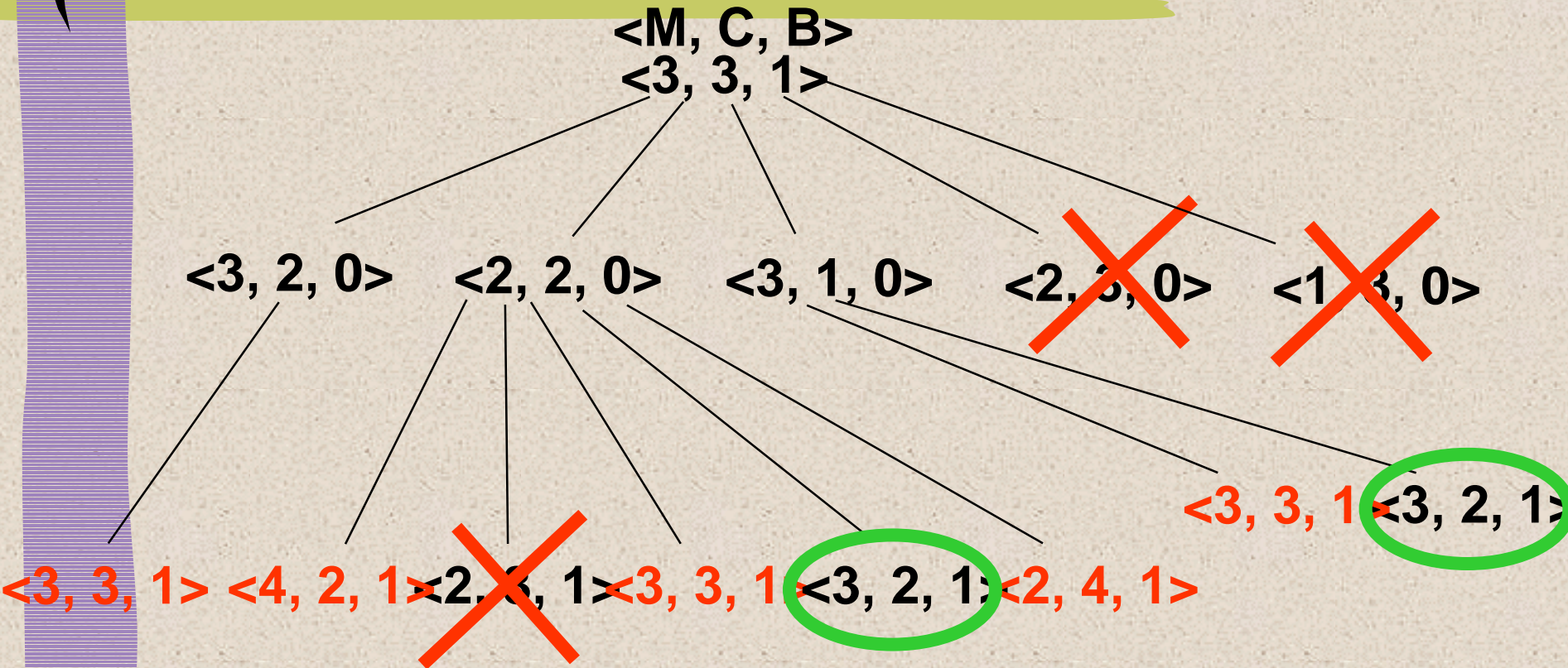


# Canibais e Missionários



# Busca de Custo Uniforme

- Completo?

- SIM. se o custo do passo  $\geq \epsilon$

- Tempo?

- nro de nodos com custo  $g \leq$  solução de custo ótimo

- Espaço?

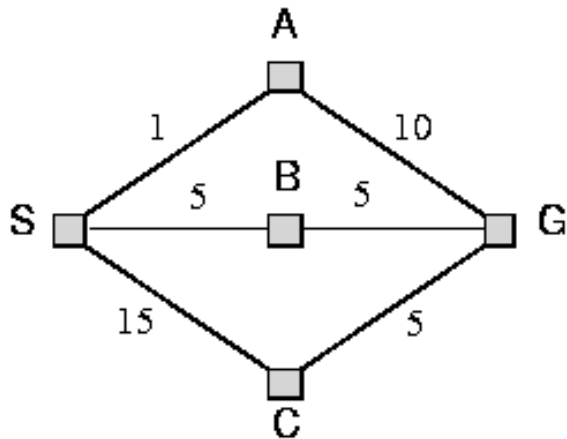
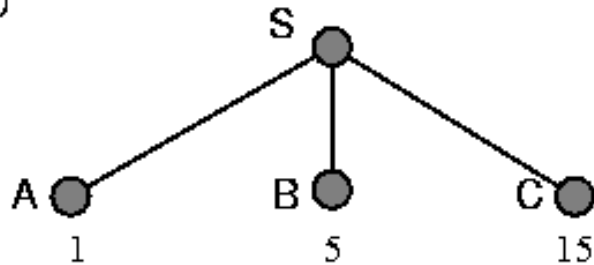
- nro de nodos com custo  $g \leq$  solução de custo ótimo

- Ótima?

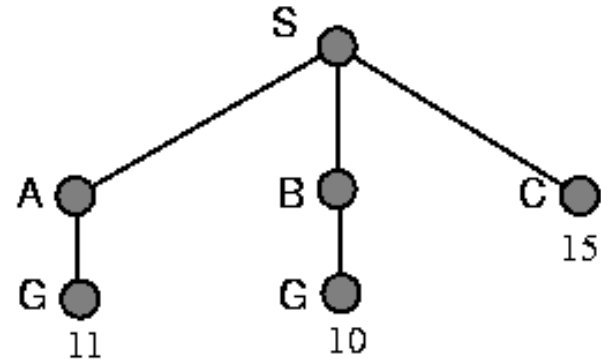
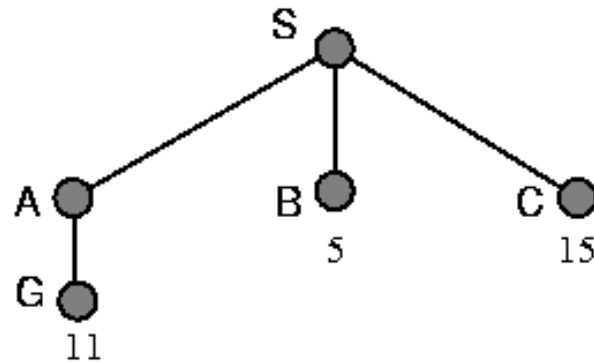
- SIM



# Busca de Custo Uniforme



(a)



(b)

# Métodos de Busca Informada

- # Busca o melhor primeiro
- # Busca A\*
- # Heurística
- # Subida de Montanha (Hill Climbing)
- # Simulated Annealing

# Relembrando a Busca Geral

```
function BUSCA_GERAL(problema, FILA_FN) returns uma solução, ou falha
nodos ← FAZ_FILA(FAZ_NODO(ESTADO_INICIAL[problema]))
loop
  if nodos = vazio
  then return falha
  nodo ← REMOVE_DA_FRENTE(nodos)
  if TESTE_OBJETIVO[problema] aplicado ao ESTADO(nodo) obtiver sucesso
  then return nodo
  nodos ← POE_NA_FILA(nodos, EXPANDE(nodo, OPERADORES[problema]))
end
```

# **Uma estratégia é definida escolhendo-se a ordem de expansão do nodo**

# Busca o Melhor Primeiro

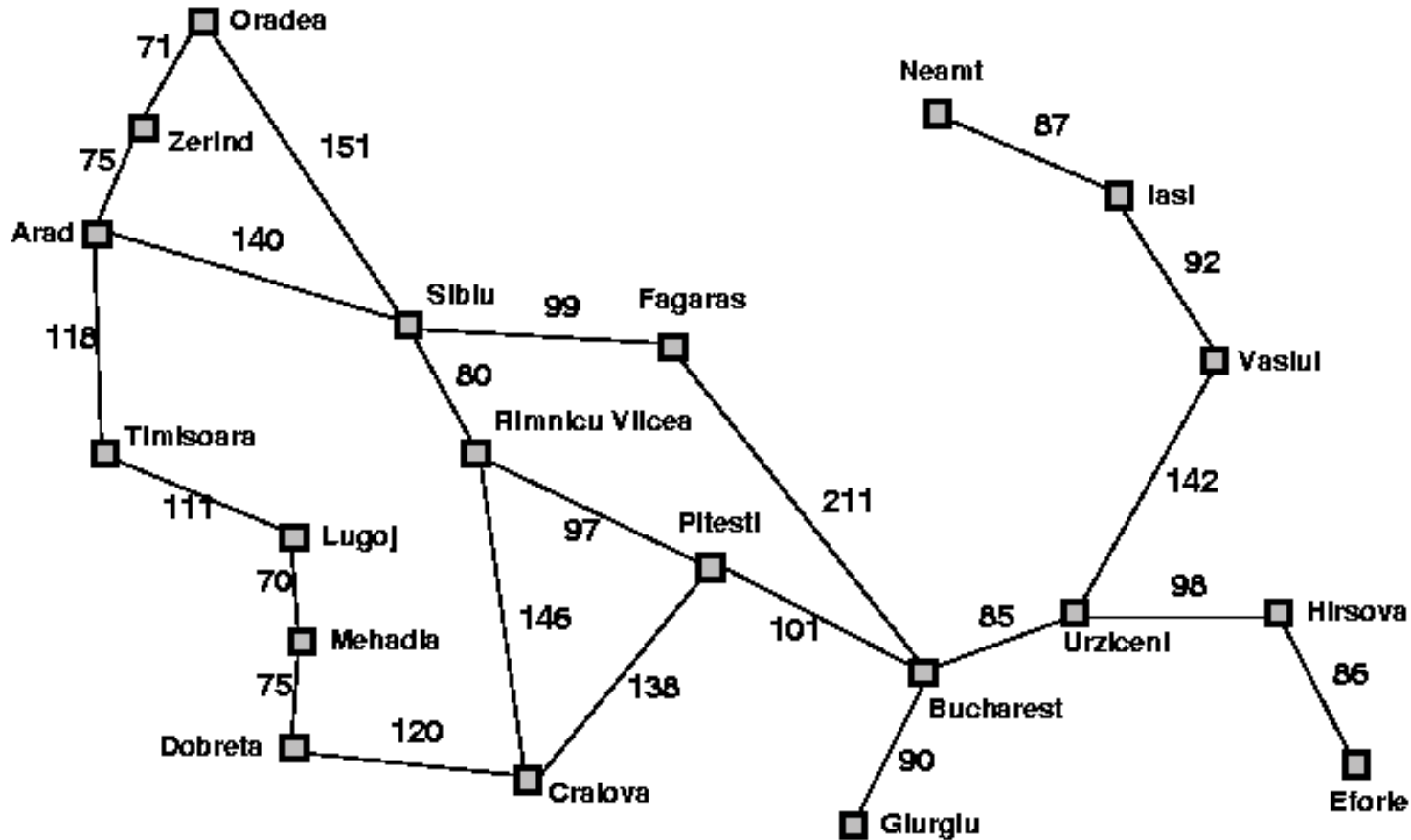
## # IDÉIA:

- ▣ Usar uma função de avaliação para cada nodo
  - ▣ estimativa de “quão desejável” é aquele nodo
- ▣ Expande o nodo mais desejável ainda não expandido

## # IMPLEMENTAÇÃO:

- ▣ POE\_NA\_FILA
  - ▣ Insere os sucessores na ordem decrescente de “desejabilidade”
- ▣ CASOS ESPECIAIS
  - ▣ Busca Greedy
  - ▣ Busca A\*

# A Romênia com custos em Km



Distância até  
Bucarest em  
linha reta

Arad	366
Bucarest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

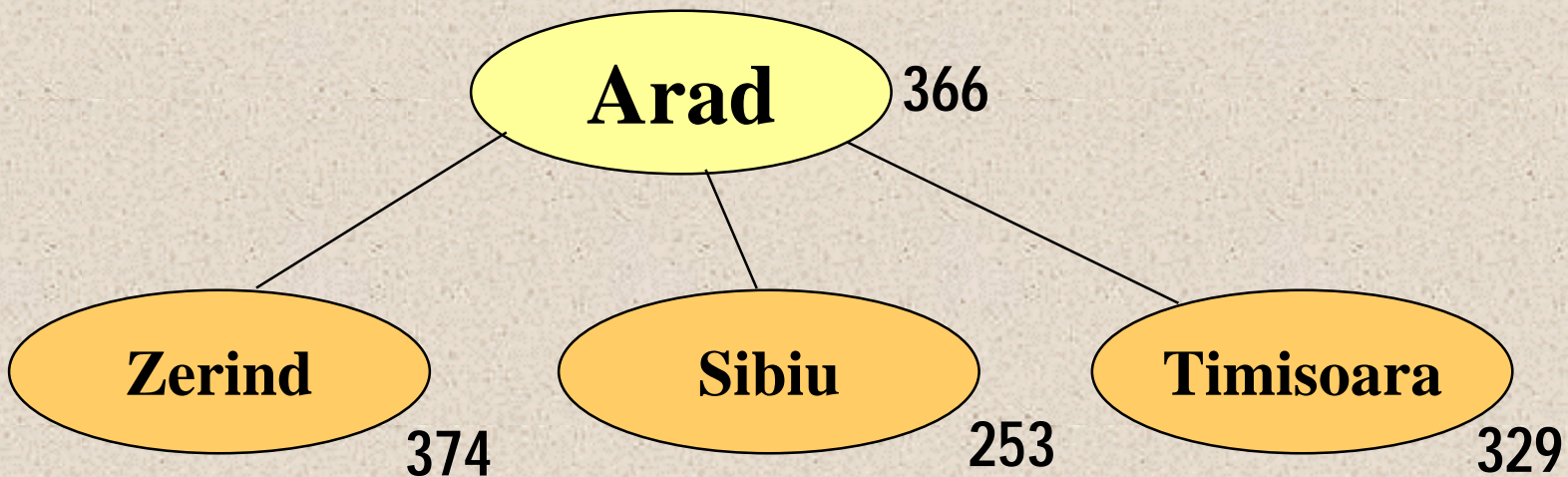
# Busca Greedy

- # **Função de Avaliação  $h(n)$  – (Heurística)**
  - ▣ **estimativa de custo para se chegar ao objetivo partindo-se de  $n$**
  
- # **Por exemplo:**
  - ▣  **$h_{DLR}(n)$  = distância em linha reta de  $n$  até Bucarest**
  
- # **A busca greedy (gulosa) expande o nodo que parece estar mais próximo do objetivo**

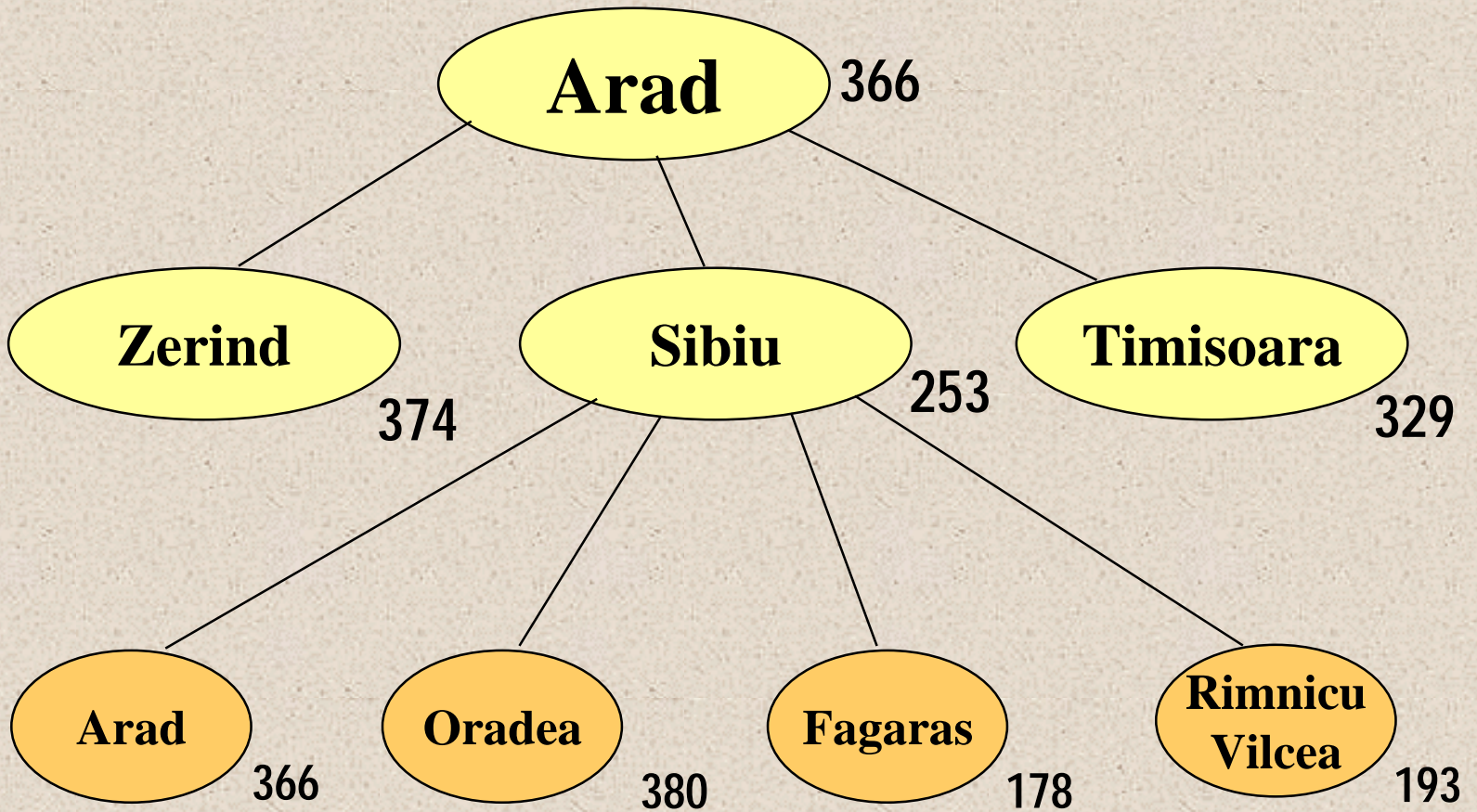
# Exemplo de Busca Greedy

**Arad** 366

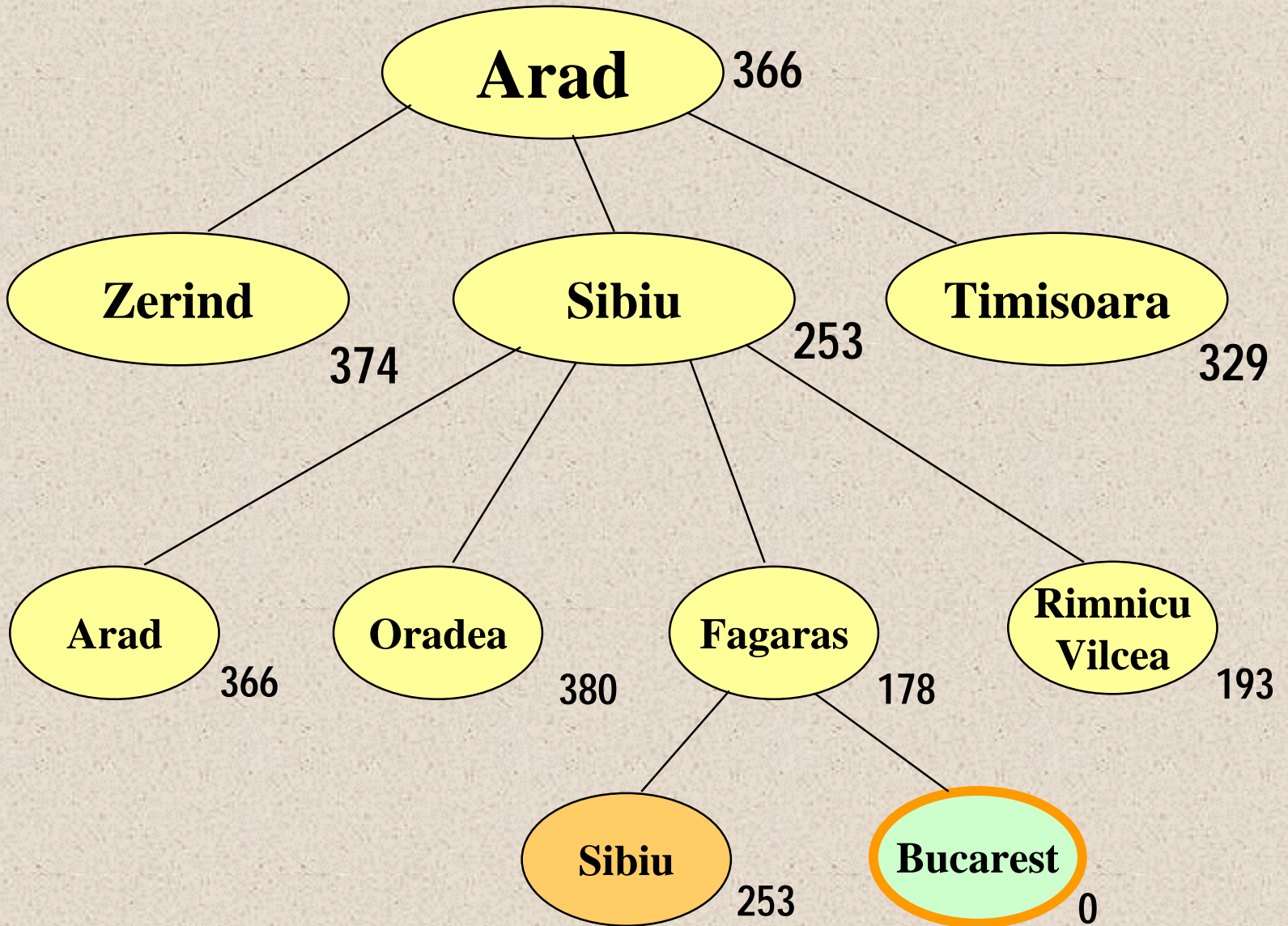
# Exemplo de Busca Greedy



# Exemplo de Busca Greedy



# Exemplo de Busca Greedy



# Propriedades da Busca Greedy

# Completa?

# Tempo?

# Espaço?

# Ótima?

# Propriedades da Busca Greedy

## # Completa?

- ▣ NÃO! Pode ficar presa em Loops
- ▣ Ex: lasi – Neamt – lasi – Neamt - .....
- ▣ Completo no espaço finito com verificação de repetições

## # Tempo?

- ▣  $O(b^m)$ , mas uma boa heurística pode melhorar muito

## # Espaço?

- ▣  $O(b^m)$ , mantem todos os nodos na memória

## # Ótima?

- ▣ Não

# Busca A\*

## # IDÉIA:

- Evitar expandir caminhos que já estão custosos
- Função de Avaliação  $f(n) = g(n) + h(n)$ 
  - $g(n)$  = custo de alcançar  $n$   
partindo do estado inicial
  - $h(n)$  = custo estimado para se atingir  
o objetivo a partir de  $n$
  - $f(n)$  = custo total estimado do caminho

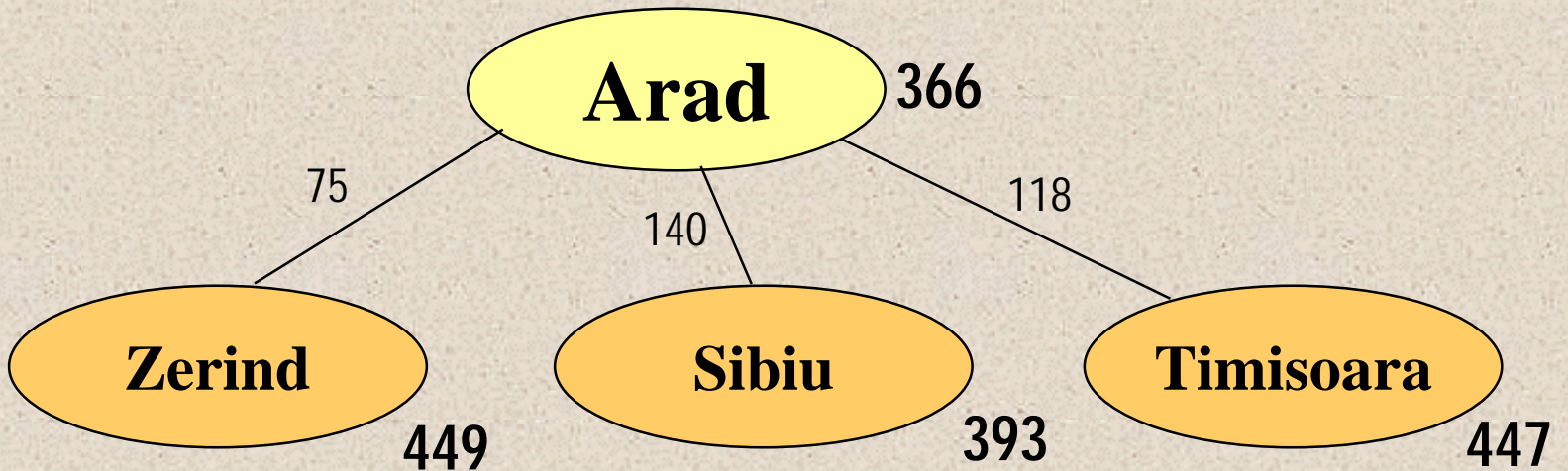
## # A\* usa uma heurística admissível

- isto é,  $h(n) \leq h^*(n)$  onde  $h^*(n)$  é o custo real de  $n$
- Por exemplo:
  - $h_{DLR}(n)$  nunca superestima a distância real pela estrada
- Teorema: A Busca A\* é ótima

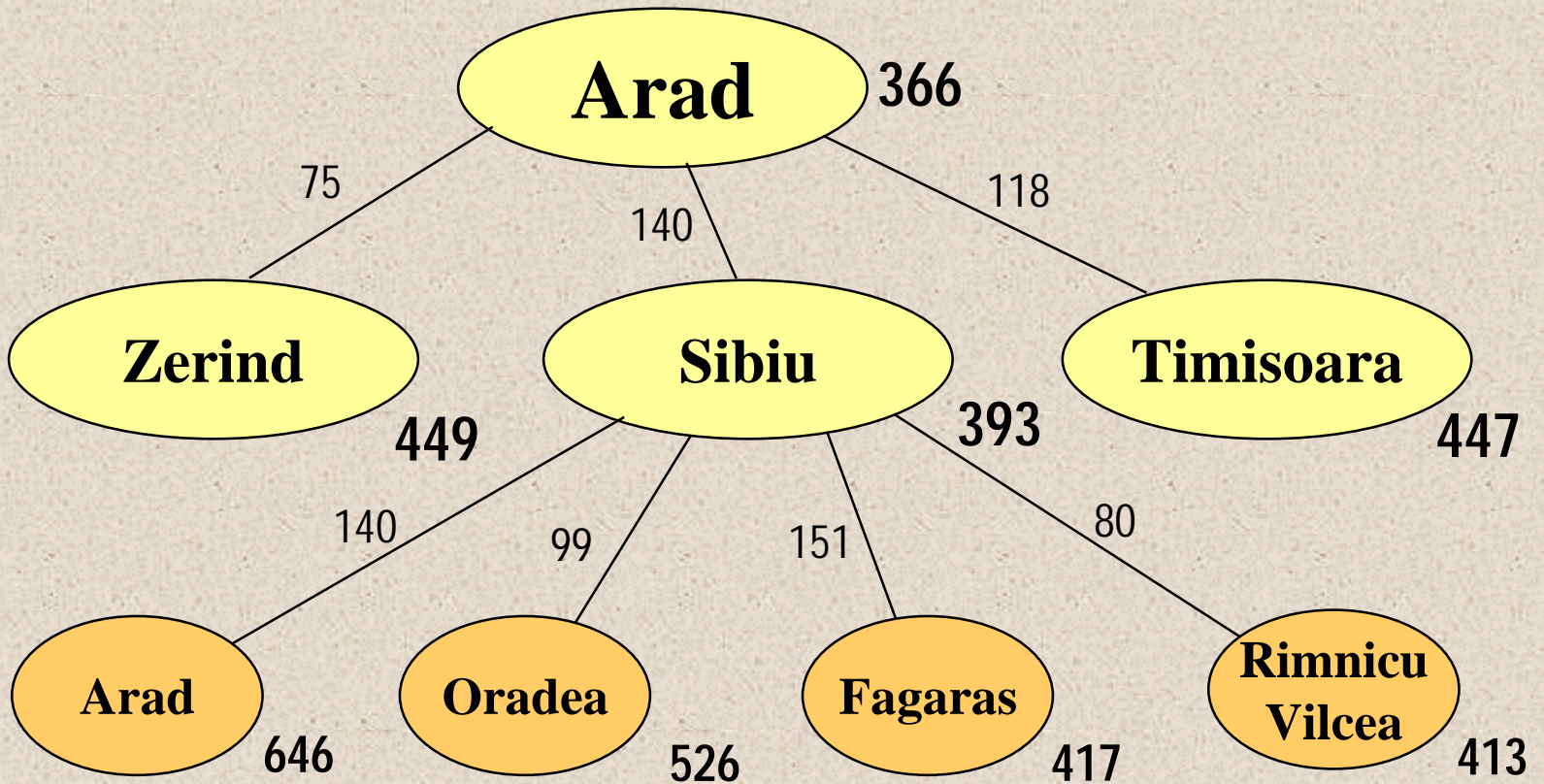
# Exemplo de Busca A\*

**Arad** 366

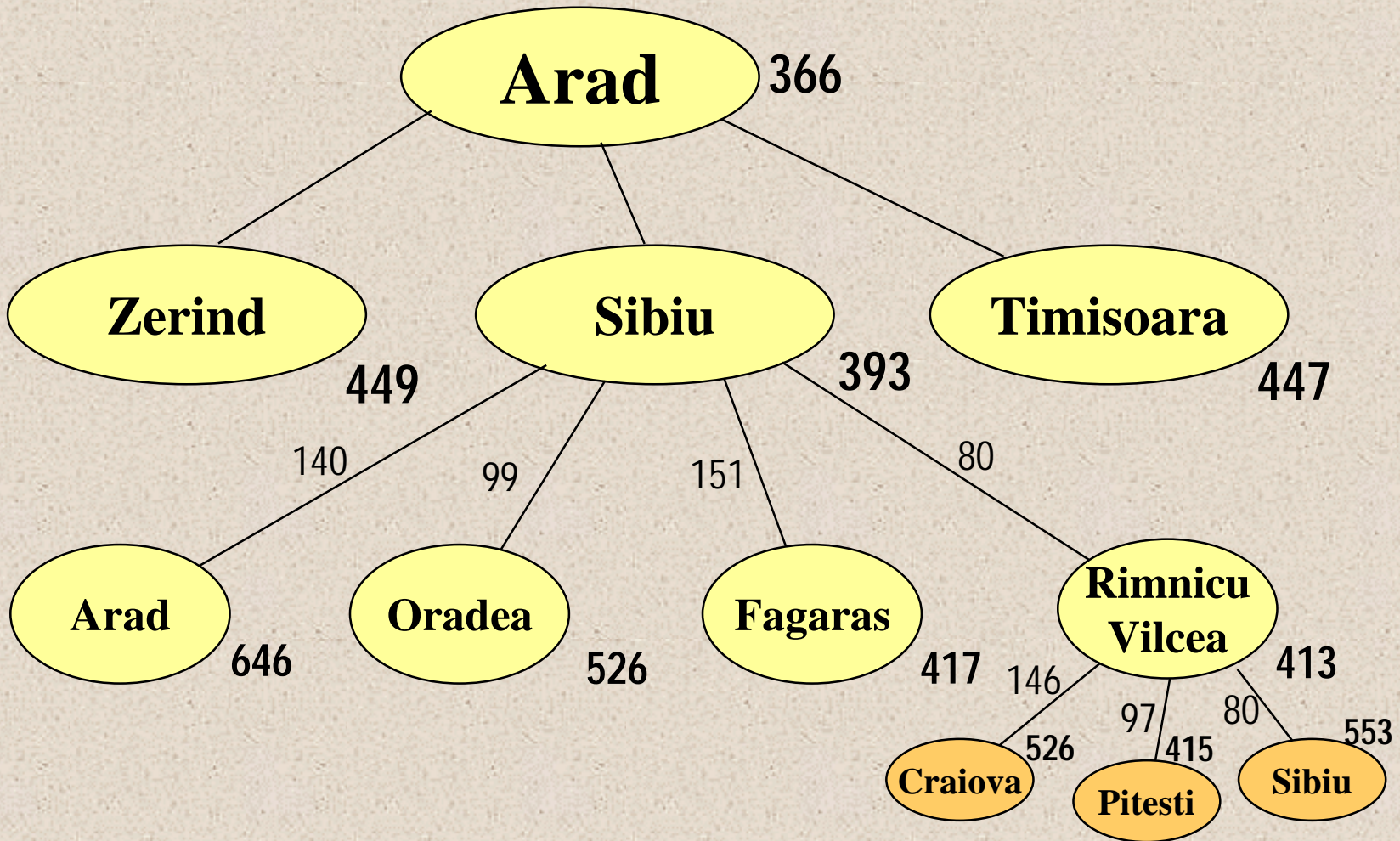
# Exemplo de Busca A\*



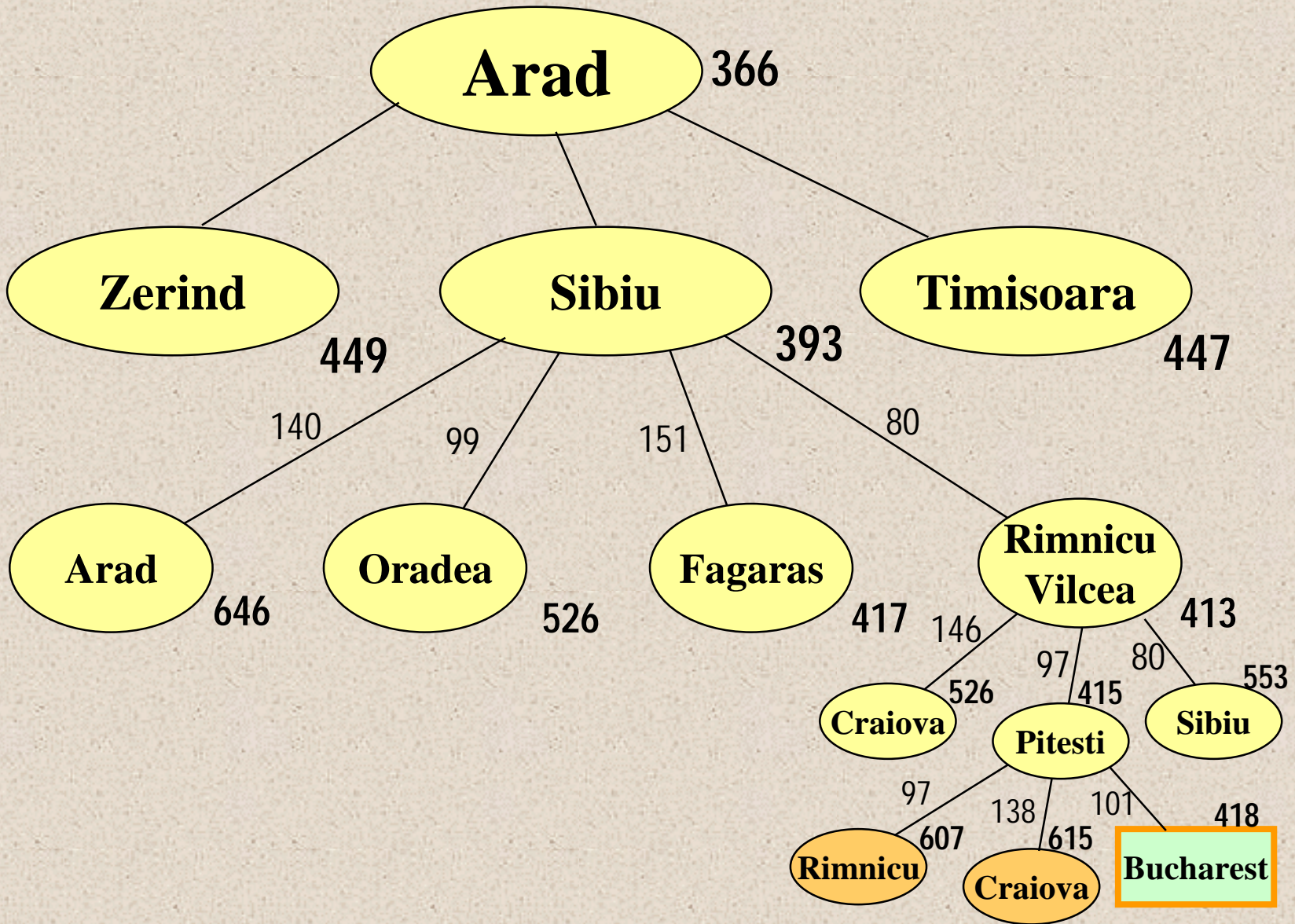
# Exemplo de Busca A\*



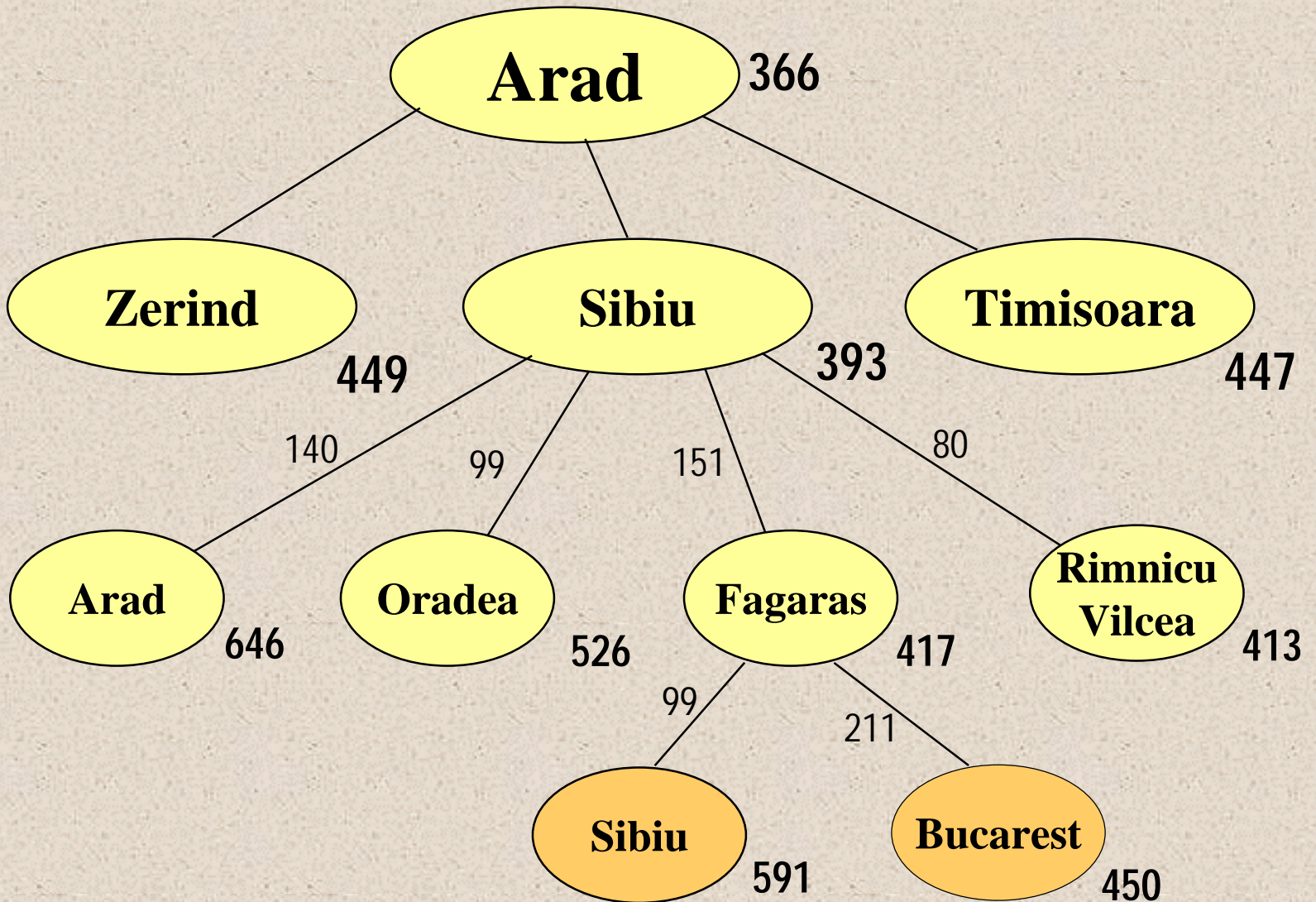
# Exemplo de Busca A\*



# Exemplo de Busca A\*

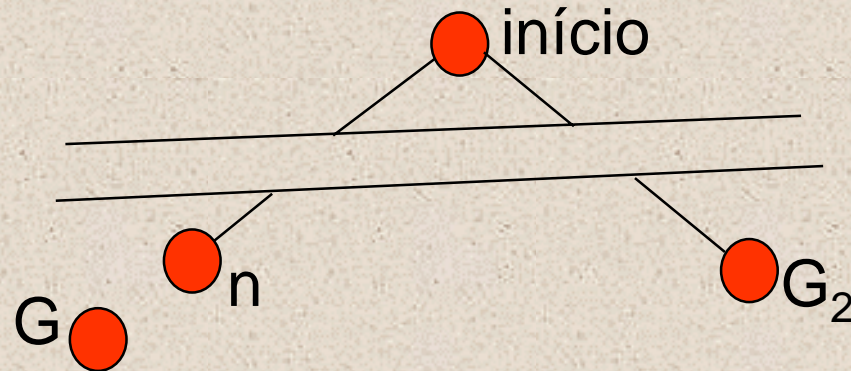


# Exemplo de Busca A\*



# Optimalidade de $A^*$ (prova padrão)

- # Suponha que algum objetivo subótimo  $G_2$  tenha sido gerado e está na fila.
- # Seja  $n$  um nodo não expandido sobre um caminho mais curto para um objetivo ótimo  $G$ .



$A^*$  expande todos os nodos  $n$  com  $f(n) < f^*$

- #  $f(G_2) = g(G_2)$   
 $> g(G)$   
 $\geq f(n)$

já que  $h(G_2) = 0$   
já que  $G_2$  é subótimo  
já que  $h$  é admissível

- # Como  $f(G_2) > f(n)$ ,  $A^*$  nunca selecionará  $G_2$  para expandir

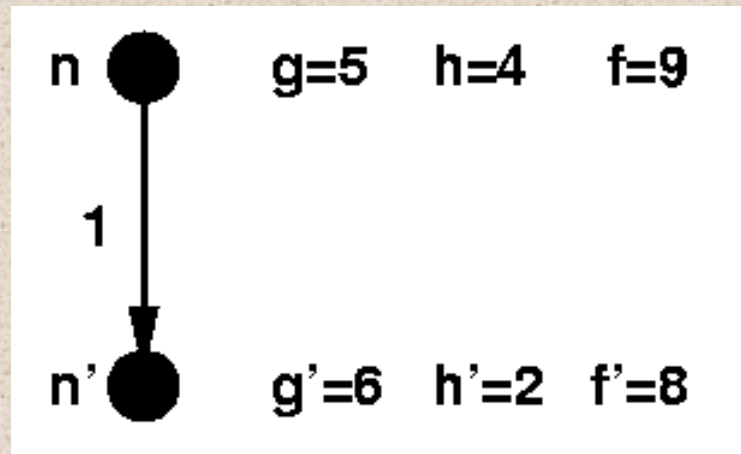
# Monotonicidade

## # Quando

- ▣ Ao longo de qualquer caminho a partir da raiz,  $f$  nunca decresce
- ▣ Há alguma heurística admissível (quase sempre)
- ▣ Diz-se que a heurística exibe Monotonicidade
- ▣ A heurística é monotônica sse obedece a desigualdade triangular
  - ▣  $d(a,b) = d(b,a)$  se  $b \neq a$ ;
  - ▣  $d(a, b) = 0$  se  $a = b$ ;
  - ▣  $d(a, c) \leq d(a, b) + d(b, c)$

# Prova do Lema: Pathmax

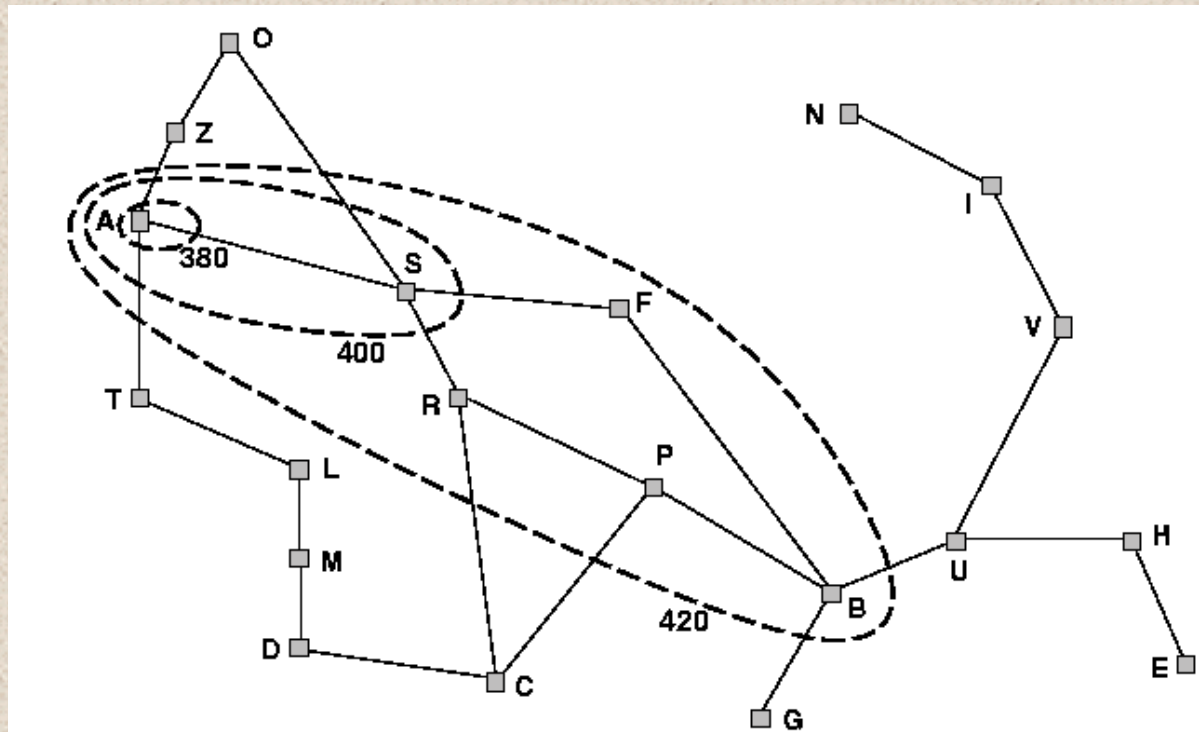
- # Para algumas heurísticas admissíveis,  $f$  pode decrescer ao longo do caminho
  - ▣ Suponha que  $n'$  seja um sucessor de  $n$



- ▣ Mas v. está jogando informação fora!!!
- ▣  $f(n) = 9 \Rightarrow$  o custo real de um caminho até  $n$  é  $\geq 9$
- ▣ Logo o custo real de um caminho até  $n'$  é  $\geq 9$  também
- # Modificação Pathmax ao A\*:
  - ▣ Em vez de  $f(n') = g(n') + h(n')$
  - ▣ Usar  $f(n') = \text{MAX}[g(n') + h(n'), f(n')]$
- # Com Pathmax,  $f$  é sempre não-decrescente ao longo de qualquer caminho

# Optimalidade de $A^*$ (mais útil)

- # Lema:  $A^*$  expande nodos em ordem crescente de valor de  $f$
- # Adiciona gradualmente “contornos- $f$ ” dos nodos (analogamente, primeiro-na-largura adiciona camadas)
- # O contorno  $i$  conterá todos os nodos com  $f = f_i$  onde  $f_i < f_{i+1}$



# Propriedades de $A^*$

## # Completo?

- ▣ SIM, a menos que sejam infinitos nodos para os quais  $f \leq f(G)$

## # Tempo?

- ▣ Exponencial em [erro relativo de  $h$  x tamanho da solução]

## # Espaço?

- ▣ Mantem todos os nodos na memória

## # Ótimo?

- ▣ SIM. Não pode expandir  $f_{i+1}$  enquanto não expandir  $f_i$

# não acredite em estatísticas (paradoxo de Simpson)

<b>Population</b>	<b>New York</b>	<b>Richmond</b>
White	4,675,174	80,895
Colored	91,709	46,733
Entirely	4,766,883	127,628

<b>Deaths</b>	<b>New York</b>	<b>Richmond</b>
White	8,365	131
Colored	513	155
Entirely	8,878	286

<b>Mortality rate</b>	<b>New York</b>	<b>Richmond</b>
White	0,00179	0,00162
Colored	0,00560	0,00332
Entirely	0,00186	0,00224

# Heurísticas Admissíveis

nunca  
superestima  
o custo do  
objetivo

# Por exemplo, para o 8-puzzle

- ▣  $h_1(n)$  = número de quadrados em locais errados
- ▣  $h_2(n)$  = distância Manhattan total

5	4	
6	1	8
7	3	2

estado inicial

1	2	3
8		4
7	6	5

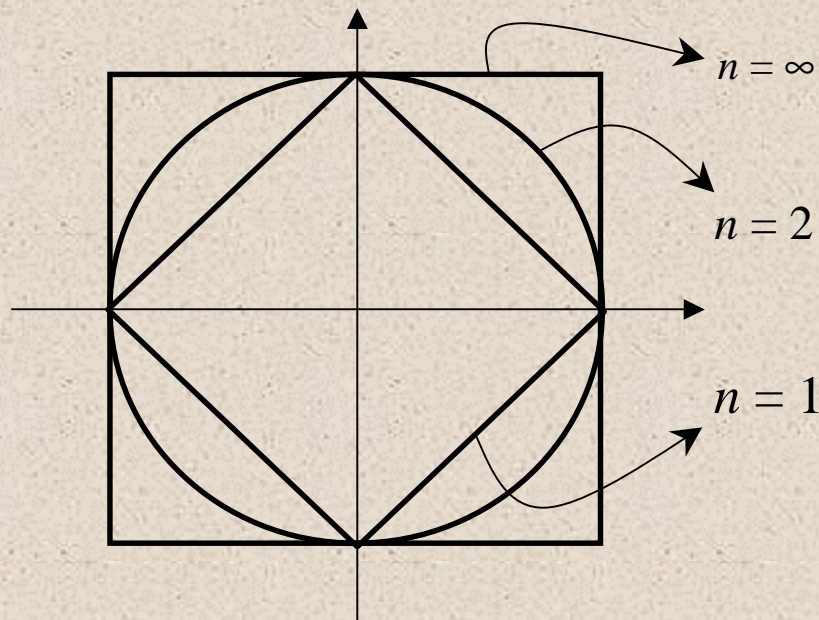
estado final

- ▣ Quais são os valores de  $h_1(s)$  e de  $h_2(s)$  ???

# Distâncias

## # Minkowski

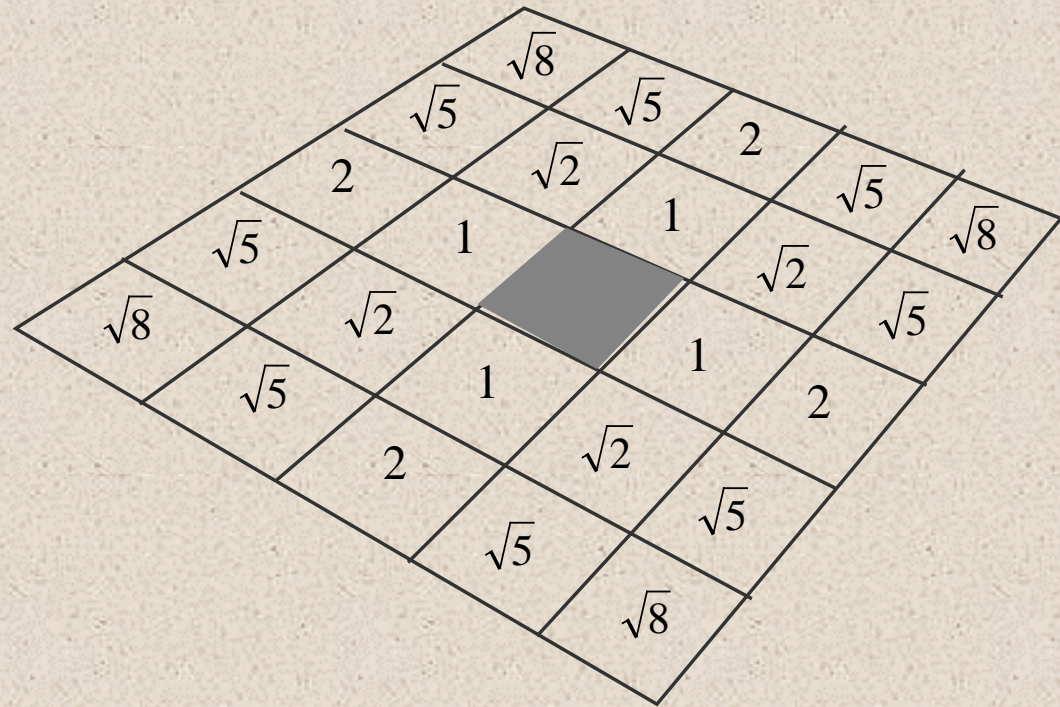
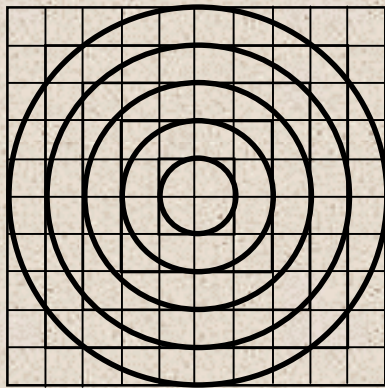
$$d_{12} = \sqrt[n]{\sum_{k=1}^D (\vec{v}_{1k} - \vec{v}_{2k})^n}$$



# Distâncias

## # Euclidiana (Minkowski com $n = 2$ )

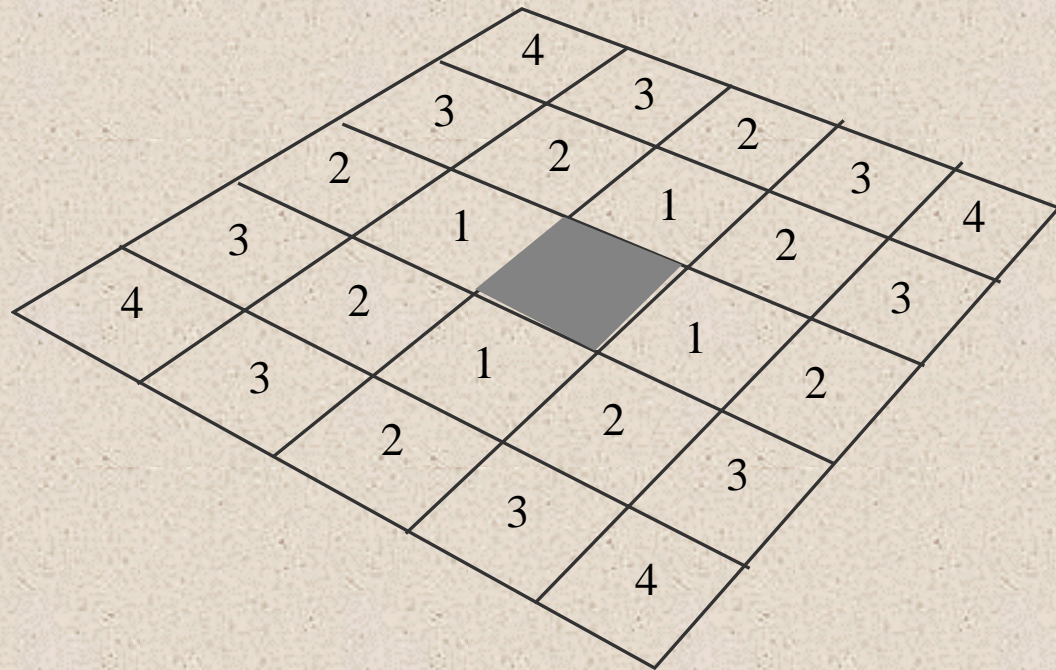
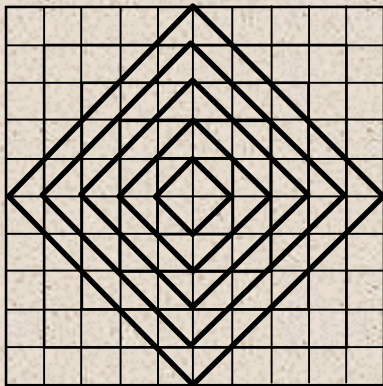
$$d_{12}^E = \sqrt{\sum_{k=1}^D (\vec{v}_{1k} - \vec{v}_{2k})^2}$$



# Distâncias

# City Block ou Manhattan (Minkowski com  $n = 1$ )

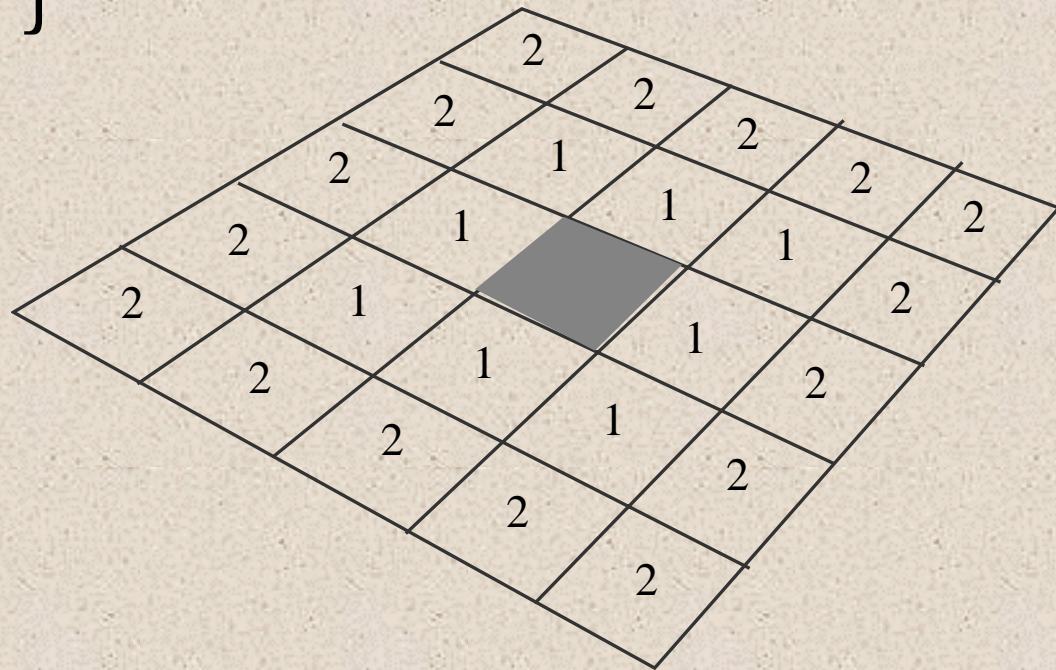
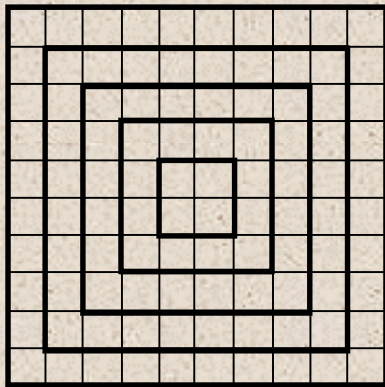
$$d_{12}^C = \sum_{k=1}^D \left| \vec{v}_{1k} - \vec{v}_{2k} \right|$$



# Distâncias

## # Chessboard

$$d_{12}^X = \max_k \left\{ \left| \vec{v}_{1k} - \vec{v}_{2k} \right| \right\}$$



# Heurísticas Admissíveis

# Por exemplo, para o 8-puzzle

- ▣  $h_1(n)$  = número de quadrados em locais errados
- ▣  $h_2(n)$  = distância Manhattan total

5	4	
6	1	8
7	3	2

estado inicial

1	2	3
8		4
7	6	5

estado final

▣  $h_1(s) = 7$      $h_2(s) = 2+3+3+2+4+2+0+2 = 18$

# Dominância

# Se  $h_2(n) \geq h_1(n)$  para todo  $n$  (ambas admissíveis) então  $h_2$  domina  $h_1$  e é melhor para a busca

# Custos típicos:

■  $d = 14$

■ Aprofundamento Iterativo = 3.473.941 nodos

■  $A^*(h_1) = 539$  nodos

■  $A^*(h_2) = 113$  nodos

■  $d = 24$

■ Aprofundamento Iterativo = muitos nodos

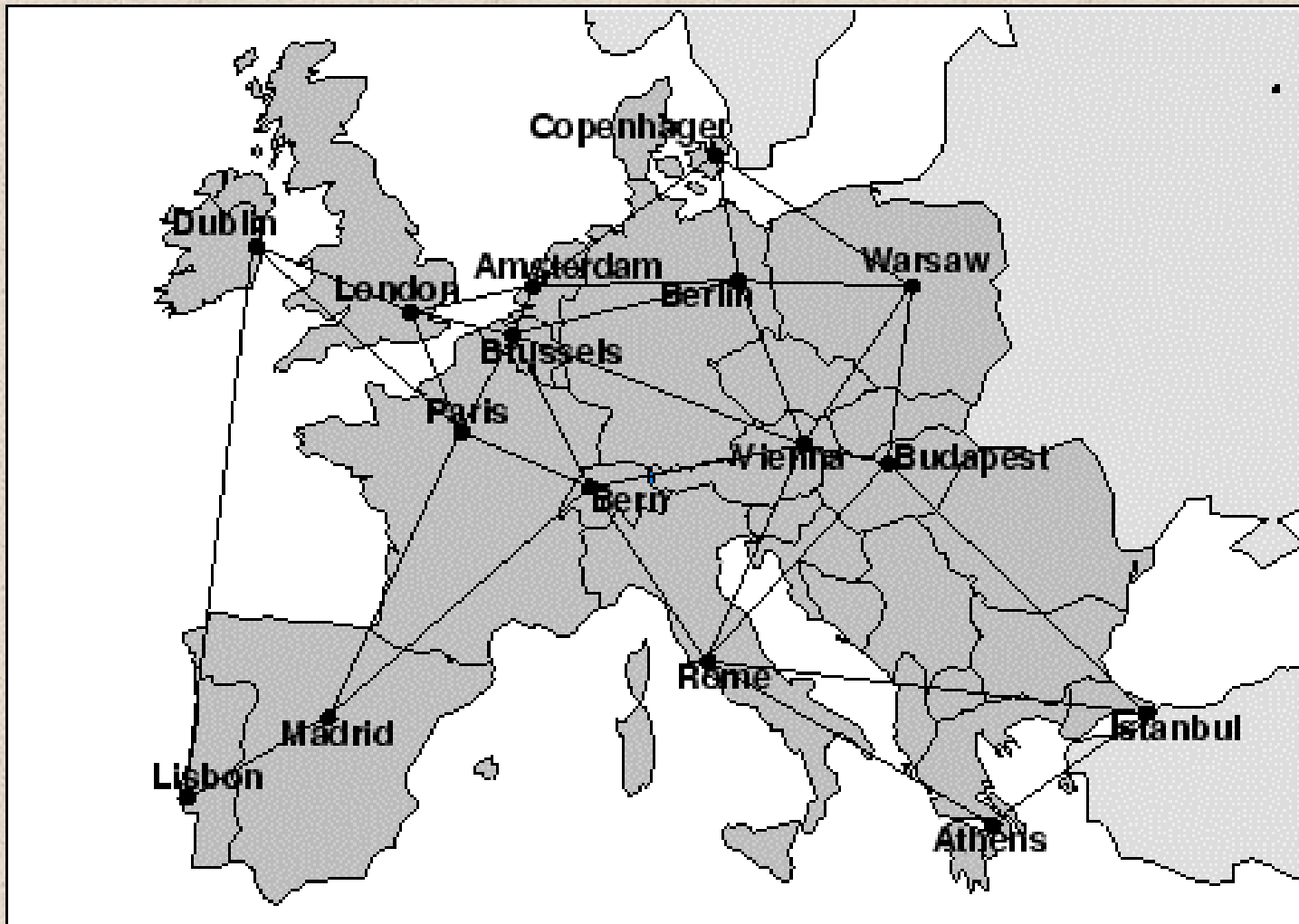
■  $A^*(h_1) = 39.135$  nodos

■  $A^*(h_2) = 1.641$  nodos

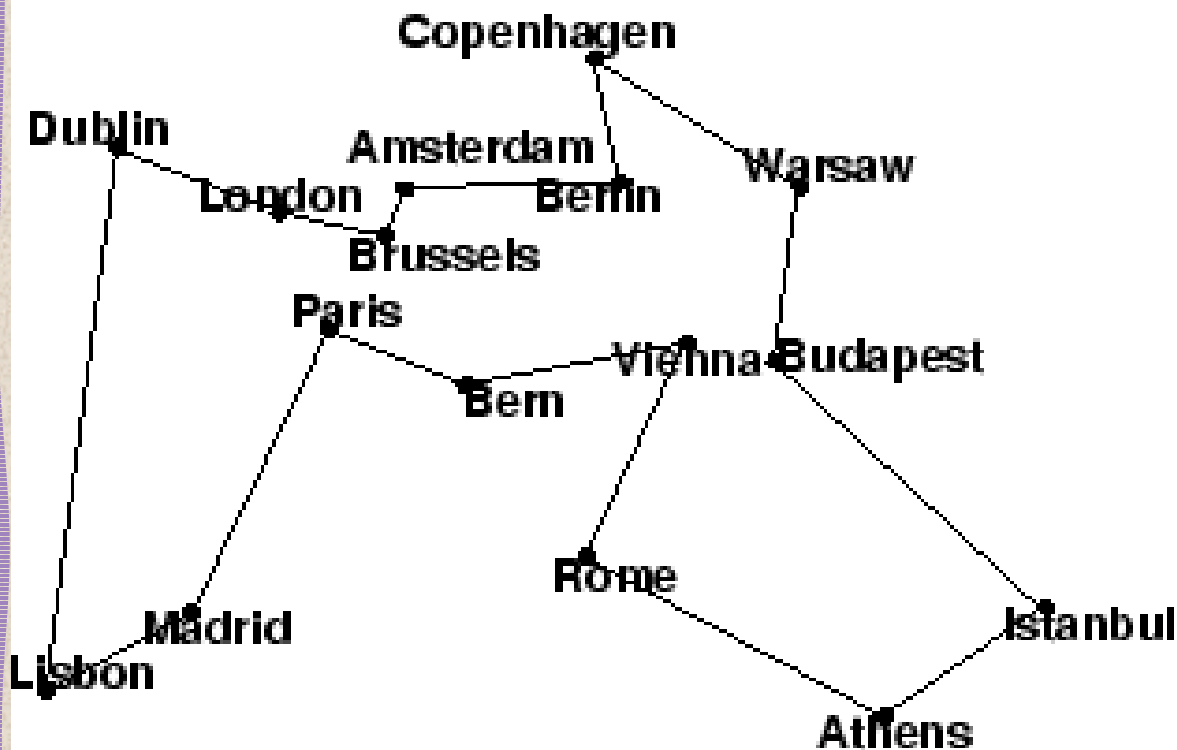
# Problemas Relaxados

- # Muitas vezes, uma boa heurística para um problema relaxado pode ser uma boa heurística para um problema relaxado
- # Heurísticas admissíveis podem ser derivadas do custo de uma solução exata a partir de uma versão relaxada do problema.
- # Se as regras do 8-puzzle forem relaxadas para que um quadrado possa se mover para qualquer lugar, então  $h_1(n)$  dá a solução de caminho mais curto
- # Se as regras forem relaxadas para que um quadrado possa mover-se para qualquer quadrado adjacente, então  $h_2(n)$  dá a solução com caminho mais curto
- # Para o problema do caixeiro viajante (TSP)
  - Permitir caminhar sobre qualquer estrutura que conecte todas as cidades → Minimum Spanning Tree

# TSP relaxado



# TSP (solução)

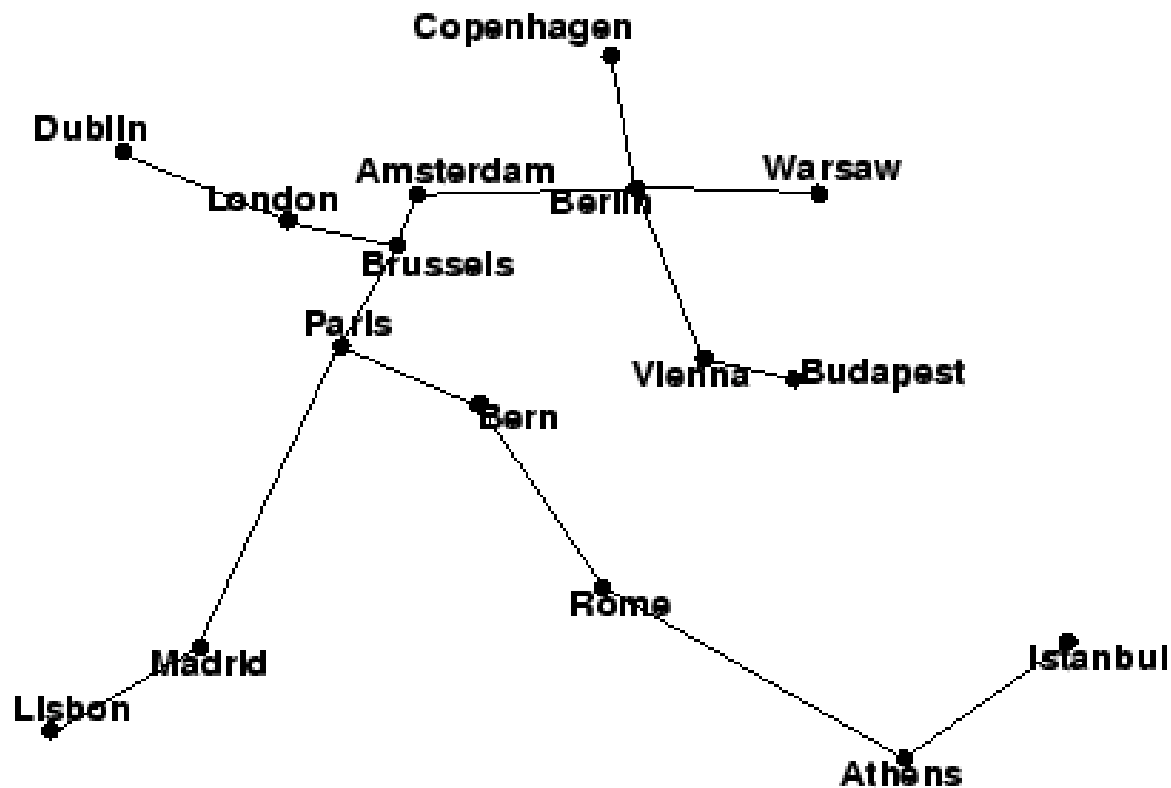


Encoding :

Dublin  
London  
Brussels  
Amsterdam  
Berlin  
Copenhagen  
Warsaw  
Budapest  
Istanbul  
Athens  
Rome  
Vienna  
Bern  
Paris  
Madrid  
Lisbon

# TSP relaxado

Min spanning tree has total distance of 563 units

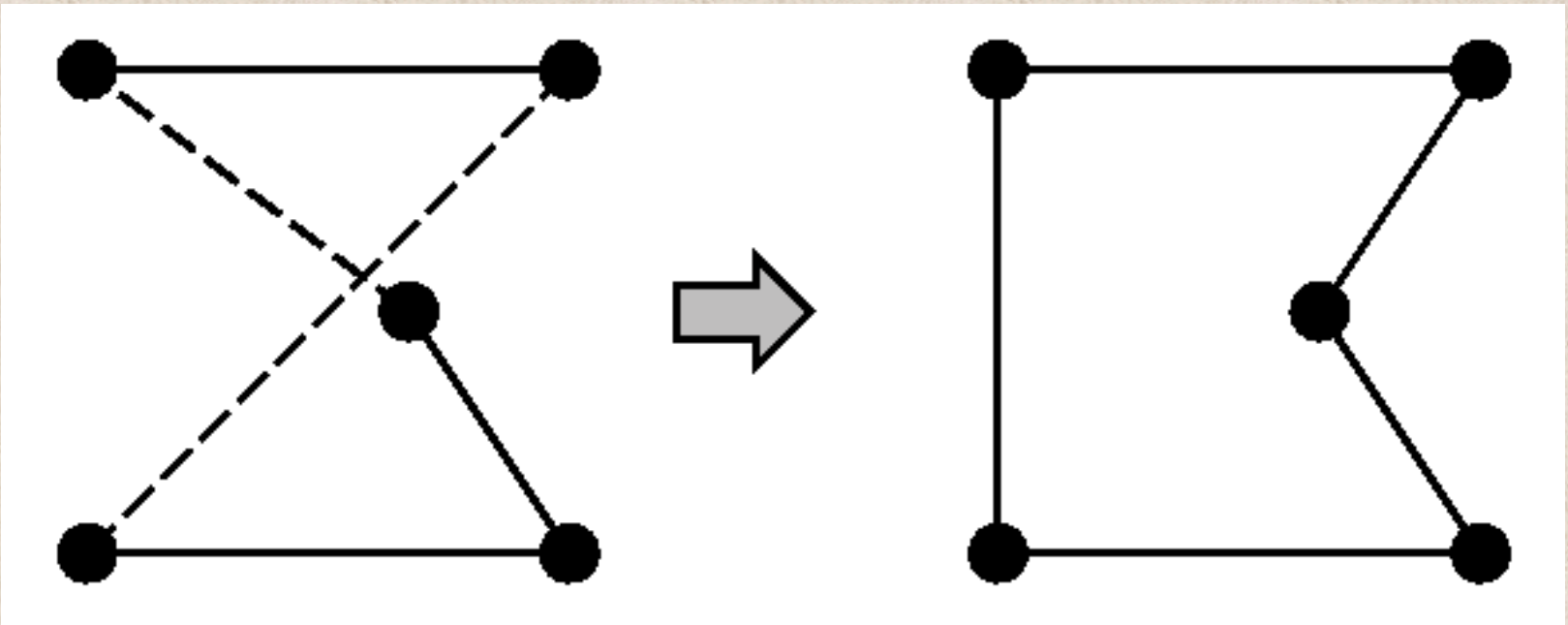


# Algoritmos de Melhoramento Iterativo

- # Em muitos problemas de otimização, o caminho é irrelevante; o estado objetivo é a solução
- # Então, o espaço estado é o conjunto completo de todas as configurações
  - Encontrar configuração ótima, por exemplo, TSP
  - Ou encontrar configurações satisfazendo restrições, por exemplo, n-rainhas
- # Em tais casos, podem ser usados algoritmos de melhoramento iterativo.
  - Manter um único estado atual e tentar melhorá-lo
  - Espaço constante. Adequado para buscas on-line e off-line

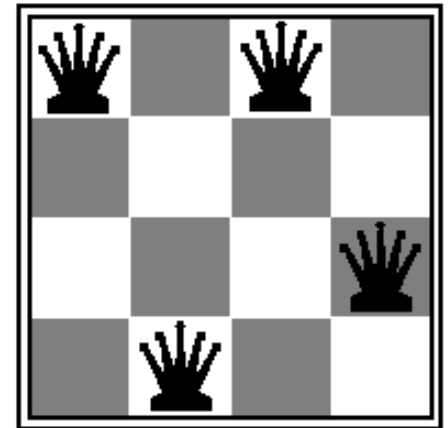
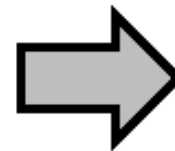
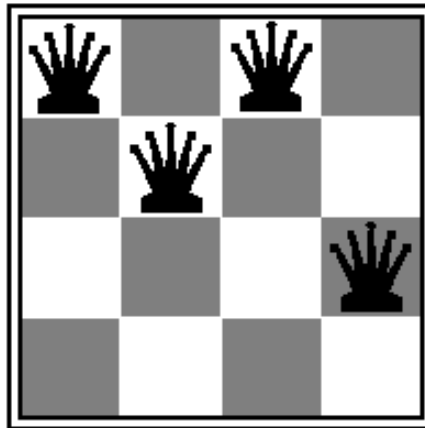
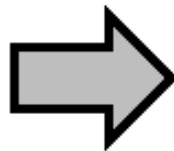
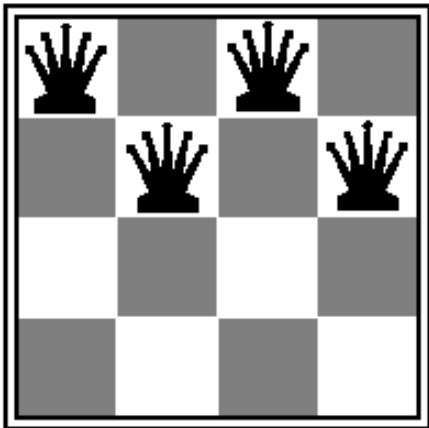
# Exemplo: Problema do Caixeiro Viajante

- # Encontre o caminho mais curto que visite todas as cidades pelo menos uma vez



# Exemplo: Problema das n-Rainhas

- ✚ Por  $n$  Rainhas em um tabuleiro  $n \times n$  sem que duas rainhas ocupem a mesma linha, nem a mesma coluna e nem a mesma diagonal



# Hill-climbing ou Descida/Subida de Gradiente

# É como escalar o Everest sob densa neblina e sofrendo de amnésia

**function** HILL\_CLIMBING(*problema*) **returns** um estado solução

**inputs:** *problema*, um problema

**local variables:** *atual*, um nodo

*próximo*, um nodo

*atual* ← FAZ\_NODO(ESTADO\_INICIAL[*problema*])

**loop**

*nodo* ← Sucessor de Maior Valor a partir de (*atual*)

**if** VALOR[*próximo*] < VALOR[*atual*]

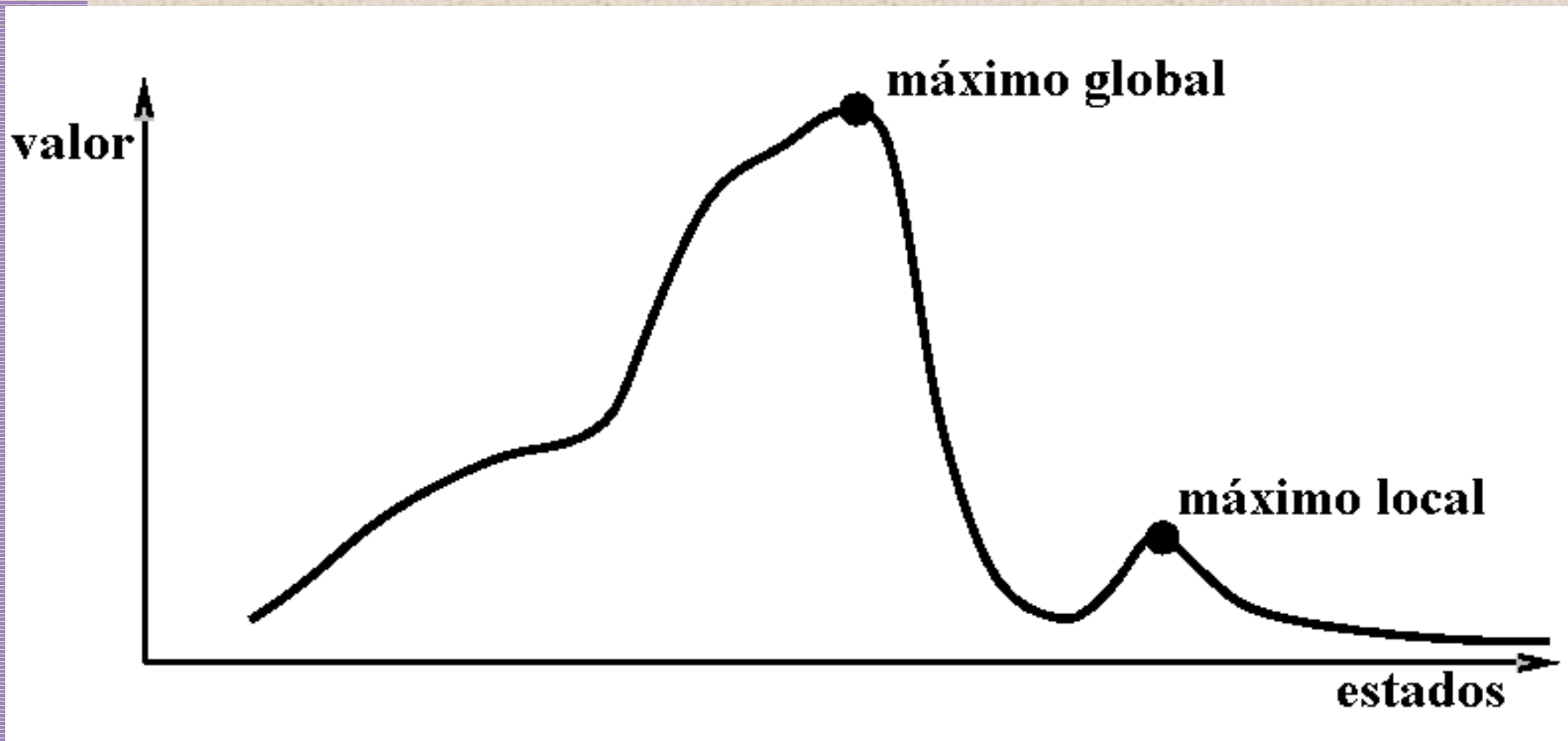
**then return** *atual*

*atual* ← *próximo*

**end**

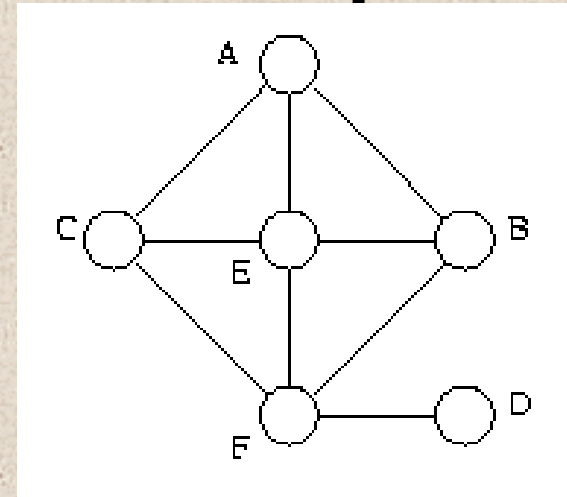
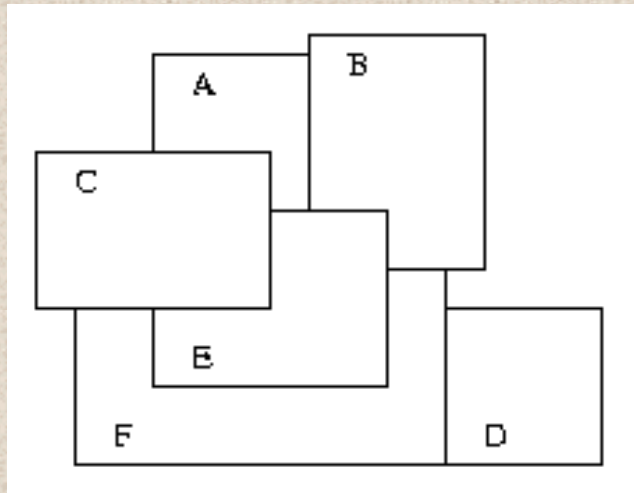
# Hill-climbing - problema

- # Dependendo do estado inicial, pode ficar preso em um máximo local



# Hill-Climbing (Busca Local)

# O problema de colorir um mapa



# Simulated Annealing

## # IDÉIA:

- fugir do máximo local permitindo alguns movimentos “ruins”, mas gradualmente decrescendo seu tamanho e frequência

**function** SIMULATED\_ANNEALING(*problema*, *escala*) **returns** um estado solução

**inputs:** *problema*, um problema

*escala*, um mapeamento do tempo pela temperatura

**local variables:** *atual*, um nodo

*próximo*, um nodo

*T*, uma temperatura controlando a probabilidade de dar passos pra baixo

*atual* ← FAZ\_NODO(ESTADO\_INICIAL[*problema*])

**for** *tempo* ← 1 **to** ∞

*T* ← *escala*[*tempo*]

**if** *T* = 0 **then return** *atual*

*próximo* ← um sucessor aleatoriamente selecionado de (*atual*)

$\Delta E$  ← VALOR[*próximo*] – VALOR[*atual*]

**if**  $\Delta E > 0$  **then** *atual* ← *próximo*

**else** *atual* ← *próximo* somente com uma probabilidade  $e^{\Delta E/T}$

**end**

# Propriedades do Simulated Annealing

- # Na “temperatura” fixa  $T$ , a probabilidade de ocupação do estado é dado pela distribuição de Boltzman

$$p(x) = \alpha e^{-\frac{E(x)}{kT}}$$

- #  $T$  decrescendo suficientemente lento  $\Rightarrow$  sempre alcança o melhor estado
- # Esta é, necessariamente, uma garantia interessante?
- # Concebido por Metropolis et al, 1953 para modelagem de processos físicos
- # Amplamente utilizado para layout de VLSI, planejamento de linhas aéreas