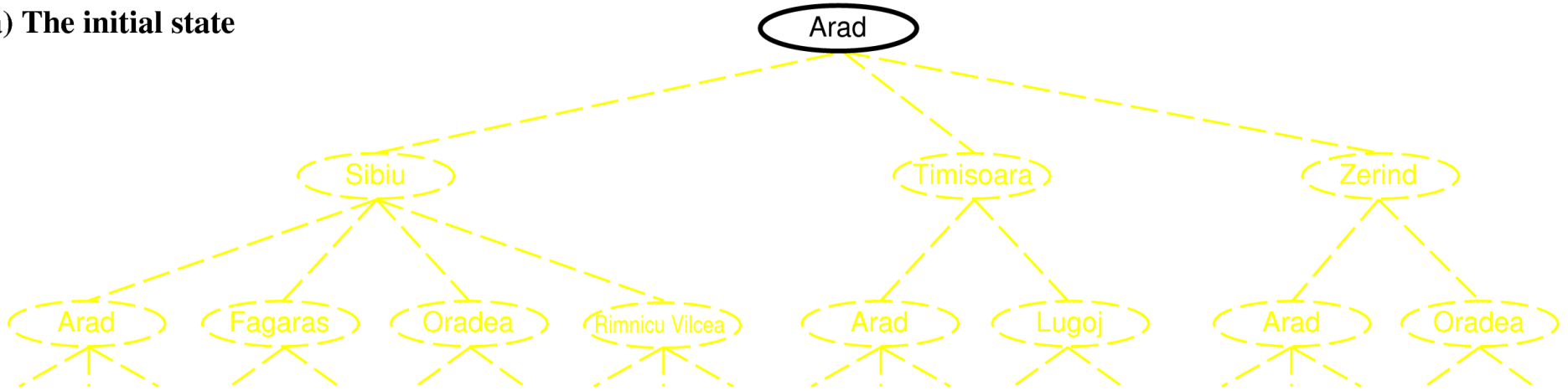
Em busca de soluções

Estado inicial

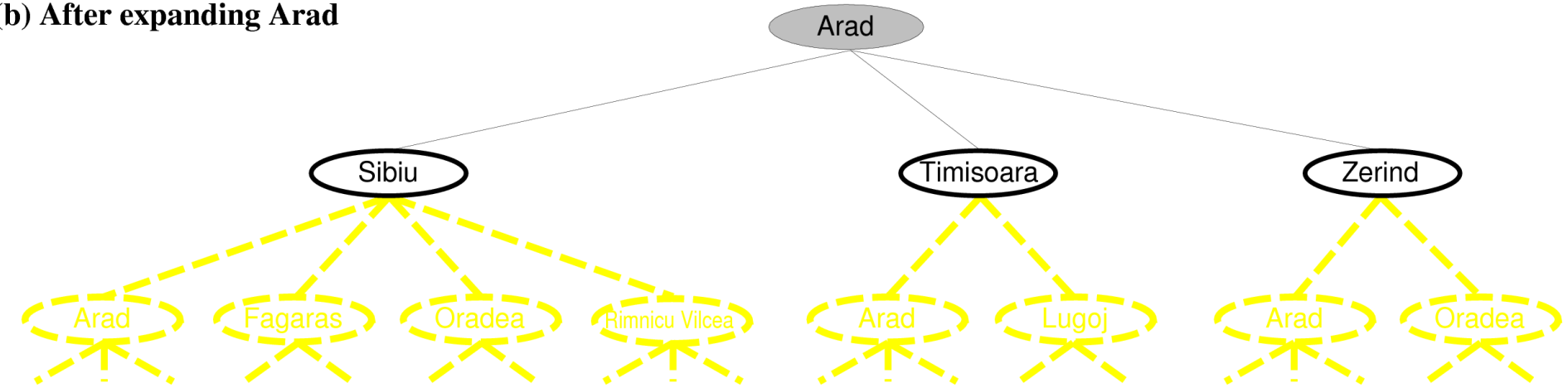




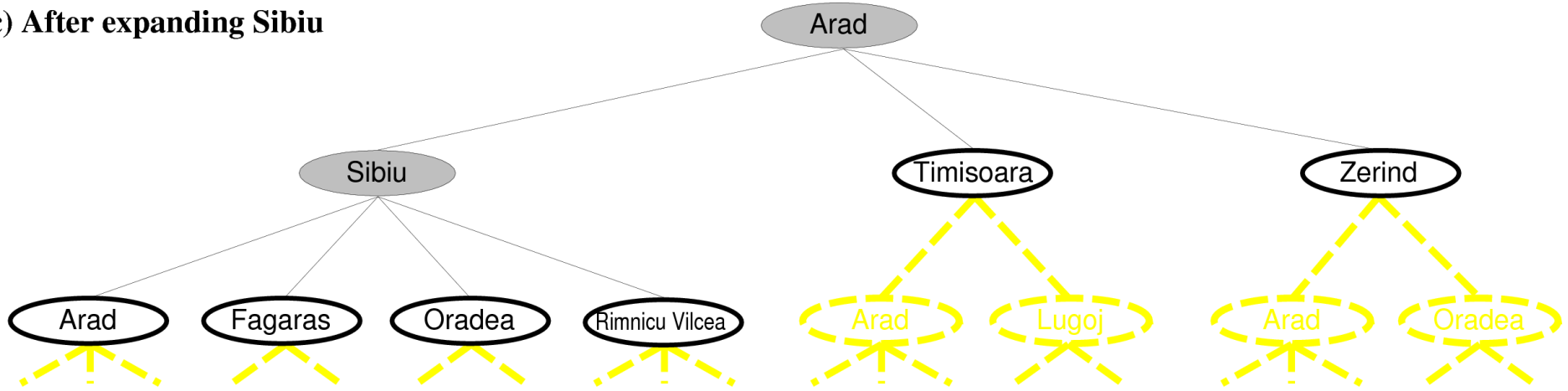
(a) The initial state

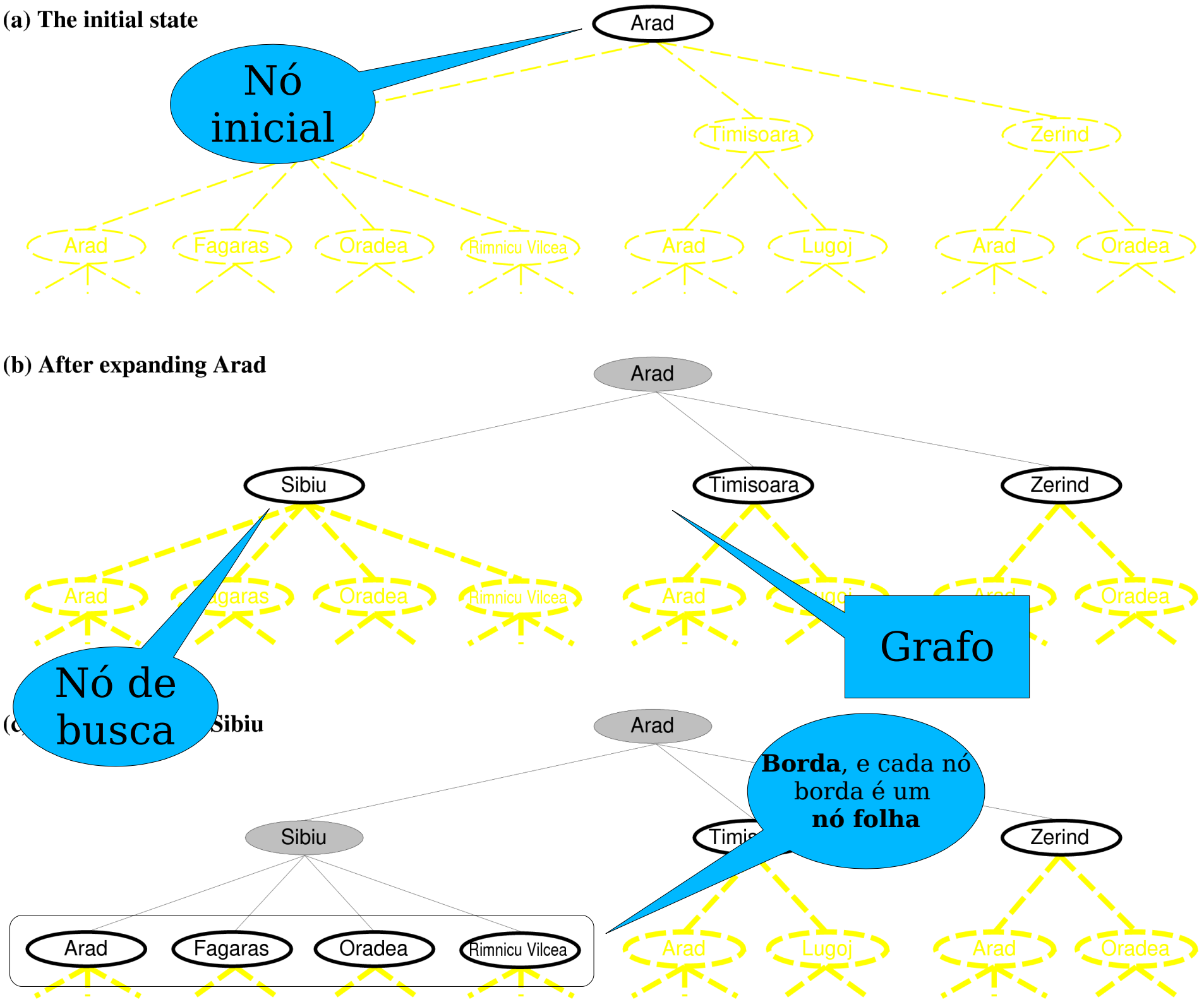


(b) After expanding Arad



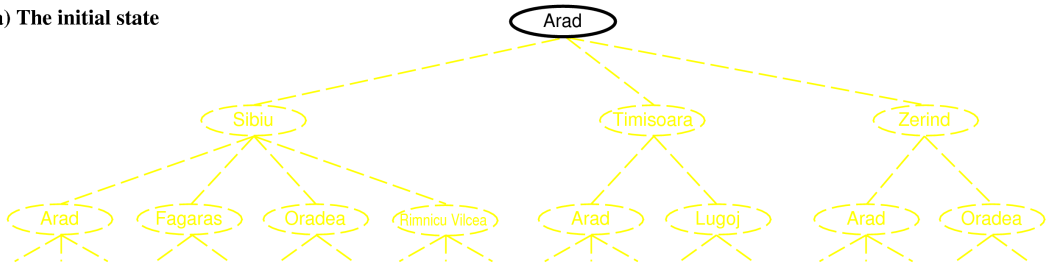
(c) After expanding Sibiu



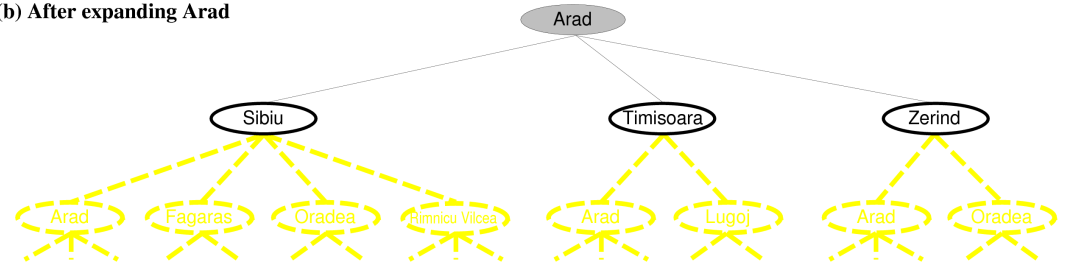

(a) The initial state
Arad
Nó inicial
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea
(b) After expanding Arad
Arad
Sibiu
Timisoara
Zerind
Arad
Oradea
Nó de busca
Grafo
Sibiu
Arad
Borda, e cada nó borda é um nó folha
Sibiu
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea

Busca em árvore

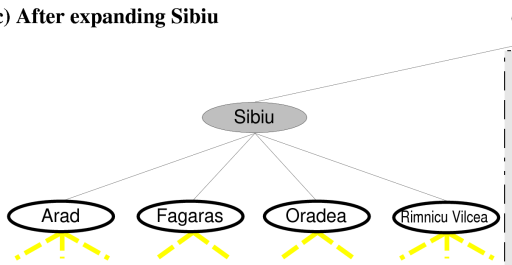
(a) The initial state



(b) After expanding Arad

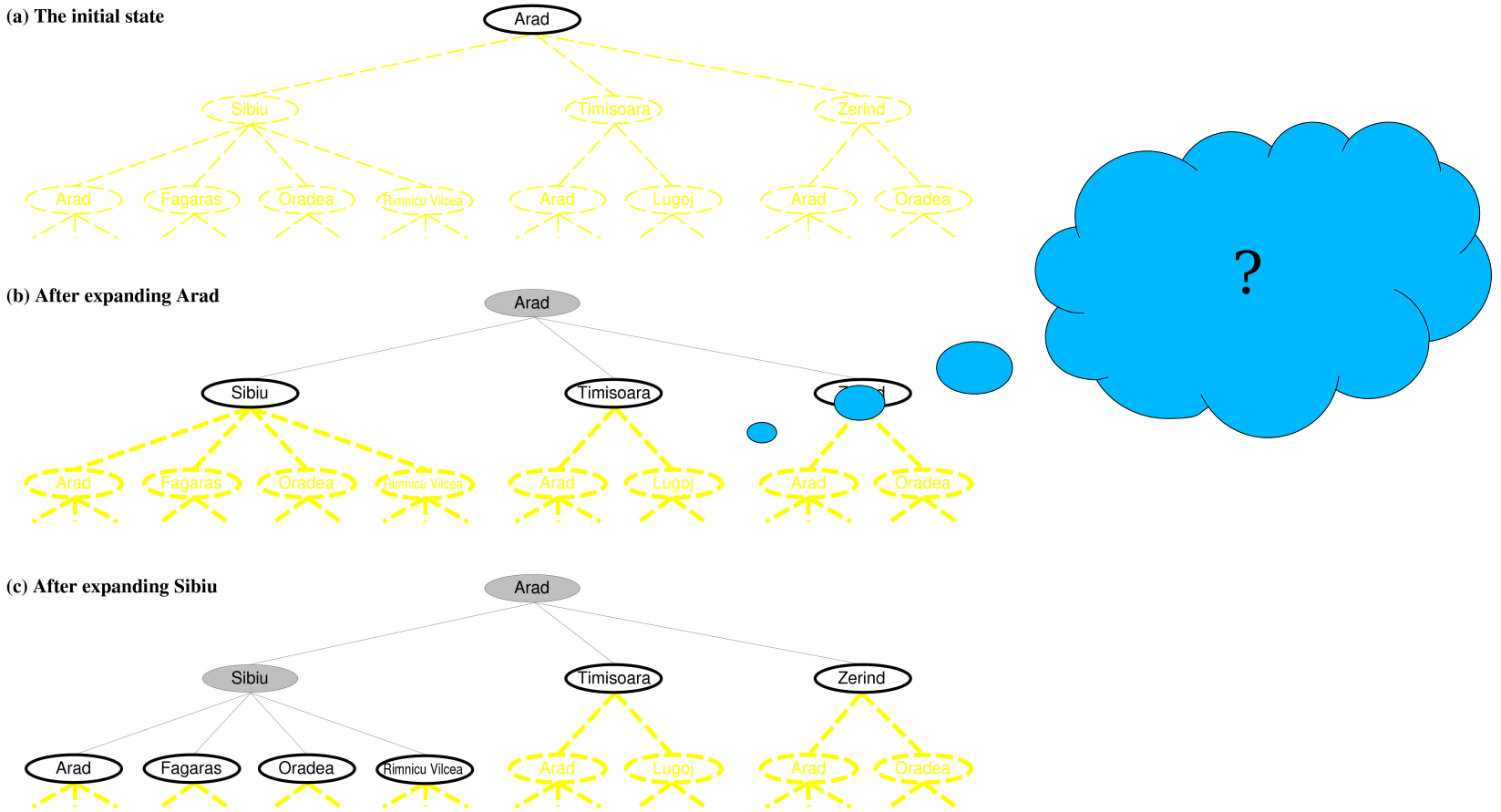


(c) After expanding Sibiu

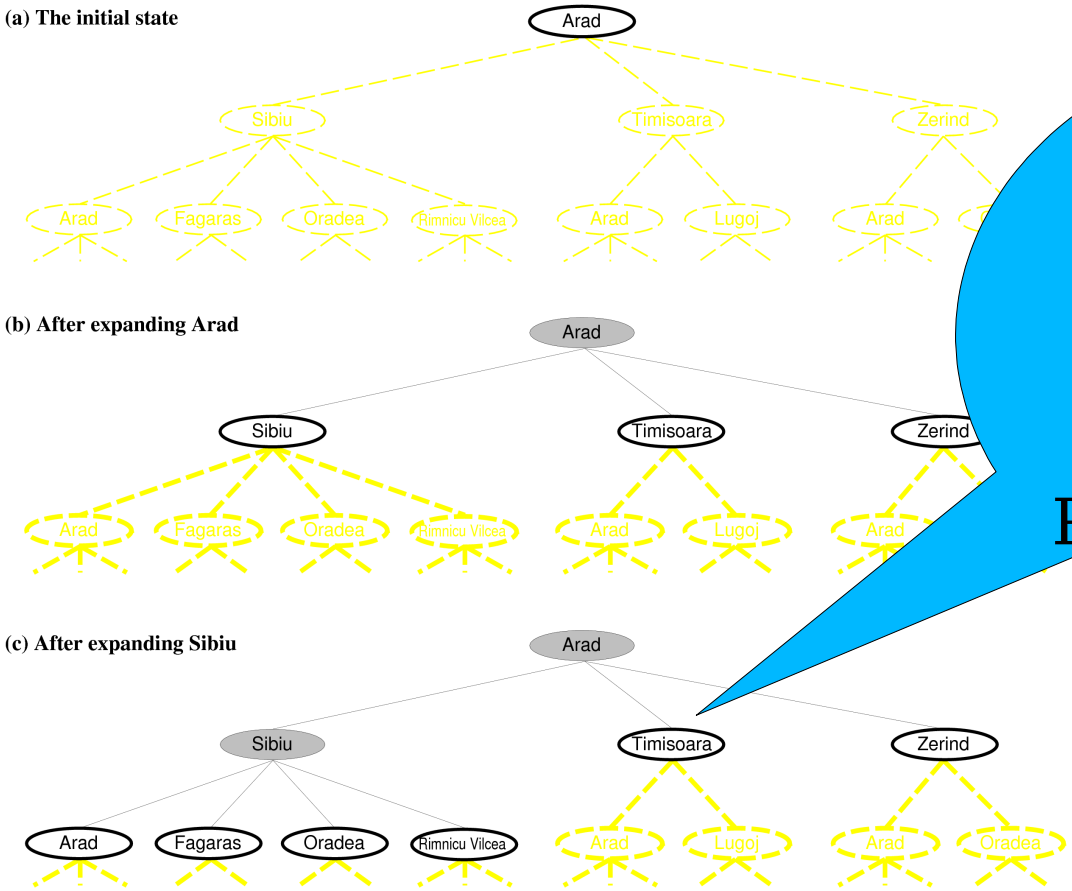


função BUSCA-EM-ÁRVORE(problema, estratégia) **retorna** uma solução ou falha
    Inicializar a árvore de busca usando o estado inicial problema
    Repita
        Se não existe nenhum candidato para expansão
        Então
            Retornar falha
        Escolher um nó folha para expansão de acordo com uma estratégia
        Se o nó contém um estado objetivo
        Então
            Retornar solução correspondente
        Senão
            Expandir o nó e adicionar os nós resultantes à árvore de busca

Estratégia de busca



Estratégia de busca



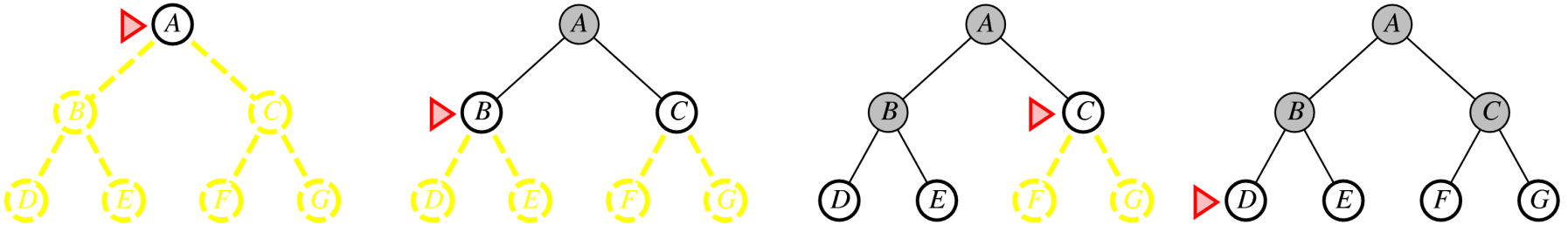
Estrutura de nó:
ESTADO
NÓ-Pai
AÇÃO(arco)
CUSTO
PROFUNDIDADE

Estratégia de busca

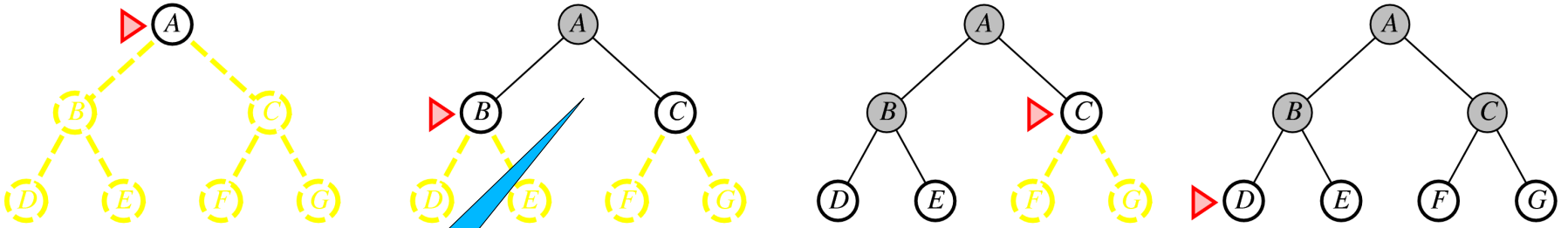
- Busca em extensão
- Busca em profundidade
- Busca em profundidade limitada
- Busca de aprofundamento iterativo em profundidade
- Busca bidirecional

  Obs.: sem informação

Busca em extensão



Busca em extensão

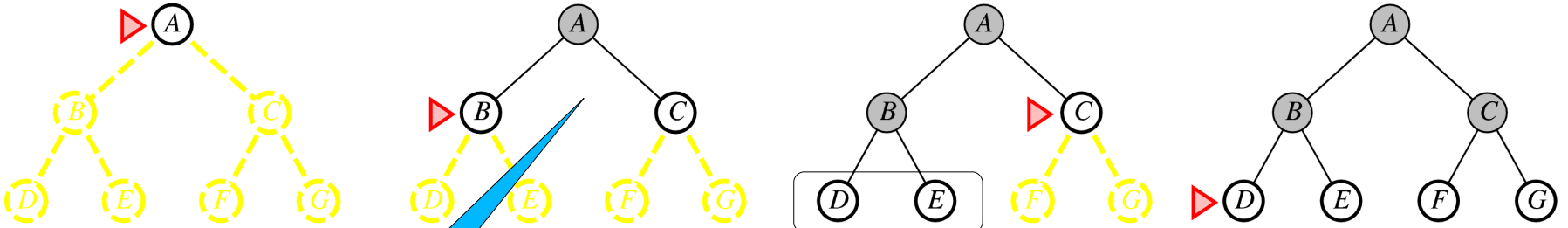


Árvore binária simples

Mantido em memória até o uso, pois:
1) expande-se o nó
2) verifica-se o nó

BUSCA em ÁRVORE com FIFO

Busca em extensão



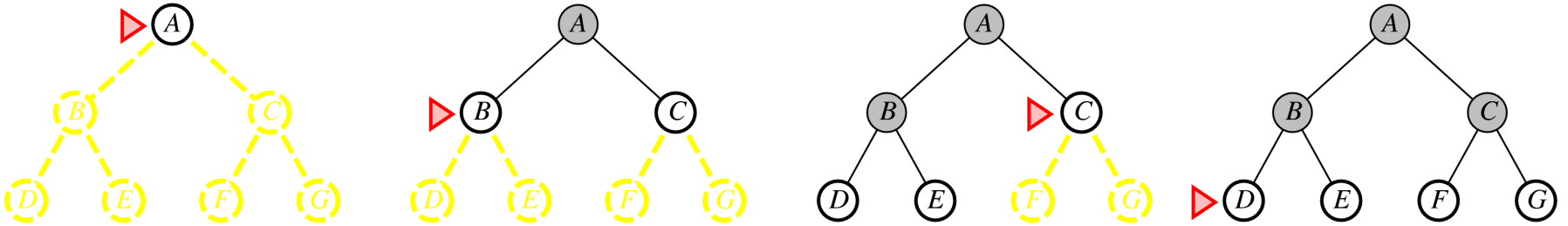
Árvore binária simples

FIFO

Borda, e cada nó borda é um **nó folha**

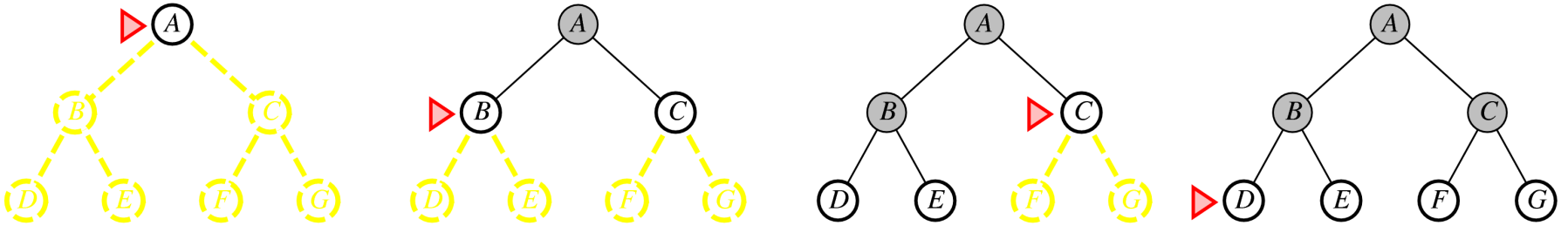
Mantido em memória até o uso, pois:
1) expande-se o nó
2) verifica-se o nó

Busca em extensão



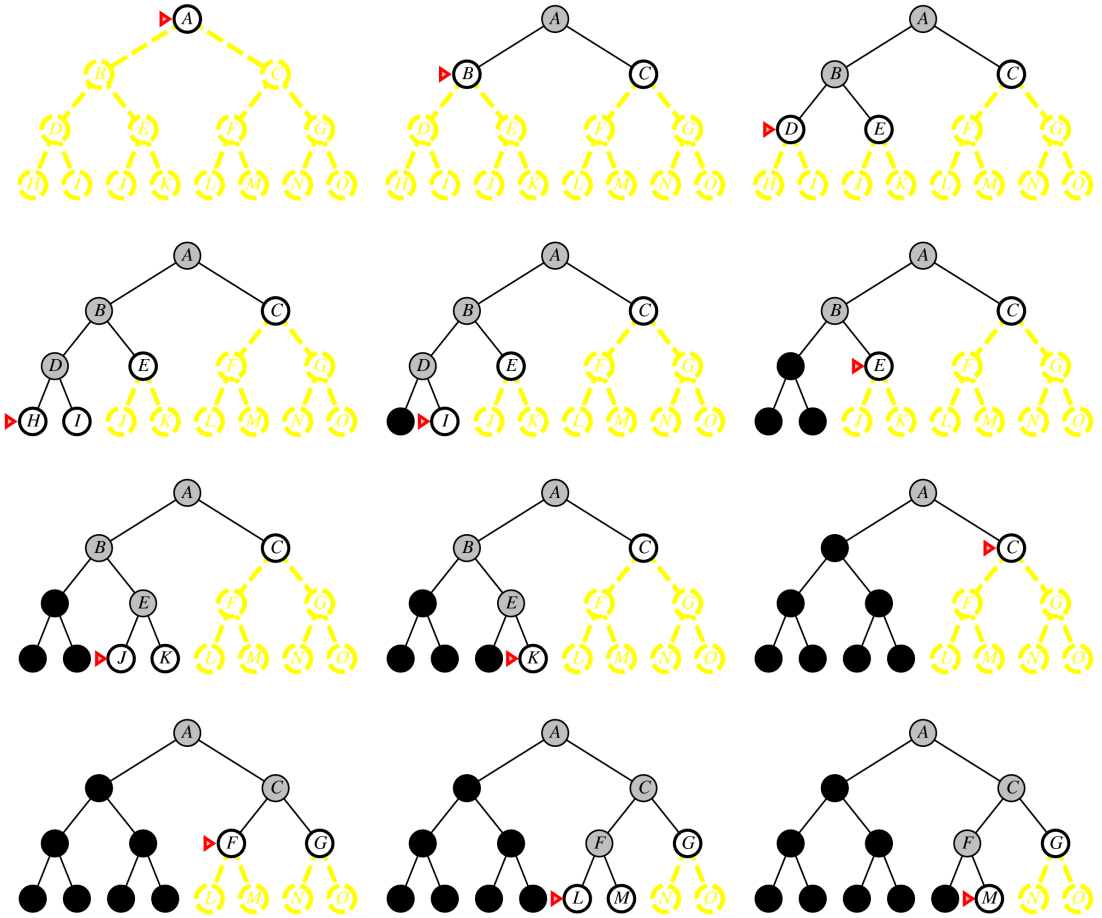
Profundidade	Nós	Tempo	Memória
2	1100	0,11 seg	1 MB
4	111.100	11 seg	106 MB
6	10^7	19 min	10 GB
7	10^9	31 horas	1TB
10	10^{11}	129 dias	101 TB
12	10^{13}	35 anos	10 PetaB
14	10^{15}	3.523 anos	1 Exabyte

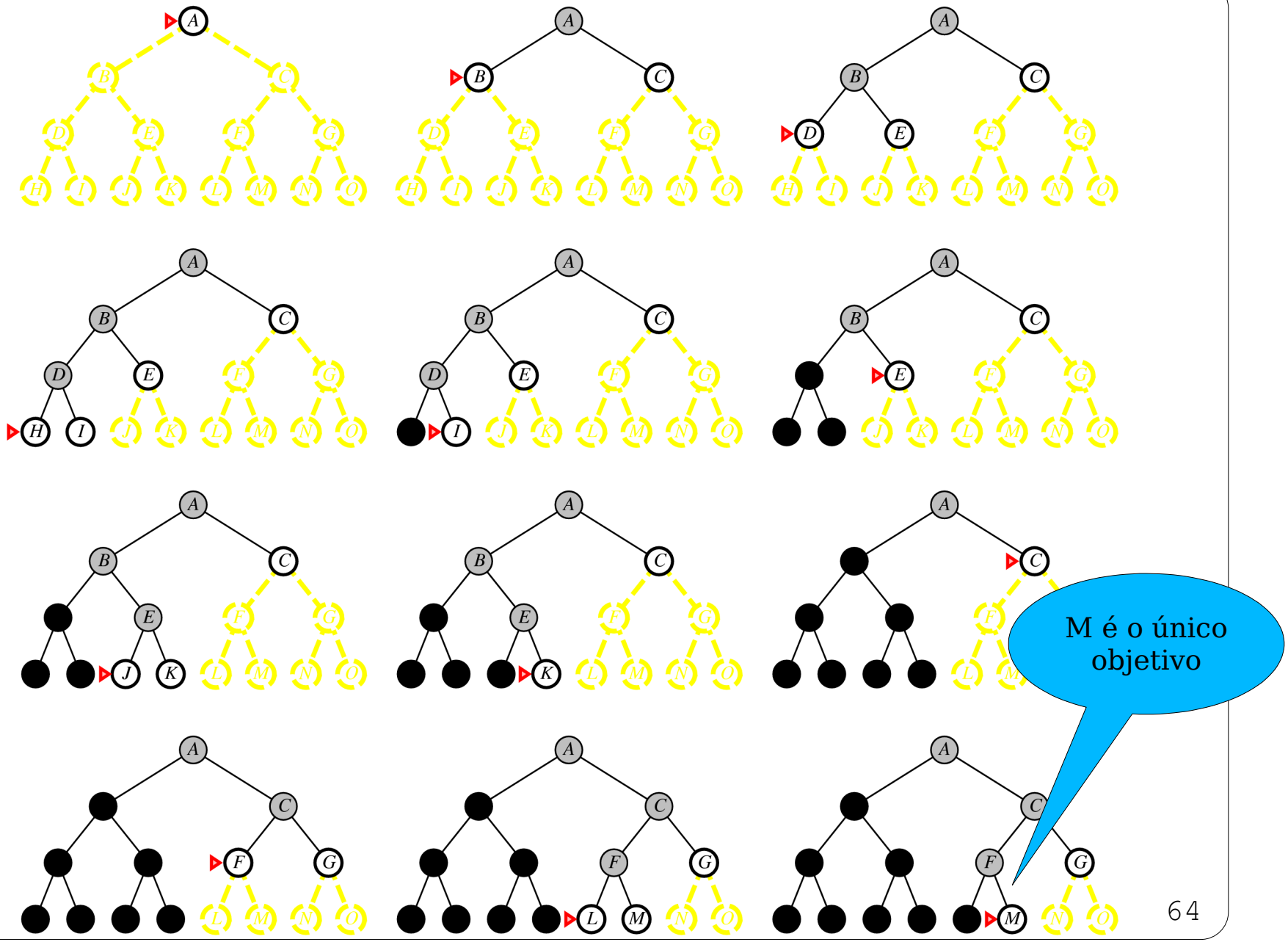
Busca em extensão

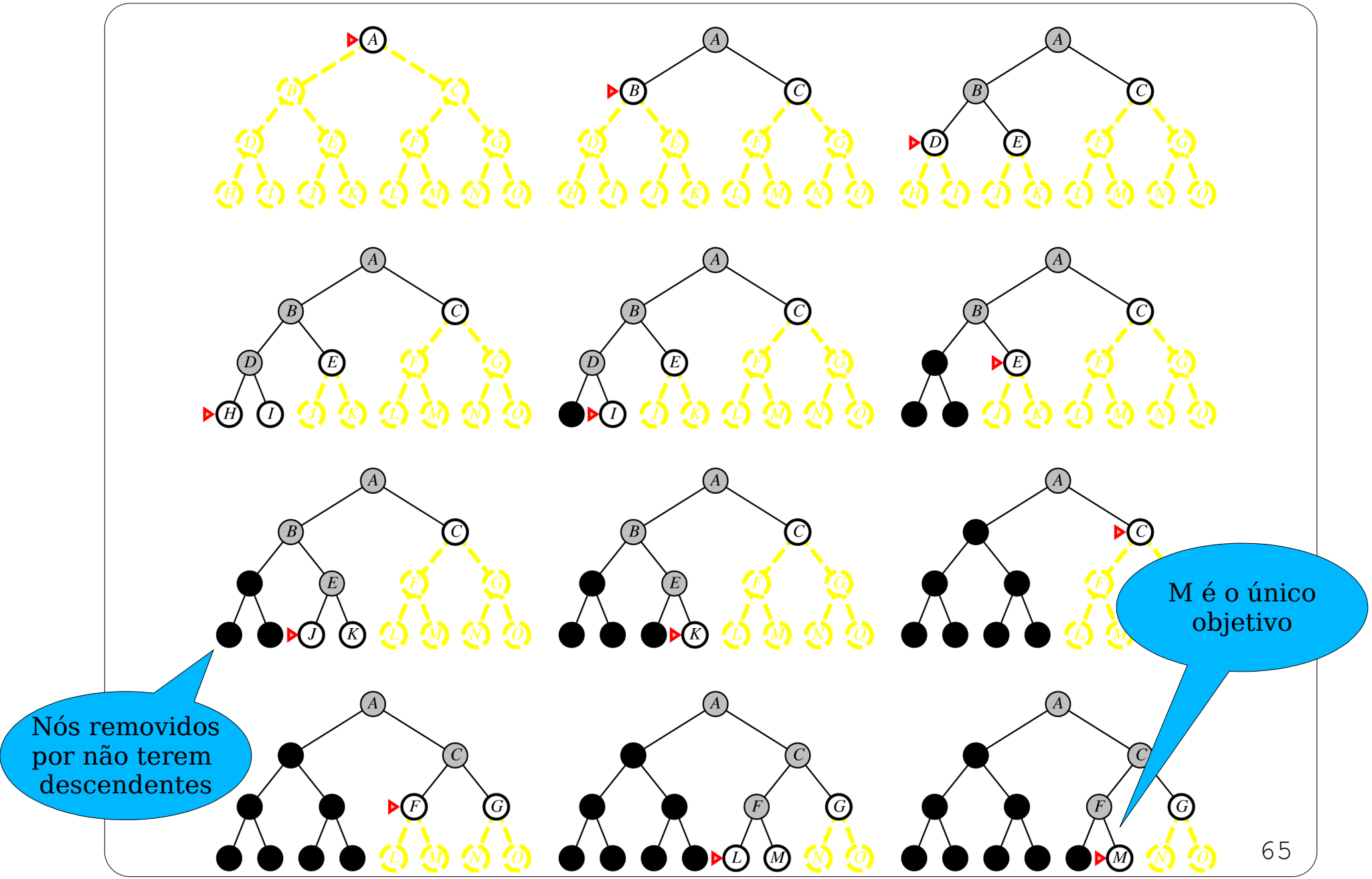


Profundidade	Nós	Tempo	Memória
2	1100	0,11 seg	1 MB
4	111.100	11 seg	106 MB
6	10^7	19 min	10 GB
7	10^9	31 horas	1TB
10	10^{11}	129 dias	101 TB
12	10^{13}	35 anos	10 PetaB
14	10^{15}	3.523 anos	1 Exabyte

Busca em profundidade



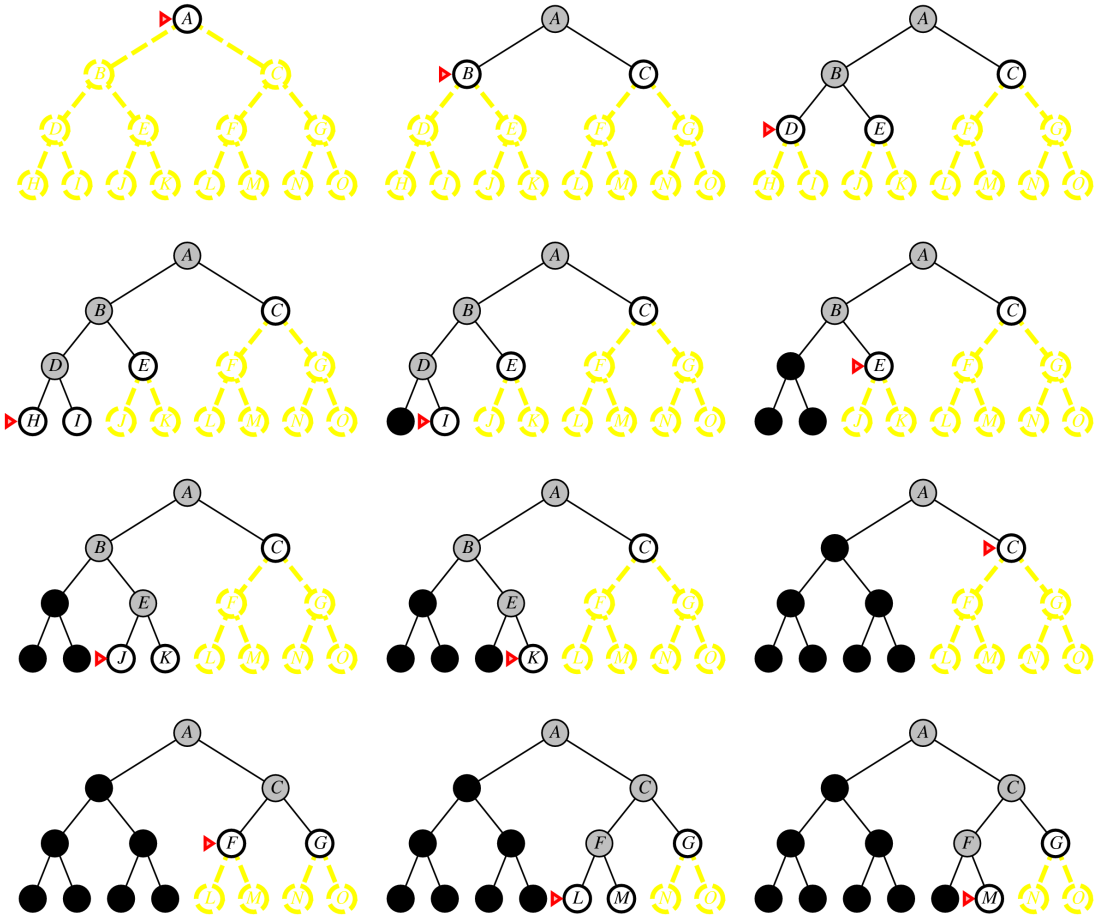

M é o único objetivo


Nós removidos por não terem descendentes
M é o único objetivo

Busca em profundidade limitada

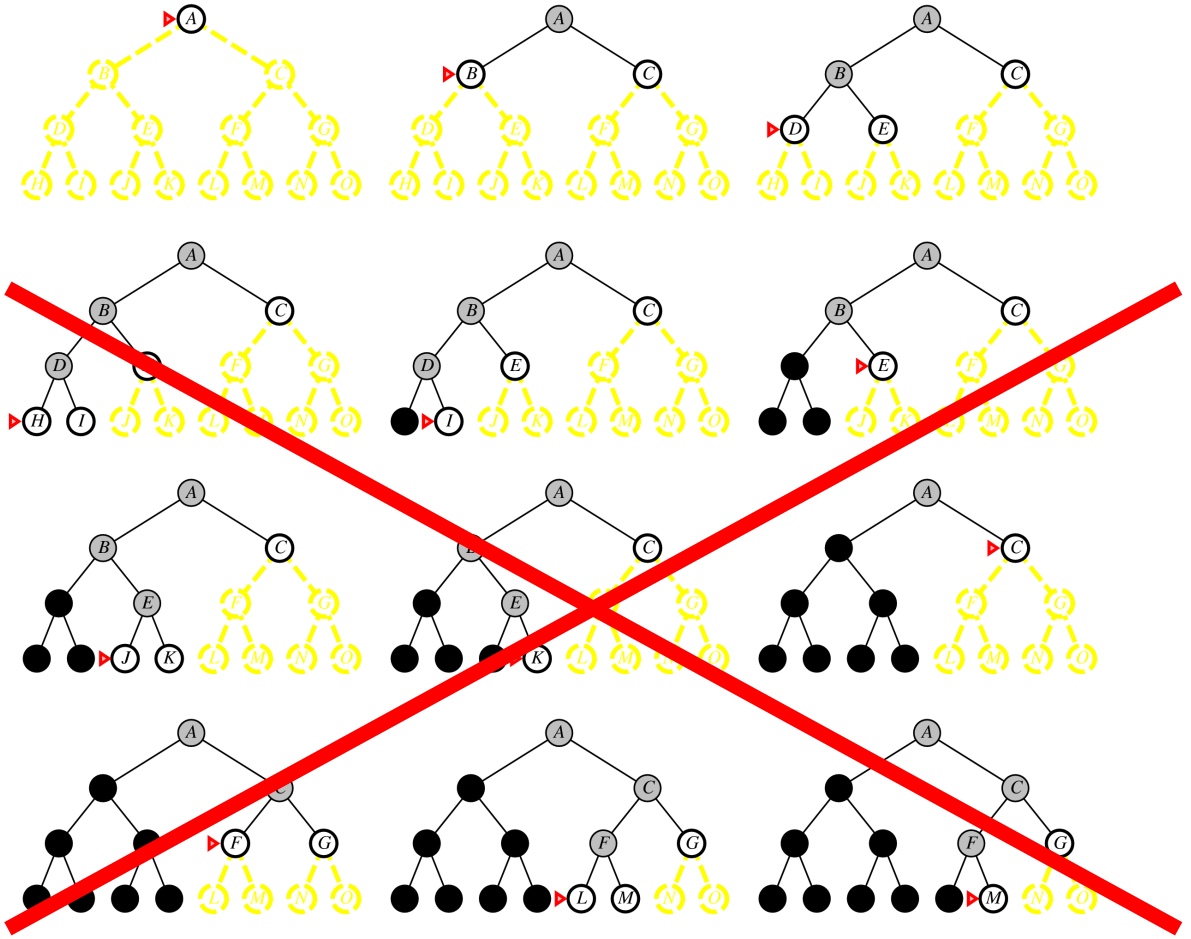
Busca em profundidade limitada

l = 2



Busca em profundidade limitada

l = 2



Útil quando se sabe algo do problema ... Ex.: mapa vs. Número de cidades

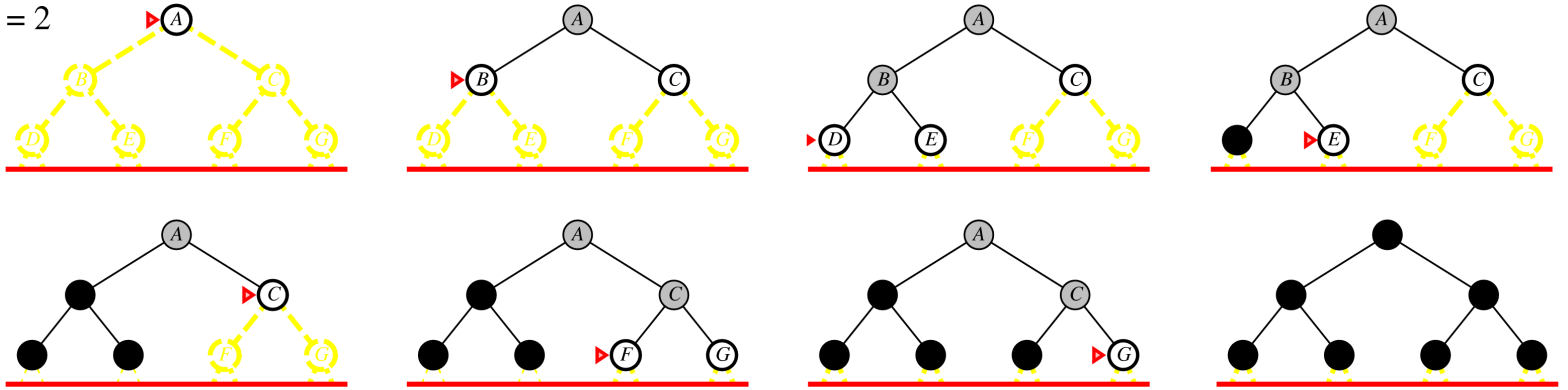
Busca de aprofundamento iterativo em profundidade

- Também conhecido como *busca em profundidade por aprofundamento interativo*
- O **limite vai sendo aumentado** gradualmente, começando em zero e depois 1, e assim por diante
- É **mais rápido** que a busca em extensão mesmo com a geração repetida de estados (maioria do nós estão em níveis inferiores)

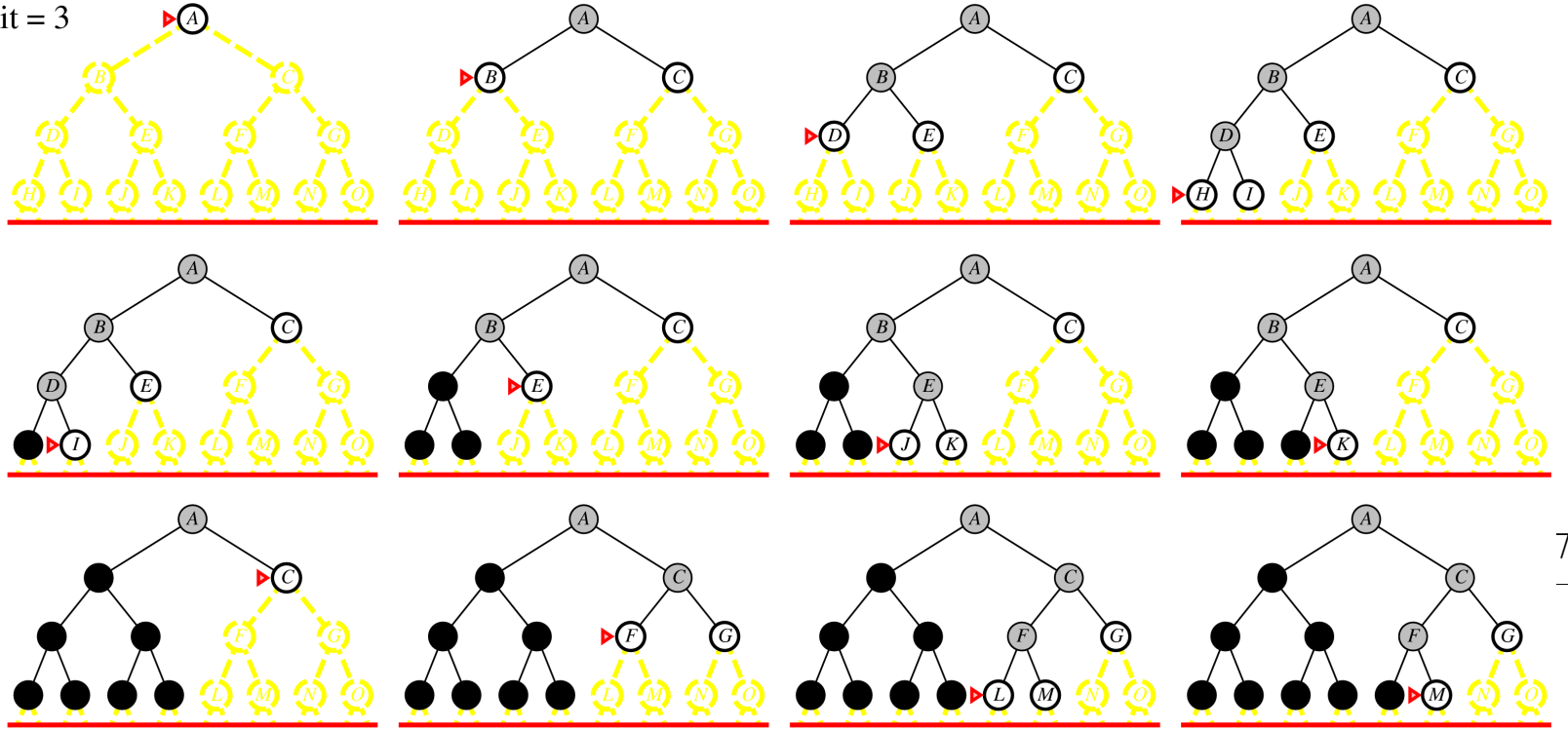
Limit = 0

Limit = 1

Limit = 2

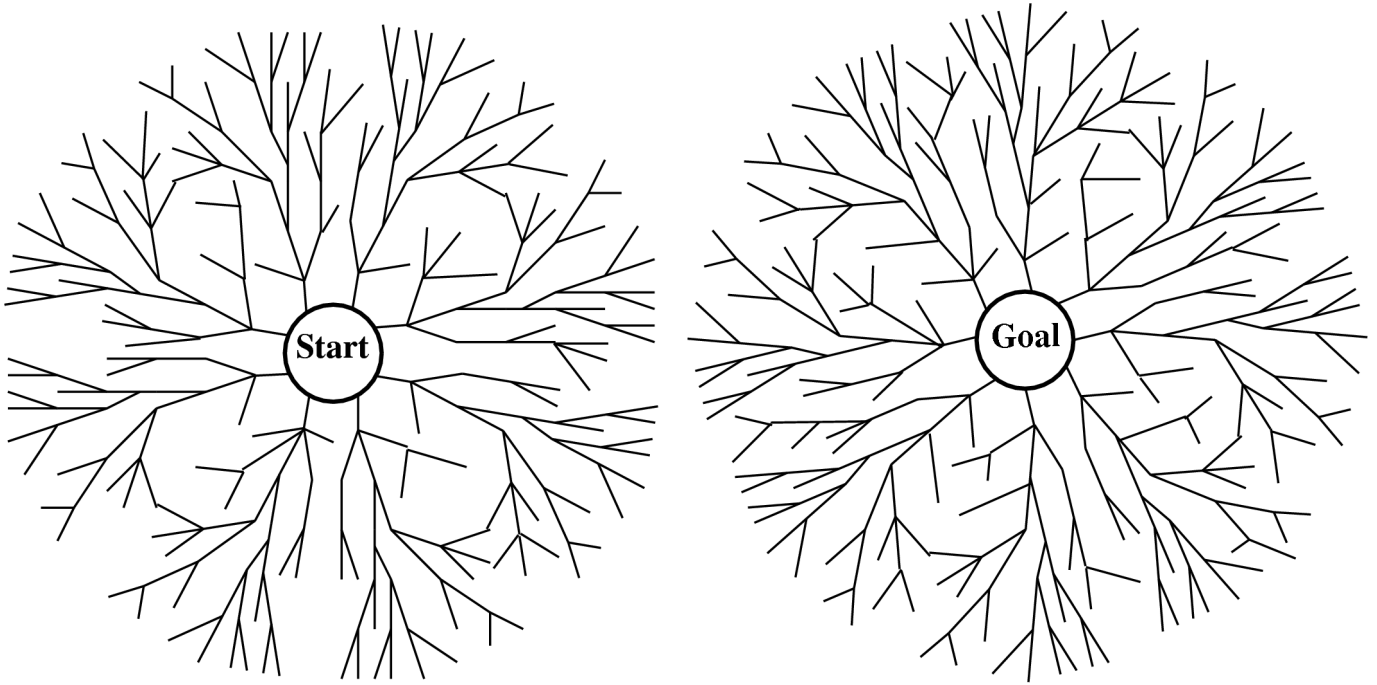


Limit = 3



Busca bidirecional

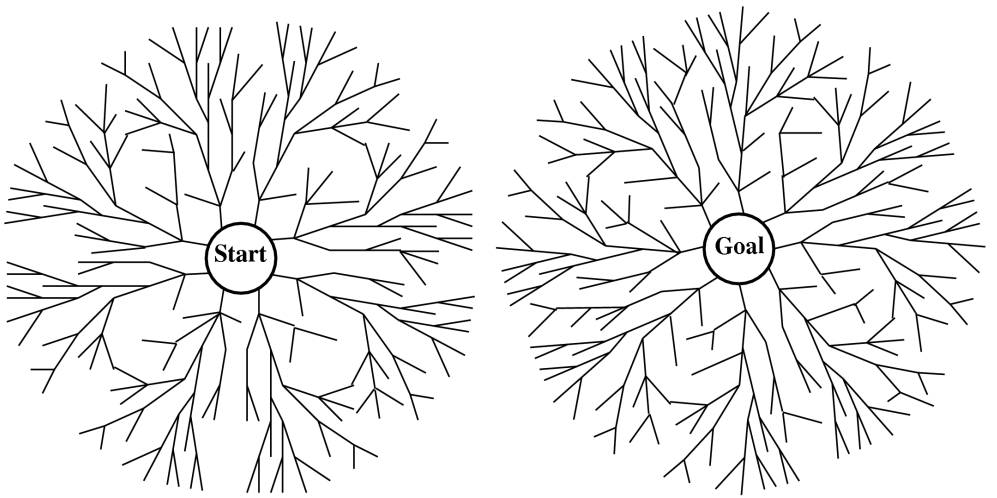
- Executa duas buscas simultâneas
  - Uma a partir do nó inicial
  - Outra a partir do objetivo



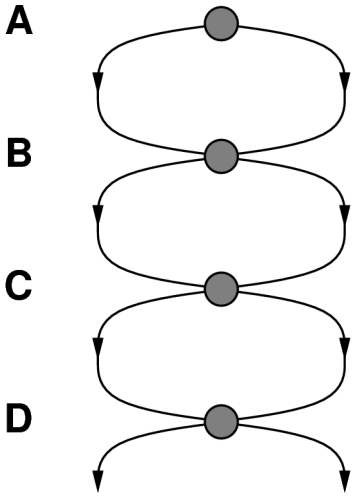
Busca bidirecional

- Executa duas buscas simultâneas
  - Uma a partir do nó inicial
  - Outra a partir do objetivo
- Verifica-se o nó antes de expandi-lo

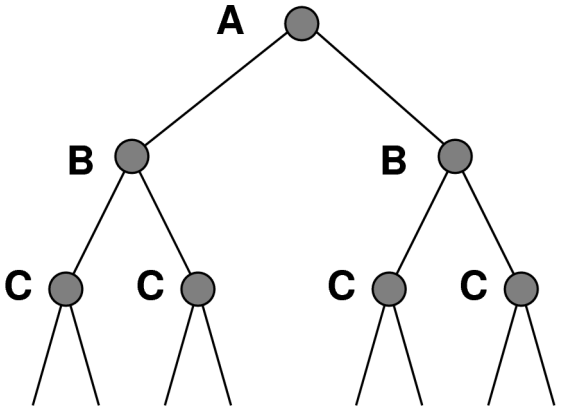
- Problemas com loopings
- Ajuda quando de tem múltiplos objetivos (evita-se nós já avaliados



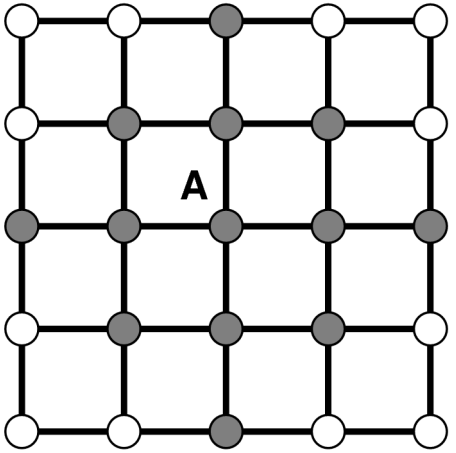
Evitando estados repetidos



(a)

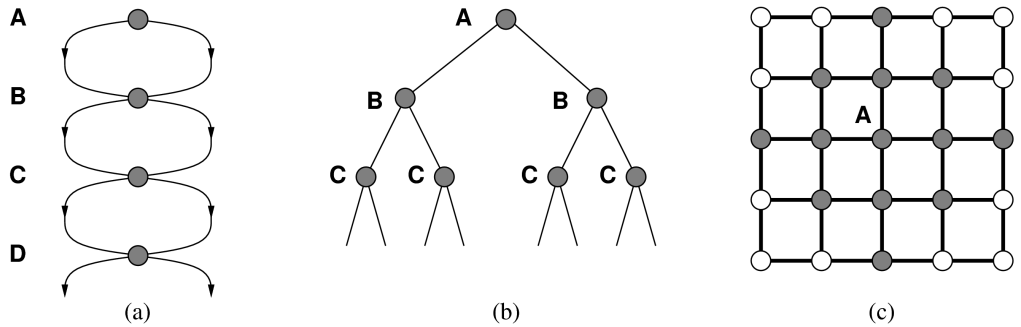


(b)



(c)

Evitando estados repetidos




```
função BUSCA-EM-GRAFO(problema, estratégia) retorna uma solução ou falha
    fechado ← um conjunto vazio
    borda ← INSERIR(CRIAR-NÓ(ESTADO-INCIAL[problema], borda)
    Repita
        Se VAZIA?(borda)
        Então
            Retornar falha
        nó ← REMOVER-PRIMEIRO(borda)
        Se TESTAR-OBJETIVO[problema](ESTADO[nó]
        Então
            Retornar SOLUÇÃO(nó)
        Se ESTADO[nó] não está em fechado
        Então
            Adicionar ESTADO[nó] a fechado
            Borda ← INSERIR-TODOS(EXPANDIR(nó, problema), borda)
```

Busca com informação e exploração (heurística)

Busca em árvore

Busca em grafo

Busca pela melhor escolha

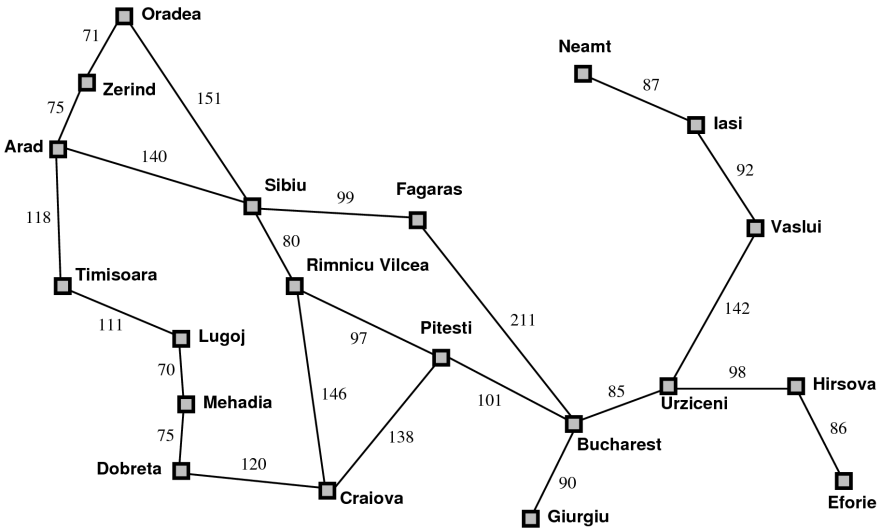
função de avaliação *f(n)*

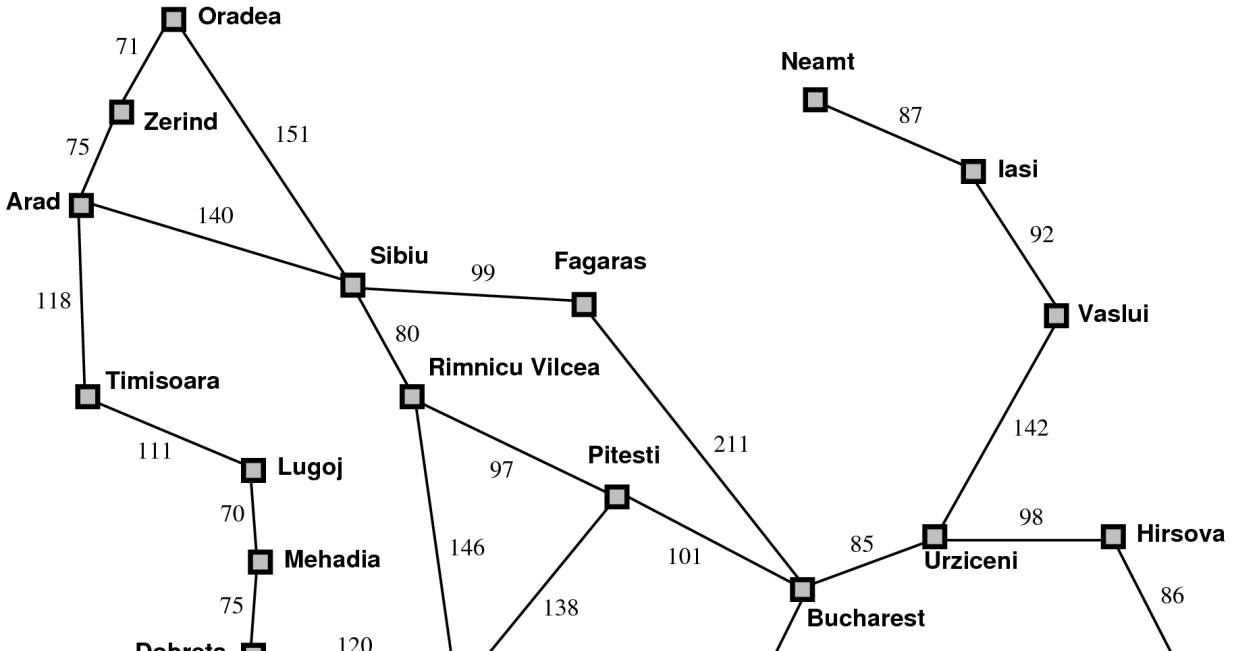
função heurística *h(n)*

- Busca gulosa pela melhor escolha
- A*

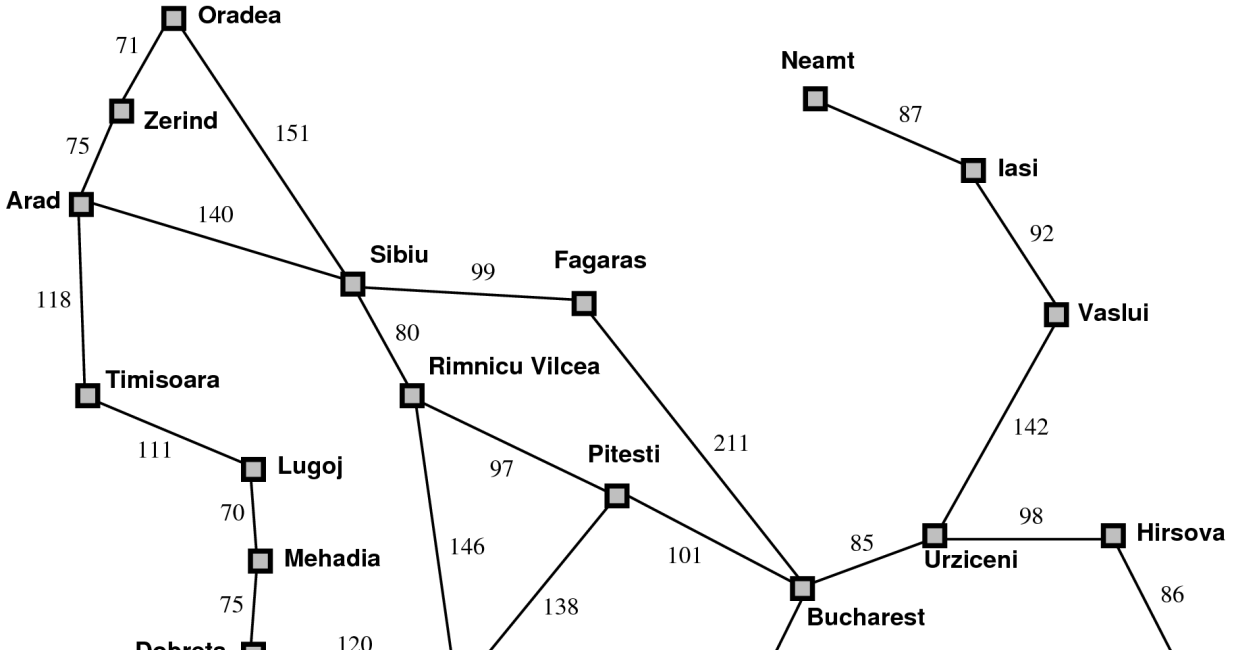
Busca gulosa pela melhor escolha

- $f(n) = h(n)$
- h(n): exemplo do mapa da Romênia
  - H: **distância em linha reta** h_{DLR}
  - h_{DLR}(em(Arad)) = 366



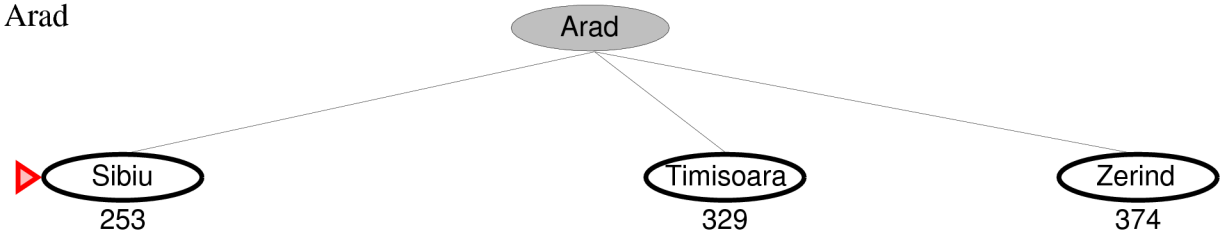

Oradea
71
Neamt
87
Zerind
151
Iasi
75
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
Pitesti
211
Lugoj
97
98
Hirsova
70
146
85
Urziceni
Mehadia
101
86
75
138
Bucharest
120

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

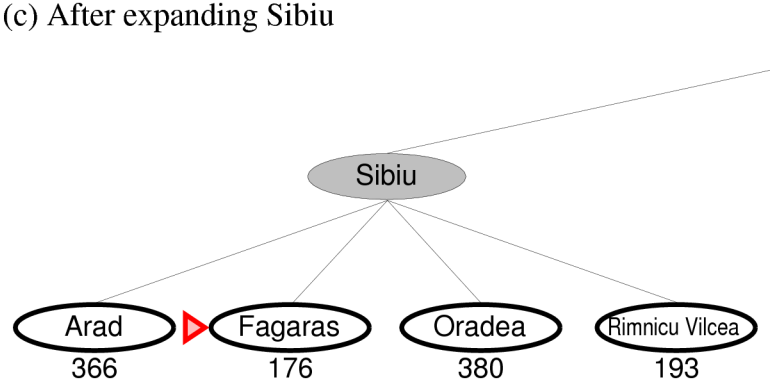


(a) The initial state

(b) After expanding Arad

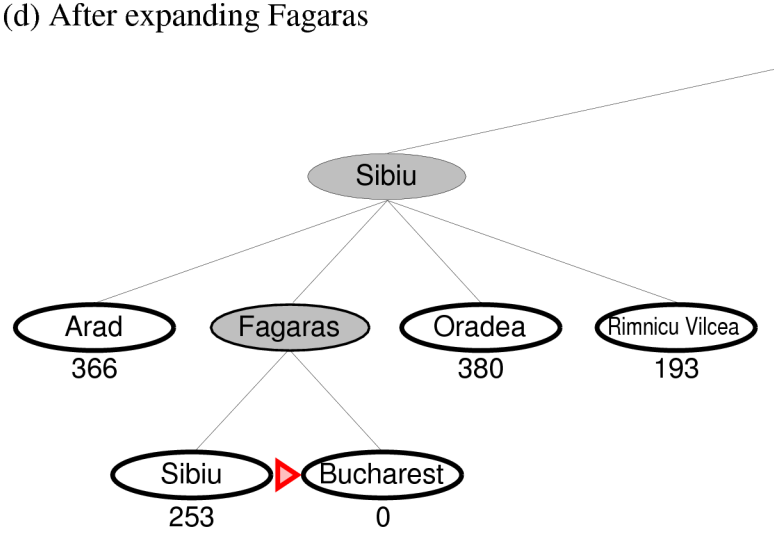


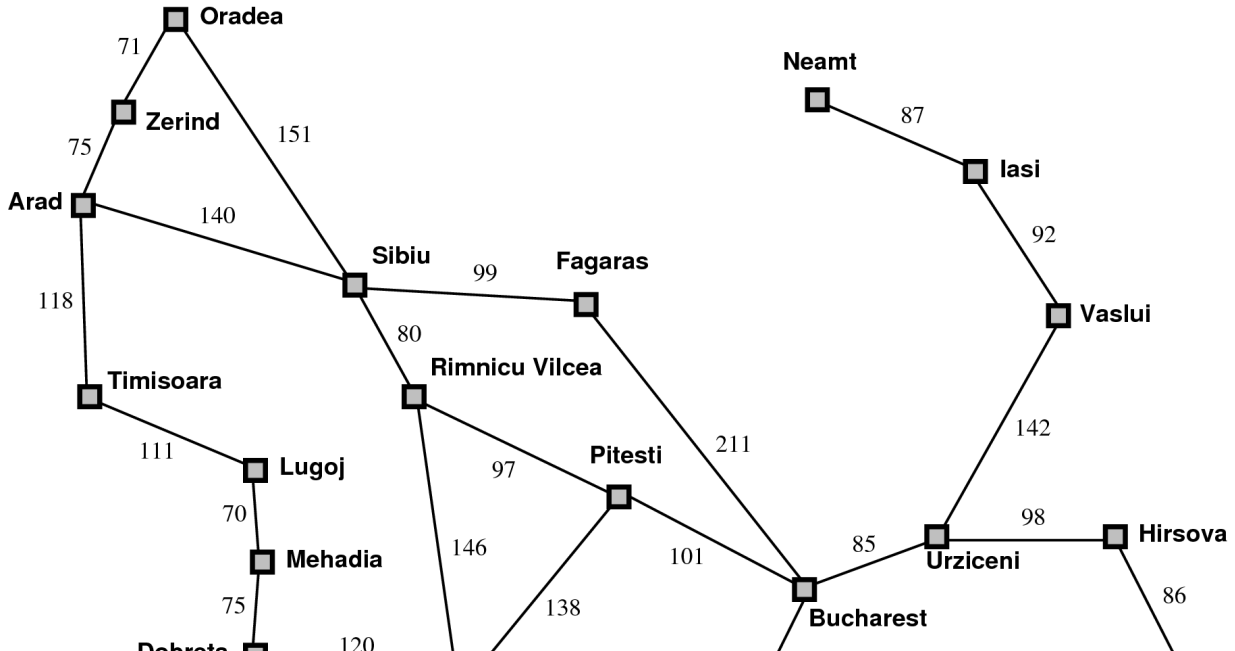
(c) After expanding Sibiu



Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

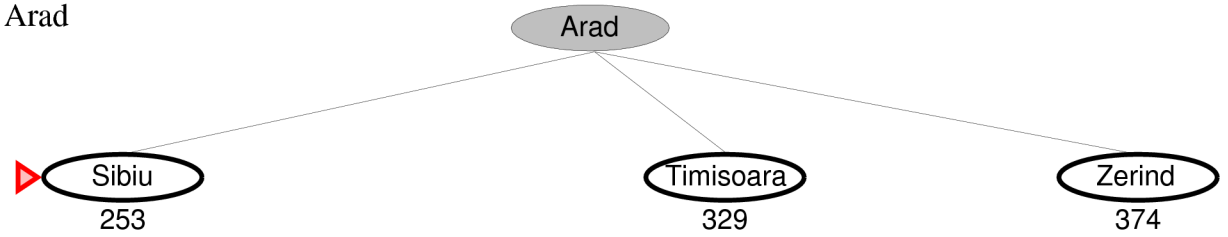
(d) After expanding Fagaras





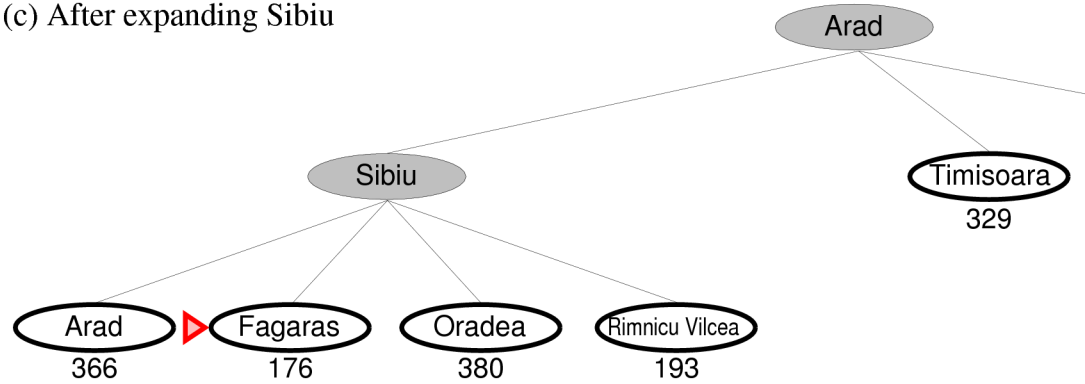
(a) The initial state

(b) After expanding Arad



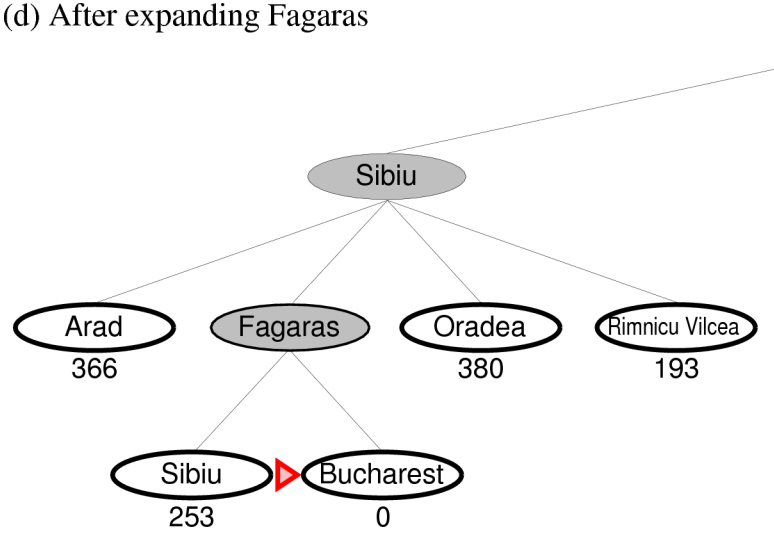
E se o início for *Iasi* e o destino Fagaras?

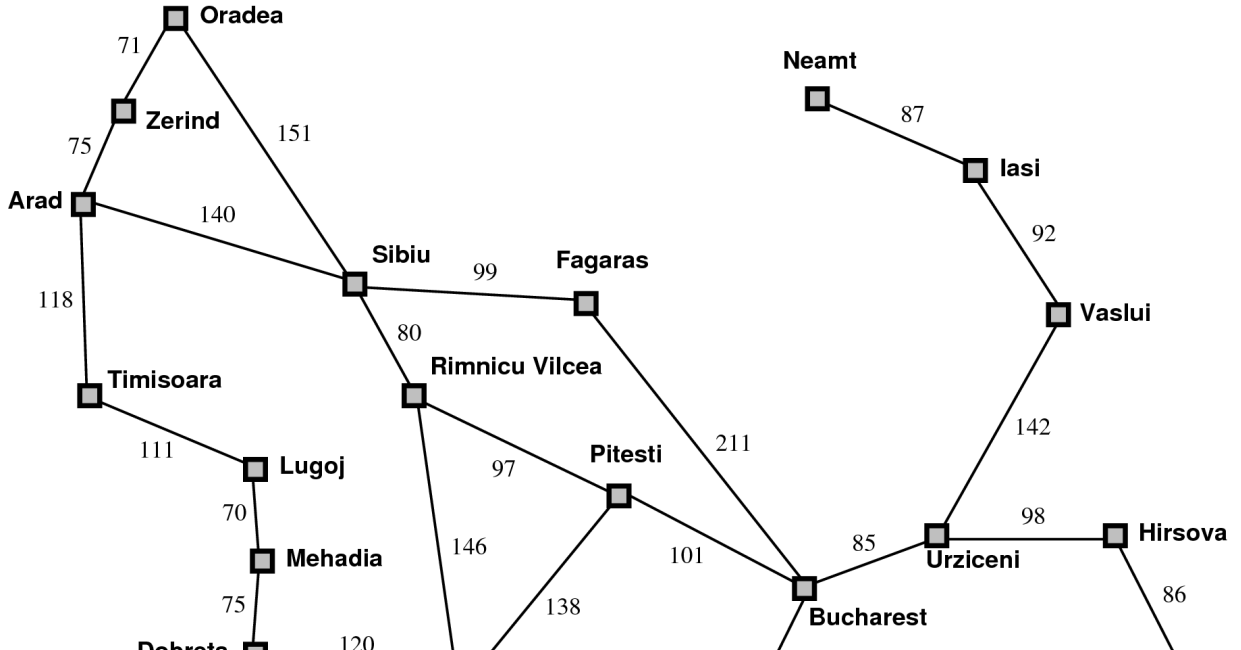
(c) After expanding Sibiu



Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

(d) After expanding Fagaras

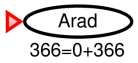




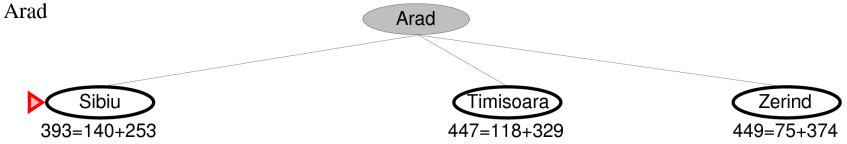
Busca A*

- Minimizar o custo total estimado da solução
  - $f(n) = g(n) + h(n)$
  - $g(n)$: custo para alcançar cada nó
  - $h(n)$: custo de ir do nó até o objetivo
  - $f(n)$: custo estimado da solução de custo mais baixo passando por n

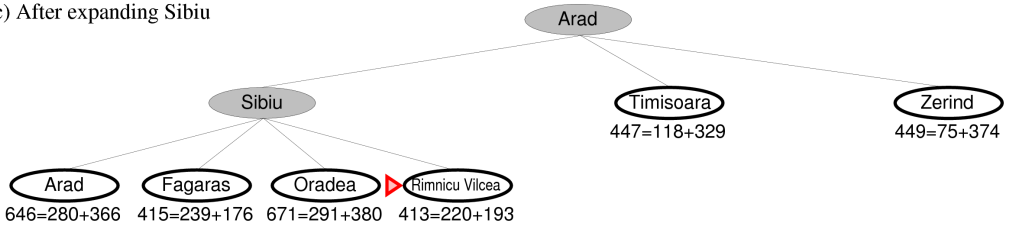
(a) The initial state



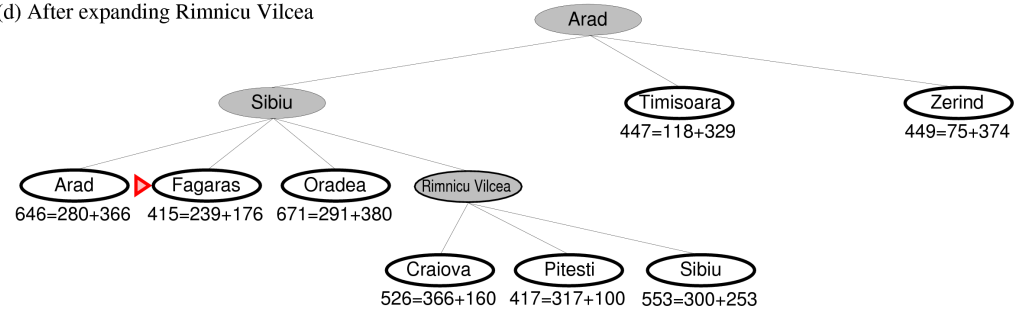
(b) After expanding Arad



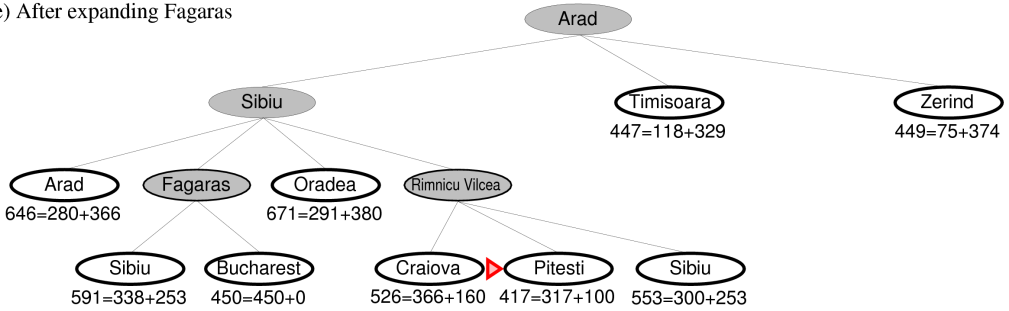
(c) After expanding Sibiu



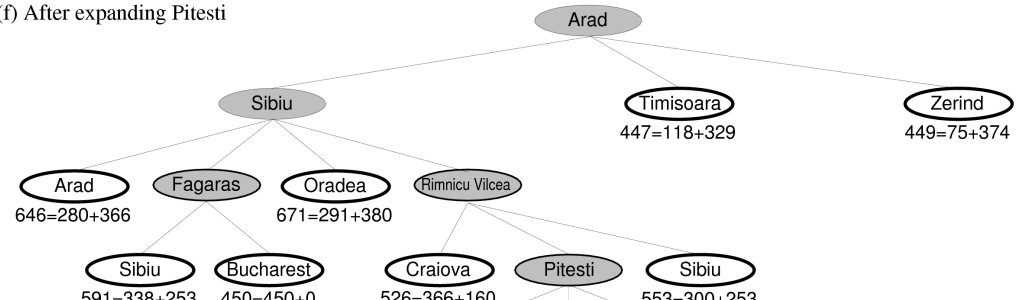
(d) After expanding Rimnicu Vilcea



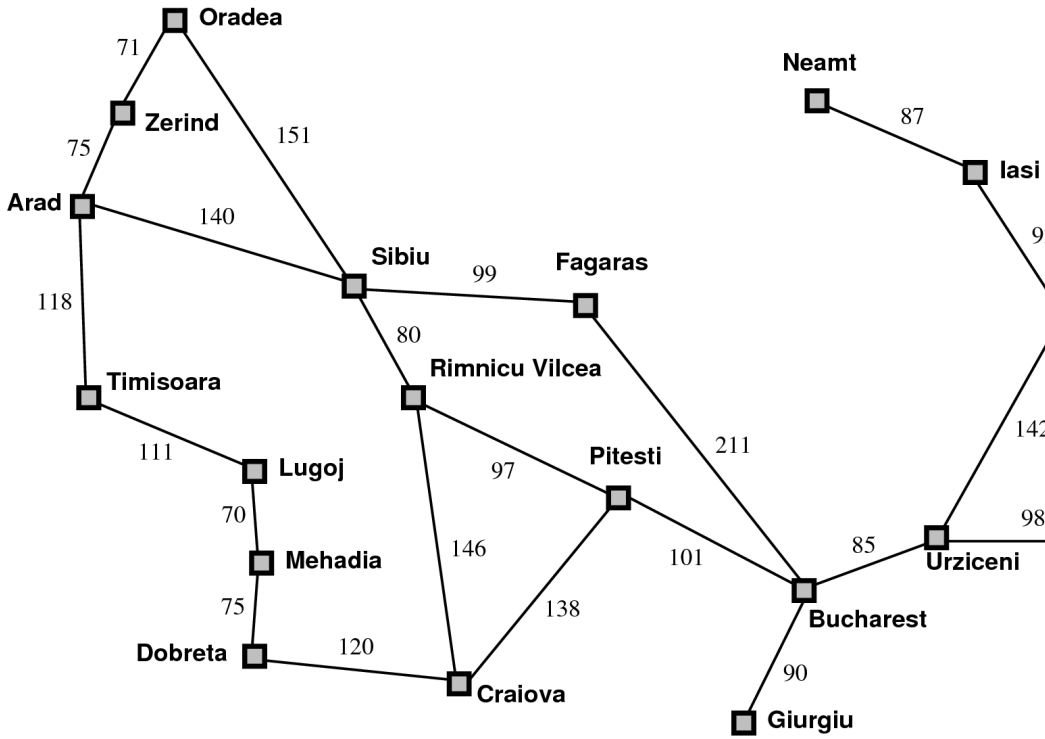
(e) After expanding Fagaras



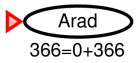
(f) After expanding Pitesti



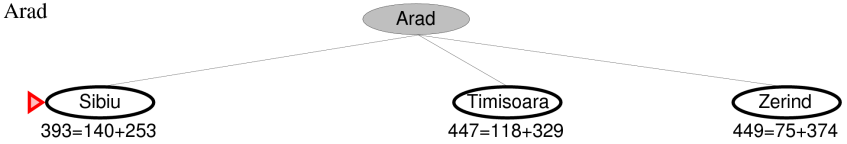
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



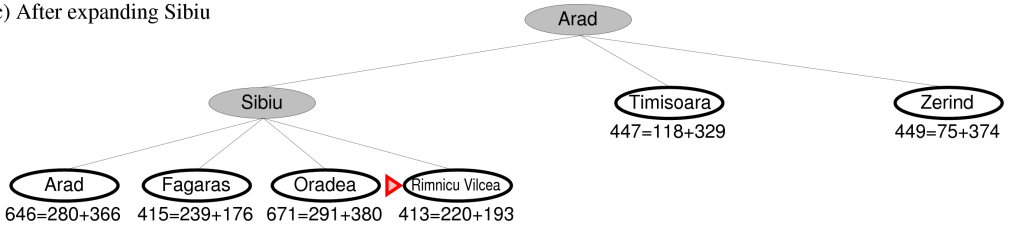
(a) The initial state



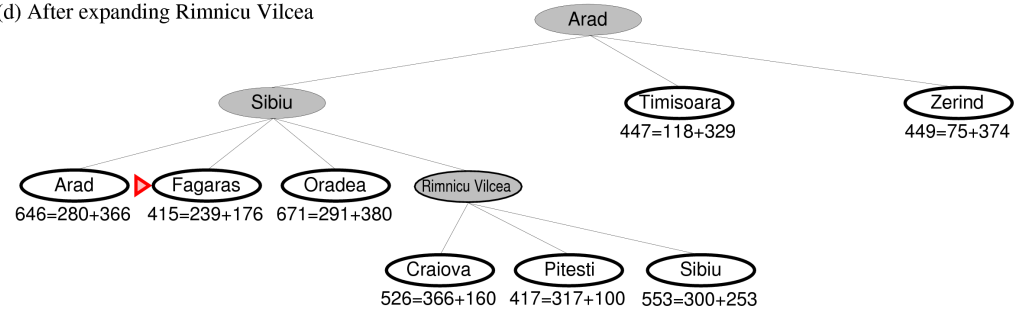
(b) After expanding Arad



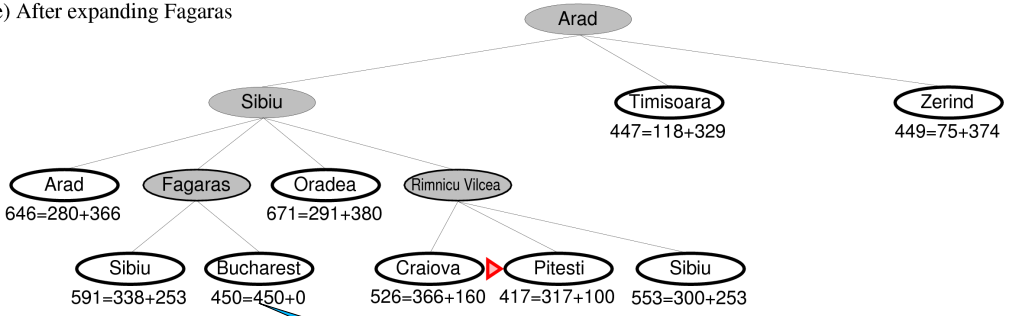
(c) After expanding Sibiu



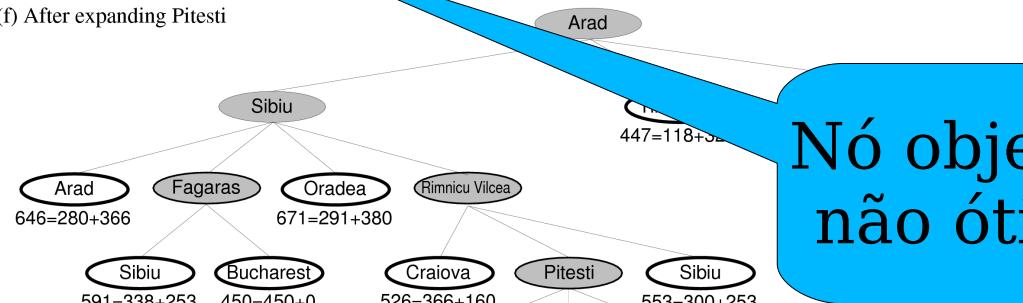
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras

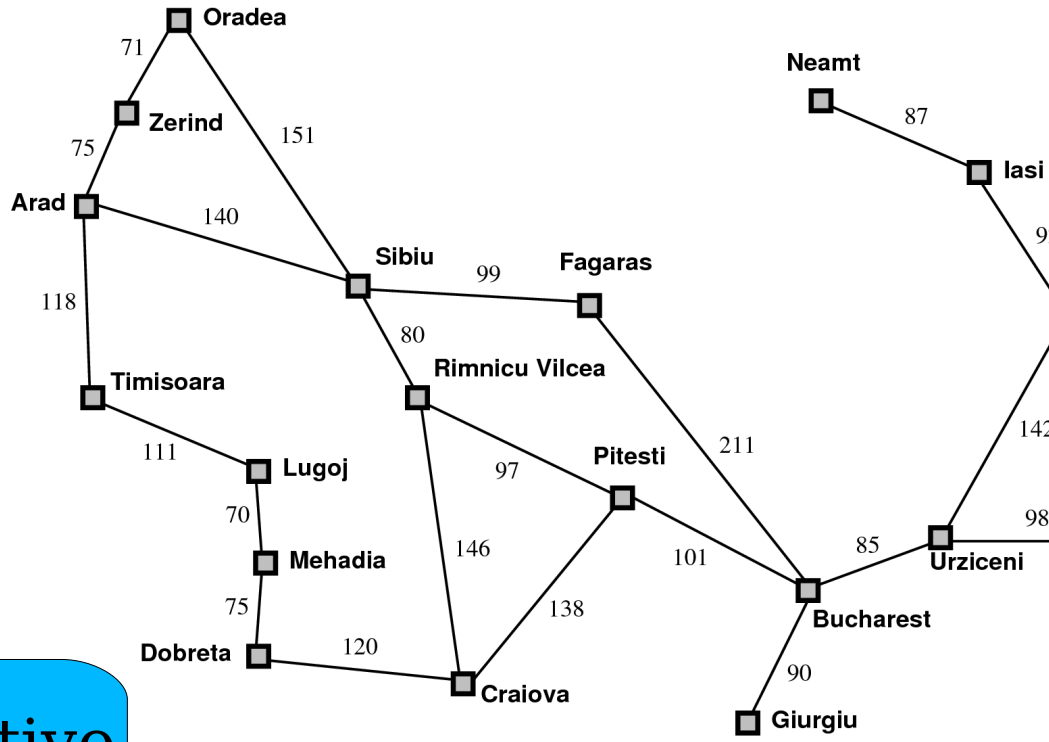


(f) After expanding Pitesti



Nó objetivo não ótimo

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374



Busca A*

- Se for utilizada BUSCA-EM-ÁRVORE
  - *h(n)* deve ser uma **heurística admissível,** ou seja, *h(n)* nunca deve superestimar o custo para alcançar o objetivo
  - $f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$
    - G_2: nó objetivo não ótimo na borda
    - C^*: custo da solução ótima
  - $f(n) = g(n) + h(n) <= C^*$

Trabalho

- Implemente um algoritmo de busca para o seguinte problema:

  "Dado um labirinto de salas na forma de um tabuleiro, uma posição inicial e uma posição de saída, busque um caminho que leve a saída."

			X		
	X		X	X	
	X	X	X	S	X
I			X		
X	X			X	
		X			

I: posição inicial
S: saída
X: obstáculo

Trabalho

- Características do labirinto
  - 10 x 10 posições
  - A saída e o ponto inicial são determinados aleatoriamente a cada execução do algoritmo
  - Obstáculos devem ser estabelecidos previamente e não se alteram a cada execução
  - Os movimentos possíveis variam entre: direita, esquerda, acima e abaixo

Execução e entrega

- Aulas em laboratório em duplas
  - 7 de Outubro
  - 14 de Outubro
- Entrega
  - 28 de Outubro
- Formato
  - Código comentado
  - Relatório