Pesquisa em árvore (revisão)

```
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
```

A estratégia de pesquisa é definida pela ordem pela qual os nós da fronteira são expandidos

Heurística

Ideia: integrar na estratégia de pesquisa conhecimento específico sobre cada problema em particular, para melhorar a eficiência do algoritmo

Prescinde-se da procura da solução ótima, muitas vezes impossível de alcançar com os recursos disponíveis

Procura-se uma boa solução, possível com os recursos disponíveis

Em vez de procurar todas as alternativas possíveis, dá-se preferência a algumas das alternativas, baseados na informação de uma função heurística que nos aconselha sobre quais são as melhores

Se a heurística for boa não se perde nada, até se pode obter a solução ótima

Pesquisa “best-first”

Ideia:

Usar uma função de avaliação que associa a cada nó uma estimativa do interesse em expandir esse nó

==> expande-se o nó com maior interesse daqueles que ainda não foram expandidos (fronteira)

Implementação:

A lista de nós fronteira é uma lista ordenada por ordem decrescente pela estimativa do interesse em expandir cada nó

Casos particulares:

Pesquisa sôfrega (ou ambiciosa, ou gulosa)

A*

Tipos de problemas de pesquisa

```
  S E N D
+ M O R E
-----------
M O N E Y
```

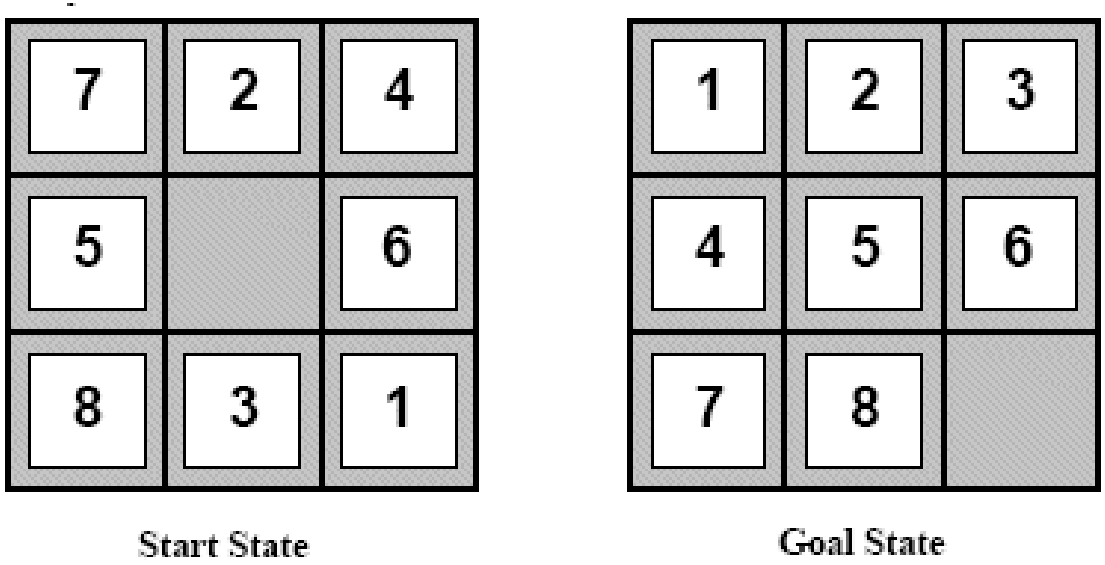
Problema:

Atribuir a cada letra um dígito (0..9), de forma que a adição faça sentido

Cada dígito só pode ser atribuído a uma letra

O estado final não é conhecido; o problema é saber qual é esse estado

Tipos de problemas de pesquisa (cont.)



Problema:

Qual é a sequência de movimentos (o caminho) que leva do estado à esquerda (estado inicial) para o estado à direita (estado final)?

O estado final é conhecido, faz parte do enunciado do problema

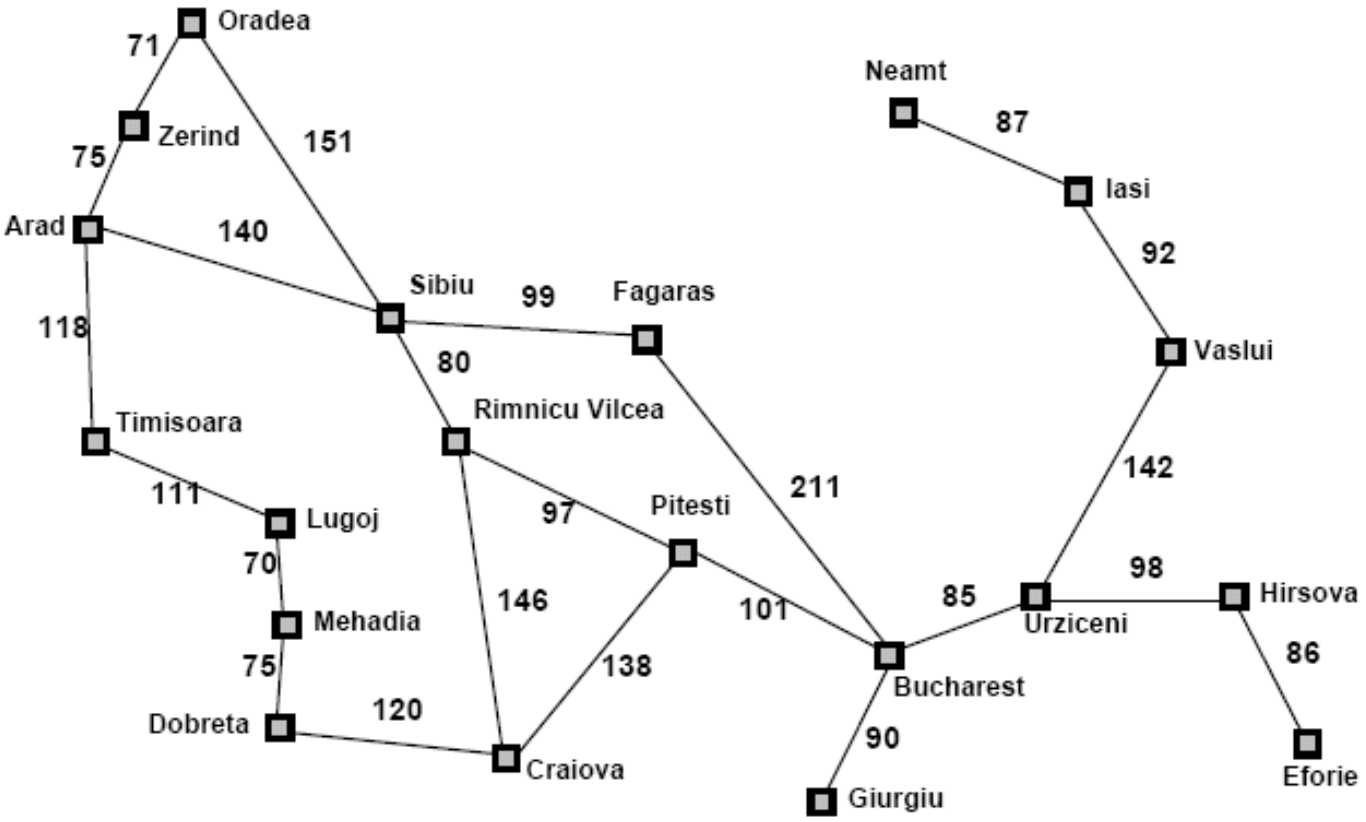
Tipos de problemas de pesquisa (cont.)

Dois tipos de problemas distintos

- problema que consiste em descobrir qual é o estado final; a solução do problema é o estado final
- pretende-se chegar rapidamente ao estado final, mesmo que o caminho seguido não seja o de mais baixo custo
- não interessa a qualidade do caminho
- a qualidade da solução é a qualidade do estado final encontrado

- problema em que o estado final (ou estados finais) são conhecidos à partida, mas se pretende saber qual o caminho (qual a sequência de ações) que nos leva do estado inicial ao estado final; a solução do problema é o caminho encontrado
- é importante que o caminho seja o de mais baixo custo
- a qualidade da solução é a qualidade do caminho encontrado

Roménia – custo de cada passo em km



Straight−line distance to Bucharest	
Arad	366
Bucharest	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

Pesquisa sôfrega

Função de avaliação h(nó) – heurística

estimativa do custo de chegar do nó ao objetivo mais próximo

exemplo:

h_{DLR}(nó) = distância em linha reta do nó até Bucareste

A pesquisa sôfrega expande o nó que parece estar mais próximo de um objetivo, de um estado final

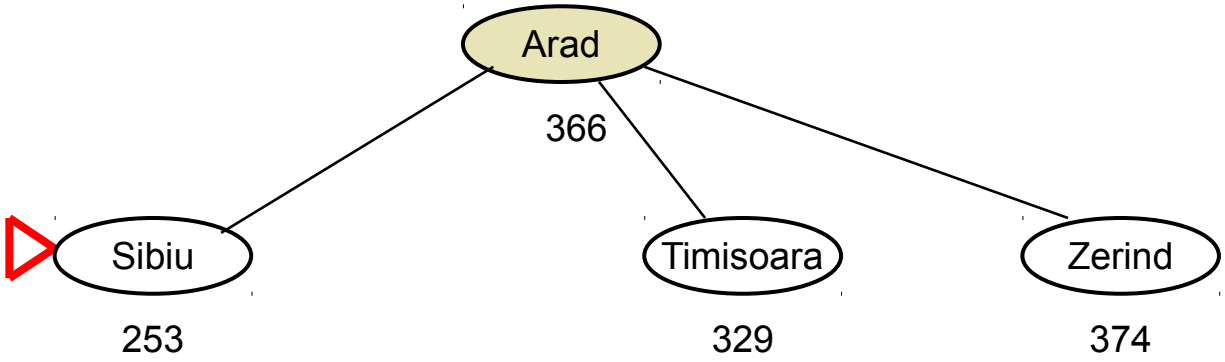
Pretende atingir rapidamente o objetivo, sem se preocupar se o caminho é o melhor. Minimiza o esforço de cálculo.

Apropriado para encontrar rapidamente estados finais, não caminhos ótimos

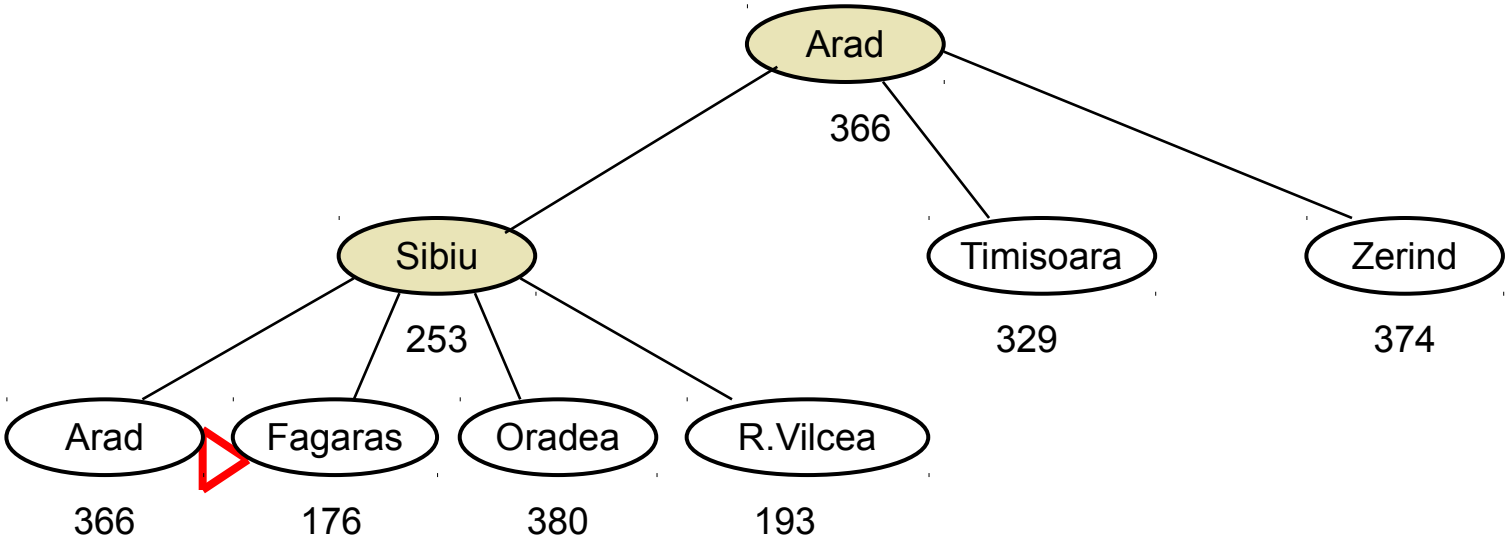
Pesquisa sôfrega -- exemplo

366

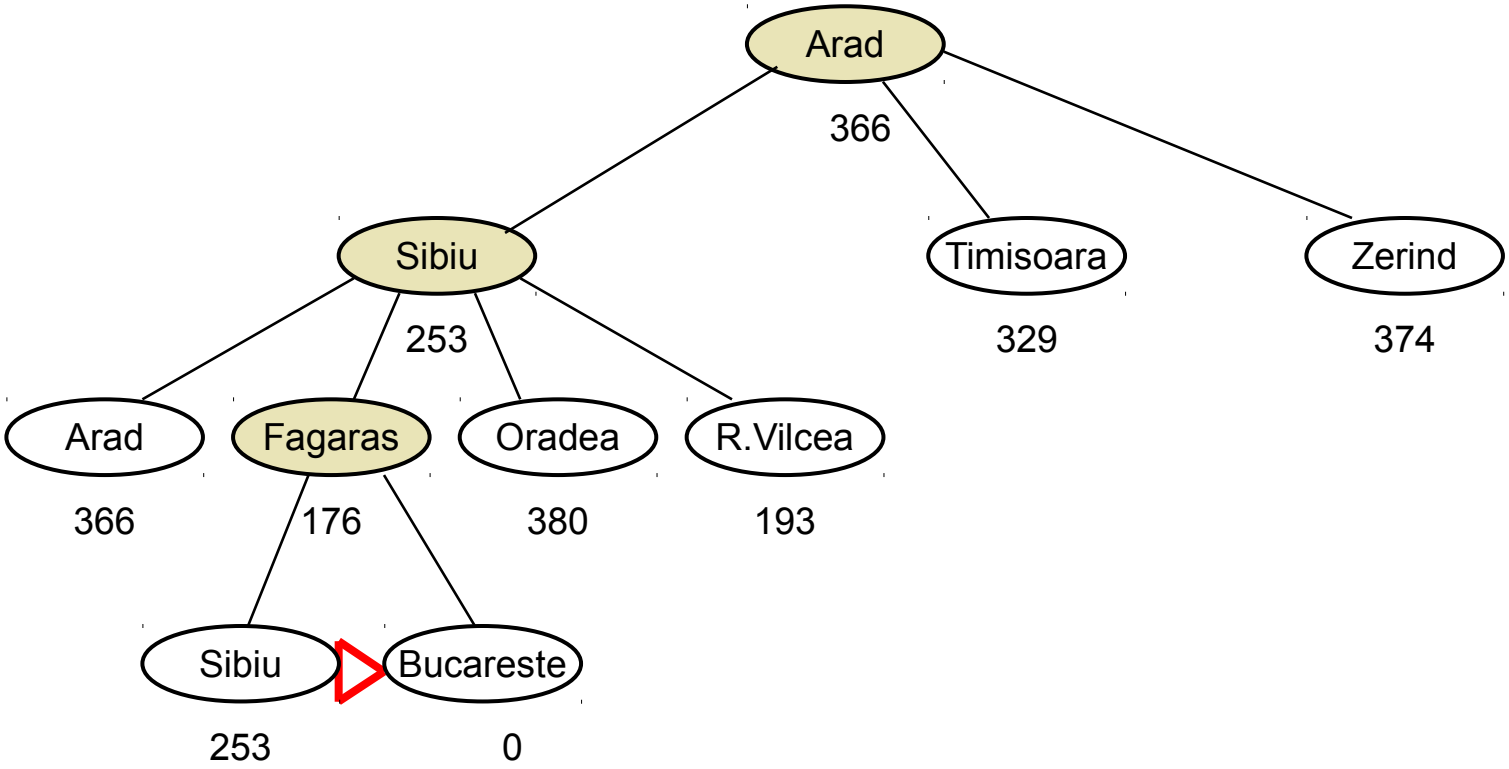
Pesquisa sôfrega – exemplo (cont.)



Pesquisa sôfrega – exemplo (cont.)



Pesquisa sôfrega – exemplo (cont.)



Pesquisa sôfrega (cont.)

Completo ??

Não. Pode ficar preso em ciclos. Por exemplo, com objetivo Oradea,

Iasi --> Neamt --> Iasi --> Neamt --> ...

É completo num espaço de estados finito, e com deteção de ciclos

Tempo ??

$O(b^m)$, mas uma boa heurística pode melhorar muitíssimo

Espaço ??

$O(b^m)$ – guarda todos os nós em memória

Ótimo ??

Não

Pesquisa A*

Ideia: evitar expandir caminhos que já têm custo elevado

Função de avaliação:

f(nó) = g(nó) + h(nó)

g(nó) : custo de chegar desde a origem até o nó

h(nó) : estimativa de chegar do nó ao objectivo mais próximo

f(nó) : estimativa do custo total do caminho que passa por nó, desde a origem até o objetivo

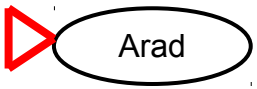
A* usa heurística admissível

h(nó) <= h*(nó), em que h*(nó) é o custo real de chegar do nó ao objetivo mais próximo – isto é, a estimativa subestima o valor real

h(nó) >= 0, portanto h(G) = 0 para qualquer objetivo G

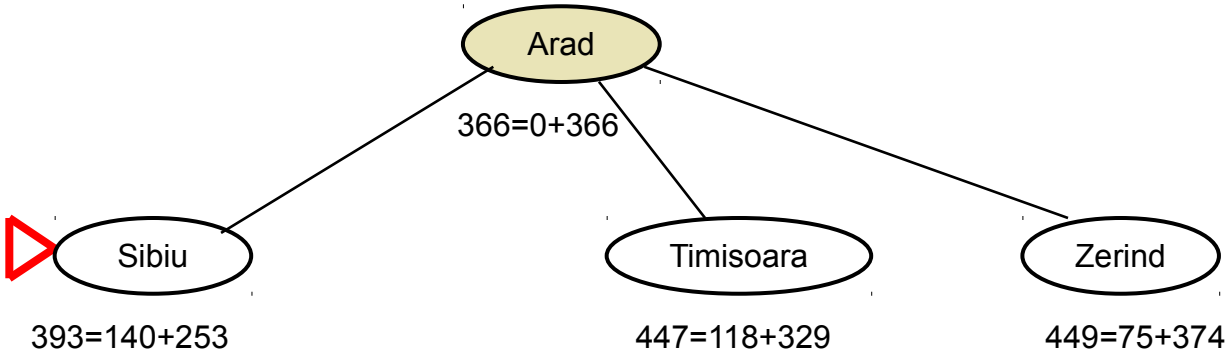
Exemplo: h_{DLR}(nó) subestima a distância real

Pesquisa A* -- exemplo

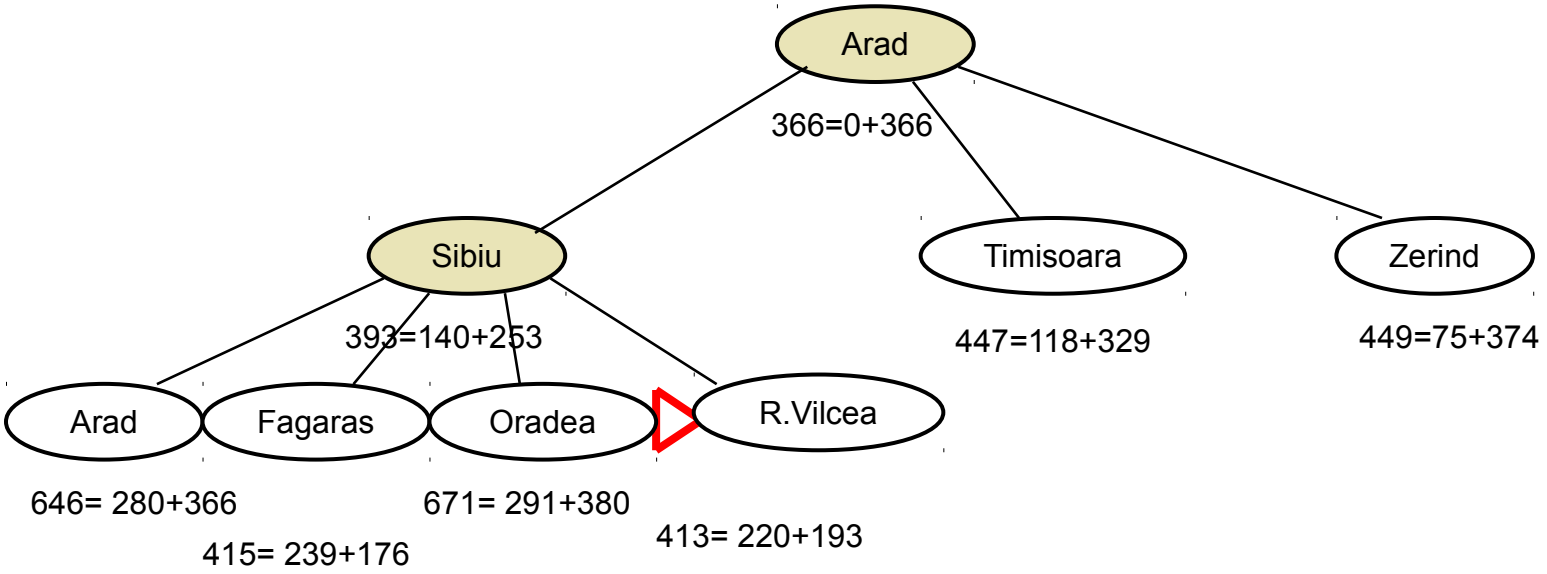


366=0+366

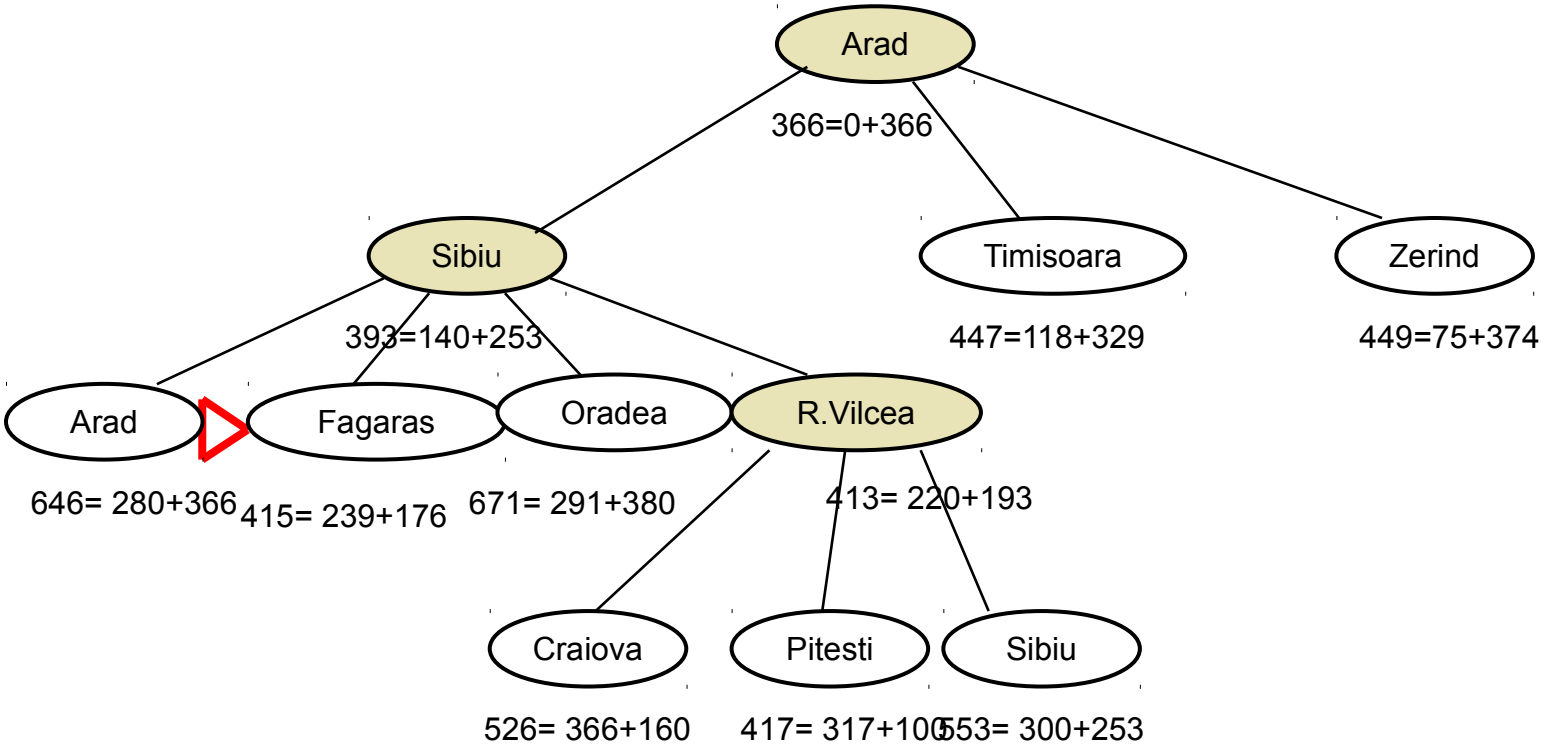
Pesquisa A* – exemplo (cont.)



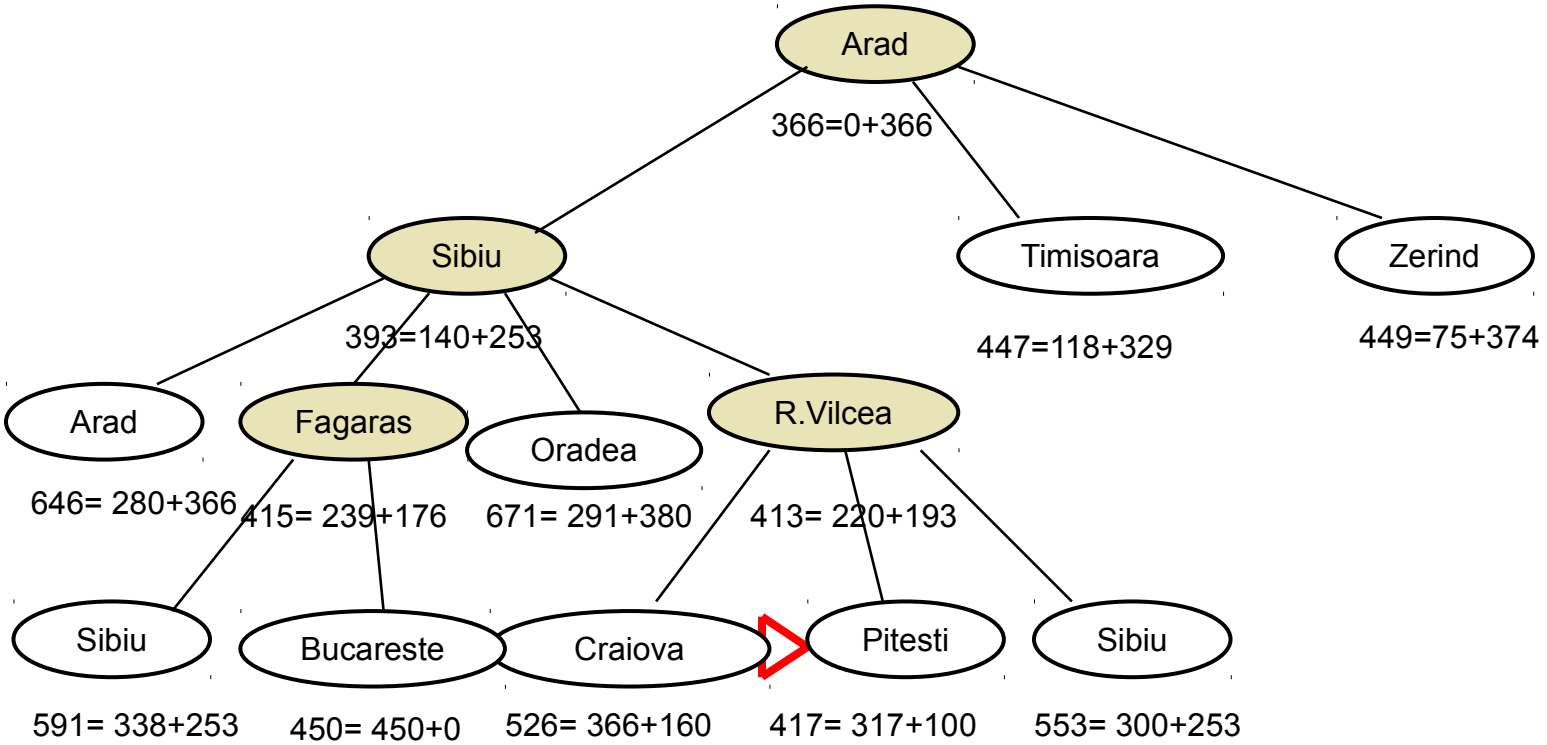
Pesquisa A* – exemplo (cont.)



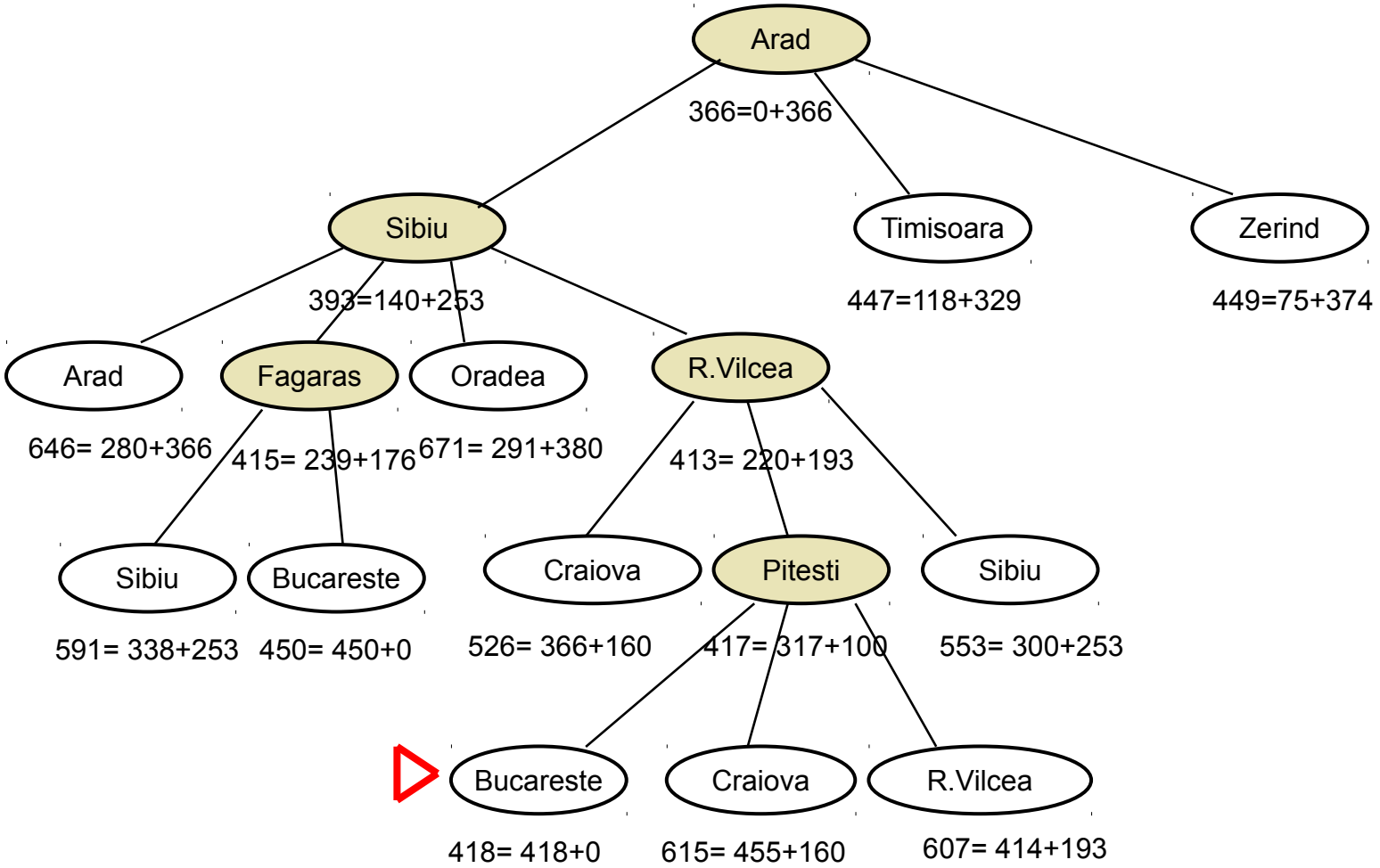
Pesquisa A* – exemplo (cont.)



Pesquisa A* – exemplo (cont.)



Pesquisa A* – exemplo (cont.)



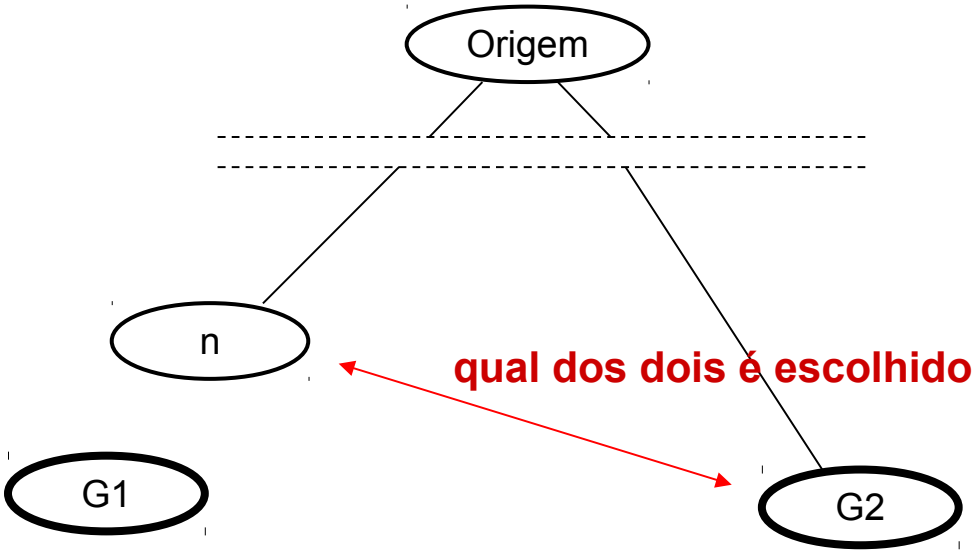
Pesquisa A* (cont.)

Teorema: a pesquisa A* é ótima

Seja G2 um objetivo não ótimo gerado e na lista fronteira

Seja n um nó não expandido num caminho ótimo para G1

Vamos ver que nunca será escolhido G2 (não ótimo) em vez de n (no caminho ótimo para G1)




```
f(G2) = g(G2), dado que h(G2) = 0, por G2 ser um objetivo
      > g(G1), dado que G2 não é ótimo, e G1 é
      >= f(n), dado que h é admissível
logo, se f(G2) > f(n), G2 nunca será escolhido em vez de n
nota : g(G1) = g(n) + h*(n) >= g(n) + h(n) = f(n), logo g(G1) >= f(n)
```

Pesquisa A* (cont.)

Completo ??

Sim, a menos que haja um número infinito de nós com f < f(G)

Tempo ??

Exponencial em [erro relativo em h x profundidade da solução]

Espaço ??

Conserva todos os nós em memória

Ótimo ??

Sim

Desde que a heurística seja admissível, o método A* é óptimo. Quanto mais próxima a estimativa dada pela função h for do custo real, mais rapidamente a solução é encontrada.

Uma heurística consistente aumenta a eficiência da pesquisa.

Consistência

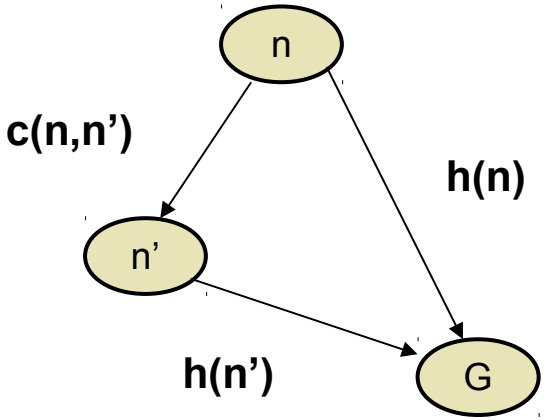
Uma heurística é consistente se

```
h(n) <= c(n,n') + h(n')
```

Se n é consistente, então

```
f(n')  = g(n') + h(n')
       = g(n) + c(n,n') + h(n')
      >= g(n) + h(n)
       = f(n)
```

isto é, f(n) é não decrescente ao longo de qualquer caminho



Pesquisa A* (cont.)

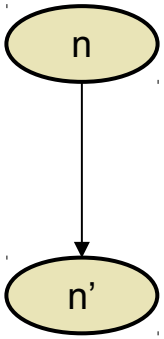
Monotonia

A consistência está ligada à monotonia

f(n) = g(n) + h(n) = 6 + 8 = 14 h(n) = 8

c(n,n') = 1

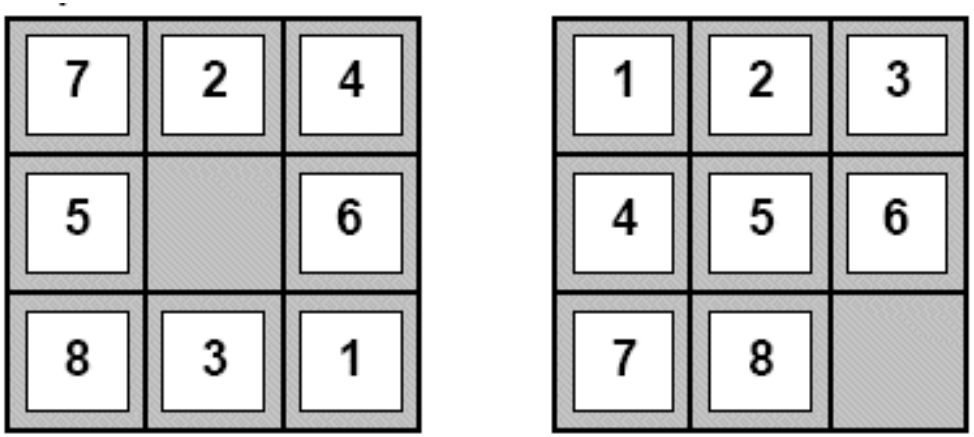
f(n') = g(n') + h(n') = 7 + 5 = 12 h(n') = 5



Mas se h é admissível, f(n') devia ser pelo menos 14!

Solução simples: f(n') = max (f(n), g(n')+h(n'))

Heurísticas admissíveis



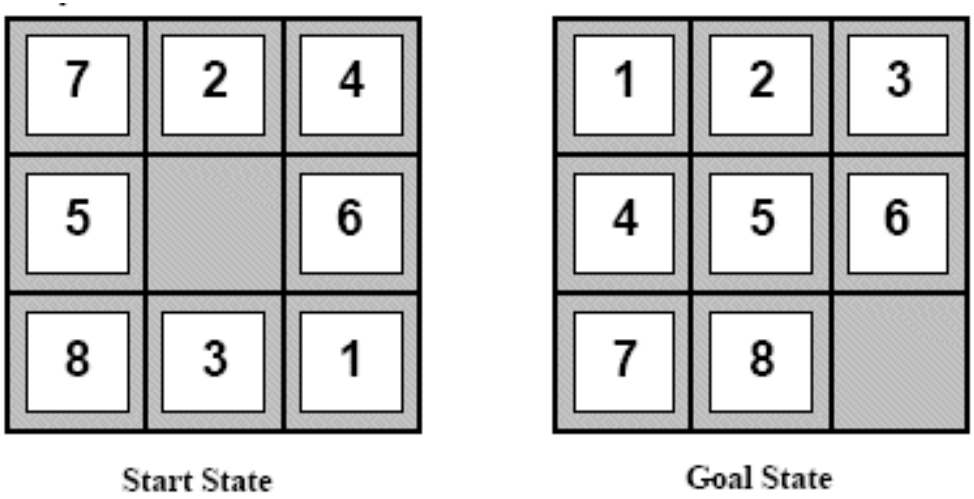
h1(n) = número de peças fora da posição final

h2(n) = soma das distâncias de Manhattan à posição final das peças

h1(S) = ??

h2(S) = ??

Heurísticas admissíveis (cont.)



h1(n) = número de peças fora da posição final

h2(n) = soma das distâncias de Manhattan à posição final das peças

h1(S) = 7

h2(S) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14

Heurísticas admissíveis (cont.)

Dominância

Se h2(n) >= h1(n) para todo o n (ambas admissíveis), então h2 domina h1 e é melhor para a pesquisa (a estimativa é mais próxima do valor real).

Exemplo de custos de pesquisa típicos para o puzzle de 8

```
d = 14     IDS = 3 473 941 nós
               A*(h1) = 539 nós
               A*(h2) = 113 nós

d = 24     IDS ≈ 54 000 000 000 nós
               A*(h1) = 39 135 nós
               A*(h2) = 1 641 nós
```

Problemas relaxados

Uma heurística admissível pode ser obtida da solução exata de uma versão relaxada do problema

Puzzle de 8:

Se uma peça se pode mover para qualquer casa, h1(n) dá a solução mais curta

Se uma peça se pode mover para qualquer casa adjacente, h2(n) dá a solução mais curta

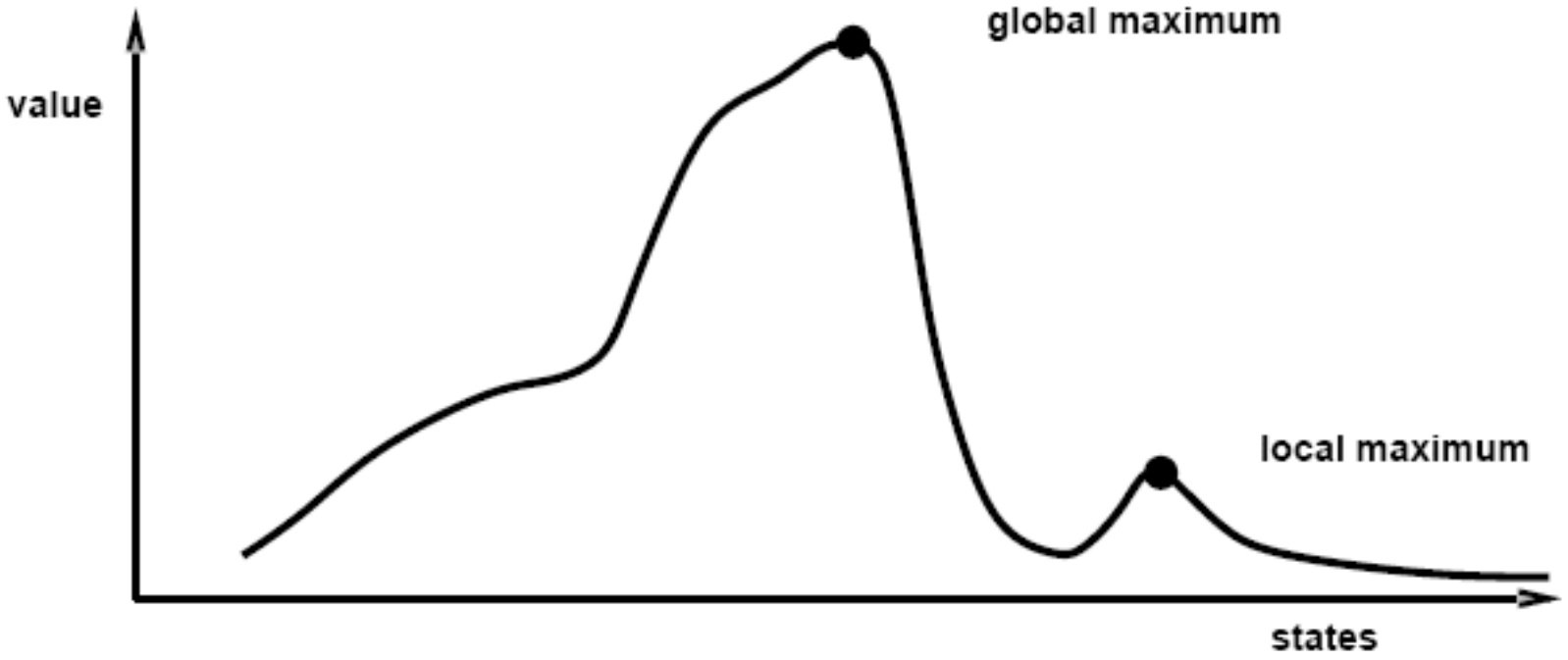
Subida da colina (*Hill climbing*)

A transição de cada nó faz-se para o “melhor” vizinho

Termina quando chegar a um nó “melhor” que todos os seus vizinhos

```
função Subida-Colina( problema) retorna um estado que é máximo
   local
    entradas: problema
    variáveis: corrente, um nó
               vizinho, um nó
    corrente ← Cria-Nó( Estado-Inicial[ problema])
    repete
       vizinho ← um sucessor de corrente com o maior valor
       se Valor[vizinho] < Valor[corrente] então
          retorna Estado[corrente]
       corrente ← vizinho
    fim-repete
```

Subida da colina (cont.)



Problema: pode ficar preso num máximo local