

# BUSCA INFORMADA

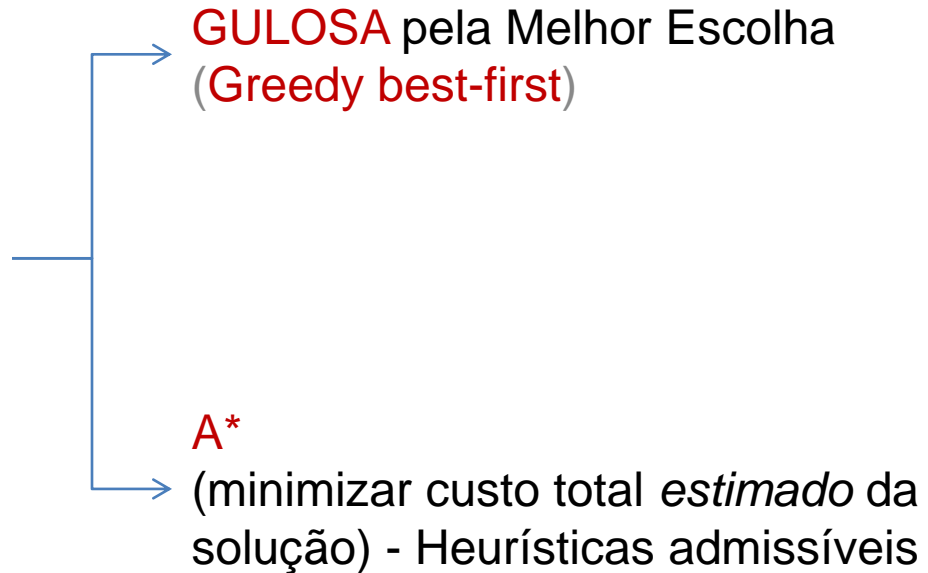
(PARTE 3 - RESOLUÇÃO DE PROBLEMAS  
POR MEIO DE BUSCA)

# Material

- Capítulo 4 - Rusell & Norvig

# Roteiro

Busca de melhor  
escolha (**Best-first**)



# BUSCA INFORMADA

- Busca informada **utiliza conhecimento do problema** para guiar a busca.
- Este conhecimento utilizado está além da própria definição (formulação) do problema.
  - Estado inicial, modelo de transição (função sucessora), custo de ação, estado objetivo
- Podem encontrar soluções de forma mais eficiente do que as buscas cegas.

Resolução de problemas por meio de busca

# **BUSCA DE MELHOR ESCOLHA (BEST-FIRST)**

# Busca Melhor Escolha (BEST-FIRST)

Melhor escolha é uma abordagem geral de busca informada. Pode ser especializada em: Gulosa e A\*

Melhor escolha seleciona o nó a ser expandido utilizando uma **função de avaliação** denominada  **$f(n)$**

**$f(n)$  é uma função de custo**, então o nó que apresentar **menor  $f(n)$**  é **expandido primeiro**.

Implementação é idêntica ao da busca de custo uniforme substituindo-se  **$g(n)$**  por  **$f(n)$**

# Busca Melhor Escolha: $F(N)$

*Ideia:* usar uma **função de avaliação  $f(n)$**  para cada nó

- **estimar o grau** em que um nó é “desejável” como caminho
- expandir os nós mais desejáveis

$$f(n) = g(n) + h(n)$$

$g(n)$  = Custo do caminho do estado inicial até o nó  $n$

$h(n)$  = Custo estimado de  $n$  ao **estado objetivo** pelo caminho mais barato

Resolução de problemas por meio de buscas

# **BUSCA GULOSA (GREEDY-FIRST)**



# Busca Gulosa

- A cada passo tenta chegar mais perto do estado objetivo sem se preocupar com os passos futuros.
- Utiliza somente a componente heurística da função  $f(n)$   
$$f(n) = \cancel{g(n)} + h(n)$$
- Logo,  $f(n) = h(n)$
- Busca que expande os nós mais baratos baseando-se somente em  $h(n)$

# Exemplo de Busca Gulosa pela Melhor Escolha

$h(n)$  = distâncias estimadas em linha reta até Bucareste

---

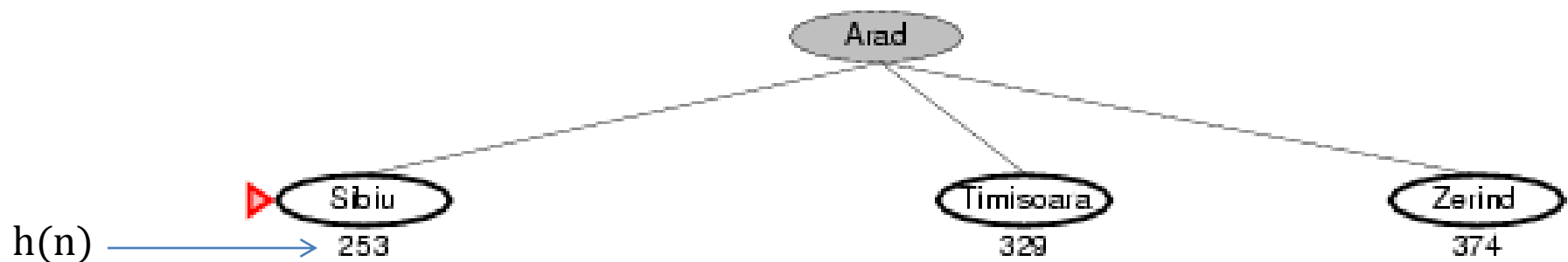
<b>Arad</b>	366	<b>Mehadia</b>	241
<b>Bucharest</b>	0	<b>Neamt</b>	234
<b>Craiova</b>	160	<b>Oradea</b>	380
<b>Drobeta</b>	242	<b>Pitesti</b>	100
<b>Eforie</b>	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	<b>Sibiu</b>	253
<b>Giurgiu</b>	77	<b>Timisoara</b>	329
<b>Hirsova</b>	151	<b>Urziceni</b>	80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

# Exemplo de Busca Gulosa pela Melhor Escolha

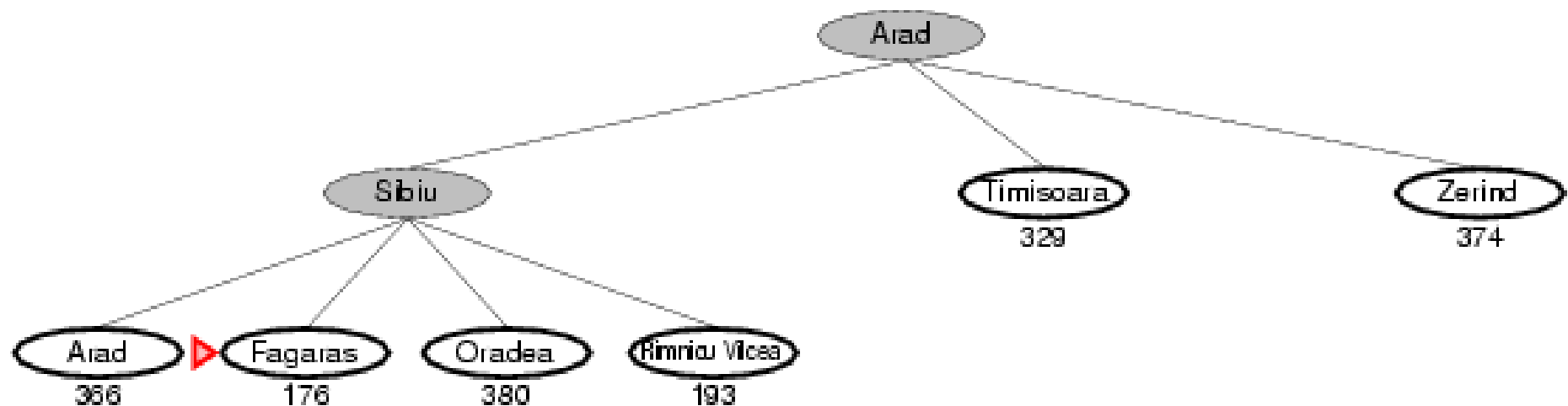
$h(n)$  = distância linha reta



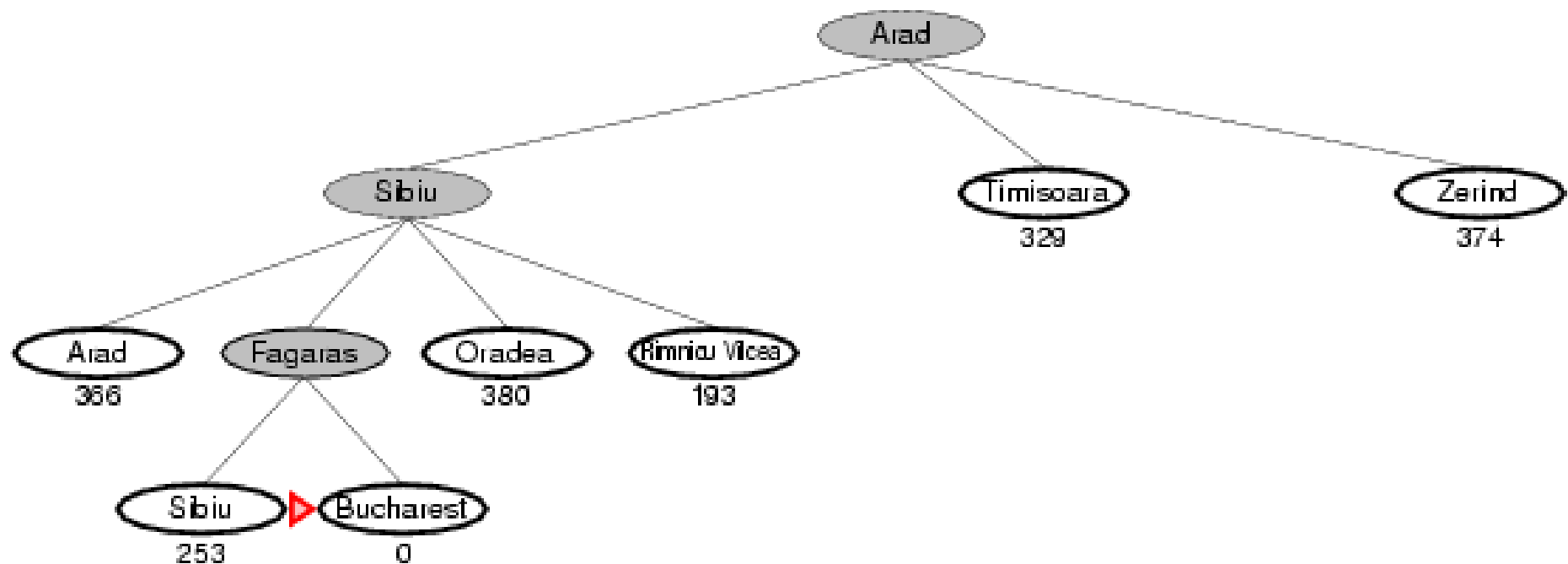
# Exemplo de Busca Gulosa pela Melhor Escolha



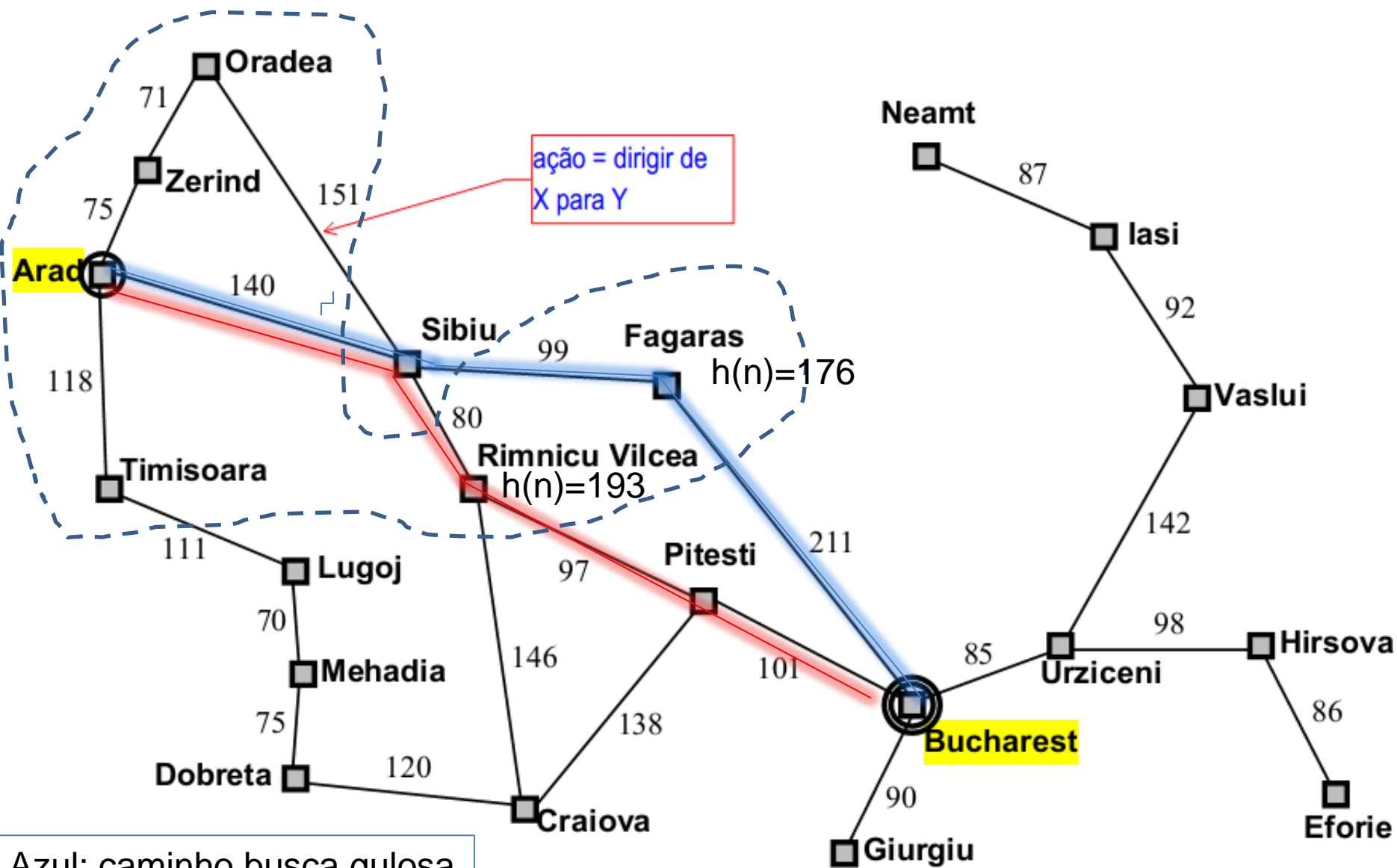
# Exemplo de Busca Gulosa pela Melhor Escolha



# Exemplo de Busca Gulosa pela Melhor Escolha



# Exemplo de Busca Gulosa pela Melhor Escolha



# Avaliação da Busca Gulosa

Espacial	$O(b^m)$
Tempo	$O(b^m)$
Completo	<b>Sim</b> , para busca em grafo, se o espaço de estados for finito
Ótimo	<b>Não</b>



Resolução de problemas por meio de buscas

**A\***

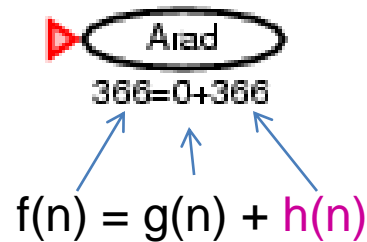
# Busca A\*

- **Ideia:** podar caminhos que são caros
- Função de avaliação  **$f(n) = g(n) + h(n)$** 
  - $g(n)$**  = custo para chegar ao nó  $n$
  - $h(n)$**  = custo estimado para ir de  $n$  até o objetivo
  - $f(n)$**  = custo estimado total do caminho para chegar do estado inicial ao objetivo passando por  $n$

O algoritmo é idêntico ao da busca de custo uniforme e gulosa exceto por:

A*	$f(n) = g(n) + h(n)$
Custo Uniforme	$f(n) = g(n)$
Gulosa	$f(n) = h(n)$

# Exemplo de Busca A\*

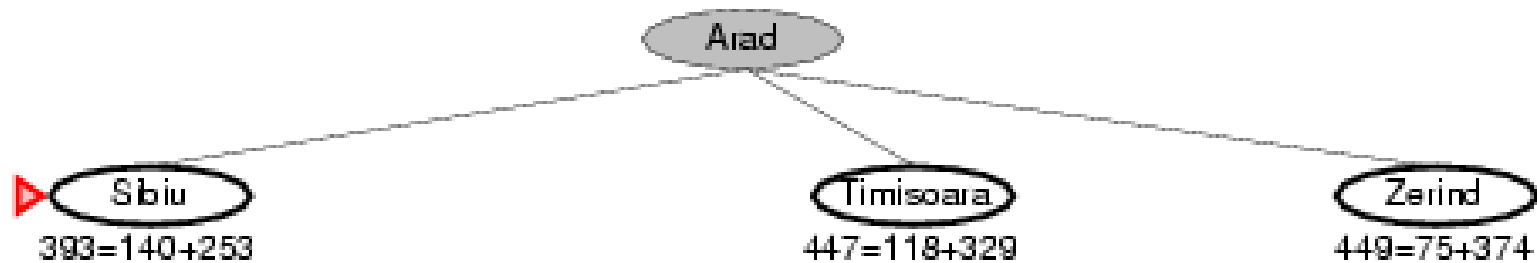


The diagram illustrates the A\* search formula for a specific node. At the top, the node name 'Aiad' is enclosed in an oval, with a red triangle pointing to it from the left. Below the oval, the equation  $366 = 0 + 366$  is displayed. Three blue arrows point upwards from the formula  $f(n) = g(n) + h(n)$  to the components of the equation above: the first arrow points from  $f(n)$  to '366', the second from  $g(n)$  to '0', and the third from  $h(n)$  to '366'.

$$f(n) = g(n) + h(n)$$
$$366 = 0 + 366$$

$h(n)$  distância em linha reta de n até  
Bucareste

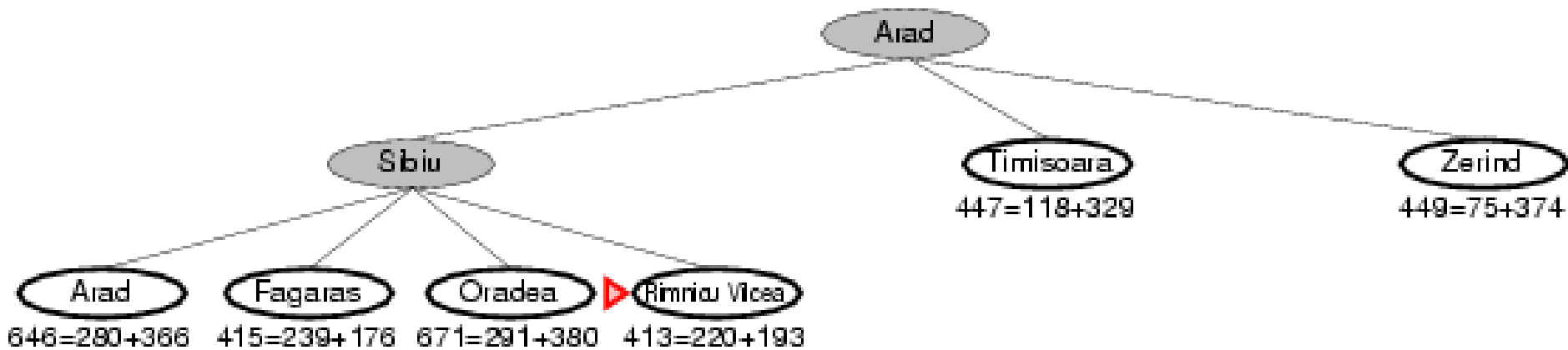
# Exemplo de Busca A\*



Fronteira lista ordenada por  $f(n)$

**Sbiu (393)** < Timisoara (447) < Zerind (449)

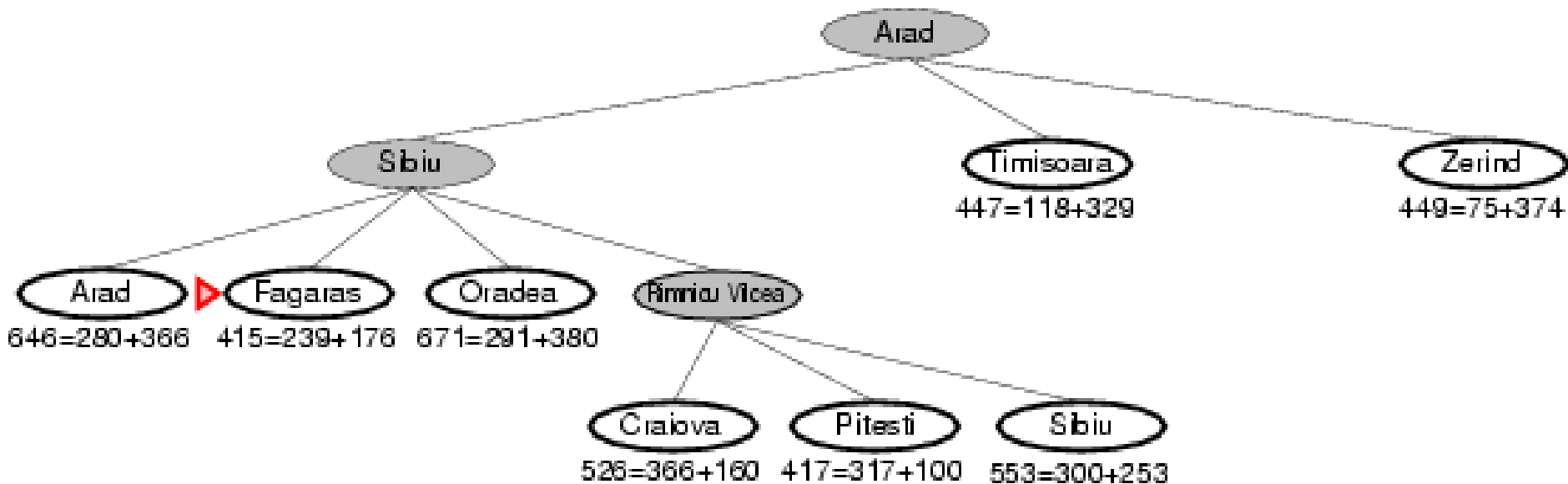
# Exemplo de Busca A\*



Fronteira lista ordenada por  $f(n)$

**Rimnicu (413)** < Fagaras (415) < Timisoara (447) < Zerind (449) < Arad (646) < Oradea (671)

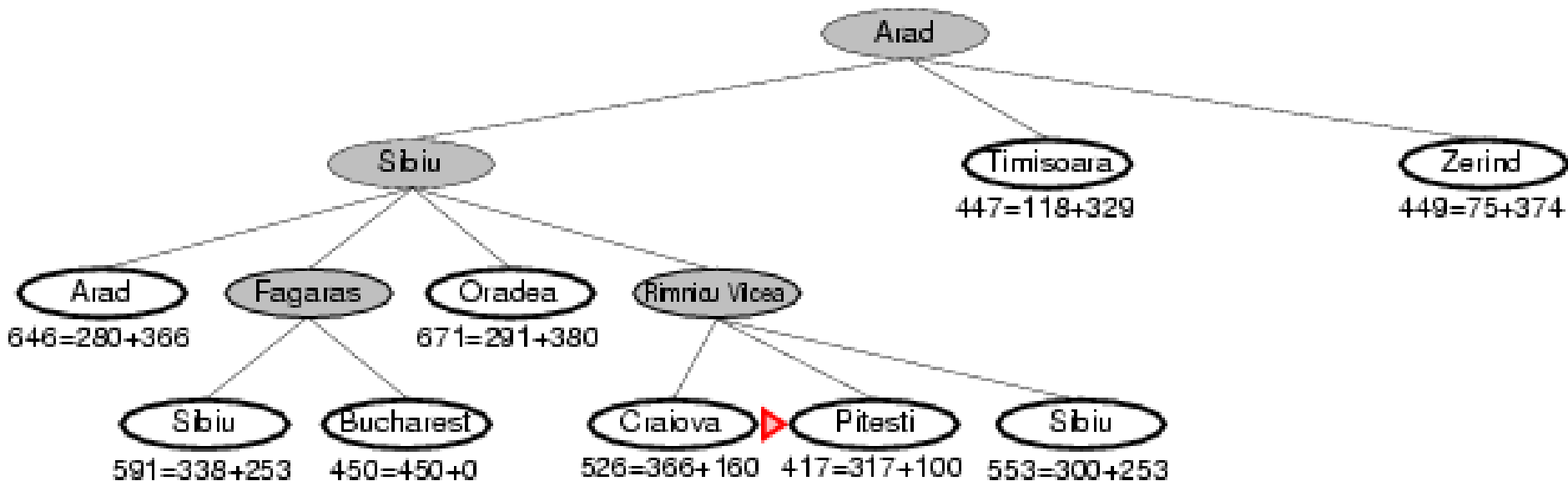
# Exemplo de Busca A\*



Fronteira lista ordenada por  $f(n)$

**Fagaras (415)** < Timisoara (447) < Zerind (449) < Craiova (526) < Arad (646) < Oradea (671)

# Exemplo de Busca A\*



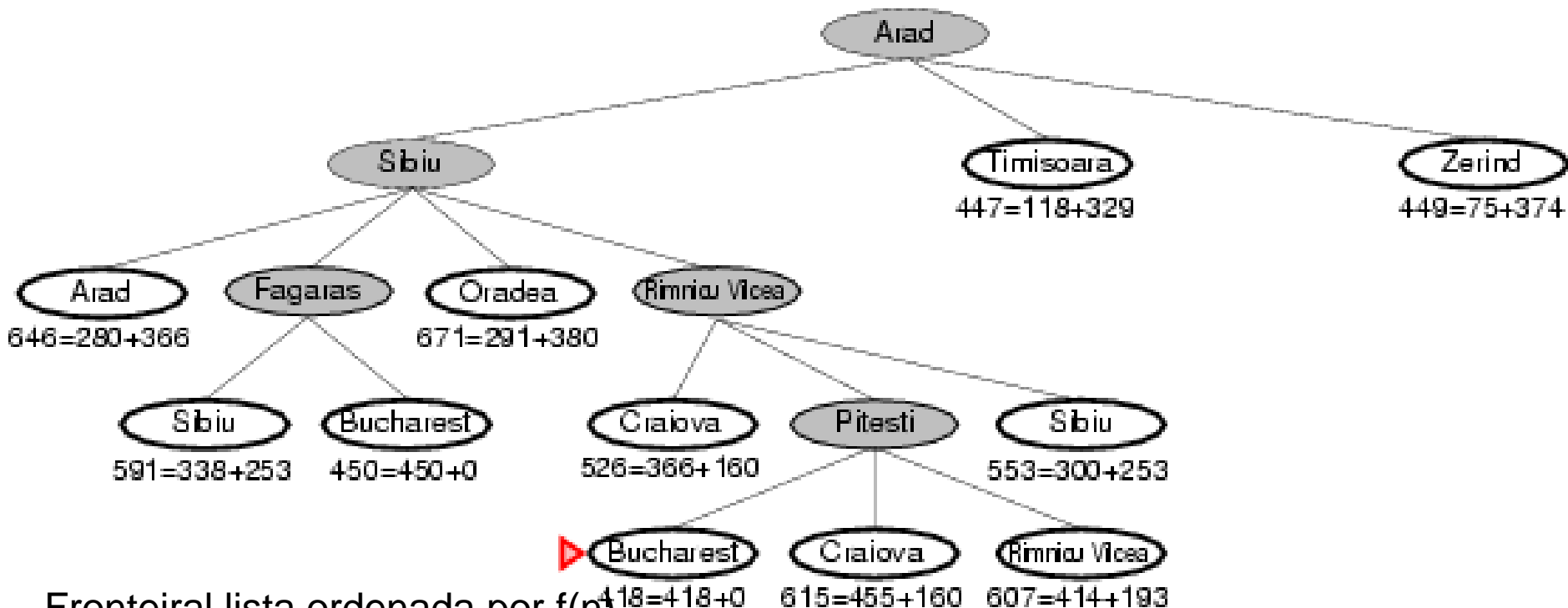
Fronteira lista ordenada por  $f(n)$

**Pitesti (417)** < Timisoara (447) < Zerind (449) < **Bucharest (450)** < Craiova (526) < Sibiu (553) < Arad (646) < Oradea (671)

## Condição parada

Ao expandir Fagaras, Bucharest aparece na fronteira.  
Porém, o algoritmo só para quando Bucharest for o primeiro da lista

# Exemplo de Busca A\*



Fronteiral lista ordenada por  $f(n)$

**Bucharest (418)** < Timisoara (447) < Zerind (449) < **Bucharest (450)** < Craiova (526) < Sibiu (553) < Rimnicu (607) < Craiova (615) < Arad (646) < Oradea (671)

condição de parada é atingida!



# ANÁLISE DE COMPLEXIDADE DE A\*

- A otimalidade de A\* depende da componente  $h(n)$
- Condições para otimalidade:
  - **$h(n)$**  deve ser uma heurística **admissível**:
    - nunca **superestimar** o custo real para alcançar o estado objetivo
    - Garante que  $f(n)$  é não-decrescente
    - *Para busca em árvore basta ser admissível*
  - **$h(n)$**  deve ser **consistente**
    - **respeitar** o princípio da desigualdade triangular
    - *Para busca em grafo tem que ser admissível e consistente*

# Heurística Admissível

Uma heurística  $h(n)$  é **admissível** se todo nó  $n$ :

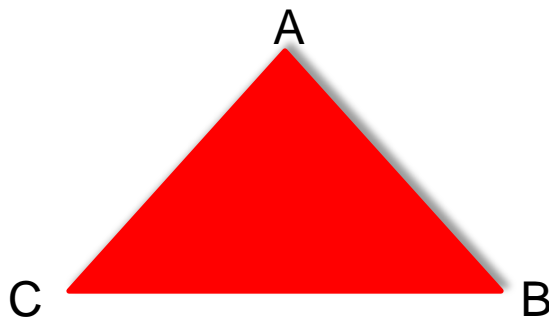
$$h(n) \leq h^*(n),$$

onde  **$h^*(n)$**  é o custo real (modelado) para se alcançar o estado-objetivo a partir de  $n$

isto é, **OTIMISTA!**

# Heurística Consistente

- Para A\* com BUSCA-EM-GRAFO há uma condição adicional:
  - ser **CONSISTENTE** ou **MONOTÔNICO**;
  - i.e. a  $f(n)$  deve ser não-decrescente
  - Deve respeitar o teorema da DESIGUALDADE TRIANGULAR



O tamanho de um lado não pode ser maior do que a soma dos tamanhos dos outros dois lados.

# Heurística Consistente (MONOTONICIDADE)

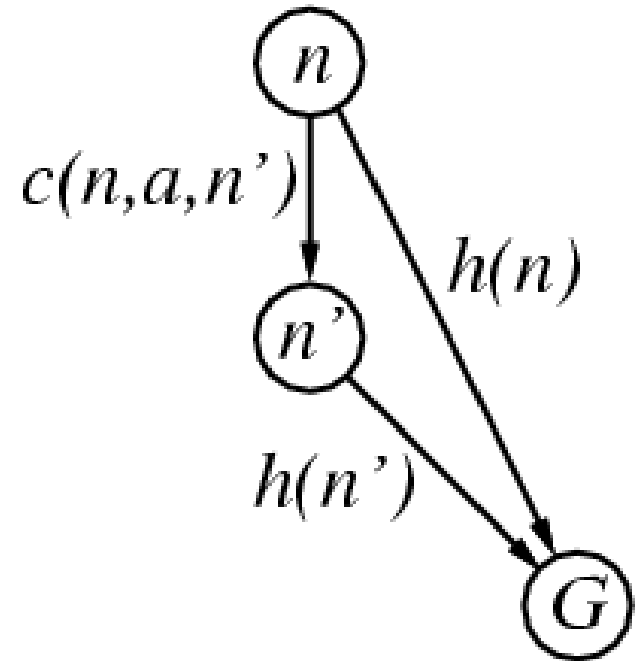
- Uma heurística é **consistente** se para cada nó  $n$ , cada sucessor  $n'$  de  $n$  gerado por qualquer ação  $a$ ,

$$h(n) \leq c(n, a, n') + h(n')$$

- Se  $h$  é consistente, então  $f(n') \geq f(n)$  ( $n'$  é o sucessor de  $n$ )

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) \end{aligned}$$

- i.e.,  **$f(n)$  é não-decrescente** ao longo de qualquer caminho.

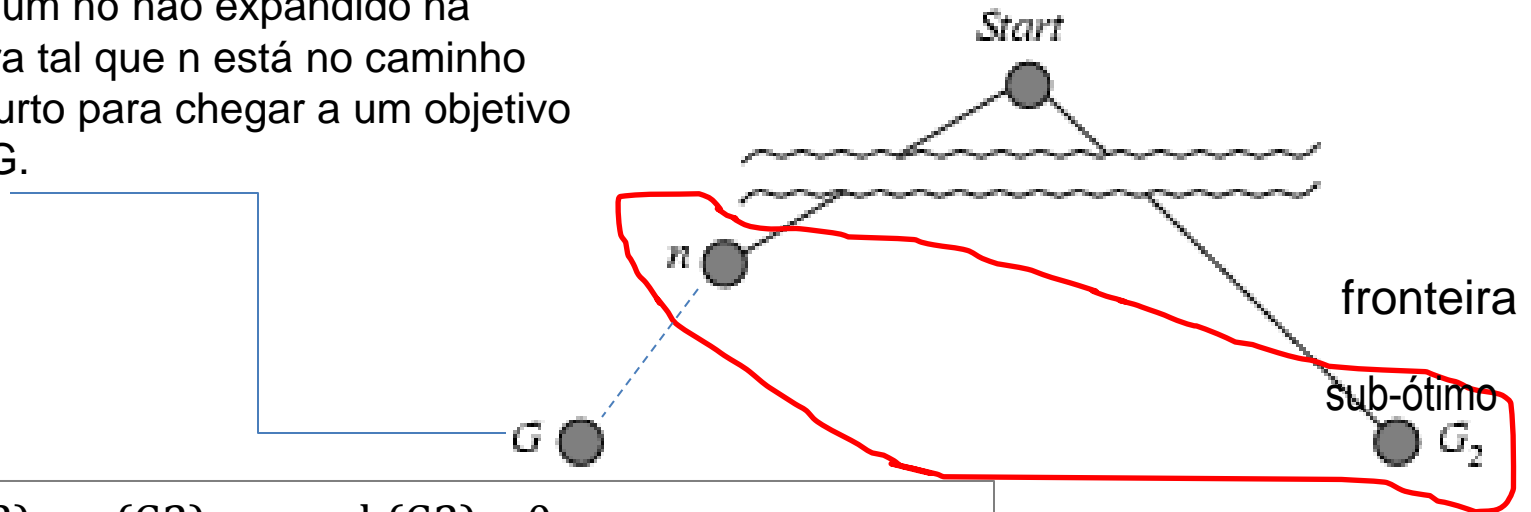


**Teorema:** Se  $h(n)$  é consistente,  $A^*$  é ótima (para busca em grafo)

# Otimidade de $A^*$ (prova)

**Provar que** mesmo que haja um objetivo sub-ótimo  $G_2$  na fronteira,  $A^*$  alcança o objetivo ótimo  $G$ , pois  $f(G_2) > f(G)$

Seja  $n$  um nó não expandido na fronteira tal que  $n$  está no caminho mais curto para chegar a um objetivo ótimo  $G$ .

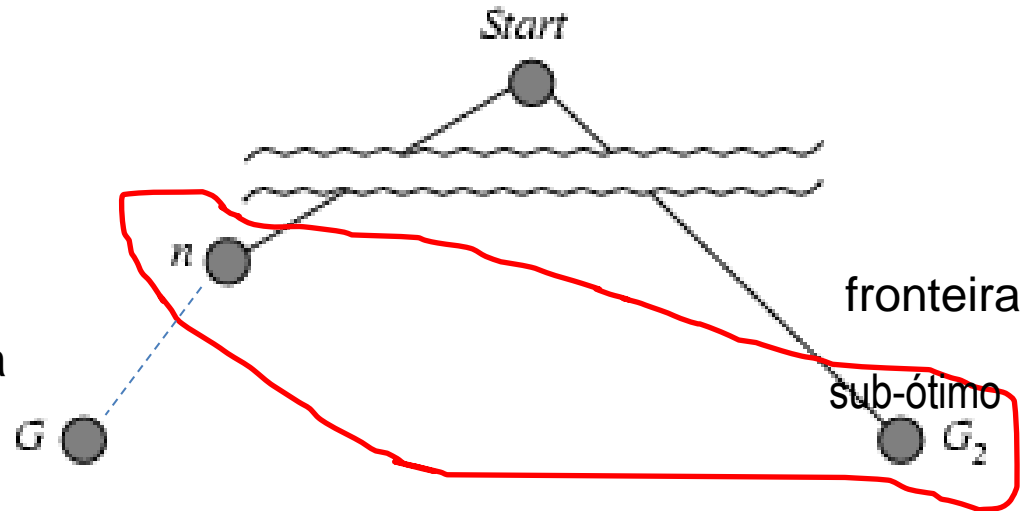


1.  $f(G_2) = g(G_2)$  como  $h(G_2) = 0$
2.  $f(G) = g(G)$  como  $h(G) = 0$
3.  $g(G_2) > g(G)$  como  $G_2$  é sub-ótimo
4.  $f(G_2) > f(G)$  como consequência de 1, 2 e 3

# Otimidade de $A^*$ (prova)

**Prova:** mesmo que um objetivo sub-ótimo ( $G_2$ ) esteja na fronteira, ele não será selecionado se há um nó num caminho mais barato para atingir o objetivo ótimo.

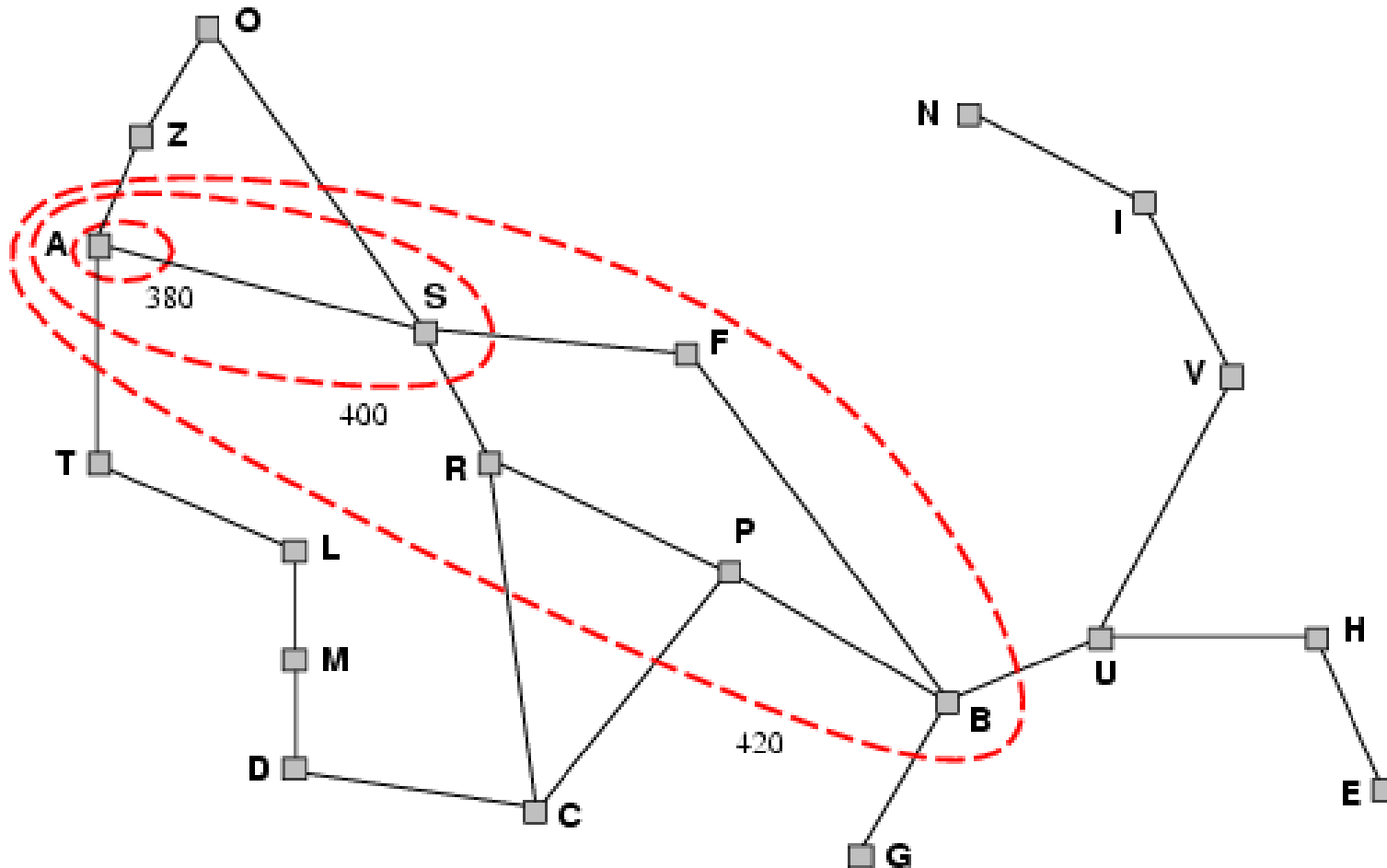
Daí  $f(G_2) > f(n)$  e  $A^*$  nunca seleciona  $G_2$  para ser expandido



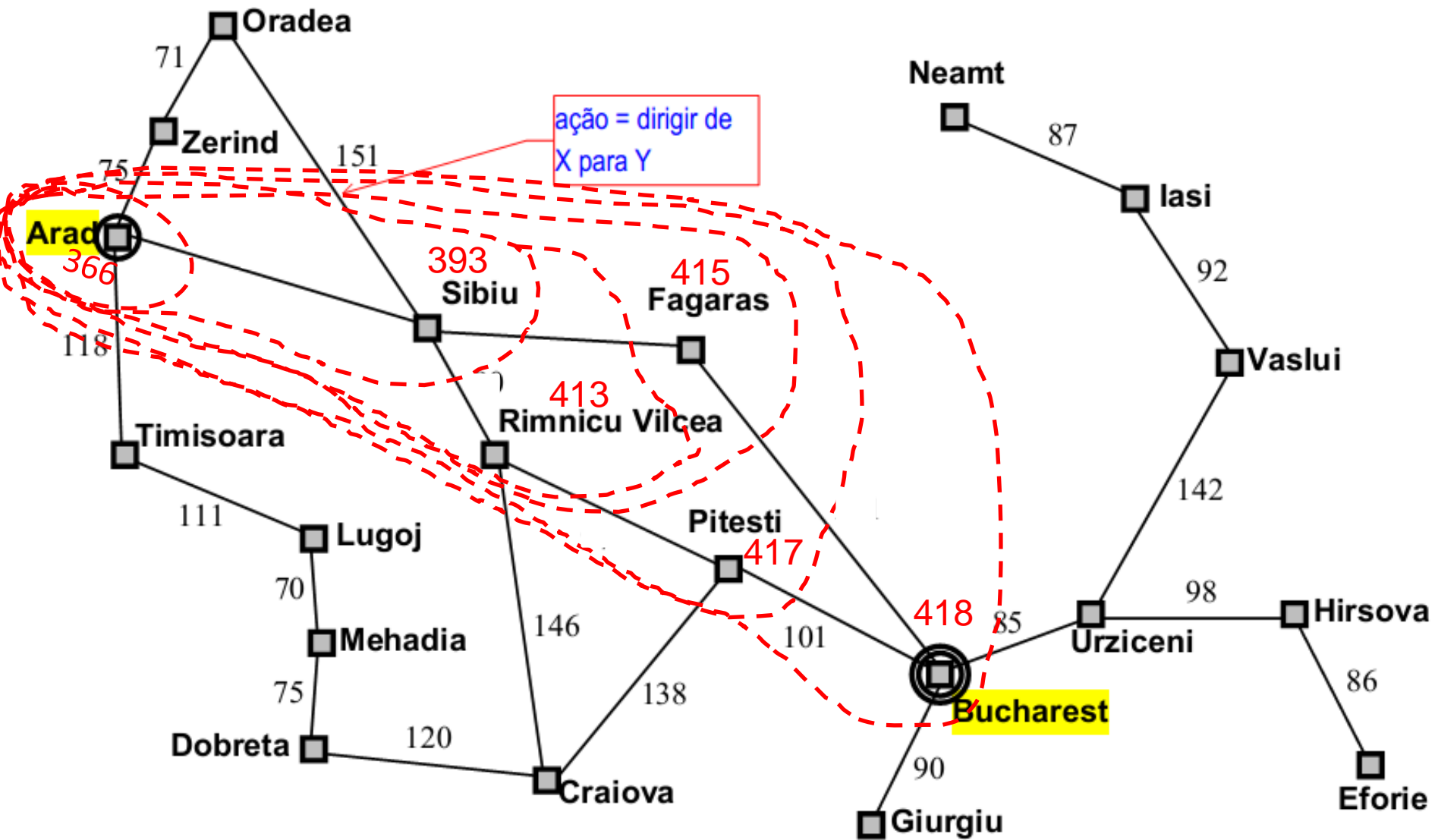
1.  $f(G_2) > f(G)$  *do exposto anteriormente*
2.  $h(n) \leq h^*(n)$  *dado que  $h$  é admissível;  $h^*(n) = \text{custo real (sem aproximação)}$*
3.  $g(n) + h(n) \leq g(n) + h^*(n) = f(G)$  *incluindo  $g(n)$  nos dois lados da ineq. 2*
4.  $f(n) \leq f(G) < f(G_2)$  *de 3 e 1*

# Otimidade de $A^*$

- $A^*$  expande nós em ordem de valores não decrescentes de  $f$
- Gradualmente vão sendo adicionados os " $f$ -contornos" = nós selecionados para expansão dos nós



$A^*$





# Propriedades de A\*

- Completa?
  - Sim, desde que não existam infinitos nós com  $f(n) \leq C^*$ 
    - $C^*$  custo do caminho da solução ótima
    - (A\* expande todos os nós tal que  $f(n) < C^*$ )
- Tempo?
  - Exponencial:  $O(b^\Delta)$ 
    - $\Delta = h^* - h$  (*erro absoluto da heurística*)
- Espaço?
  - Guarda todos os nós na memória
    - problema: normalmente explode para espaços de estados grandes
- Ótima?
  - Sim,
    - se heurística for admissível e consistente

Solução de problemas por meio de busca

# **HEURÍSTICAS ADMISSÍVEIS**

# Heurísticas Admissíveis

E.g., para o quebra-cabeça de 8 peças

- $h_1(n)$  = número de pedras fora do lugar
- $h_2(n)$  = distância total à la Manhattan

(i.e., número de quadrados da localização desejada de cada pedra)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $h_1(S) = ?$
- $h_2(S) = ?$

# Heurísticas Admissíveis

E.g., para o quebra-cabeça de 8 peças

- $h_1(n)$  = número de pedras fora do lugar
- $h_2(n)$  = distância total à la Manhattan

(i.e., número de quadrados da localização desejada de cada pedra)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $h_1(S) = ?$  8
- $h_2(S) = ?$   $3+1+2+2+2+3+3+2 = 18$

# Dominância

- **Se  $h_2(n) \geq h_1(n)$  para todo  $n$  (ambas admissíveis)**
  - então  $h_2$  domina  $h_1$
  - $h_2$  é melhor para busca
- Custos de busca típicos (número médio de nós expandidos):
- $d=12$                       BAI = 3.644.035 nós                      Busca por Aprofundamento Iterativo
  - $A^*(h_1) = 227$  nós
  - $A^*(h_2) = 73$  nós
- $d=24$                       BAI = muitos nós!!!
  - $A^*(h_1) = 39.135$  nós
  - $A^*(h_2) = 1.641$  nós

# Problemas com menos restrições

- Um problema com menos restrições nas ações é chamado de problema relaxado → relaxação permite obter heurísticas admissíveis/consistentes
- O custo da solução ótima para um **problema relaxado** é uma heurística admissível para o problema original
- $h_1$  = relaxa as regras do quebra-cabeças com 8 peças tal que uma pedra possa ser movimentada para qualquer posição (passando por cima das outras)
- $h_2$  = relaxa as regras para que as pedras possam ir para qualquer quadrado adjacente (menos relaxada que  $h_1$ )