

---

# Métodos de Busca Informada (best first search)

## Capítulo 4

### Parte I

Leliane Nunes de Barros

leliane@ime.usp.br

# Busca não informada: geração sistemática de estados

---

- **Busca em profundidade:**

- boa quando não se deseja encontrar a solução ótima (em especial, se existem múltiplas soluções); economia de espaço.
- ruim se não existir um estado meta naquela sub-árvore; se compromete muito com um ramo específico da árvore.

- **Busca em largura:**

- não se compromete com um ramo específico mas consome muita memória e tempo.

- **Propriedades das estratégias de busca:**

- completeza: visita todos os nós (garantia de se encontrar uma solução, se ela existir).
- sistematicidade: nunca visita um nó duas vezes.

# Questão

---

- Como podemos modificar os algoritmos gerais de busca para incluir mais conhecimento sobre o problema?

(mais conhecimento: que vá além do que está na descrição do problema, isto é, descrição de estado, ações, função custo e teste de estado meta)

# Busca informada: *Best-first Search*

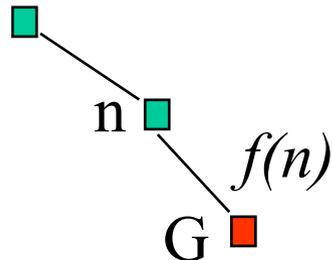
---

- procedimento não sistemático  $\Rightarrow$  prefere selecionar nós que “prometem” dirigir a busca mais rapidamente ao estado meta.
- usa informação específica do domínio que indica qual “parece” ser o melhor caminho para se atingir o estado meta, ou seja, o próximo melhor nó a ser expandido.

# Best-first Search

Conhecimento específico do problema  $\Rightarrow$  função de avaliação  $f(n)$ :

- incorpora uma estimativa do custo do caminho entre o nó corrente e o estado meta
- recebe descrições de estados e devolve um valor real
- convenção: valores menores de  $f(n)$  indicam os melhores nós;  $f(G)=0$



# *Best-first Search*

---

- introduz conhecimento na estratégia geral de busca (QUEUEING-FN)
- expande primeiro o nó “aparentemente melhor” (o nó que possui a melhor avaliação).
- termina quando o nó a ser expandido for o estado meta.

# *Best-First Search*

---

**função** *BestFirstSearch(problema, estratégia)* **devolve** uma solução,  
ou falha

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

**laço faça**

se não há mais candidatos para expandir **então devolve** falha

escolha o primeiro nó da lista de nós terminais

se o nó contém um estado gol **então devolve** a solução

**senão** expanda o nó de acordo com a *estratégia*

**fim**

Obs: *estratégia* = QUEUING-FN (ordenação da fila de acordo com uma função de avaliação)

# Questão

---

*Best-First Search* expande sempre o melhor nó primeiro?

- Se soubessemos qual é o melhor nó, não haveria busca

# Família de algoritmos BFS

---

- Busca de Custo Uniforme (ou de melhor custo)
- Greedy Best-First Search (busca gulosa)
- Busca  $A^*$
- Busca IDA\*

# Questão

---

Qual é a diferença entre os métodos:  
busca uniforme e *best-first search*?

# Função de avaliação - custo uniforme

---

- Podem incorporar informação sobre o custo da solução
- Exemplo: Busca de Custo Uniforme
  - $f(n) = g(n)$ : estimativa do custo do caminho
  - não dirige a busca para a solução

# Função heurística - $h(n)$

---

- $f(n) = h(n) \longrightarrow$ 
  - $h(n)$ : **estimativa** do custo do caminho que leva a solução
  - $h(n) = 0$  se  $n$  é o estado meta
- constrói-se uma  $h(n)$  específica para o problema

# Função de avaliação para o 8-puzzle

Quais seriam algumas funções de avaliação para o 8-puzzle ?

5	4	
6	1	8
7	3	2

Estado corrente

1	2	3
8		4
7	6	5

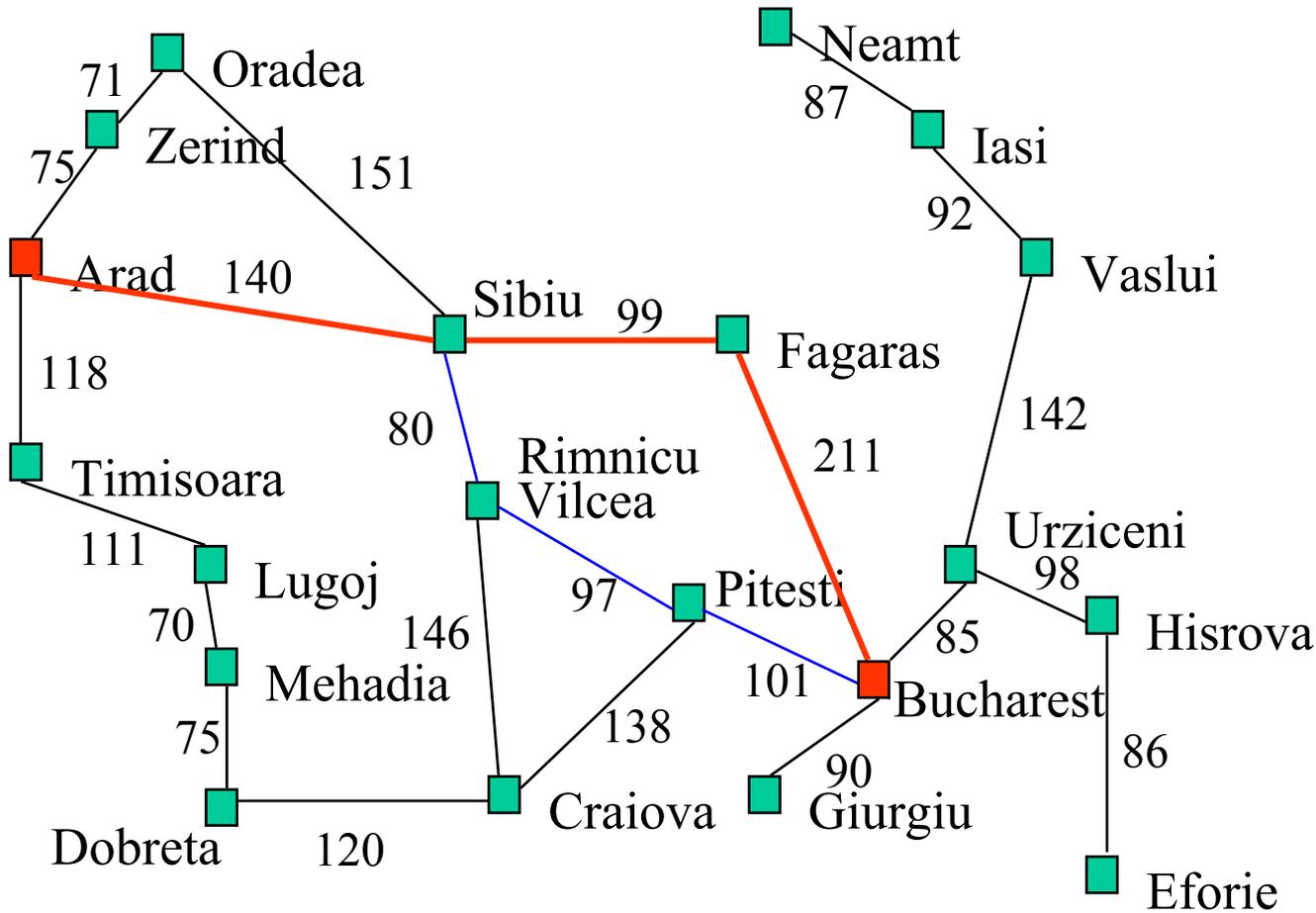
Estado meta

# *Greedy Best-First Search*

---

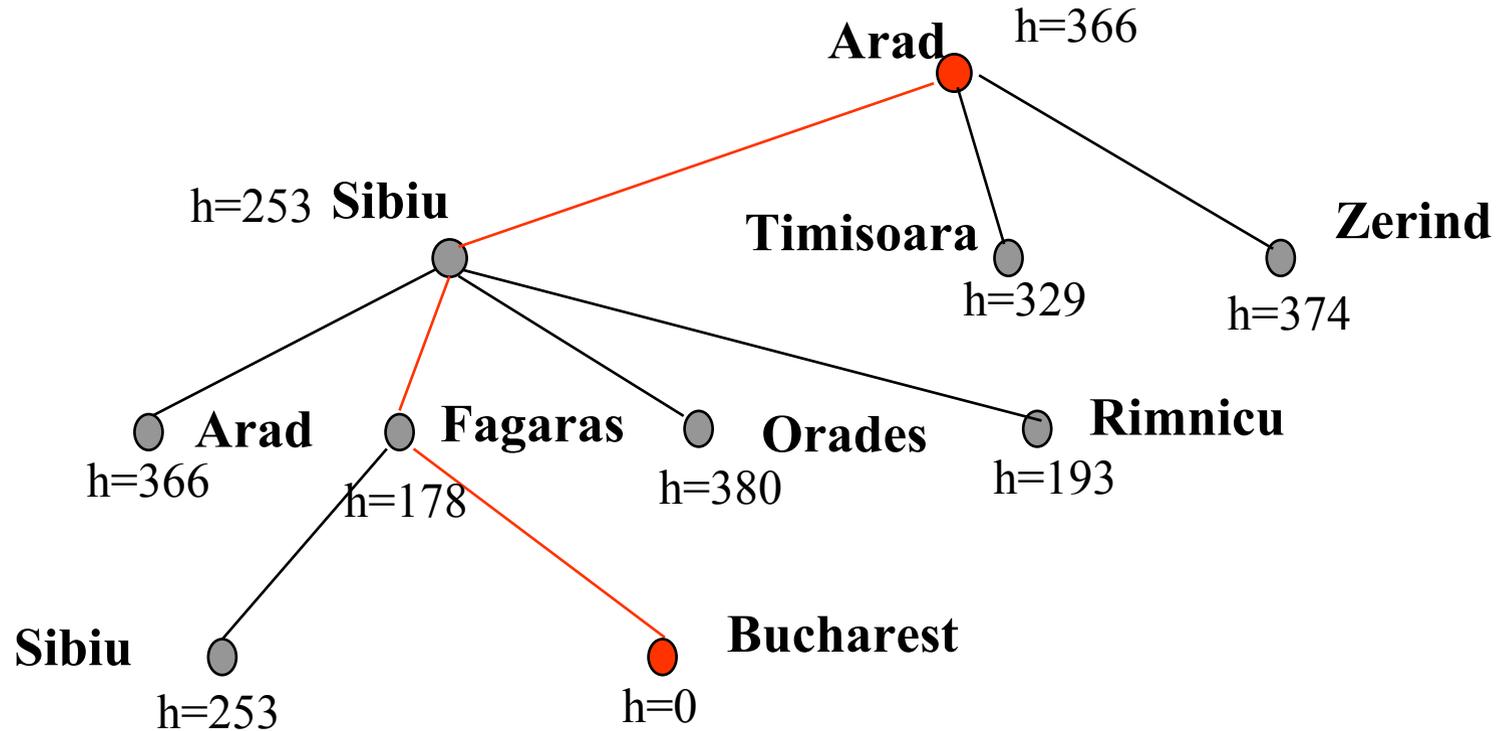
- algoritmo *Best-first Search* que usa  $h$  para selecionar nós a serem expandidos
- minimiza o custo estimado para se atingir o estado meta ( $h(G)$ )
- tenta encontrar uma solução rápida mas que nem sempre é a ótima (algoritmo não ótimo)
- como o *DFS*: não completo
- como na *BrFS*: guarda todos os nós gerados
- complexidade de tempo e espaço  $O(b^m)$  - pode ser reduzida com uma boa função heurística (depende fortemente da heurística)

# Exemplo 1: ir de Arad para Bucharest



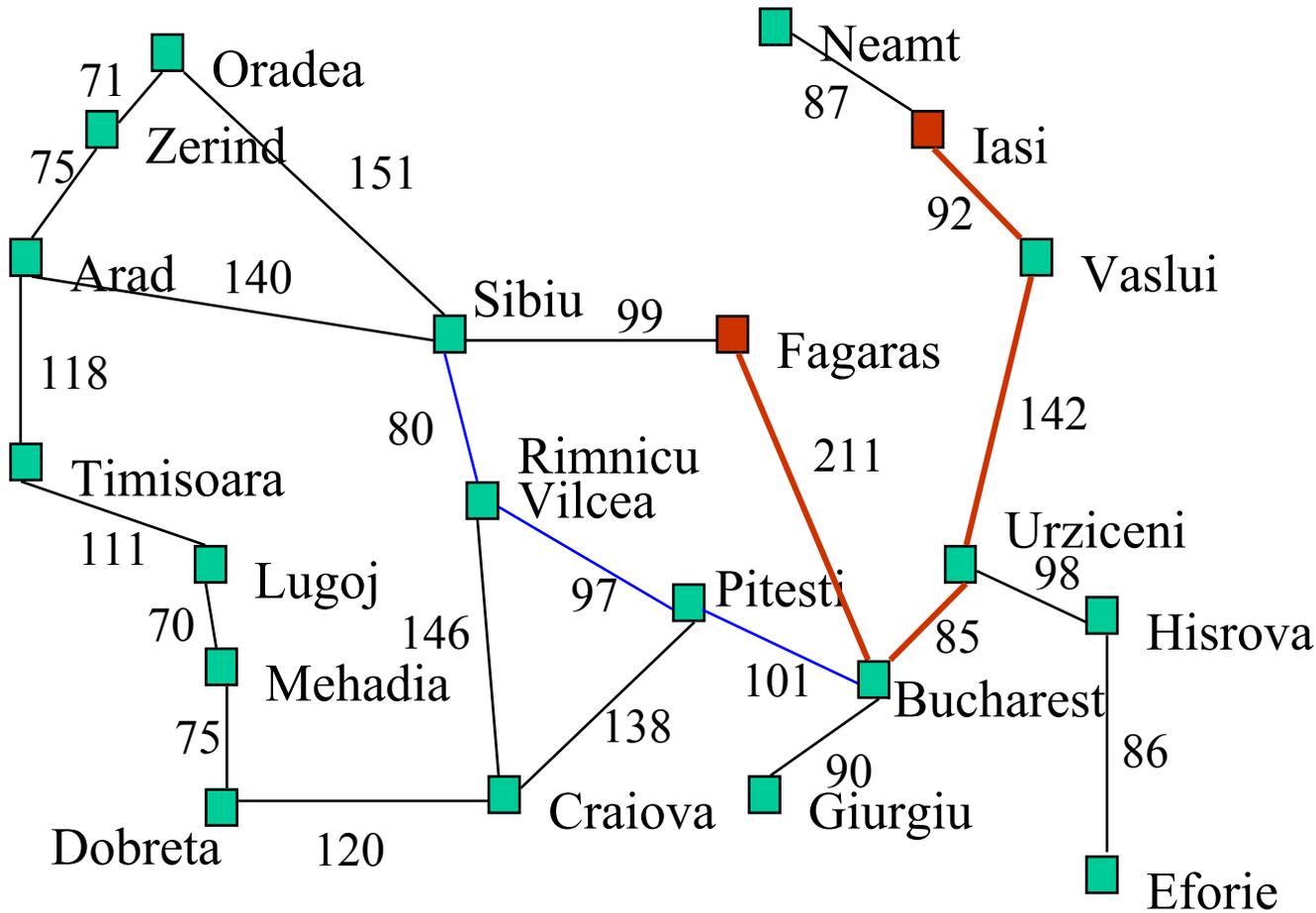
<b>Arad</b>	<b>36</b>
<b>Bucharest</b>	<b>0</b>
<b>Craiova</b>	<b>16</b>
<b>Dobreta</b>	<b>24</b>
<b>Eforie</b>	<b>16</b>
<b>Fagaras</b>	<b>17</b>
<b>Giurgiu</b>	<b>7</b>
<b>Hirsova</b>	<b>15</b>
<b>Iasi</b>	<b>22</b>
<b>Lugoj</b>	<b>24</b>
<b>Mehadia</b>	<b>24</b>
<b>Neamt</b>	<b>23</b>
<b>Oradea</b>	<b>38</b>
<b>Pitesti</b>	<b>98</b>
<b>Rimnicu Vilcea</b>	<b>19</b>
<b>Sibiu</b>	<b>25</b>
<b>Timisoara</b>	<b>32</b>
<b>Vaslui</b>	<b>25</b>

# Espaço de busca



$h(n)$ : distância à cidade de destino em linha reta

# Exemplo 2: ir de Iasi para Fagaras



<b>Arad</b>	<b>36</b>
<b>Bucharest</b>	
<b>Craiova</b>	<b>16</b>
<b>Dobreta</b>	<b>24</b>
<b>Eforie</b>	<b>16</b>
<b>Fagaras</b>	<b>17</b>
<b>Giurgiu</b>	<b>7</b>
<b>Hirsova</b>	<b>15</b>
<b>Iasi</b>	<b>22</b>
<b>Lugoj</b>	<b>24</b>
<b>Mehadia</b>	<b>24</b>
<b>Neamt</b>	<b>23</b>
<b>Oradea</b>	<b>38</b>
<b>Pitesti</b>	<b>98</b>
<b>Rimnicu Vilcea</b>	<b>19</b>
<b>Sibiu</b>	<b>25</b>
<b>Timisoara</b>	<b>32</b>
<b>Vaslui</b>	<b>25</b>

# Busca A\*

---

- Combina busca de custo uniforme com busca heurística
- $f(n) = g(n) + h(n)$ 
  - $g(n)$  = custo gasto até  $n$
  - $h(n)$  = custo estimado para atingir a meta a partir de  $n$
  - $f(n)$  = custo estimado do caminho que passa por  $n$  para atingir o estado meta
- A\* é uma estratégia de busca completa e ótima

# Busca A\*

A\* usa uma heurística admissível:

- $h$  não deve superestimar o custo real para se alcançar a solução: *heurística admissível* (otimista).

Ex.: distância em linha reta.

$h(n) \leq h^*(n)$ , sendo  $h^*(n)$  o custo real a partir de  $n$

# Espaço de busca

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

$$f=140+253$$
$$=393$$

**Sibiu**



**Arad**  $f=0+366$



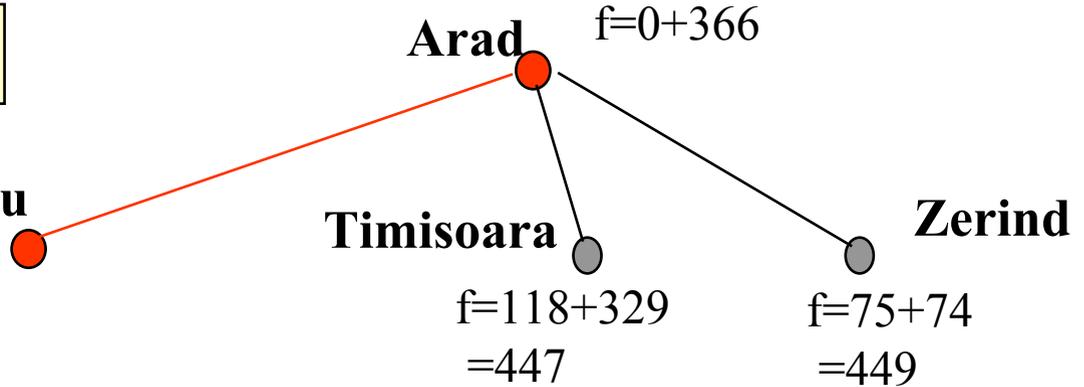
**Timisoara**

$$f=118+329$$
$$=447$$



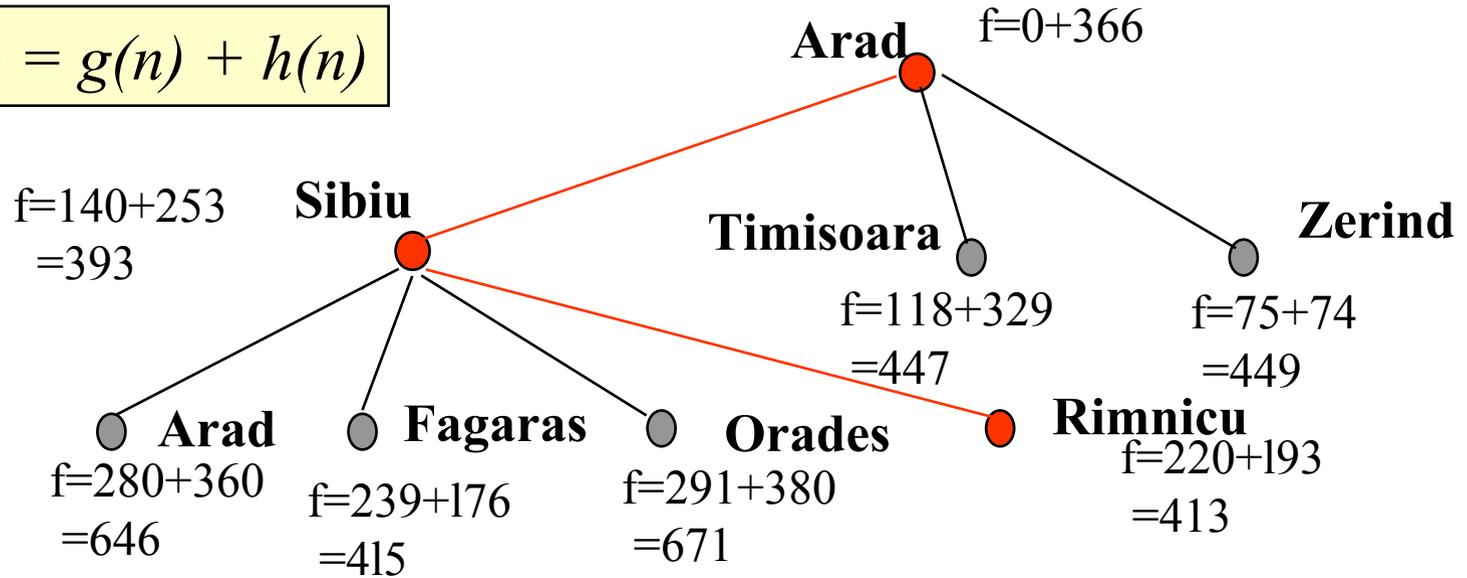
**Zerind**

$$f=75+74$$
$$=449$$



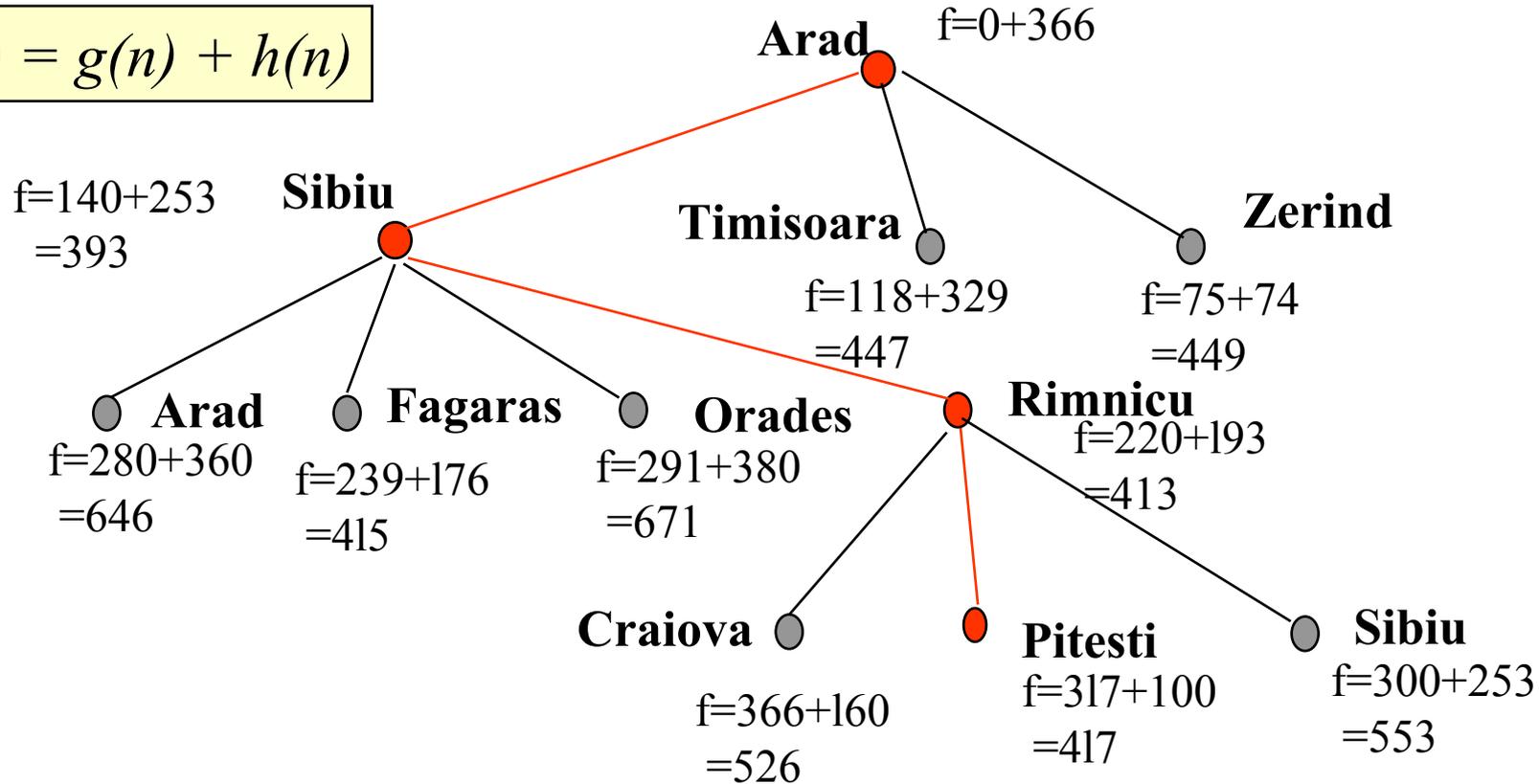
# Espaço de busca

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



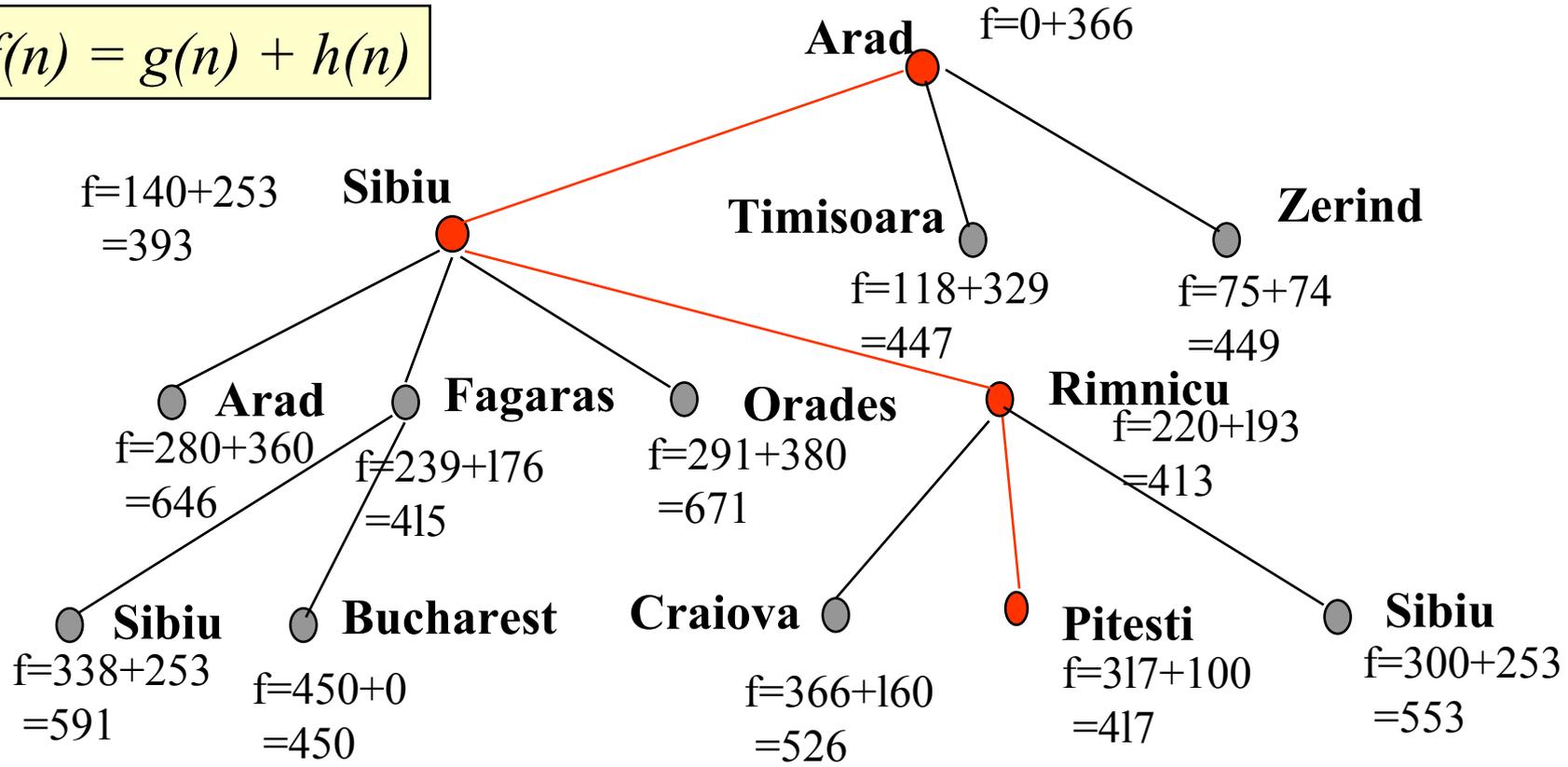
# Espaço de busca

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



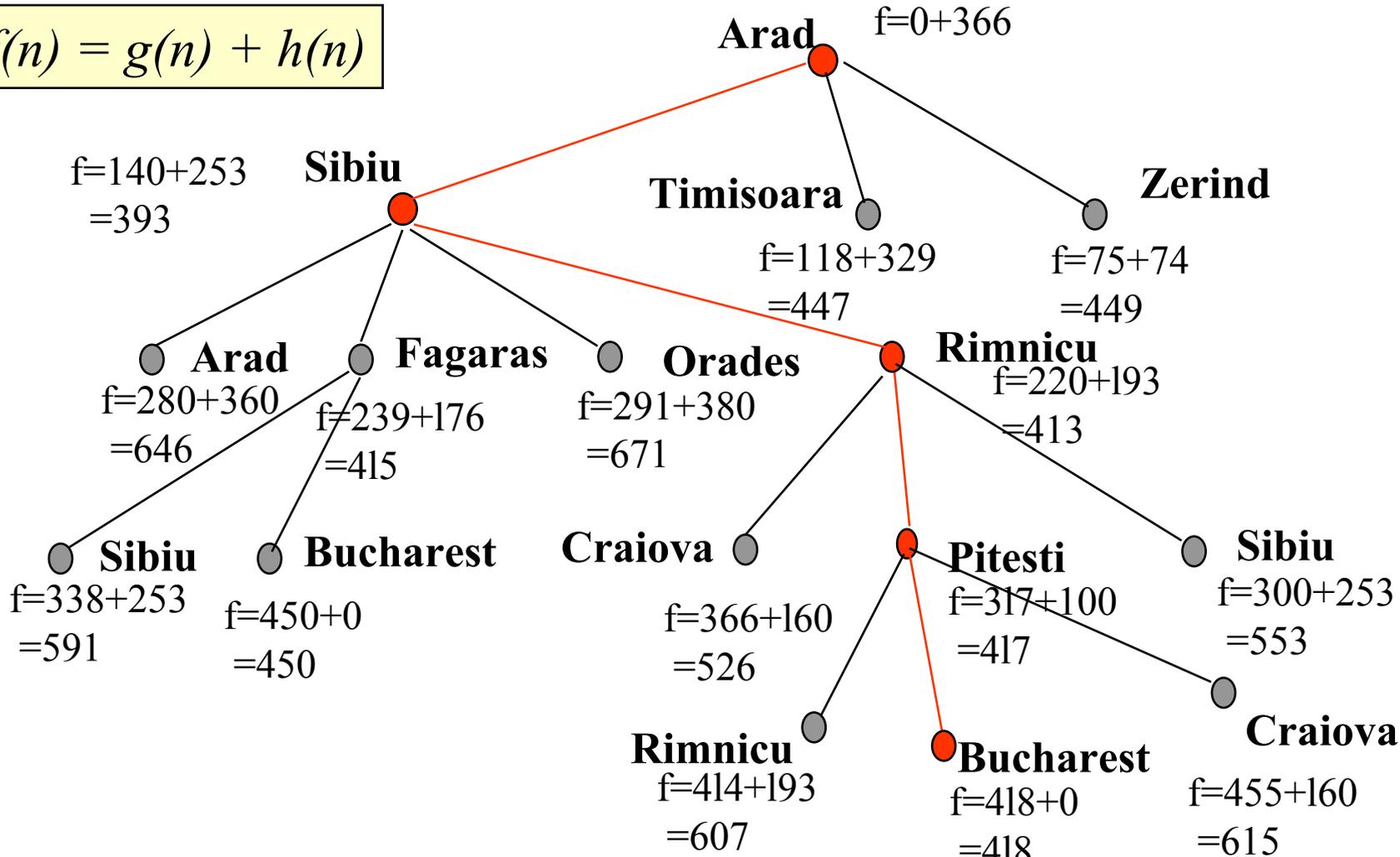
# Espaço de busca

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



# Espaço de busca

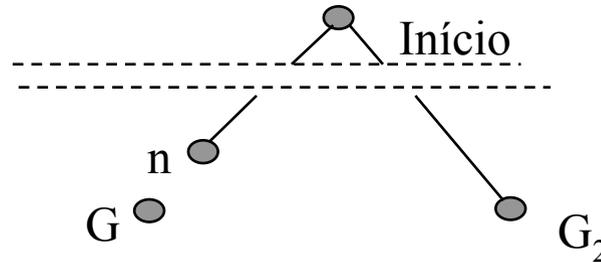
$$f(n) = g(n) + h(n)$$



# Busca $A^*$ - generalização

$g(n)$	$h(n)$	características
1	0	<i>BrFS</i>
custo	0	<i>Uniform Cost Search</i>
0	0	<i>random search</i>
$g(n)$	admissível	busca ótima de acordo com $g(n)$

# A\* encontra a solução ótima



- Seja  $G$  o estado meta ótimo e  $C^*$  o custo do da solução ótima
- Seja  $G_2$  o estado meta sub-ótimo, isto é:

$$f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$$

- Seja  $n$  um nó folha no caminho de  $G$  (que deixou de ser expandido pela busca que encontrou a solução em  $G_2$ ). Supondo que  $n$  não é escolhido quando comparado com  $G_2$ , logo:

(1) sendo  $h$  admissível:  $f(n) \leq C^*$

(2)  $n$  não é expandido:  $f(G_2) \leq f(n)$

de (1) e (2):  $f(G_2) \leq C^*$

como  $f(G_2) = g(G_2)$ :  $g(G_2) \leq C^*$  **contradição**

# Heurísticas para o 8-puzzle

- solução típica possui 20 passos
- fator de ramificação: 3 ou 4  $\Rightarrow \approx 3^{20} \Rightarrow \approx 3.5 \cdot 10^9$
- função heurística que nunca superestima a distância à meta ?

5	4	
6	1	8
7	3	2

Estado Inicial

1	2	3
8		4
7	6	5

Estado Meta

# 8-puzzle: Heurísticas admissíveis

$h_1(n)$  = número de pastilhas no lugar errado

$h_2(n)$  = total das distâncias de Manhattan

5	4	
6	1	8
7	3	2

Estado Inicial

1	2	3
8		4
7	6	5

Estado meta

$$h_1(I) = ?$$

$$h_2(I) = ?$$

# 8-puzzle: Heurísticas admissíveis

$h_1(n)$  = número de pastilhas no lugar errado (admissível)

$h_2(n)$  = total das distâncias de Manhattan (admissível)

5	4	
6	1	8
7	3	2

Estado Inicial

1	2	3
8		4
7	6	5

Estado Meta

$$h_1(I) = 7$$

$$h_2(I) = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$$

# Teste de função heurística

---

Se  $h_2(n) \geq h_1(n)$  para todo  $n$

então  $h_2$  domina  $h_1$  e é melhor para a busca (expande nós onde  $f(n) < f^*$ , ou seja,  $h(n) < f^* - g(n)$  )

Custo típico :

$d = 14$             IDS = 3.473.941 nós

$A^* (h_1) = 539$  nós

$A^* (h_2) = 113$  nós

$d = 24$             IDS = muitos nós

$A^* (h_1) = 39.135$  nós

$A^* (h_2) = 1.641$  nós

# Como “inventar” uma boa heurística?

---

Heurísticas admissíveis podem ser derivadas a partir do custo da solução exata de uma versão relaxada do problema (menos restrita)

- **Versão 1:** as pastilhas podem se mover para qualquer casa  $\Rightarrow h_1(n)$  é melhor
- **Versão 2:** as pastilhas podem se mover para qualquer casa adjacente (incluindo casas ocupadas)  $\Rightarrow h_2(n)$  é melhor

# Geração automática de heurísticas

---

- Problema descrito em linguagem formal
- Exemplo de três problemas relaxados com a remoção de 1 ou mais condições:
  - (a) mover de A para B *se* A é adjacente a B
  - (b) mover de A para B *se* B está vazio
  - (c) mover de A para B
- ABSOLVER gerou a melhor heurística para o 8-puzzle e a primeira heurística útil para o Cubo Mágico

# A melhor heurística

---

- Heurísticas admissíveis:

$$h_1(n), \dots, h_m(n)$$

- Heurística composta

$$h(n) = \max( h_1(n), \dots, h_m(n) )$$

- Análise estatística da execução do método com diferentes heurísticas para um número grande de casos

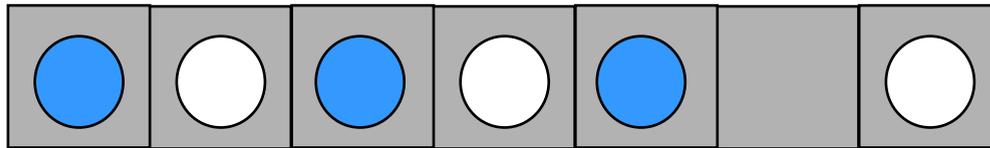
# Custo da função heurística

---

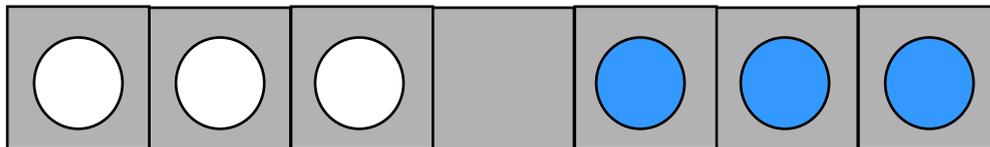
- Suposição: custo de se calcular  $h(n)$  = ao custo de se expandir um nó
- nem sempre minimizar o número de nós a serem expandidos com uma boa heurística é uma boa coisa  $\implies$  o cálculo de  $h(n)$  pode ser equivalente a expandir centenas de nós
- uma heurística precisa: heurística que faz busca em largura “*on the fly*”!!
- Uma boa heurística deve ser eficiente e precisa!

# Função heurística para régua-puzzle

Quais seriam algumas funções de avaliação para o régua-puzzle ?



Estado corrente



Estado Meta

# Algumas variações do A\*

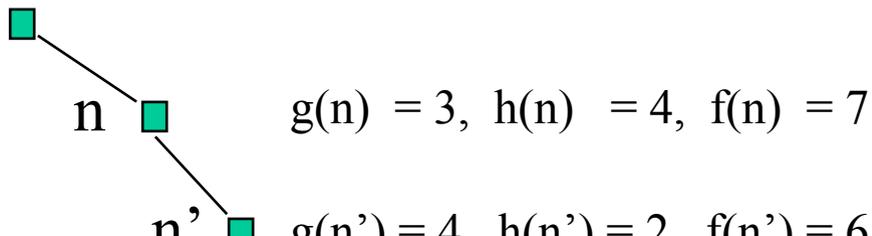
---

- Apesar da existência de algoritmos de busca informada, alguns problemas são realmente difíceis por natureza
- Dois algoritmos utilizados para economizar memória.
  - IDA\* extensão da Busca em Profundidade Iterativa (IDS) que usa informação heurística
  - SMA\*, é similar ao A\*, mas restringe o tamanho da lista de nós para caber na memória disponível.

# Busca A\*

- A função de avaliação deve ser monotônica!
- Funções com heurísticas admissíveis, em geral, são monotônicas
- Pode existir uma heurística não monotônica:  $f(n)$  decresce. Para eliminar problemas desse tipo devemos usar o custo do pai ao invés do calculado para o seu sucessor, ou seja:

$$f(n') = \max(f(n), g(n') + h(n')) \quad (\text{equação do } \mathit{pathmax})$$

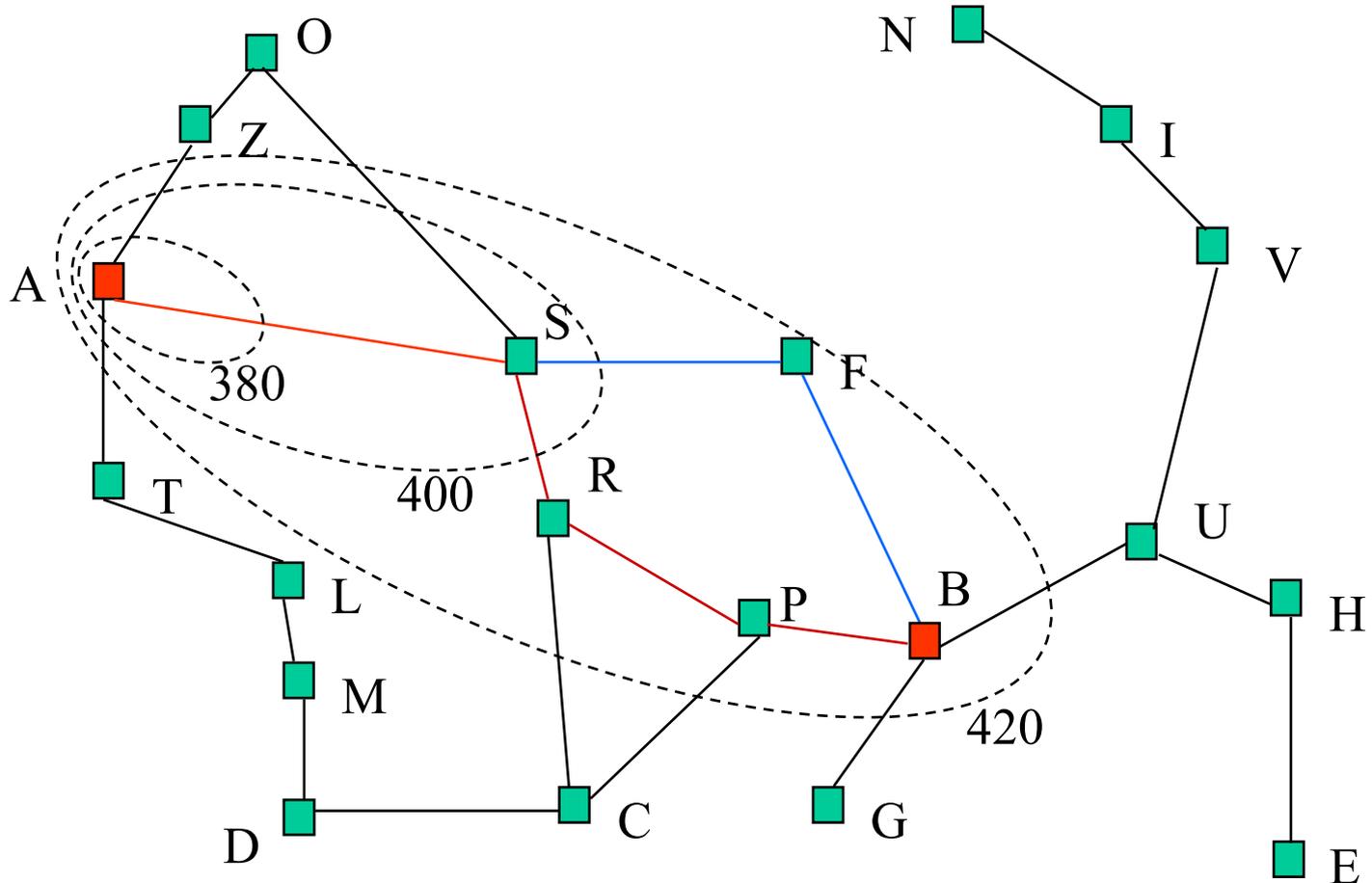


# Contorno no espaço de estados

---

- Garantindo que  $f$  nunca decresce (monotonicidade), podemos desenhar **contornos** no espaço de busca

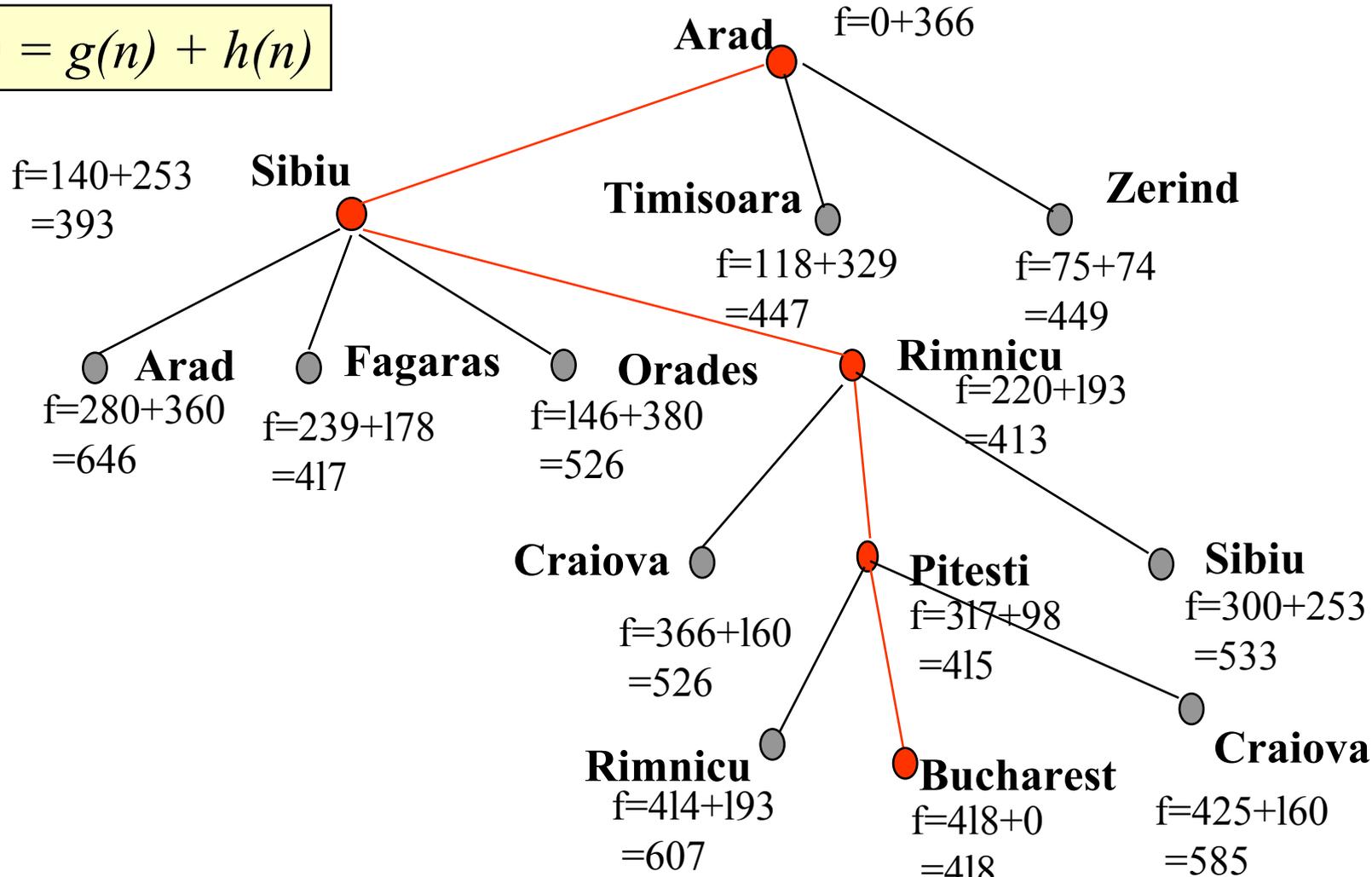
# Contornos no espaço de estados



Note como os contornos se "esticam" em direção à meta.

# Espaço de busca

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



# Busca $A^*$

---

- Para  $f^*$  sendo o custo do caminho da solução ótima,  $A^*$  expande todos os nós com  $f(n) < f^*$ .
- $A^*$  pode expandir alguns nós dentro do contorno do estado meta antes de selecionar o estado meta  $\implies$  a primeira solução encontrada é a ótima
- nenhum outro algoritmo garante expandir menos nós do que  $A^*$

# Busca $A^*$

---

- Para a maioria dos problemas de busca o número de nós dentro do contorno da solução é exponencial com o tamanho da solução

# Busca em Profundidade Iterativa

## A\* (IDA\*)

---

- IDS reduz requisitos de memória.
- Em IDA\* cada iteração é DFS, como na busca IDS.
- Ao invés de adotar valores crescentes de profundidade limite, IDA\* adota valores de função de avaliação limite.
- a cada iteração expande-se todos os nós dentro de um **contorno** no espaço de estados.

# O algoritmo IDA\*

**Função** IDA\* (*problema*) **devolve** a sequência da solução

**Entradas:** *problema*

**Variáveis locais:** *f-limite* (limite atual de *f*)

*raiz* (um nó)

*raiz* ← Faça-nó (Estado-Inicial(*problema*))

*f-limite* ← *f*(*raiz*)

**laço faça**

*solução, f-limite* ← DFS-contorno(*raiz, f-limite*)

**Se** *solução* não nula **então devolve** *solução*

**Se** *f-limite* =  $\infty$  **então devolve** falhou; fim

# Função DFS-contorno

**Função** DFS-contorno(*nó*, *f-limite*) **devolve** a solução e um novo *f-limite*

**Entradas:** *nó*, *f-limite* (limite corrente)

**Variáveis locais:** *next-f* (próximo valor de *f-limite*, inicialmente  $\infty$ )

**Se**  $f[nó] > f-limite$  **então devolve** nulo,  $f(nó)$  /\*mudou de contorno\*/

**Se** Testa-meta(*nó*) **então devolve** *nó*, *f-limite* /\*achou solução \*/

**Para cada** *nó* *s* **em** Sucessores(*nó*) **faça**

*solução*, *new-f*  $\longleftarrow$  DFS-contorno(*s*, *f-limite*)

**Se** *solução* não nula **então devolve** *solução*, *f-limite*

**Senão** *next-f*  $\longleftarrow$  Min(*new-f*, *next-f*); **fim**

**devolve** nulo, *next-f* /\*busca em contorno maior\*/

# IDA\*

---

completo e ótimo como A\*

porque usa DFS requer espaço proporcional ao caminho mais longo a ser explorado

complexidade de tempo depende do cálculo da heurística

Tipicamente  $f$  cresce 2 ou 3 vezes

IDA\* guarda menos nós que A\* e encontra a solução ótima mais rapidamente (não usa lista de prioridades)

# SMA\* - Simplified Memory-Bounded A\*

---

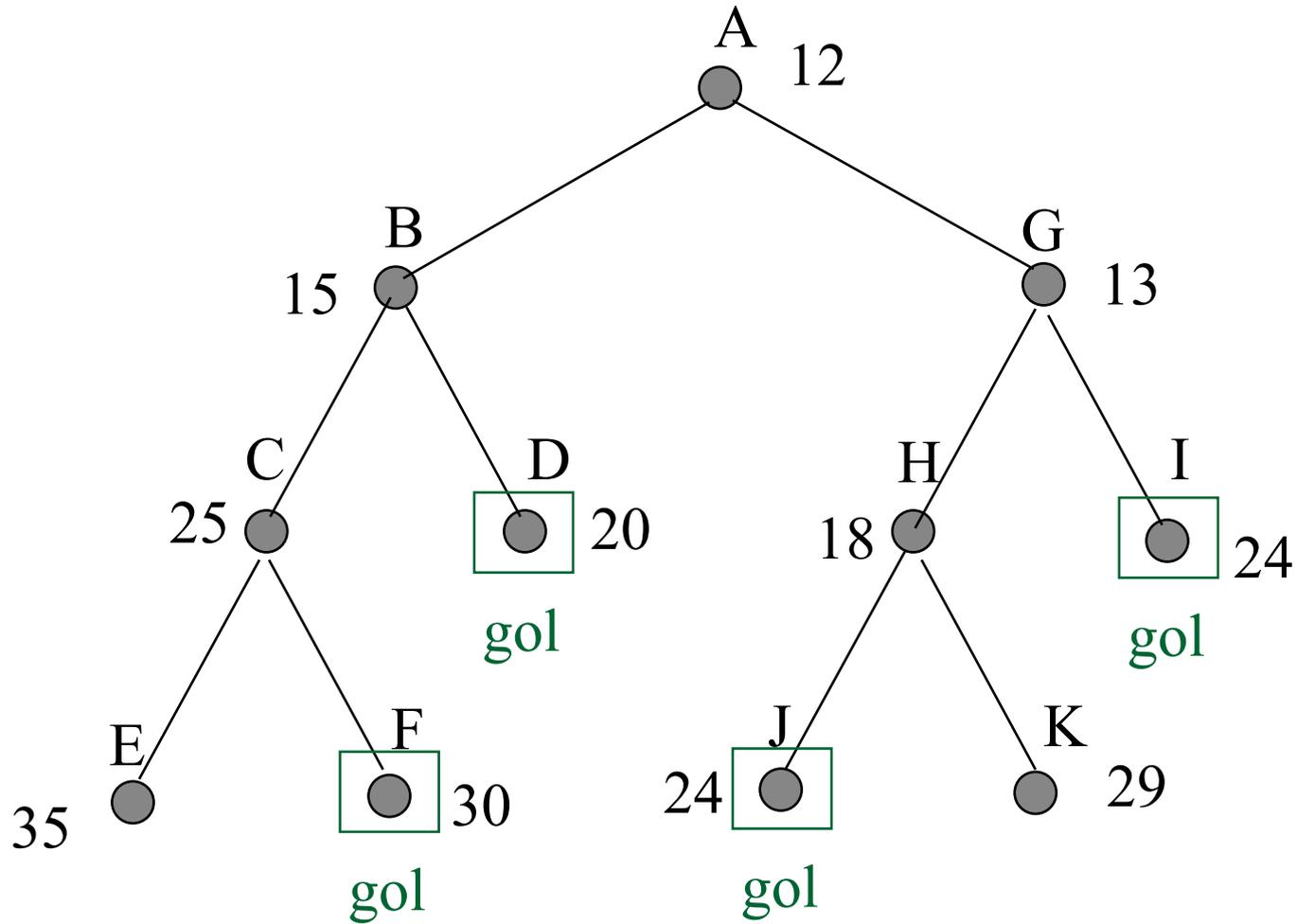
- somente usa a memória disponível para busca (quanto mais memória disponível, melhor a eficiência)
- evita explorar nós repetidos
- é completo se houver memória para guardar o caminho da solução de menor profundidade
- é ótimo se houver memória para guardar o caminho da solução ótima
- melhor desempenho que A\* (se existir suficiente memória disponível)

# SMA\* - estratégia

---

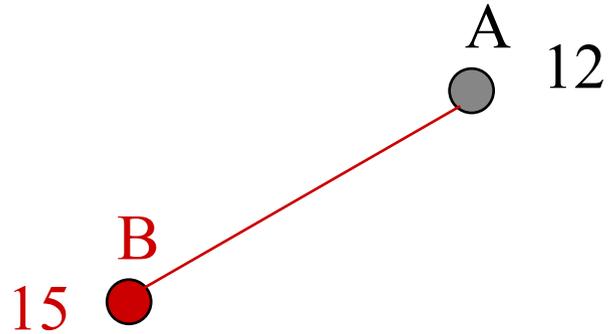
- para gerar um sucessor sem existir espaço na memória: joga fora nós com alto  $f$ -cost (*nós esquecidos*)
- guarda nos nós ancestrais informação sobre a qualidade do melhor caminho da sub-árvore esquecida
- somente regenera a sub-árvore quando todos os outros caminhos se mostraram piores que o caminho esquecido

# SMA\* - com memória de três nós

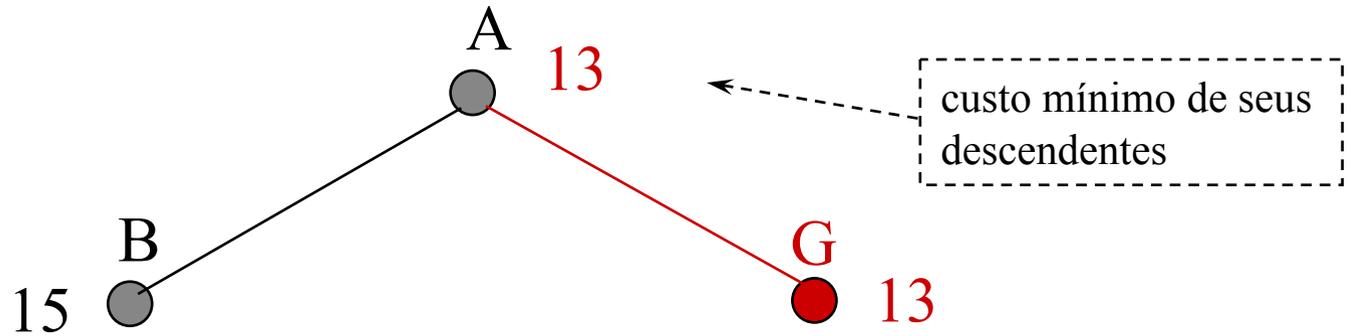


# SMA\* - progresso

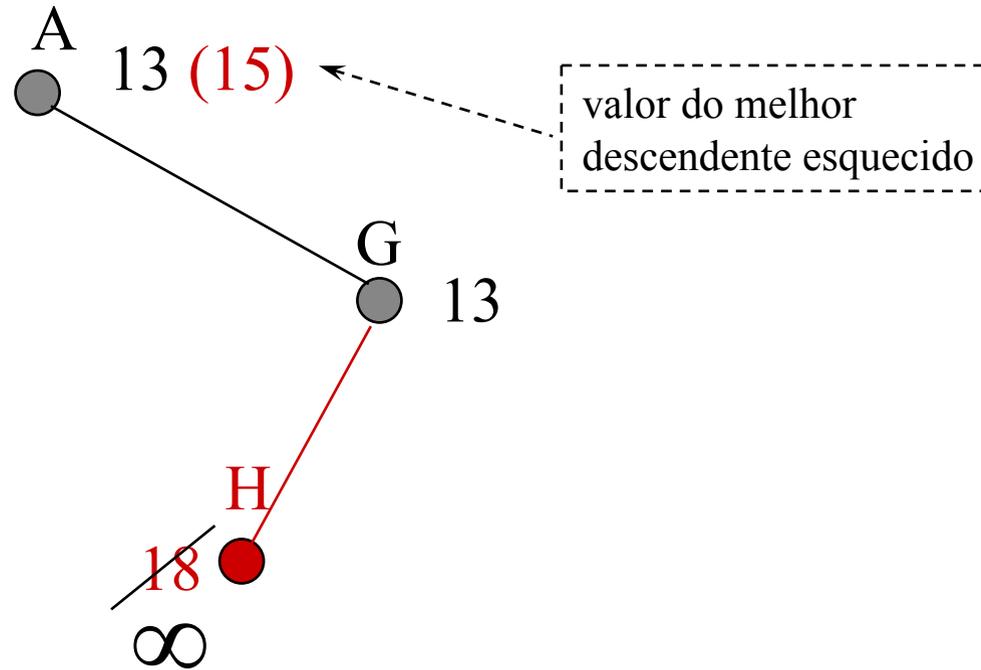
---



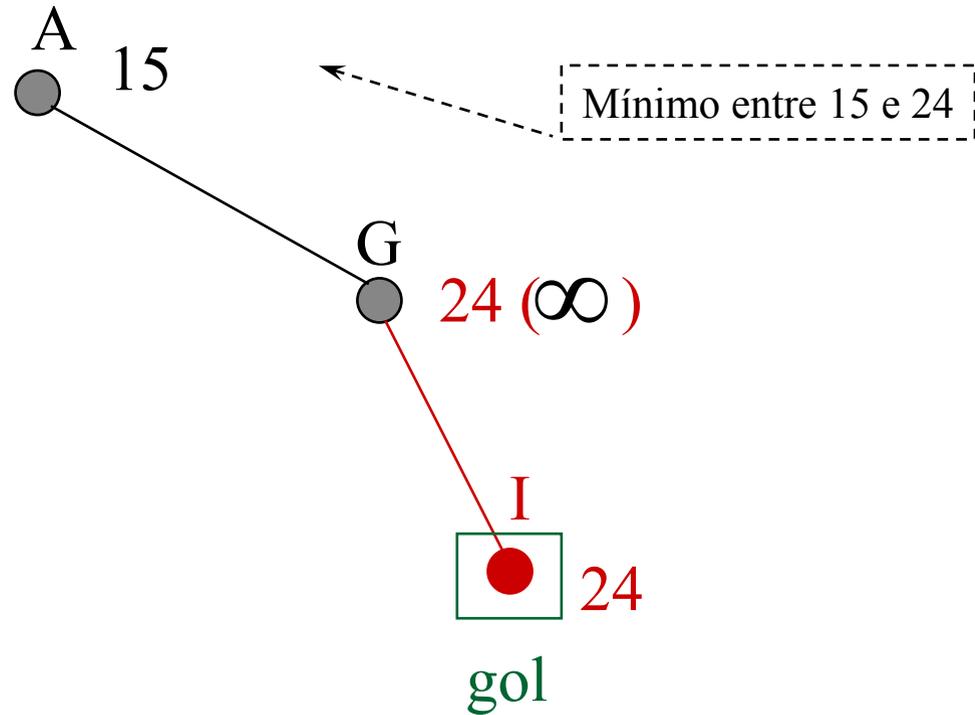
# SMA\* - progresso



# SMA\* - progresso

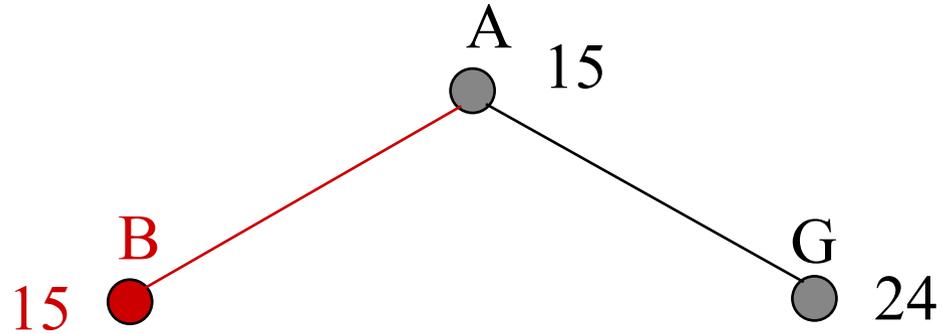


# SMA\* - progresso



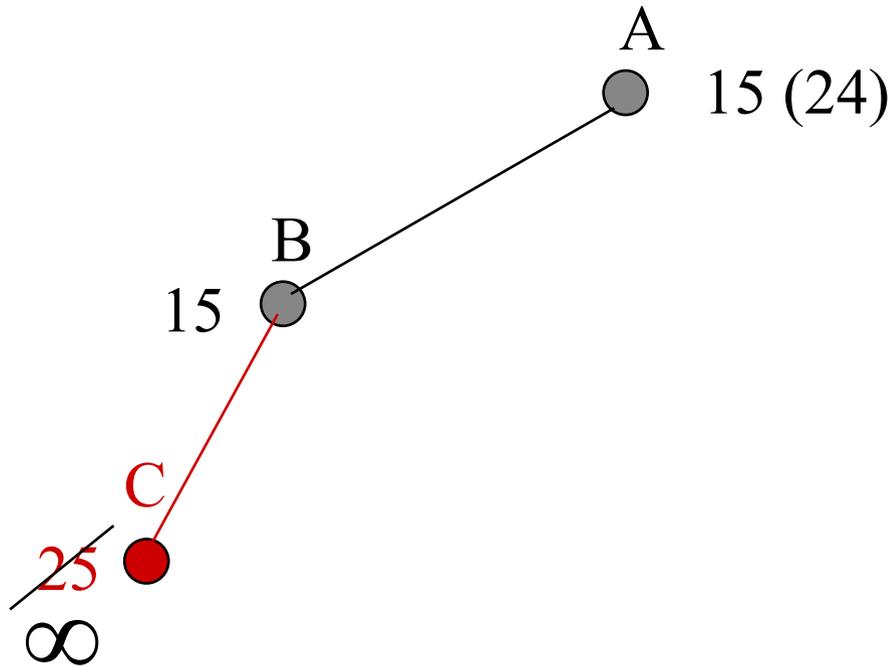
# SMA\* - progresso

---



# SMA\* - progresso

---



# SMA\* - progresso

