Estratégias de Busca

Busca Informada

ou

Busca Heurística

Solange Rezende
Departamento de Ciências de Computação
ICMC-USP, São Carlos
solange@icmc.usp.br

Esta aula descreve algumas estratégias de busca informada em espaço de estados

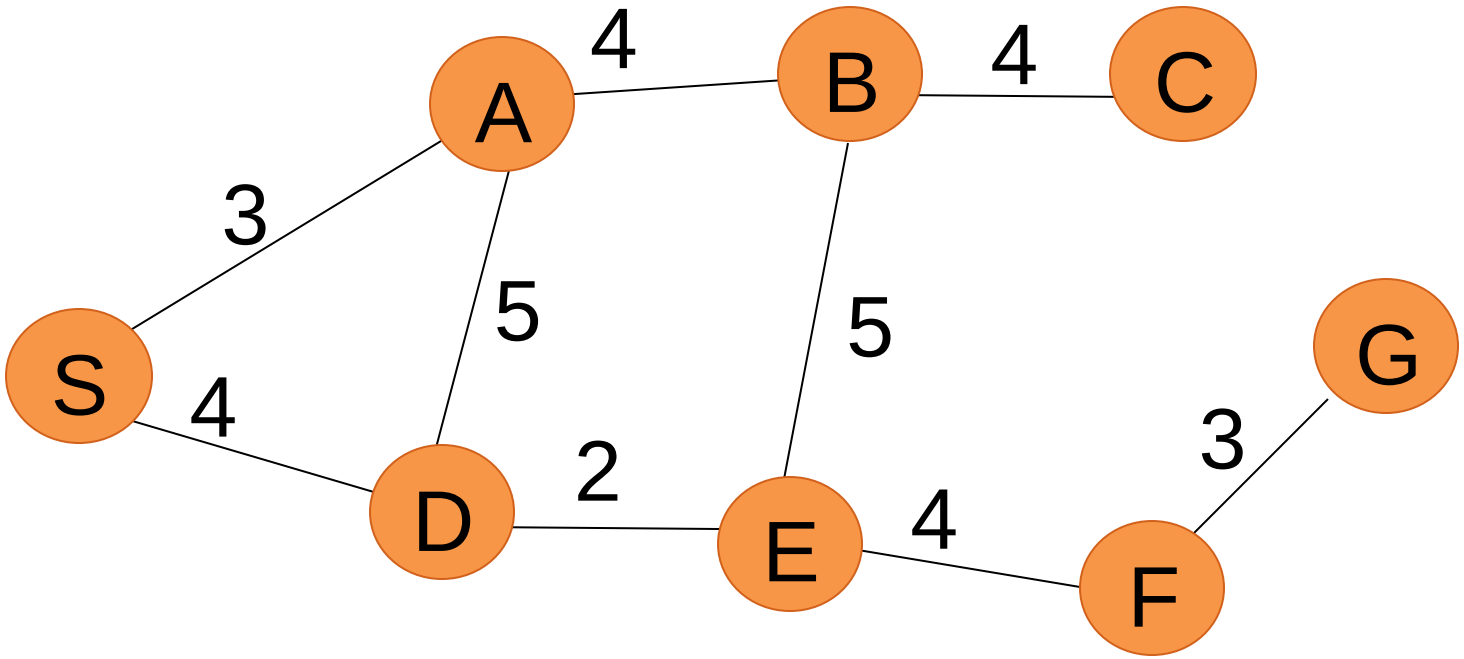
A palavra **heurística** vem da palavra grega heuriskein, que significa "**descobrir**," que é também a origem de eureka, a famosa exclamação de Arquimedes ("Eu achei").

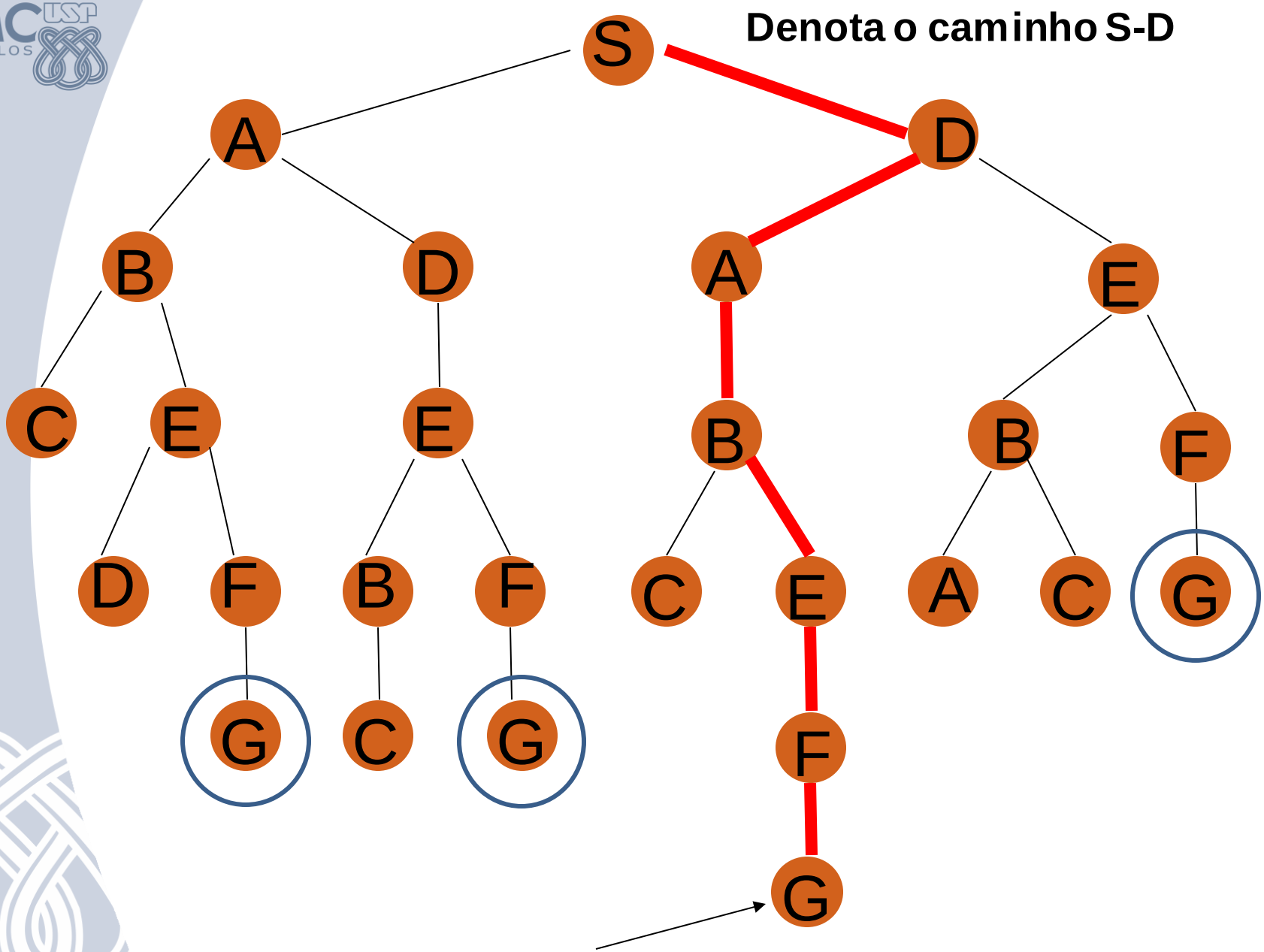
- A **busca em grafos** pode atingir uma **complexidade elevada** devido ao número de alternativas. Existem várias formas de reduzir o tempo/custo de busca.
- A estratégia de **busca com informação utiliza conhecimento específico do problema, além da definição do próprio problema** e pode encontrar solução de forma mais eficiente que uma estratégia sem informação – Busca Heurística.
- O principal **papel da heurística é eliminar** (ou podar) **ramos da busca**.

- Alguns autores consideram as **heurísticas** como funções baseadas em cálculos numéricos.
  - É conveniente diferenciar e chamar essas funções baseadas em cálculos numéricos de funções de avaliação.
- Uma **função de avaliação**, também conhecida como **função heurística de avaliação** ou **função estática de avaliação** é uma função matemática utilizada para estimar o valor ou uma boa posição e será **representada por *h(n)*.**
  - A função de avaliação é escrita para ser rápida e precisão e procura na posição atual e não explora os movimentos possíveis (é estática).
- A *tradução* de uma **heurística** para um valor numérico é realizado pela ***função de avaliação***.
  - São não negativas e tal que o estado associado com o menor valor é considerado o estado mais promissor e frequentemente, a FA do estado meta é zero.

EXEMPLO

- Dado o grafo abaixo, **encontrar a menor distância** de **S** a **G**




Denota o caminho S-D
S
A
D
B
D
A
E
C
E
E
B
B
F
D
F
B
F
C
E
A
C
G
G
C
G
F
G
Um caminho é: S-D-A-B-E-F-G

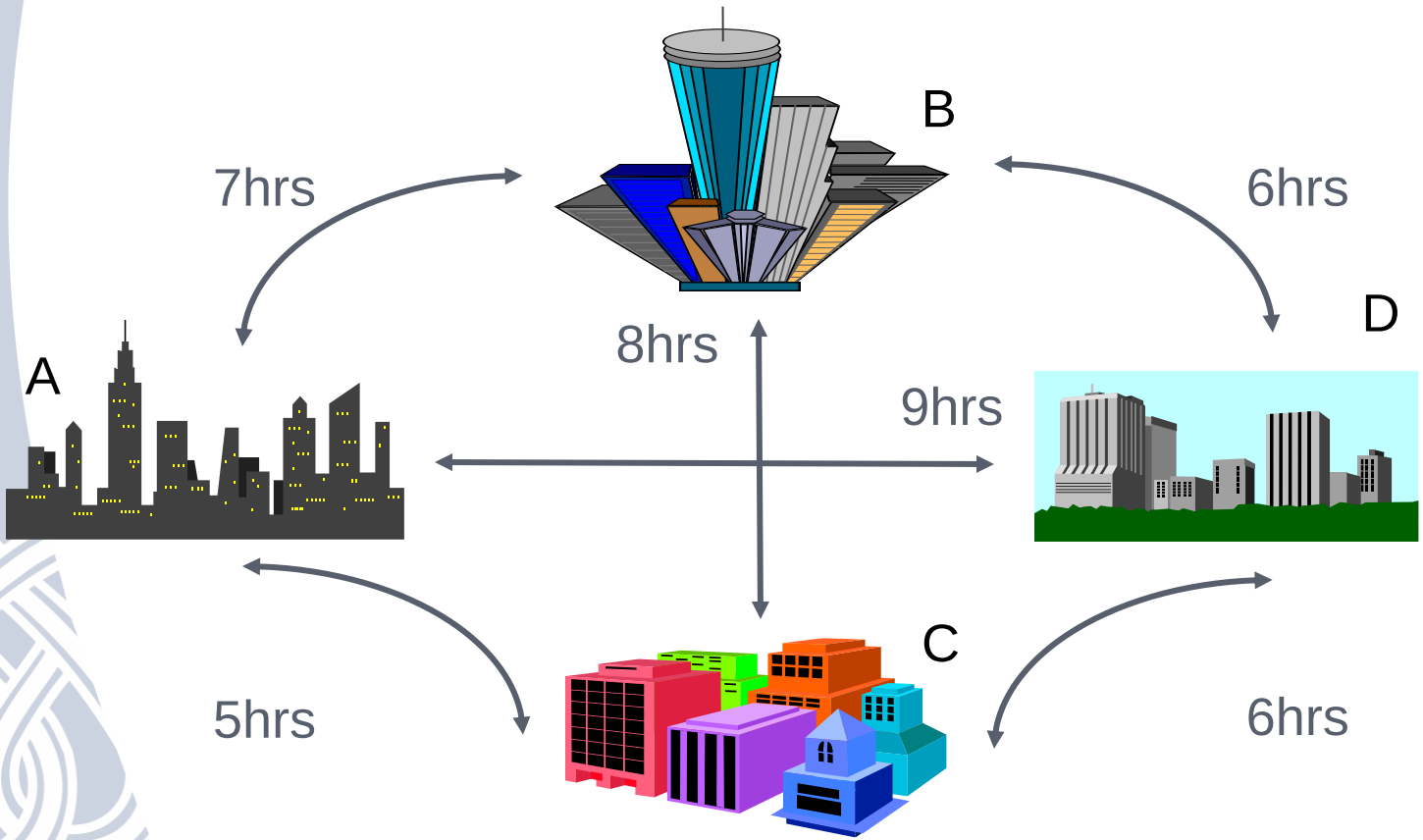
PROBLEMA DO CAIXEIRO VIAJANTE
(TSP – TRAVELLING SALESMAN PROBLEM)

- Um vendedor tem uma **lista de cidades**, que ele deve visitar apenas uma vez. Existem **estradas diretas** ligando cada par de cidades da lista. **Encontre a rota** que o vendedor deve seguir para a **viagem mais curta** possível que começa e termina em uma das cidades.

- A cada viagem esta associado um custo
  - É recomendado que o caixeiro percorra a rota mais curta

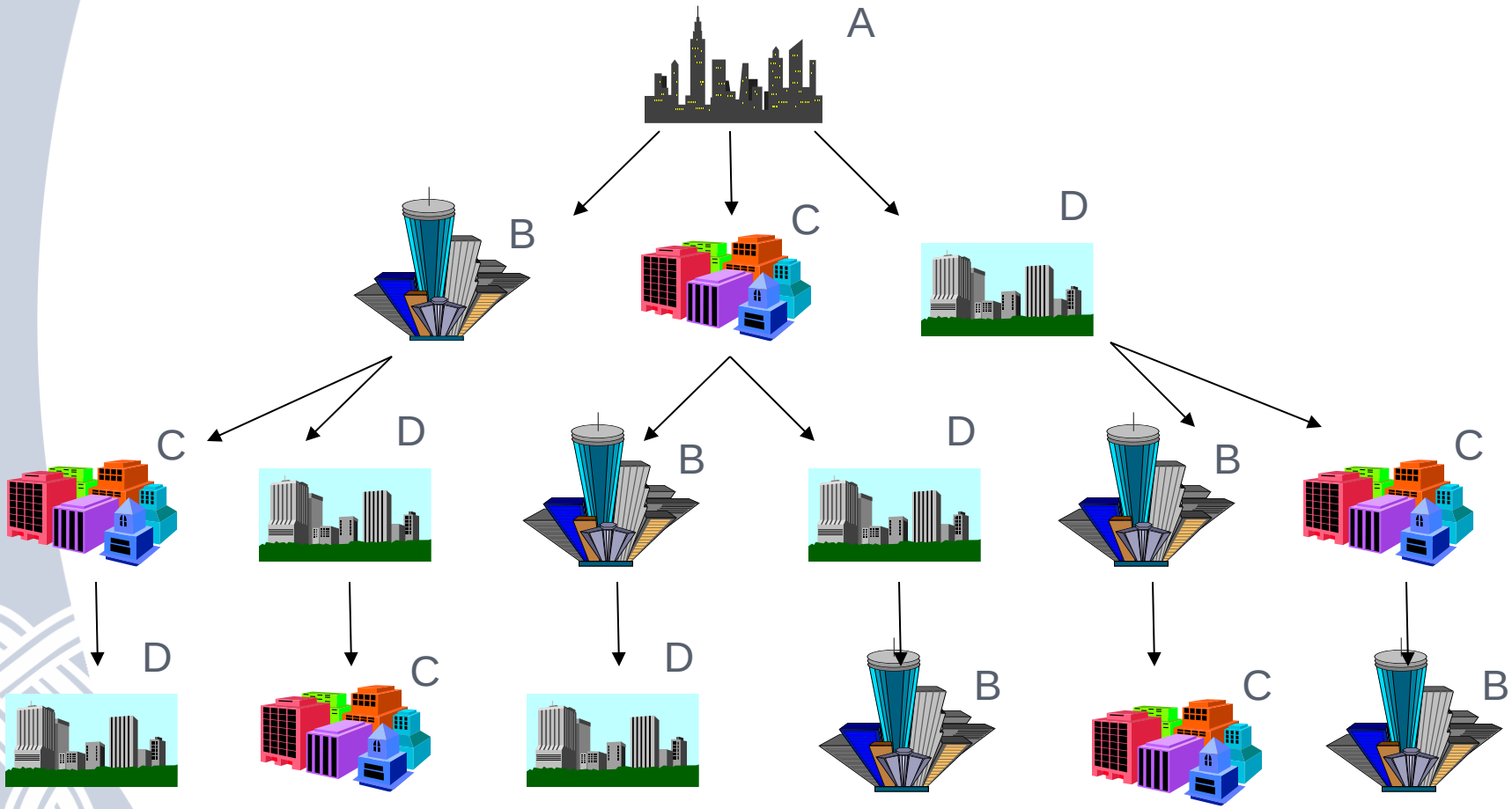
O PROBLEMA Caixeiro Viajante

Considere as rotas definidas entre estas 4 cidades:



O PROBLEMA TSP

representado como árvore de busca



PROBLEMA: EXPLOSÃO COMBINATÓRIA

- Com **quatro cidades**, temos **6 caminhos** possíveis.
- Com **dez cidades**, temos **362.880 caminhos** possíveis.
- Quanto mais cidades adicionarmos ao problema do caixeiro viajante, mais caminhos possíveis há.
  - O que nos leva a uma *explosão combinatória.*
  - Como prevenir ou pelo menos limitar isto?

BUSCA INFORMADA

- O problema do caixeiro viajante é basicamente um problema de busca.
- **Limitar** o **espaço de busca**, e assim tornar o processo de busca mais rápido e eficiente.
- Humanos utilizariam “macetes”; em IA são chamados de **Heurísticas**.
- **Estratégias de busca informada** utilizam **informação heurística** sobre o problema para calcular estimativas para os nós no espaço de estados. Essa estimativa indica o quanto o nó é promissor com relação a atingir a meta.

Heurística do vizinho mais próximo

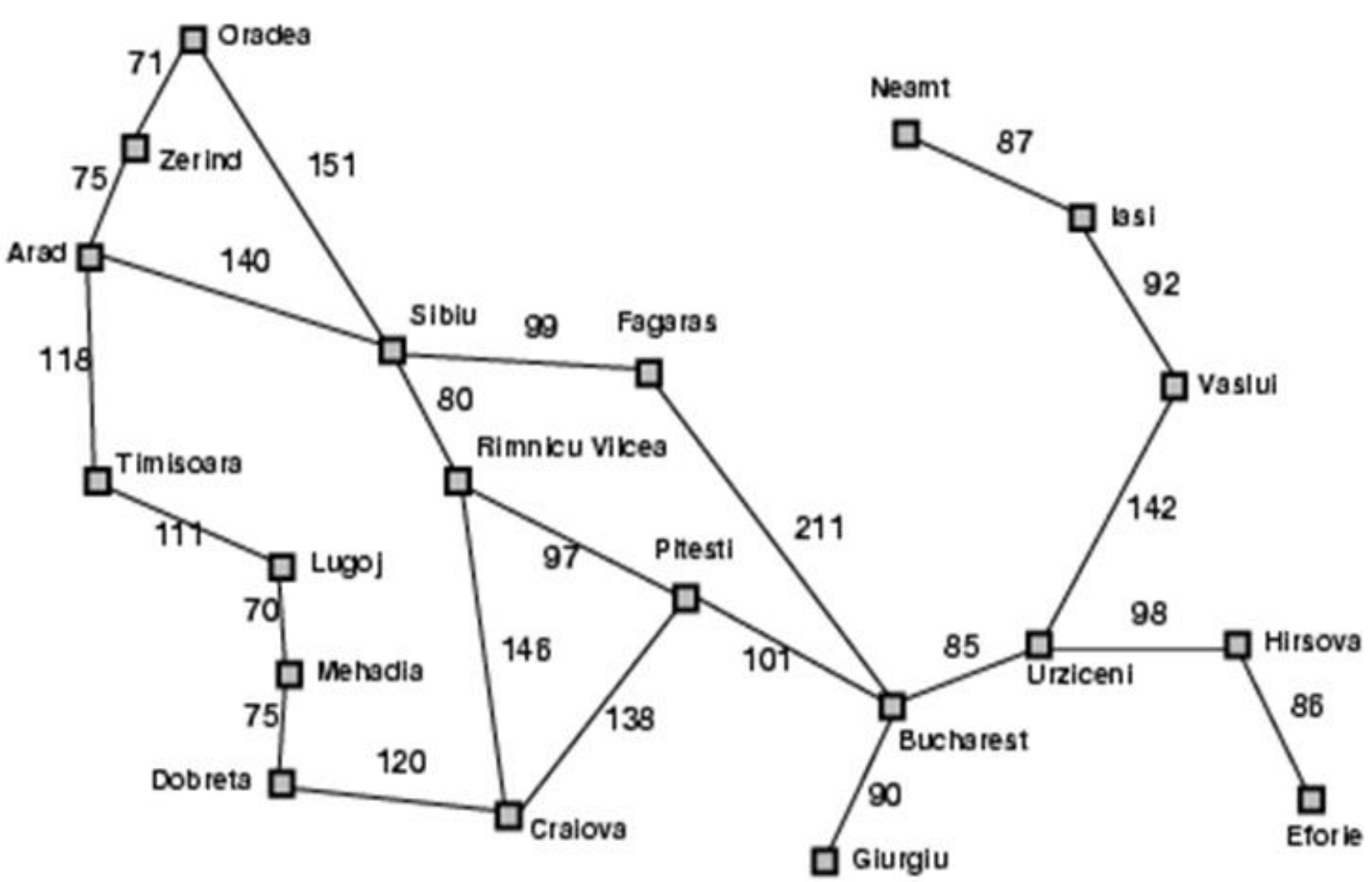
1. Arbitrariamente selecione uma cidade inicia
2. Para selecionar a próxima cidade, olhe todas as cidades ainda não visitadas e **selecione a mais próxima** a cidade corrente. Vá a ela.
3. Repita o passo 2 até que todas as cidades tenham sido visitadas.

Este procedimento é executado em tempo proporcional a N^2, uma melhora significativa sobre *N!*, que seria o tempo gasto caso fosse usada uma estratégia não informada (que geraria uma explosão *combinatorial*).

Busca pela Melhor Escolha

- Ideia: usar uma função de avaliação ***f(n)*** para cada nó
  - estimativa da "desejabilidade"
  - expandir o nó não expandido mais desejável
- Implementação:
  - Ordenar os nós na lista ABERTOS na ordem decrescente de "desejabilidade"
- Casos especiais:
  - busca gulosa pela melhor escolha
  - busca A*

Vamos considerar Romênia com Custo para explorar Busca Heurística Fonte: Russel e Norvig



Straight-line distance to Bucharest	
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

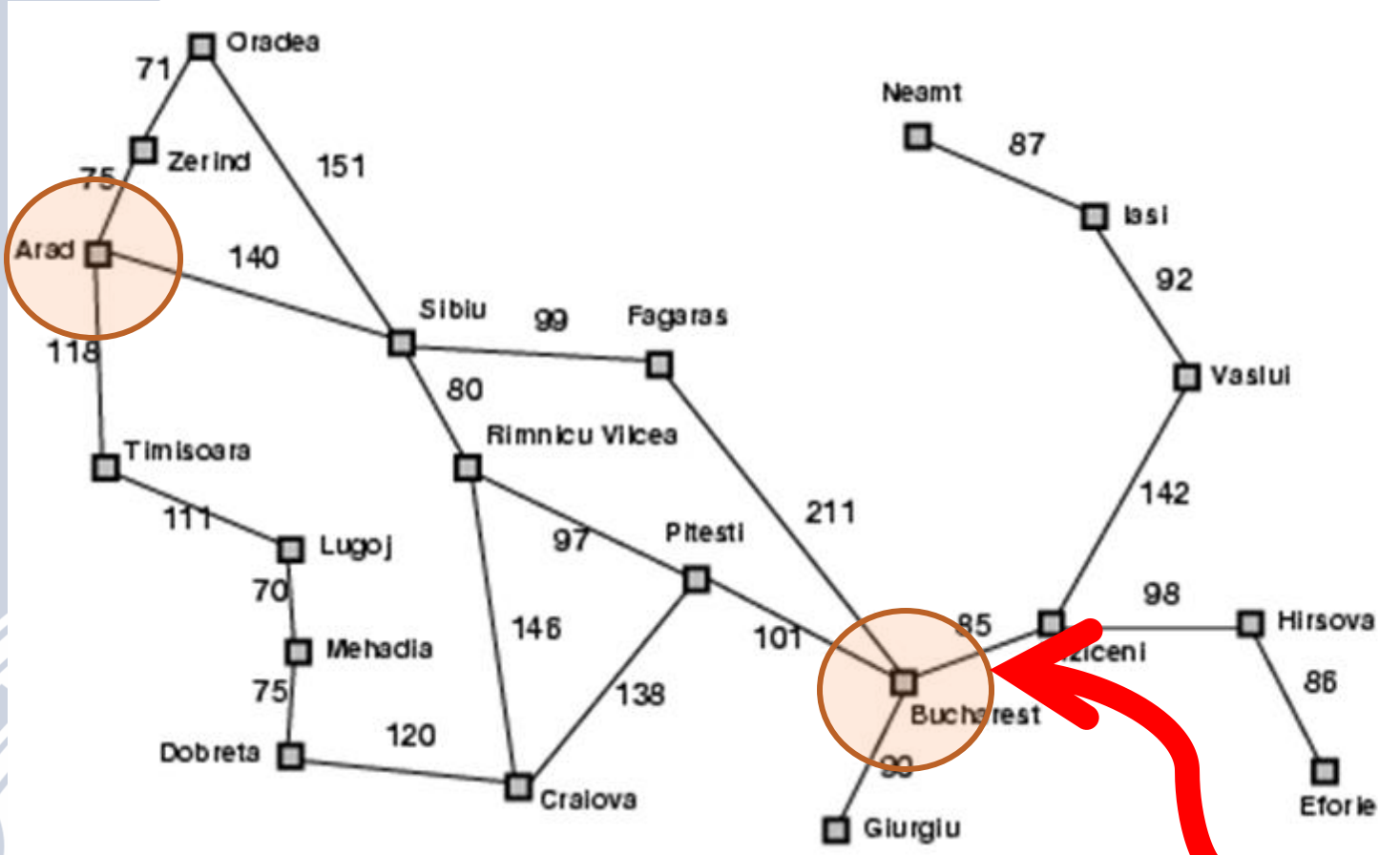
Busca Gulosa pela Melhor Escolha
GREEDY BEST-FIRST (GULOSO)

- A função de avaliação f(n) = ***h(n)*** (heurística) estima o custo de ***n*** ao **objetivo**
- No exemplo a seguir $h(n)$ será h_{DLR}(n) = a distância em linha reta de ***n*** a ***Bucharest*** (estado final)
- Busca gulosa pela melhor escolha expande o nó que parece levar mais perto do objetivo

Problema de localização de rotas na Romênia

Qual o caminho para sair de *Arad* e chegar em *Bucharest* usando

Busca gulosa pela Melhor Escolha?

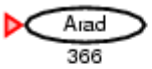


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Estado final

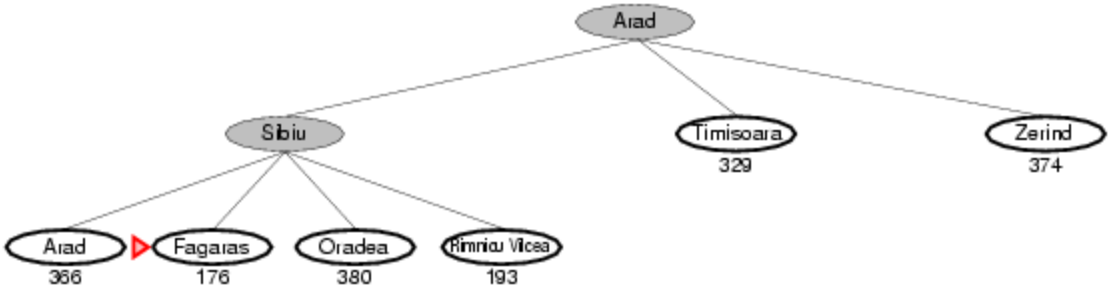
Exemplo: Livro Russel

GREEDY BEST-FIRST
Exemplo

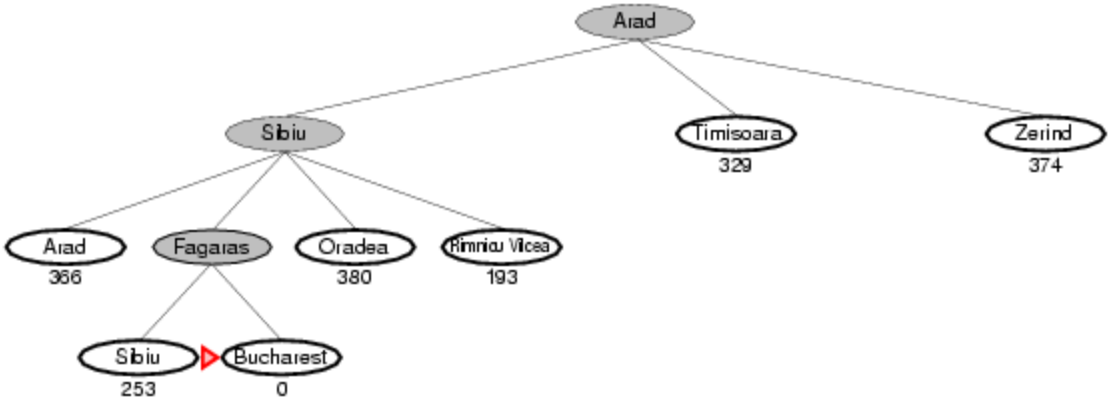


GREEDY BEST-FIRST
Exemplo

GREEDY BEST-FIRST
Exemplo



GREEDY BEST-FIRST
Exemplo



Propriedades da Busca Gulosa pela Melhor Escolha

- Completa? Não – pode ficar presa em loops (Lasi→Neamt→Lasi→Neamt →)
- Tempo? $O(b^m)$, mas uma boa heurística pode melhorar ele muito
- Espaço? $O(b^m)$ – mantém todos os nós em memória
- Ótimo? Não

FUNÇÕES DE AVALIAÇÃO E CUSTO

- Uma ***função de avaliação***, é uma função utilizada para estimar o valor ou uma boa posição e é **representada** por **h(*N*).** São usadas para evitar expandir caminhos que são caros.
- Além das FAs há outras funções que podem auxiliar o processo de busca, denominadas ***funções de custo***.
  - São funções não negativas que medem a dificuldade de ir de um estado para um outro.
- É importante observar que:
  - **FAs** se referem ao **futuro, denominadas h(n)**, "estimam" o quão perto está um estado ***n*** do estado meta,
  - **FCs** se referem ao **passado, denominadas g(n),** sabem quão longe está um estado ***n*** do estado inicial. Assim as FCs são mais concretas que as FAs

Heurística

A função heurística é a estimativa de custo de ir do **estado inicial** ***s*** até o estado final ***t*** passando pelo nó ***n*** e é representada por

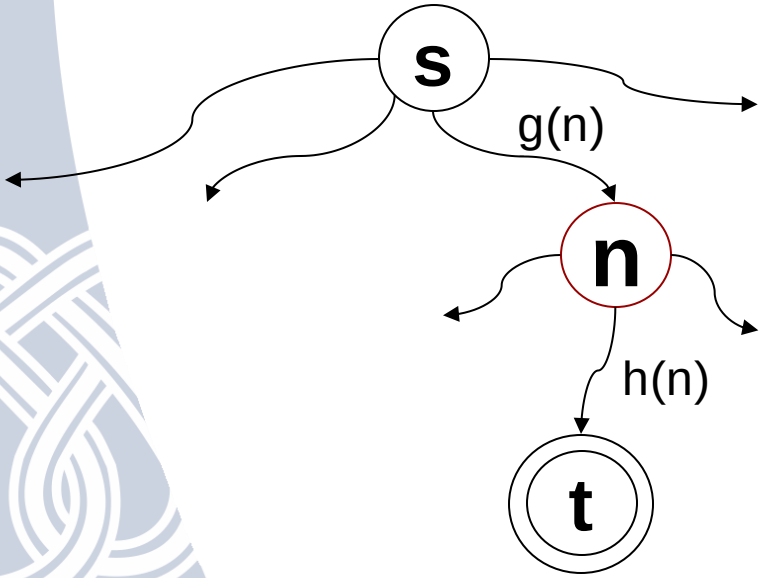
$$f(n) = g(n) + h(n)$$

no qual

- ***g(n)*** a **FC** do nó ***n***, *i.e.* o custo do caminho de **s** até ***n*** **(o passado)**
- ***h(n)*** é **FA** do no ***n***, *i.e.* uma estimativa para ir de n até **t (o futuro)**

Então considere:

- **g(n)** a FC do nó **n**, *i.e.* o custo do caminho de **s** até **n** **(o passado)**
- **h(n)** é FA do nó **n**, *i.e.* uma estimativa para ir de **n** até **t** **(futuro)**
- **f(n) = g(n) + h(n)** é a estimativa de custo de ir da raiz **s** até a meta **t** passando pelo nó **n**



Os **efeitos da heurística são *locais*** no sentido de "oferecer um conselho" referente a escolha do **sucessor de um estado específico**, mas não referente a toda a estratégia de busca.

A* - minimizando o custo total estimado da solução

- A ideia do A* é **adicionar os valores da FC e FA dos sucessores de um dado estado** e usar esses valores para **selecionar o estado sucessor mais promissor** para evitar a expansão de caminhos que são muito custosos
  - Para isso é importante que as unidades de ambas as funções sejam iguais.
  - Usada quando conhece tanto a FA quanto a FC (busca pela melhor escolha).

- O processo de busca visto como um conjunto de **sub-processos**, cada um explorando sua própria alternativa, ou seja, sua própria sub-árvore

A*

- Dentre todos os sub-processos apenas **um mantém-se ativo a cada momento**: aquele que lida com a alternativa atual mais promissora -- aquela com **menor valor de f**

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

- Os sub-processos restantes aguardam até que a estimativa **f** atual se altere e alguma outra alternativa se torne mais promissora
  - Então, a atividade é comutada para esta alternativa
  - O algoritmo usa um mecanismo de ativação-desativação

A*

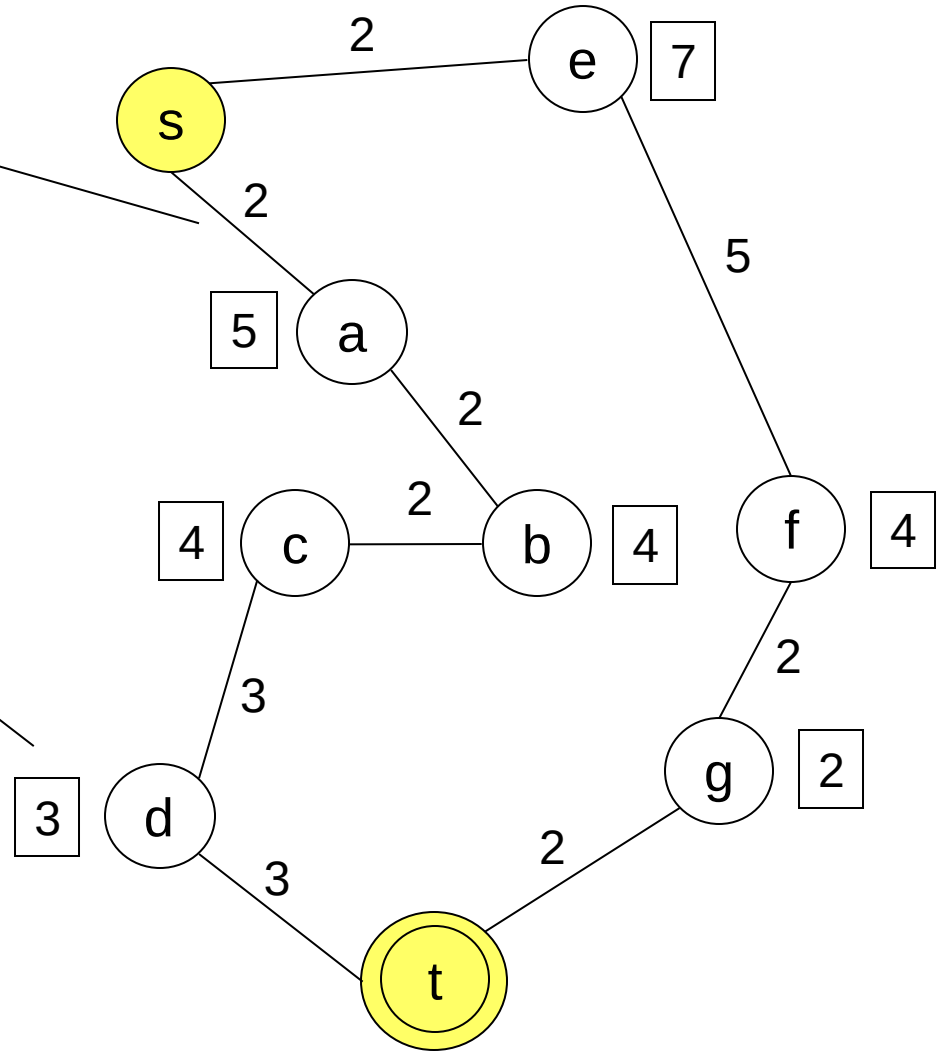
- A busca, começando pelo nó inicial continua **gerando novos nós sucessores**, sempre expandindo na direção mais promissora de acordo com os valores **f**
- Podemos imaginar o mecanismo de ativação-desativação da seguinte forma
  - O processo trabalhando na alternativa atual recebe um orçamento limite e permanece ativo até que o orçamento seja exaurido
  - Durante o período em que está ativo, o processo continua expandindo sua sub-árvore e relata uma solução caso um nó final seja encontrado
  - O orçamento limite para essa execução é definido pela estimativa heurística da alternativa competidora mais próxima

A*

- Considere o Grafo Exemplo 1

Distância entre duas cidades via um caminho (rodovia)
FC **g(n)**

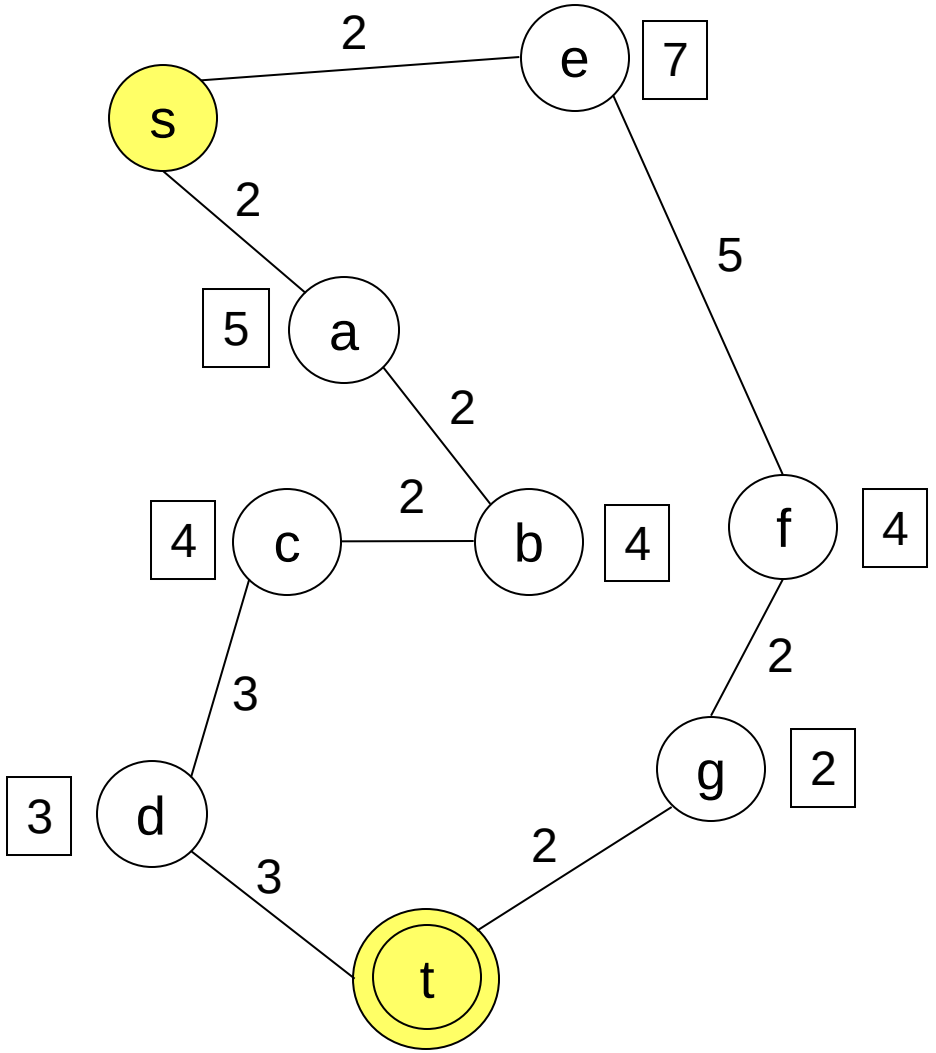
Distância entre a cidade em questão e a cidade destino (t) em linha reta
FA **h(n)**



A*

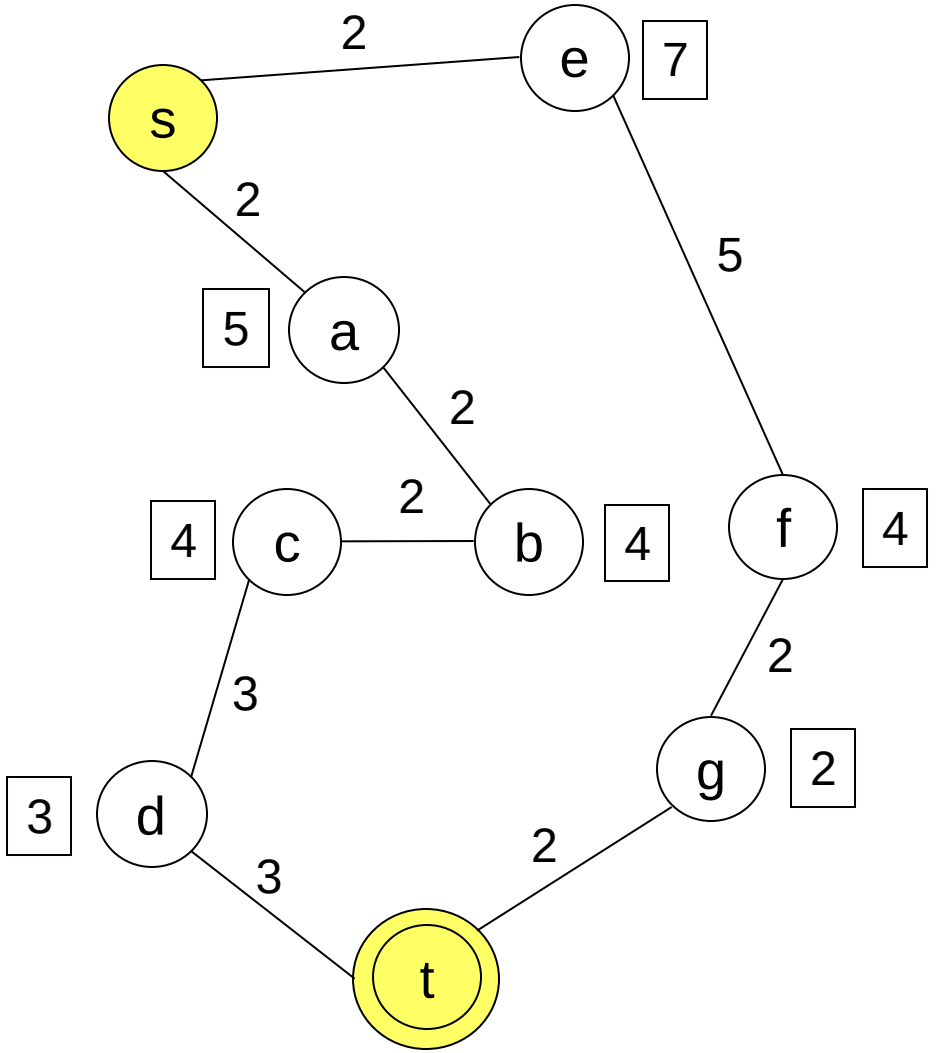
- Dado um mapa, o objetivo é encontrar o caminho mais curto entre a cidade inicial **s** e a cidade destino **t**
  - **FC** (g(X)) - Distância entre duas cidades
  - Para estimar –**FA** (h(X))- uma heurística para calcular o caminho restante da cidade X até a cidade **t** é utilizada a distância em linha reta denotada por **dist(X,t)**

- **f(X) = g(X) + h(X) =**
  = g(X) + dist(X,t)



A*

- Neste exemplo, podemos imaginar a busca consistindo em dois processos, cada um explorando um dos caminhos alternativos
- Processo 1 explora o caminho via **a**
- Processo 2 explora o caminho via **e**

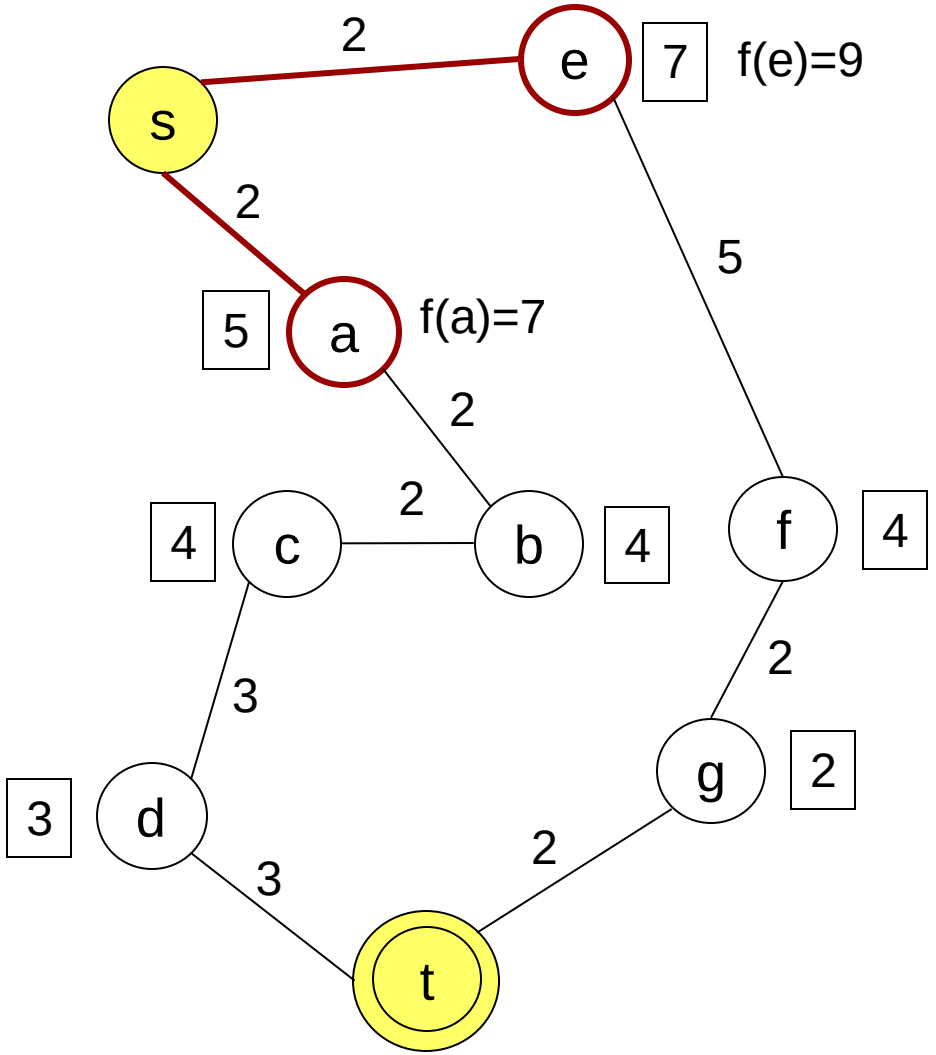


A*

- f(a)=g(a)+dist(a,t)=2+5=7
- f(e)=g(e)+dist(e,t)=2+7=9
- Como o valor-f de **a** é menor do que de **e**,
- processo 1 (busca via **a**) permanece ativo

enquanto

- processo 2 (busca via **e**) fica em estado de espera



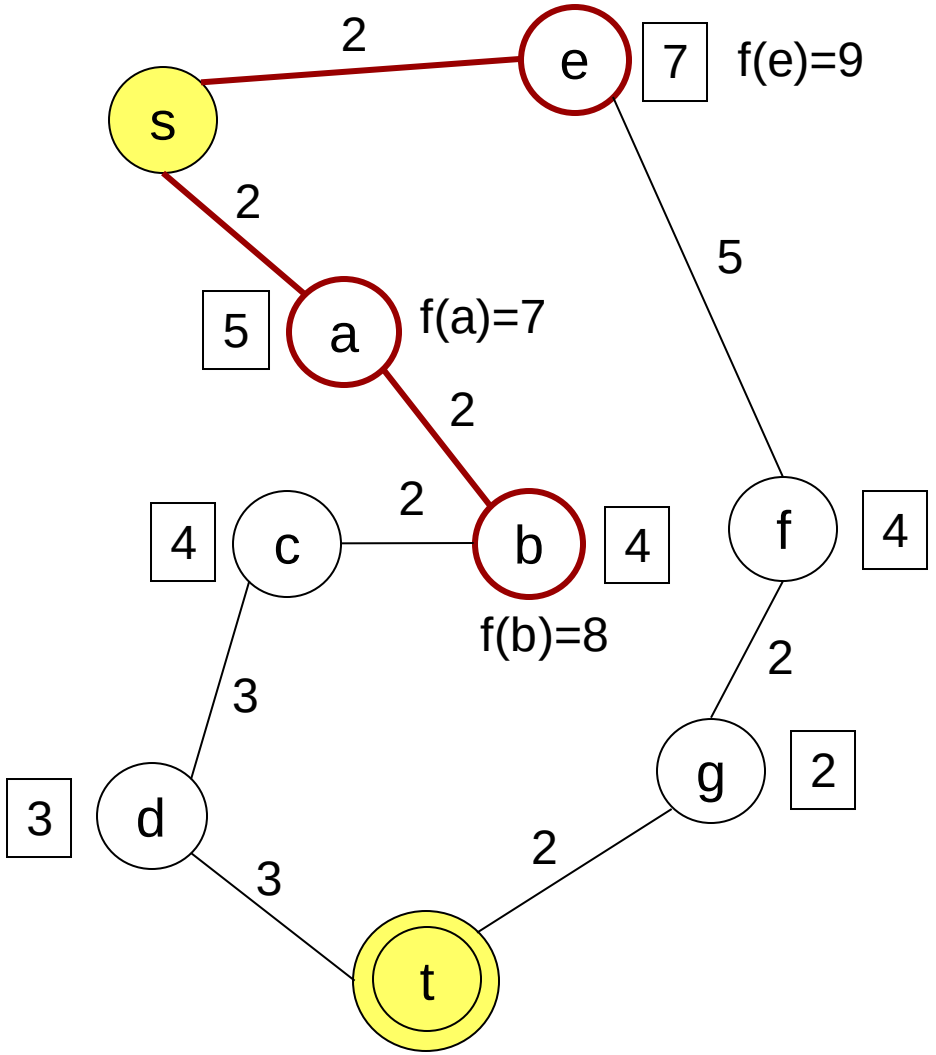
A*

- f(a)=g(a)+dist(a,t)=2+5=7
- f(e)=g(e)+dist(e,t)=2+7=9
- Como o valor-f de **a** é menor do que de **e**,
- processo 1 (busca via **a**) permanece ativo

enquanto

- processo 2 (busca via **e**) fica em estado de espera

- f(b)=g(b)+dist(b,t)=4+4=8

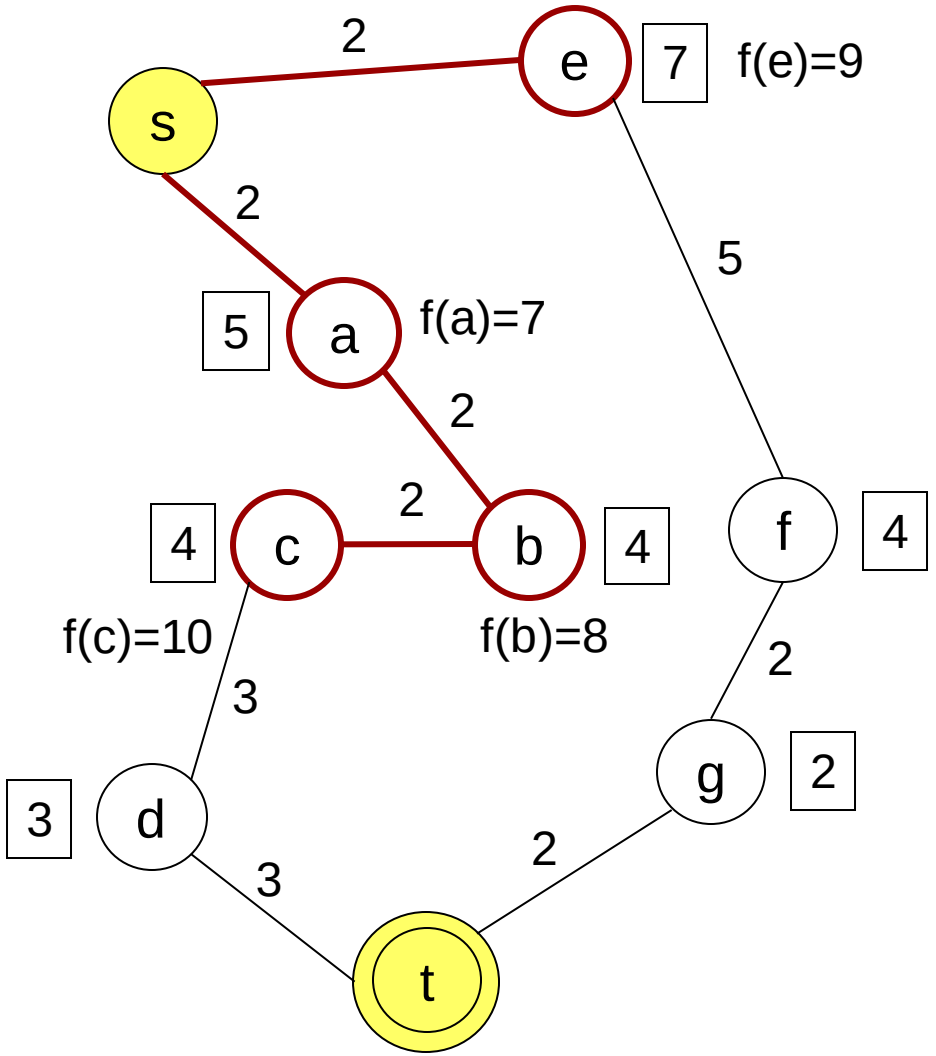


A*

- f(a)=g(a)+dist(a,t)=2+5=7
- f(e)=g(e)+dist(e,t)=2+7=9
- Como o valor-f de **a** é menor do que de **e**,
- processo 1 (busca via **a**) permanece ativo

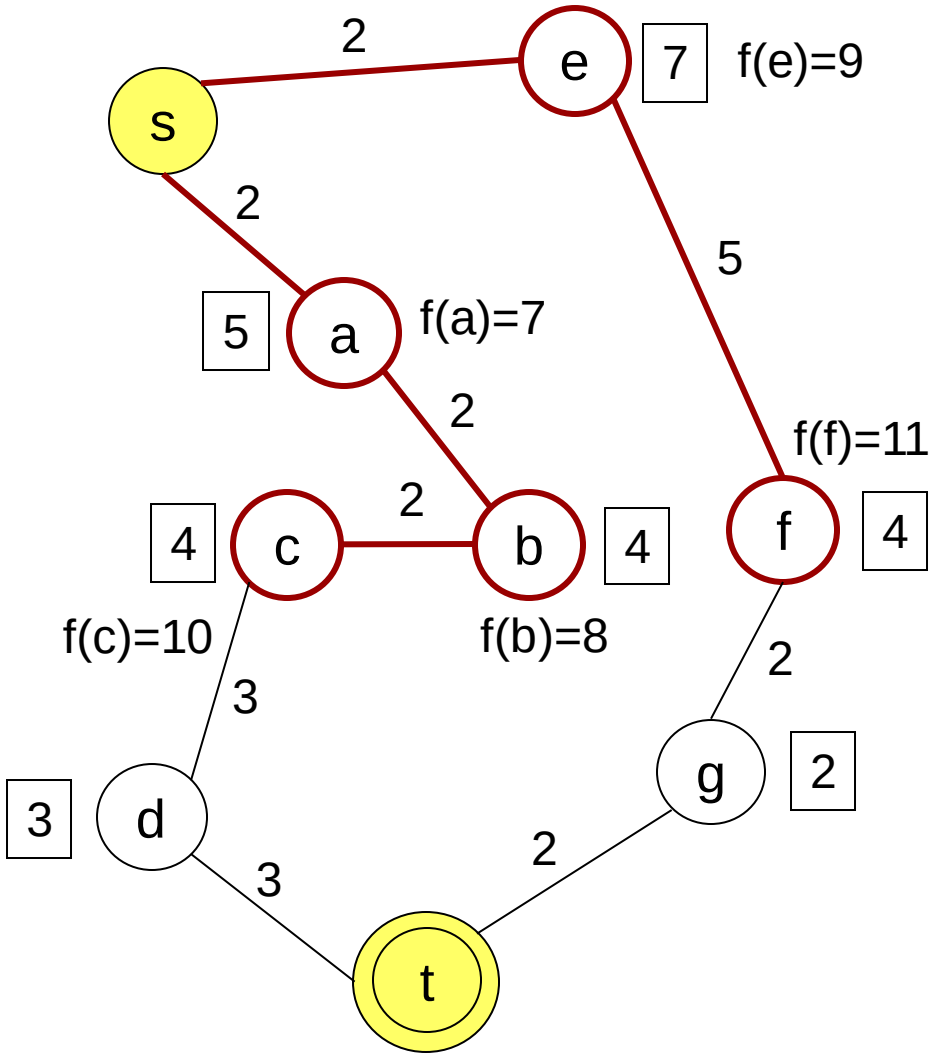
enquanto o

- processo 2 (busca via **e**) fica em estado de espera
- f(b)=g(b)+dist(b,t)=4+4=8
- f(c)=g(c)+dist(c,t)=6+4=10
- Como f(e)<f(c) agora o processo 2 prossegue para a cidade **f**



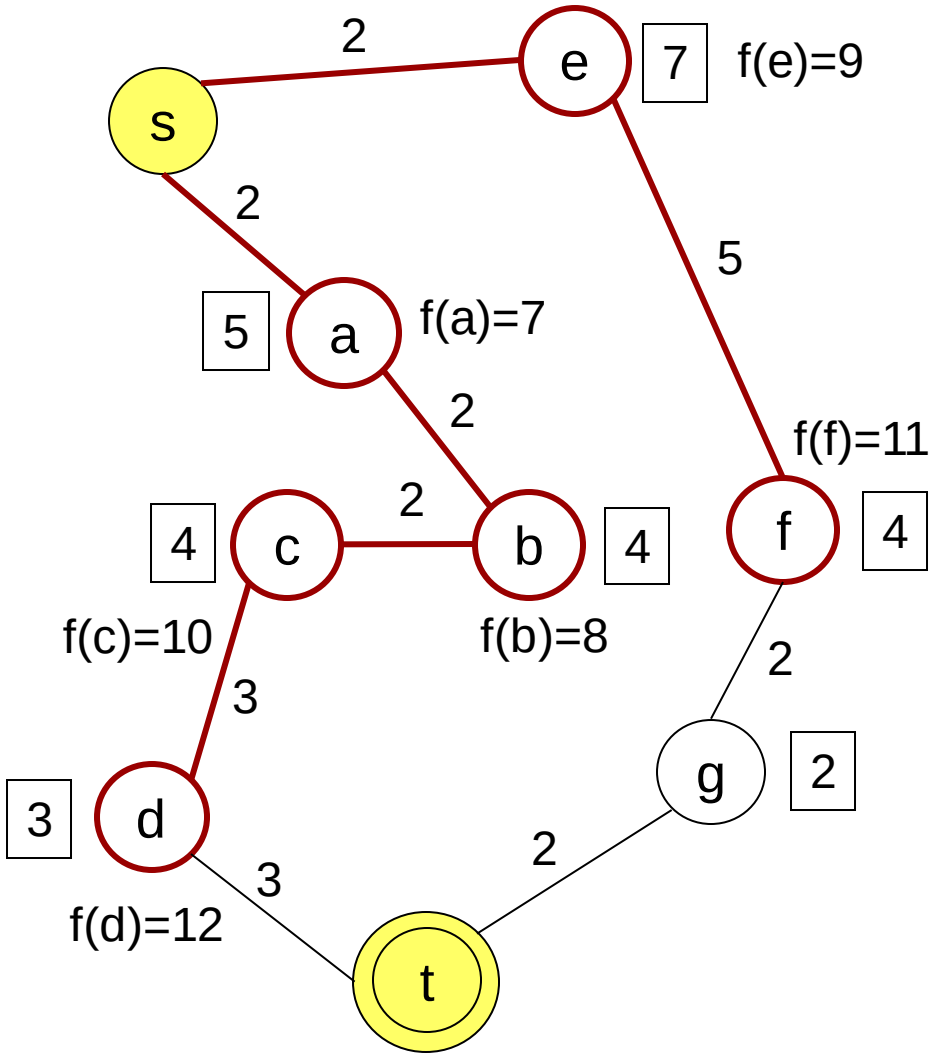
A*

- f(f)=g(f)+dist(f,t)=7+4=11
- Como f(f)>f(c) agora
- processo 2 espera e
- processo 1 prossegue



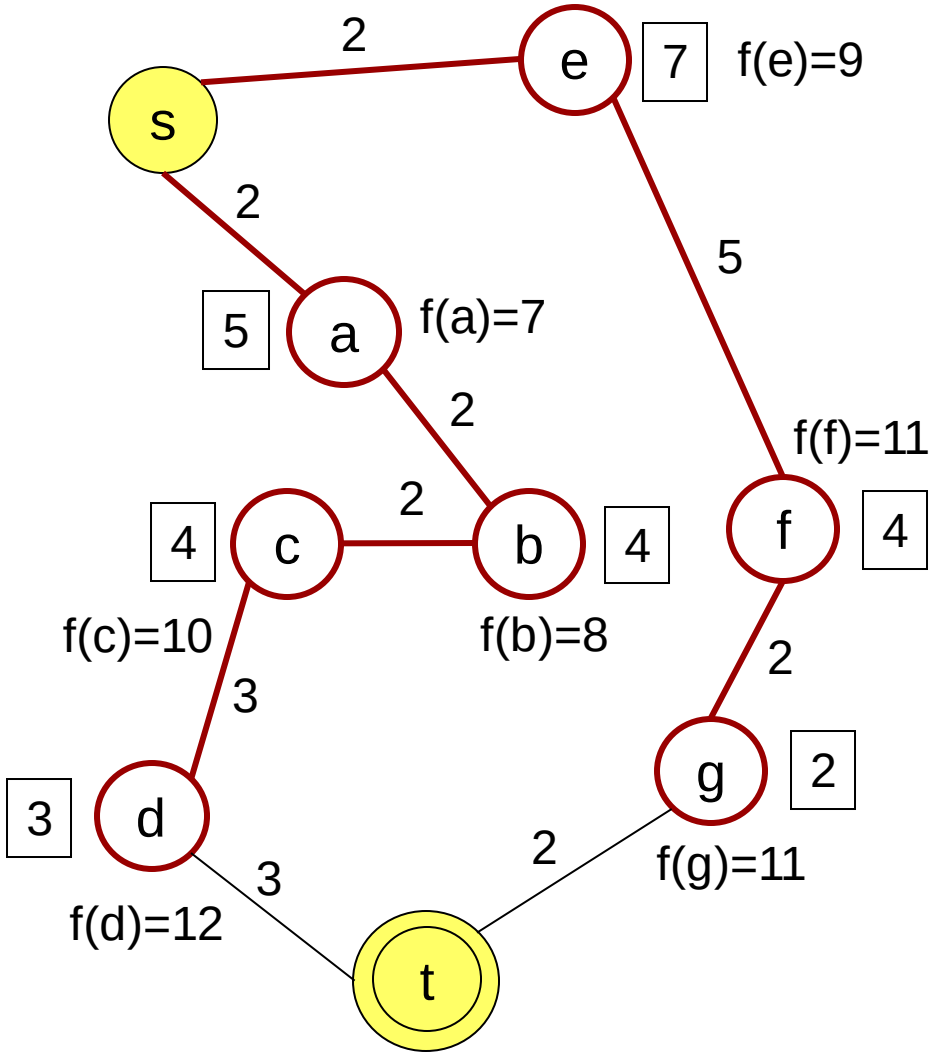
A*

- f(f)=g(f)+dist(f,t)=7+4=11
- Como f(f)>f(c) agora
- processo 2 espera e
- processo 1 prossegue
- f(d)=g(d)+dist(d,t)=9+3=12
- Como f(d)>f(f)
- o processo 2 reinicia



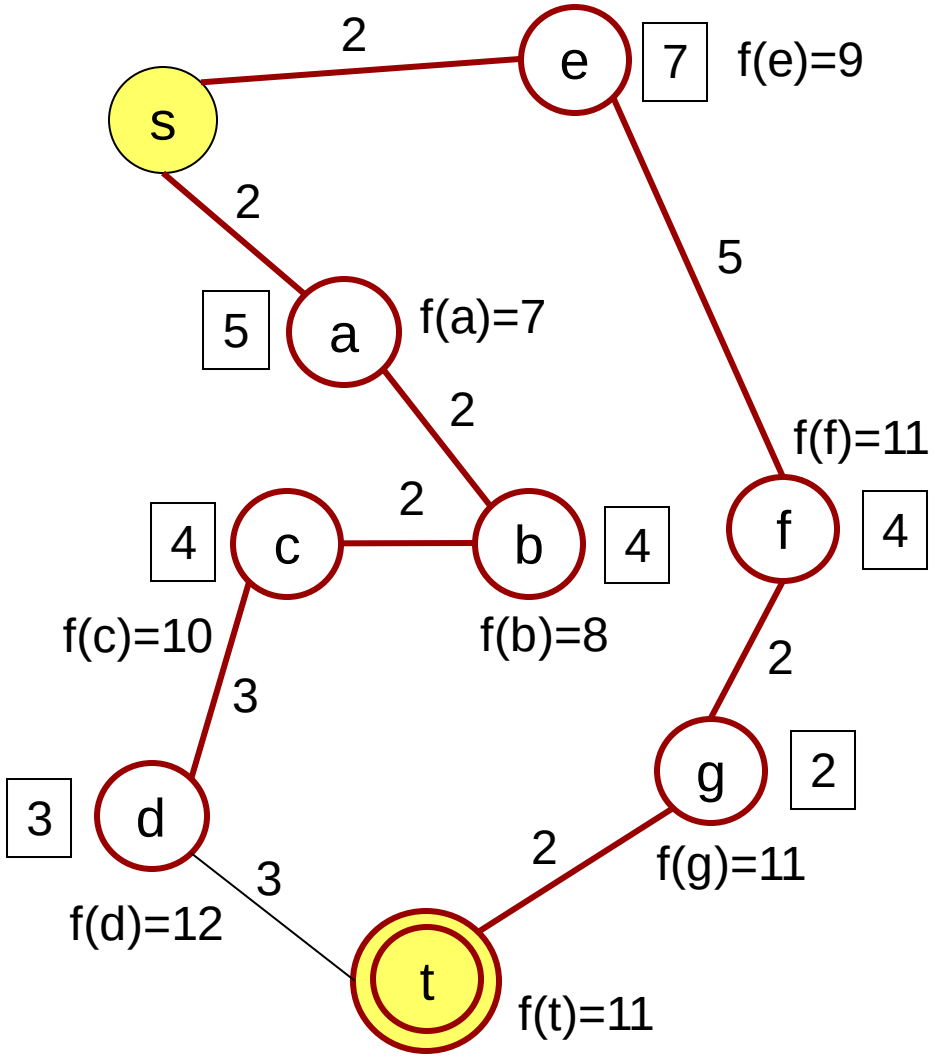
A*

- f(f)=g(f)+dist(f,t)=7+4=11
- Como f(f)>f(c) agora o processo 2 espera e o processo 1 prossegue
- f(d)=g(d)+dist(d,t)=9+3=12
- Como f(d)>f(f) o processo 2 reinicia **chegando até o destino t**
- f(g)=g(g)+dist(g,t)=9+2=11



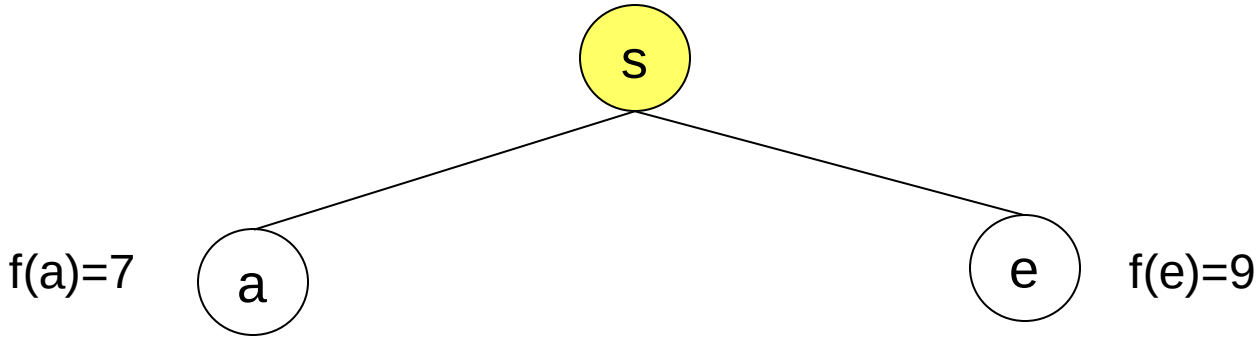
A*

- f(f)=g(f)+dist(f,t)=7+4=11
- Como f(f)>f(c) agora o processo 2 espera e o processo 1 prossegue
- f(d)=g(d)+dist(d,t)=9+3=12
- Como f(d)>f(f) o processo 2 reinicia chegando até o destino t
- f(g)=g(g)+dist(g,t)=9+2=11
- f(t)=g(t)+dist(t,t)=11+0=11



A*

- Exemplo 1 – Arvore de Busca

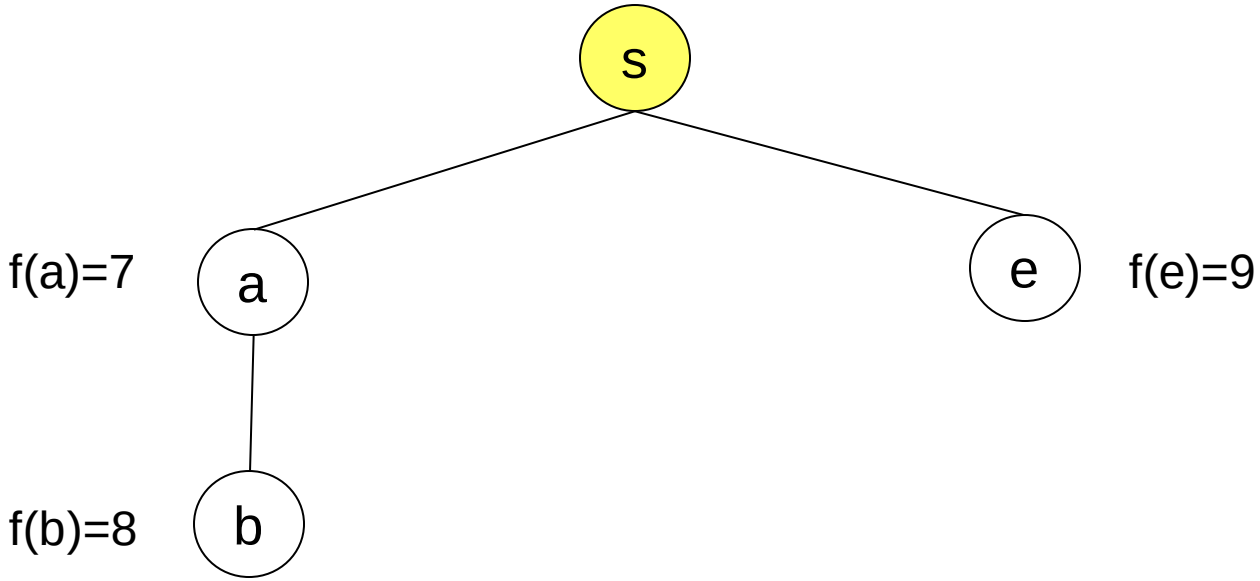


- Durante o processo, uma **árvore de busca** é gerada tendo como raiz o **nó inicial** e

- o algoritmo A* continua expandindo a árvore de busca até que uma solução seja encontrada

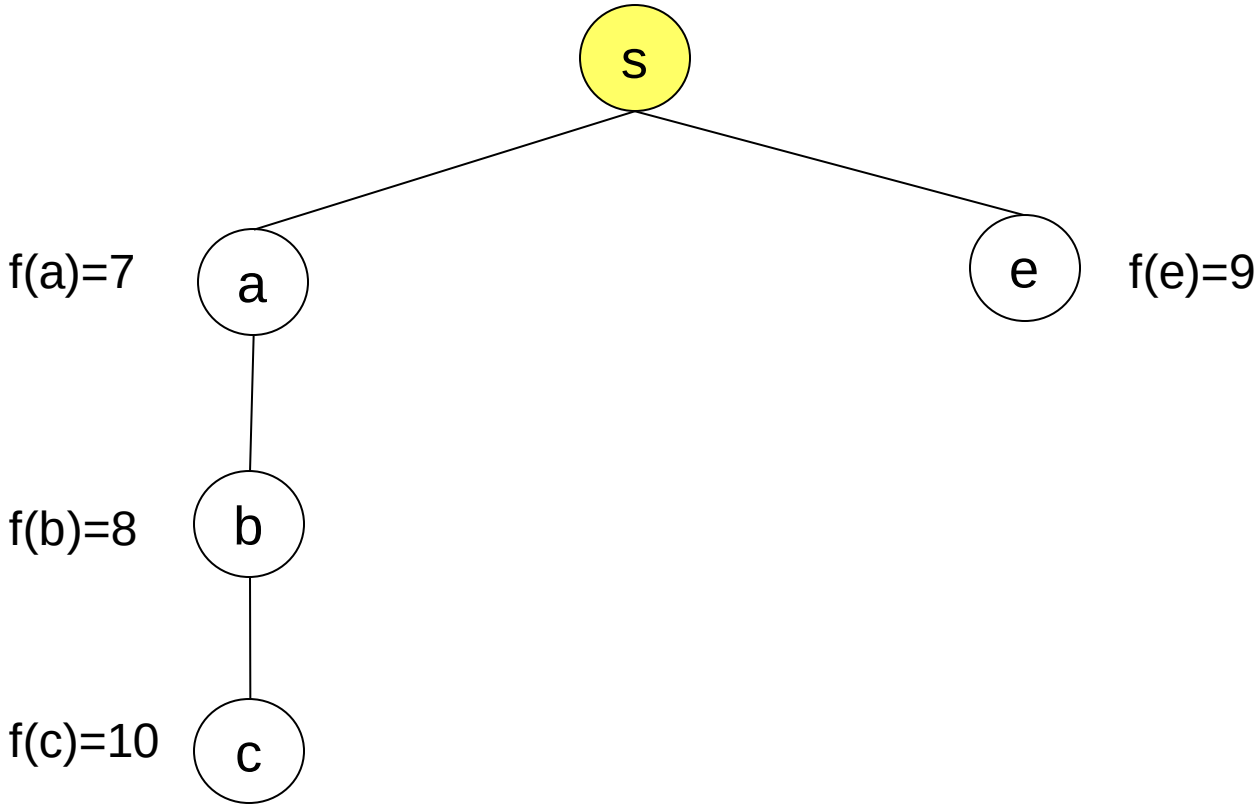
A*

- Exemplo 1 – Arvore de Busca



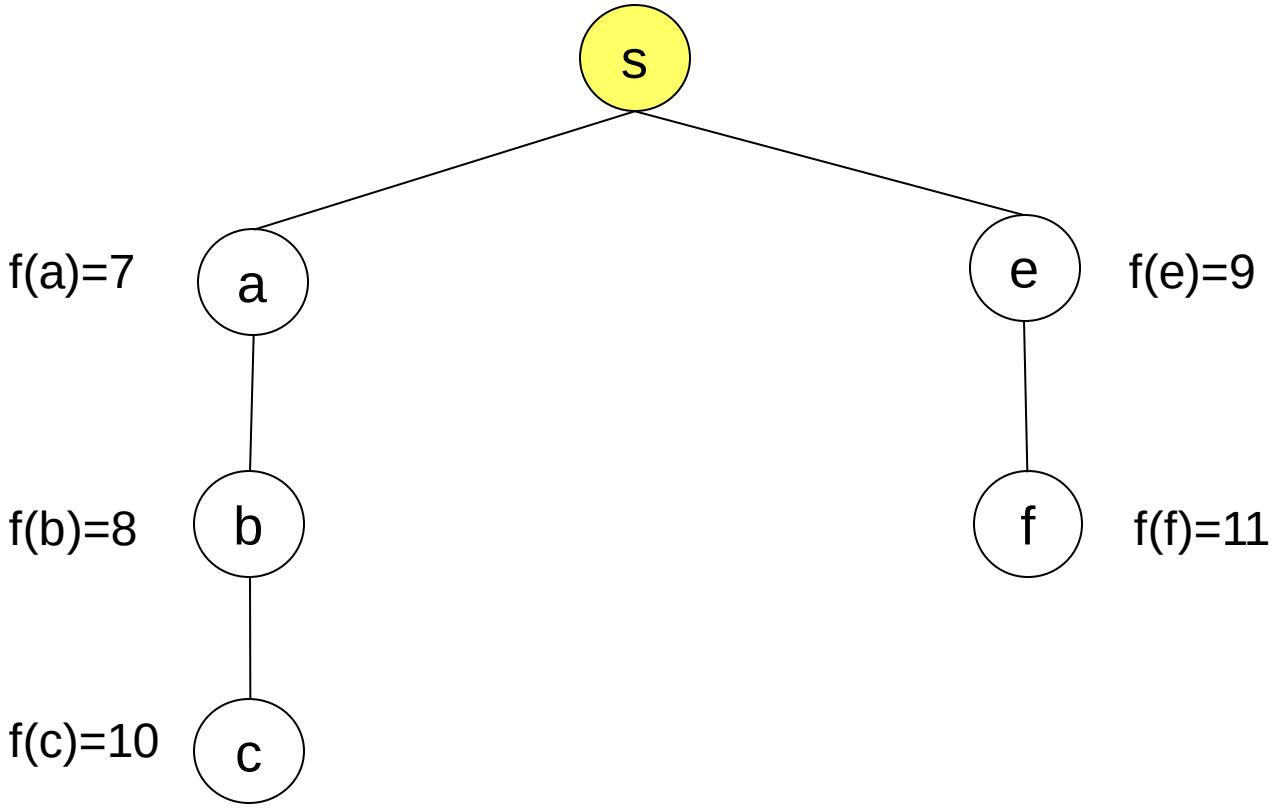
A*

- Exemplo 1 – Arvore de Busca



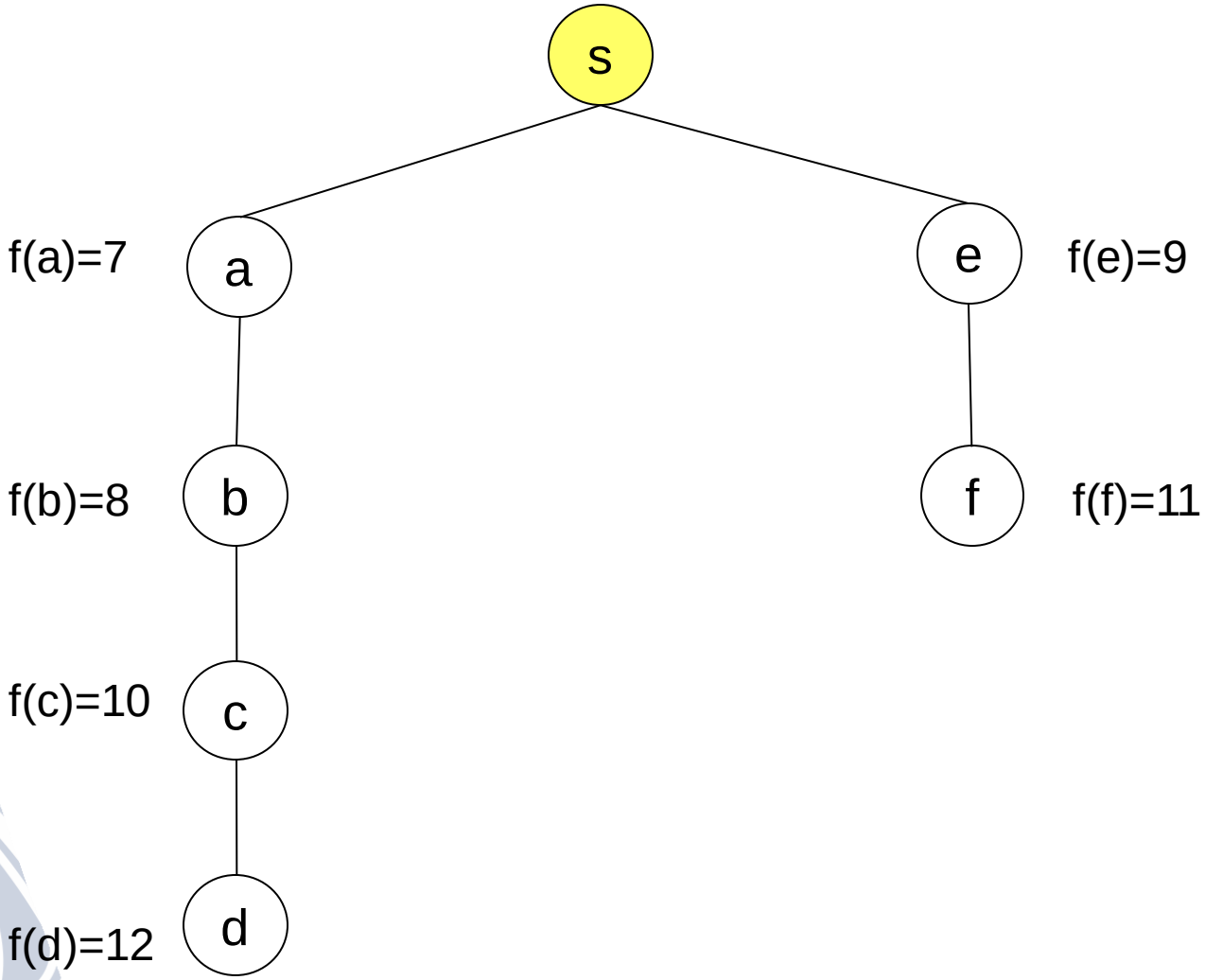
A*

- Exemplo 1 – Arvore de Busca



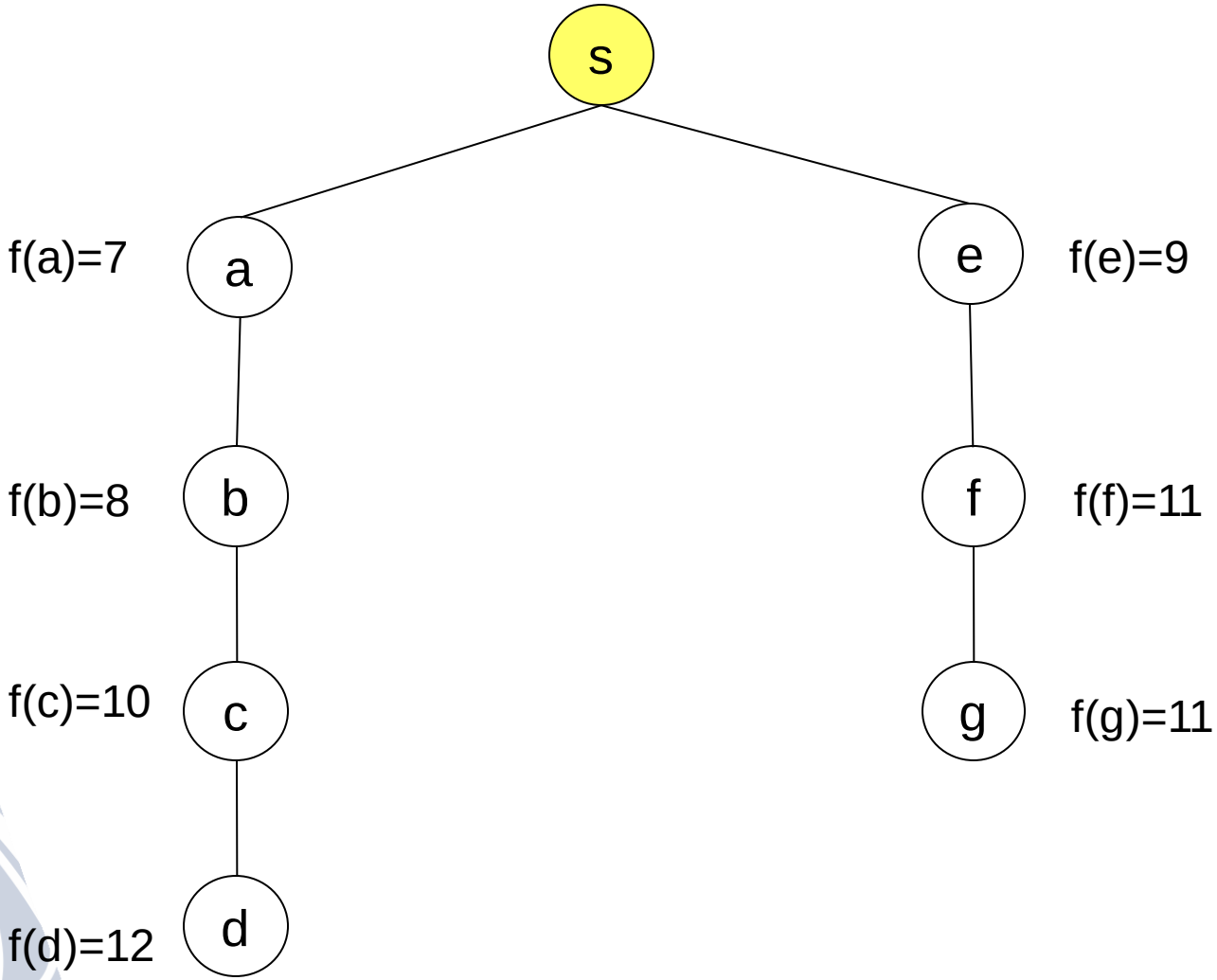
A*

- Exemplo 1 – Arvore de Busca



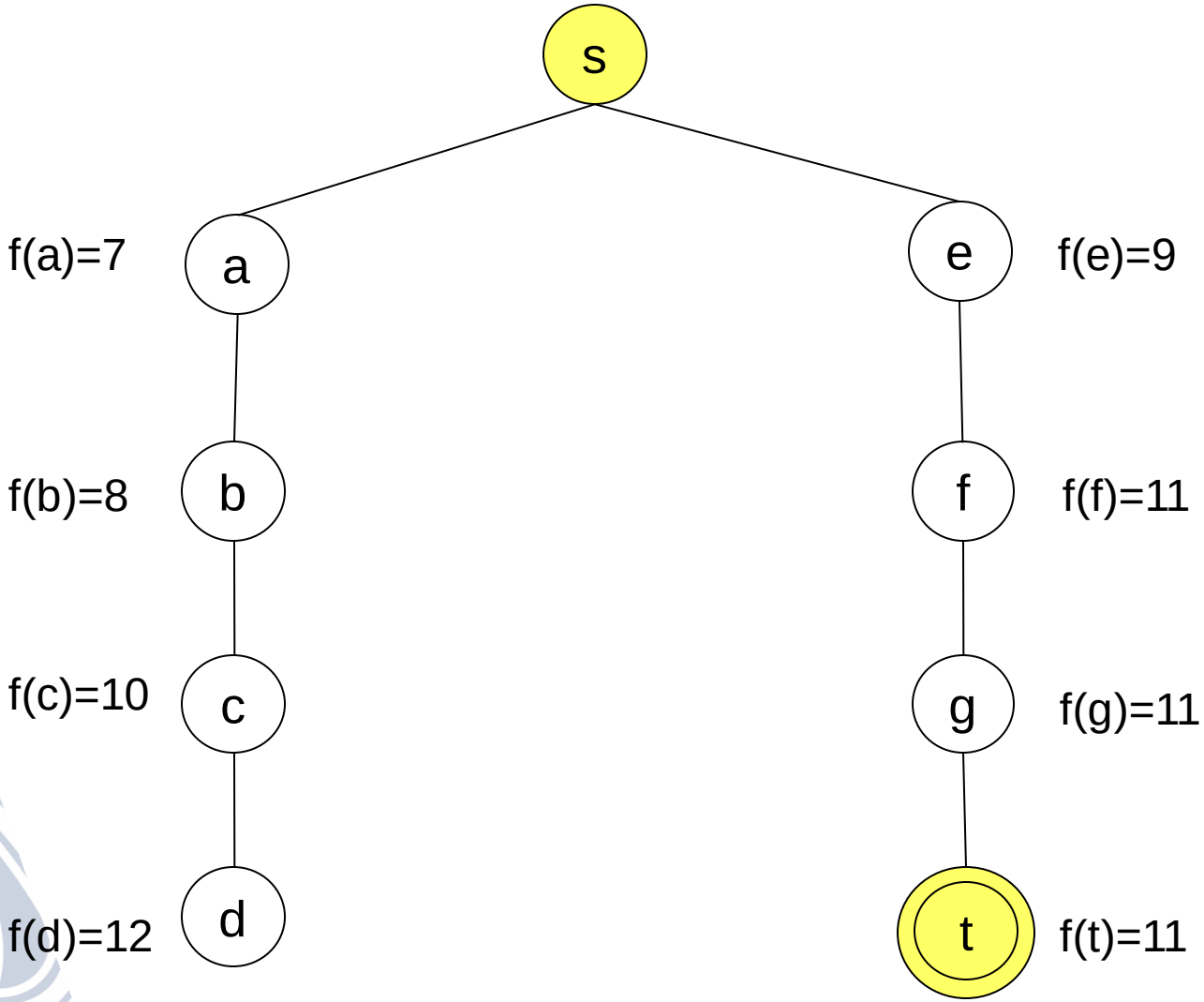
A*

- Exemplo 1 – Arvore de Busca

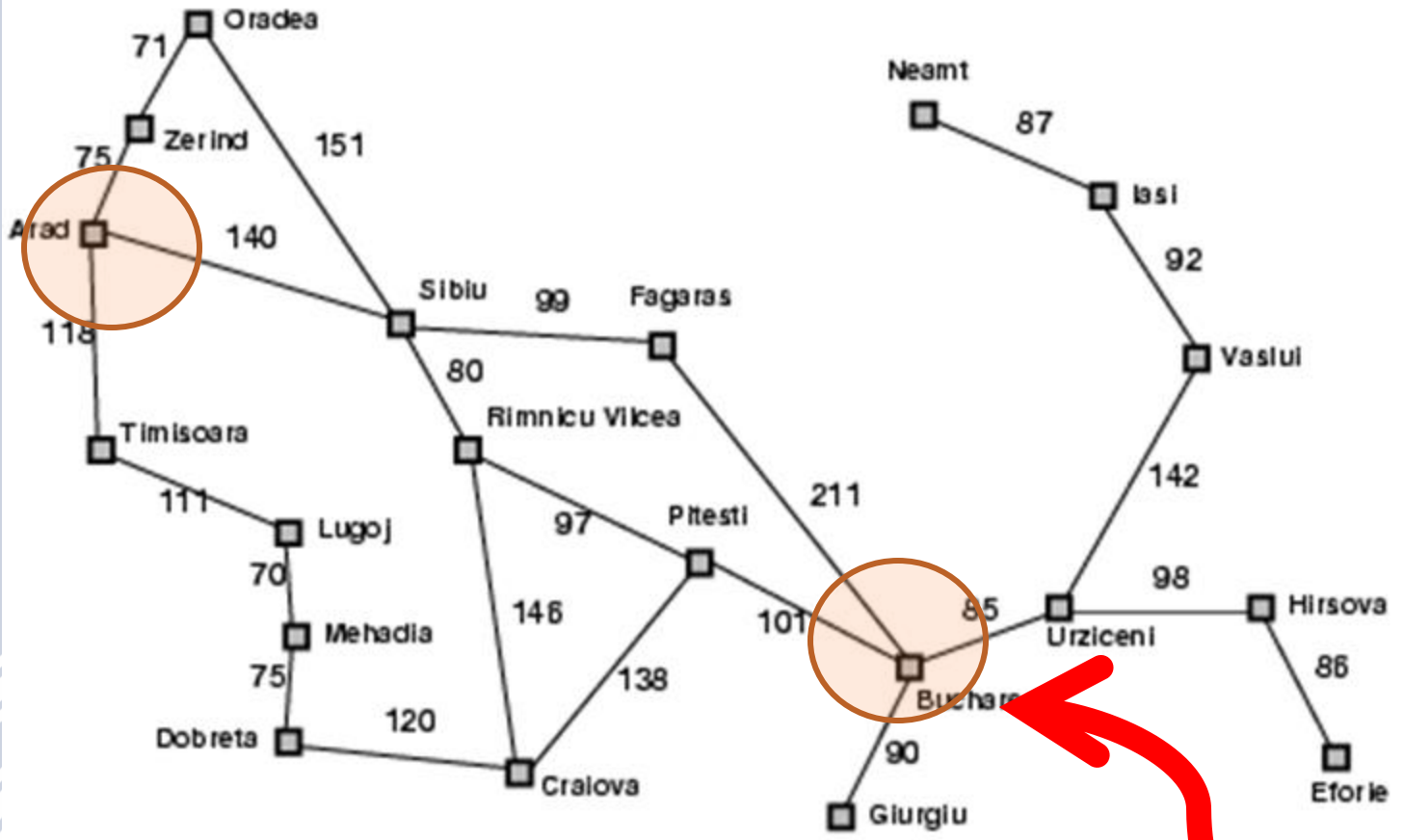


A*

- Exemplo 1 – Arvore de Busca



Problema de localização de rotas na Romênia
Qual o caminho para sair de *Arad* e chegar em *Bucharest* usando **A***?

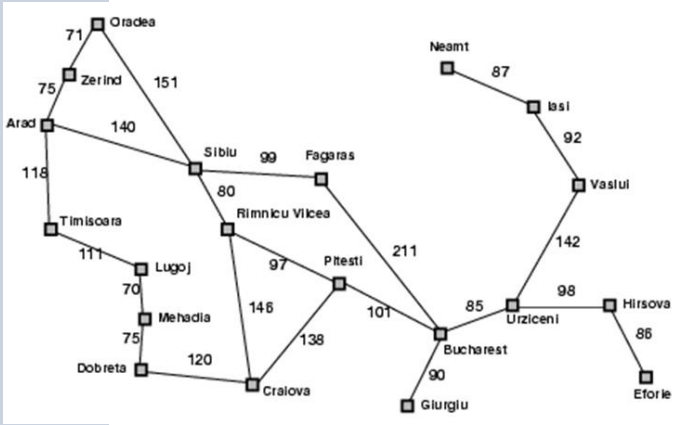


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Estado final

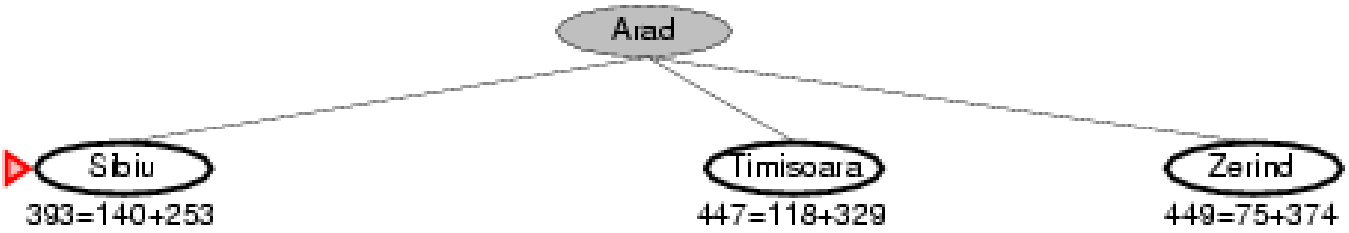
Exemplo: Livro Russel

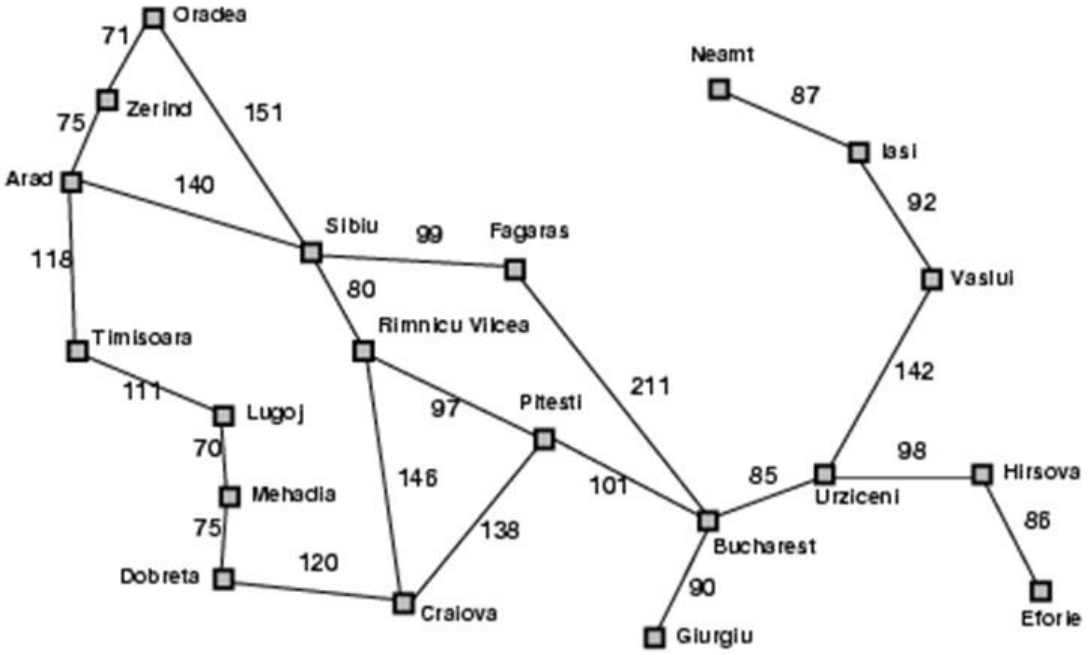
Qual o caminho para sair de *Arad* e chegar em *Bucharest* usando **A***?



Straight-line distance to Bucharest	
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

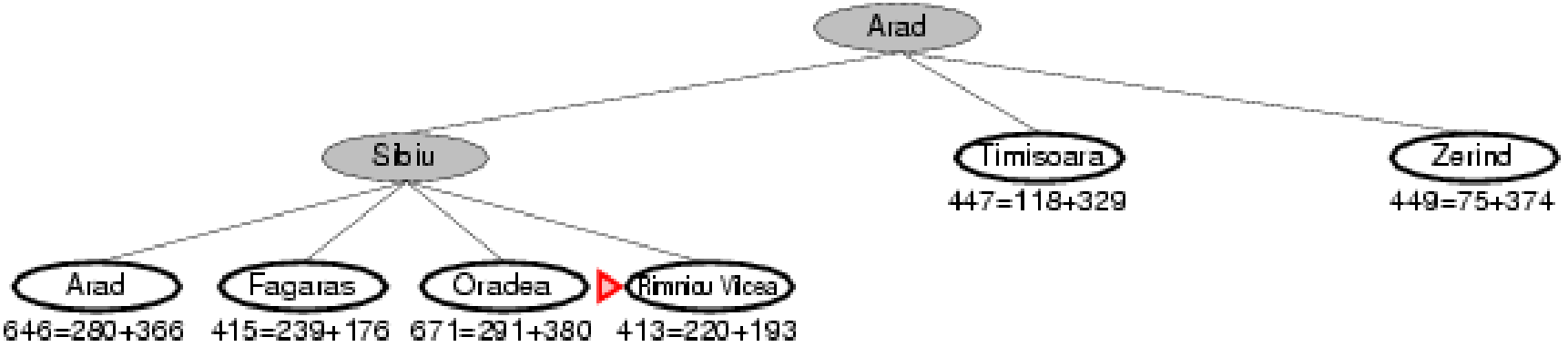
A*

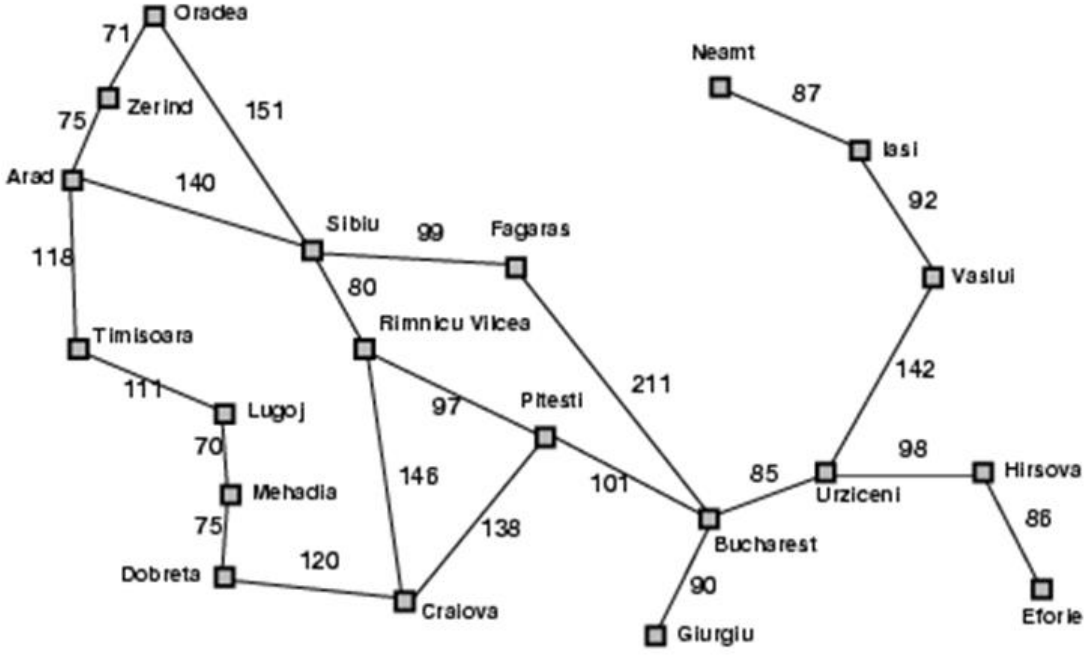




Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A*

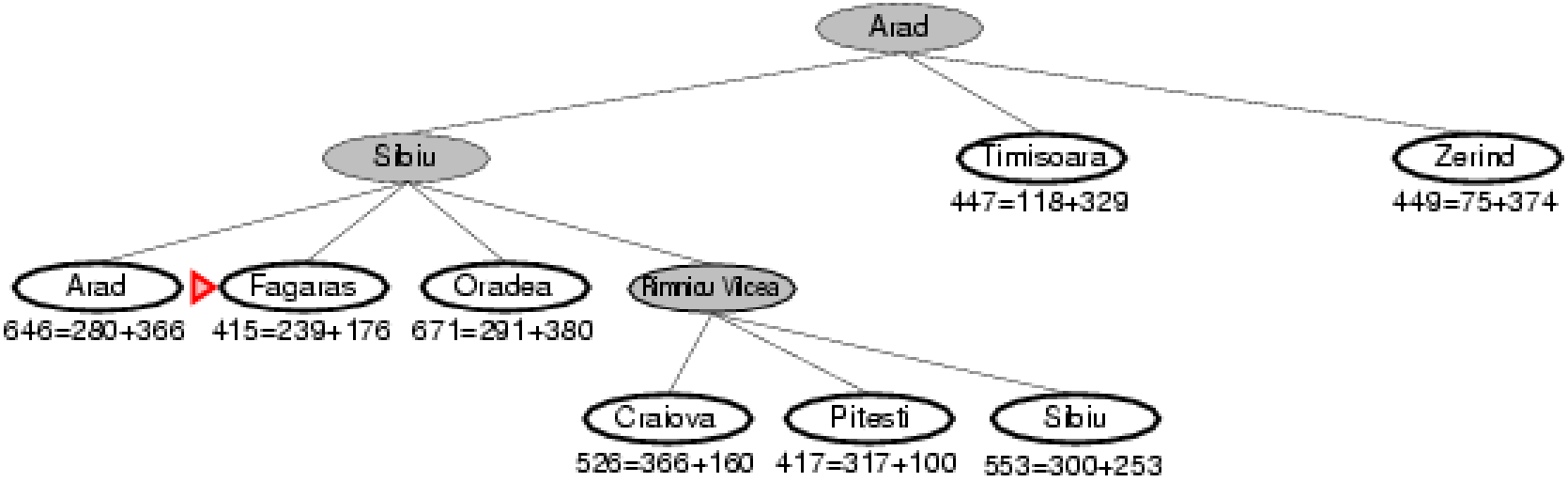


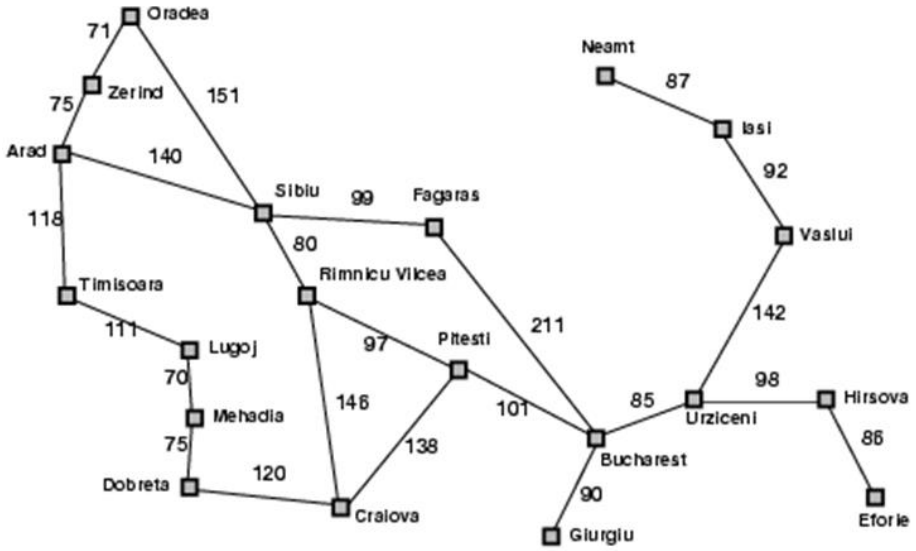


Straight-line distance to Bucharest

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*

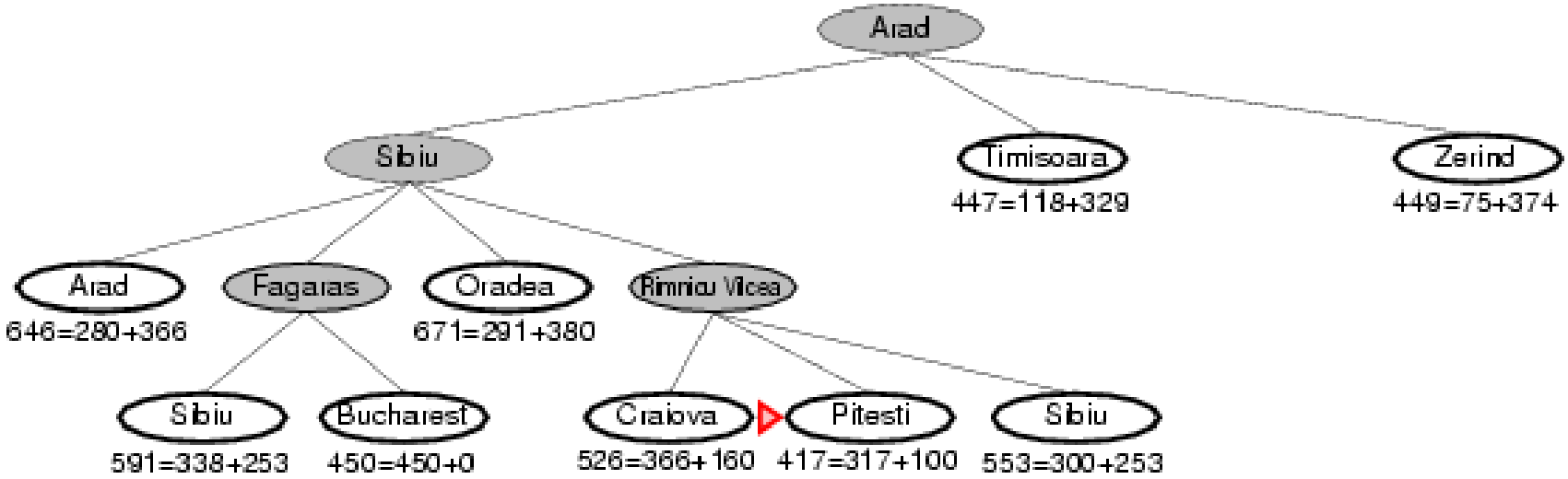


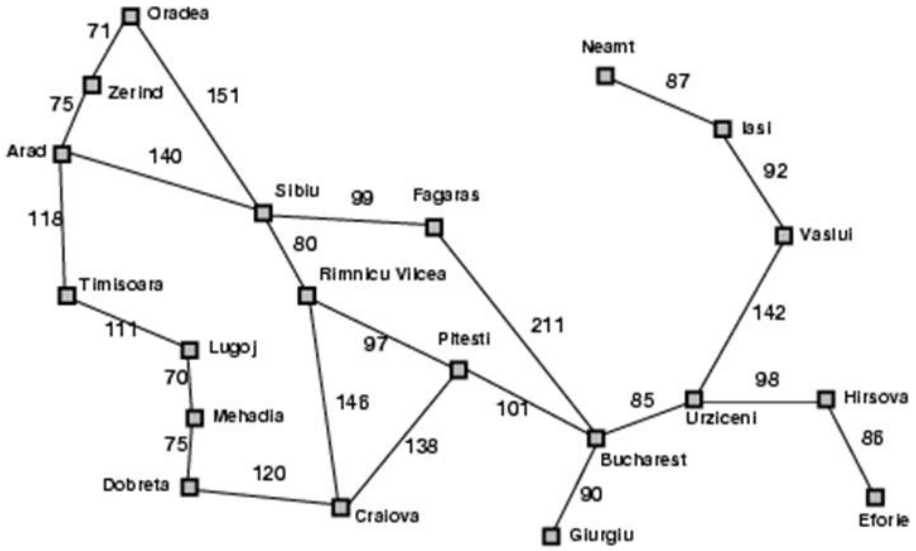


Straight-line distance to Bucharest

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

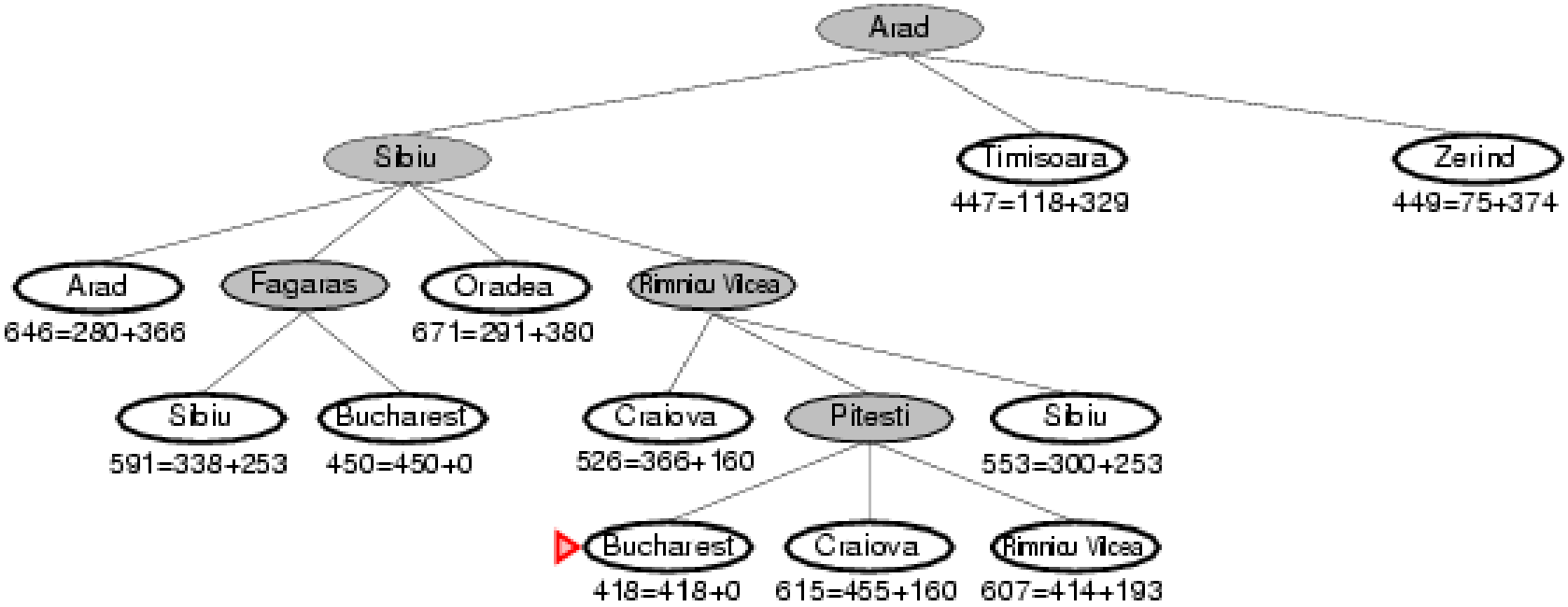
A*





Straight-line distance to Bucharest	
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*


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

A*

- **A*** possui uma propriedade muito importante:
  - Se a FA para qualquer estado ***e*** é sempre menor ou igual ao custo real de ***e*** para a **meta**, então o **primeiro caminho** encontrado pela **estratégia de busca A*** é o **caminho de custo mínimo** (ótimo).

HEURÍSTICA ADMISSÍVEL

- Uma heurística ***h(n)*** é admissível

  se para cada nó ***n***, $h(n) \leq h^*(n)$, em que $h^*(n)$ é o custo real de atingir o estado objetivo desde ***n***.

- Uma heurística admissível nunca superestima o custo de atingir o objetivo; ela é otimista

  - **Obs:** A distância em linha reta é uma heurística admissível (nunca superestima a distância rodoviária real)

***Teorema*:**

Se *h(n) é* admissível, a estratégia A^* que usa busca em grafos é ótima

Busca Admissível

- Um algoritmo de busca é **admissível** se ele sempre produz uma solução ótima (caminho de custo mínimo), assumindo que uma solução exista
- Para cada nó ***n*** no espaço de estados vamos denotar ***h*(n)*** como sendo o custo de um **caminho ótimo** de ***n*** até **um nó final**
- Um teorema sobre a admissibilidade de A* diz que um algoritmo A* que utiliza uma função heurística,

i.e. **FA *h* tal que para todos os nós no espaço de estados**

***h(n)* <= *h*(n)* é admissível**

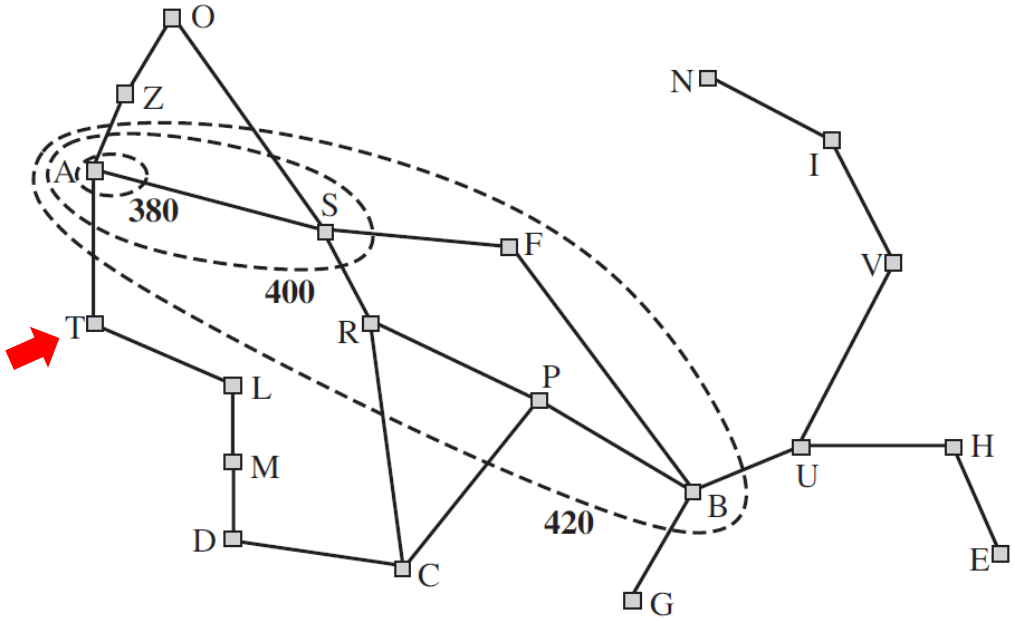
Admissibilidade

- Considerando $h^*(n)$ o custo real para atingir o objetivo, então $h(n) \leq h^*(n)$. Há um limite inferior trivial
  - $h(n) = 0$ para todo n no espaço de estados
- Embora este limite trivial garanta admissibilidade sua desvantagem é que não há nenhuma heurística e assim não há como fornecer nenhum auxílio para a busca, resultando em alta complexidade
- A* usando h=0 comporta-se de forma similar à busca em largura

$h(n) = 0$

Contornos

- A* expande nós na forma de contornos. Se C^* é o custo da solução ótima:
  - Todos os nós $f(n) < C^*$ são expandidos
  - Nenhum nó $f(n) > C^*$ é expandido. Por exemplo, Timisoara não é expandido, mesmo sendo vizinho a Arad



Admissibilidade

- Portanto é interessante utilizar $h>0$ e o mais próximo possível de h^* ($h \leq h^*$) para garantir eficiência
- Se múltiplas heurísticas estão disponíveis busque a melhor
  - $h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\}$
- De maneira ideal, se h^* é conhecida, podemos utilizar h^* diretamente
  - A* utilizando h^* encontra uma solução ótima diretamente, sem precisar realizar *backtracking*

Sobre A* e Heurísticas Admissíveis e

- Este resultado tem grande valor prático
- Mesmo que não conheçamos o exato valor de h*, nós só precisamos encontrar um limite inferior para h* e utilizá-la como FA h em A*
- Isto é suficiente para garantir que A* irá encontrar uma solução ótima

Algoritmo A*

A **fila F** é uma fila ordenada pela função **f(n) = g(n) + h(n)**

1) Insere na fila F o nó **u** e marque-o como alcançado
2) Enquanto fila F não está vazia faça
   v ← elemento da frente da fila
   (retire v (elemento com menor h(n) da fila)
   se v não é o estado final
      para todo w que partir de **v** que ainda não foi alcançado
         - marque **w** como alcançado
         - insira **w** na fila **F** (ordenado por f(n))

Propriedades A*

- A utilização de heurística para guiar o algoritmo reduz a busca apenas a uma região do espaço do problema
- **Apesar da redução no esforço da busca**, a ordem de **complexidade é ainda exponencial** na profundidade de busca
  - Isso é válido para tempo e memória uma vez que o algoritmo mantém todos os nós gerados
- Em situações práticas o espaço de memória é mais crítico e A* pode utilizar toda a memória disponível em questão de minutos

Propriedades de A*

- Completa? Sim

(a menos que existam infinitos nós com $f \leq f(G)$)

- Tempo? Exponencial

- Espaço? Mantém todos os nós em memória

- Ótima? Sim

ALGUNS PROBLEMAS CLÁSSICOS DE BUSCA

- Encontrar um caminho para um objetivo
- Missionários e canibais
- N-rainhas
- Jogos
  - Xadrez
  - Gamão
- Torres de Hanói
- Simplesmente encontrar um objetivo
- Problema do tabuleiro de xadrez danificado

HEURÍSTICA – EXEMPLOS GERAIS

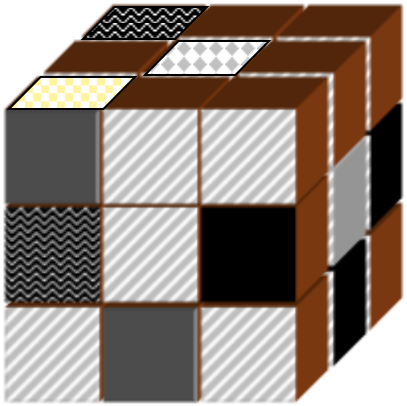
Exemplos de heurísticas:

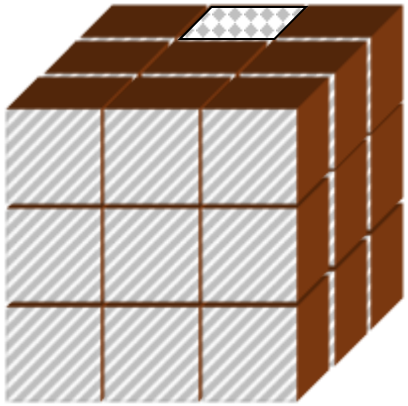
- Para planejar um caminho dentro de uma cidade, por exemplo:
  - não vire sucessivamente a esquerda (ou direita) em uma rua, pois fazendo isso há uma tendência a voltar ao mesmo lugar
  - virar quando se chega aos limites da cidade
- Para reparar máquinas, retire primeiro as peças externas mais pequenas, etc...

OBSERVAR QUE ALGUNS ESTADOS SÃO MELHORES QUE OUTROS...

7	8	4
3	5	1
6	2	

1	2	3
4	5	6
7		8





FUNÇÕES DE AVALIAÇÃO

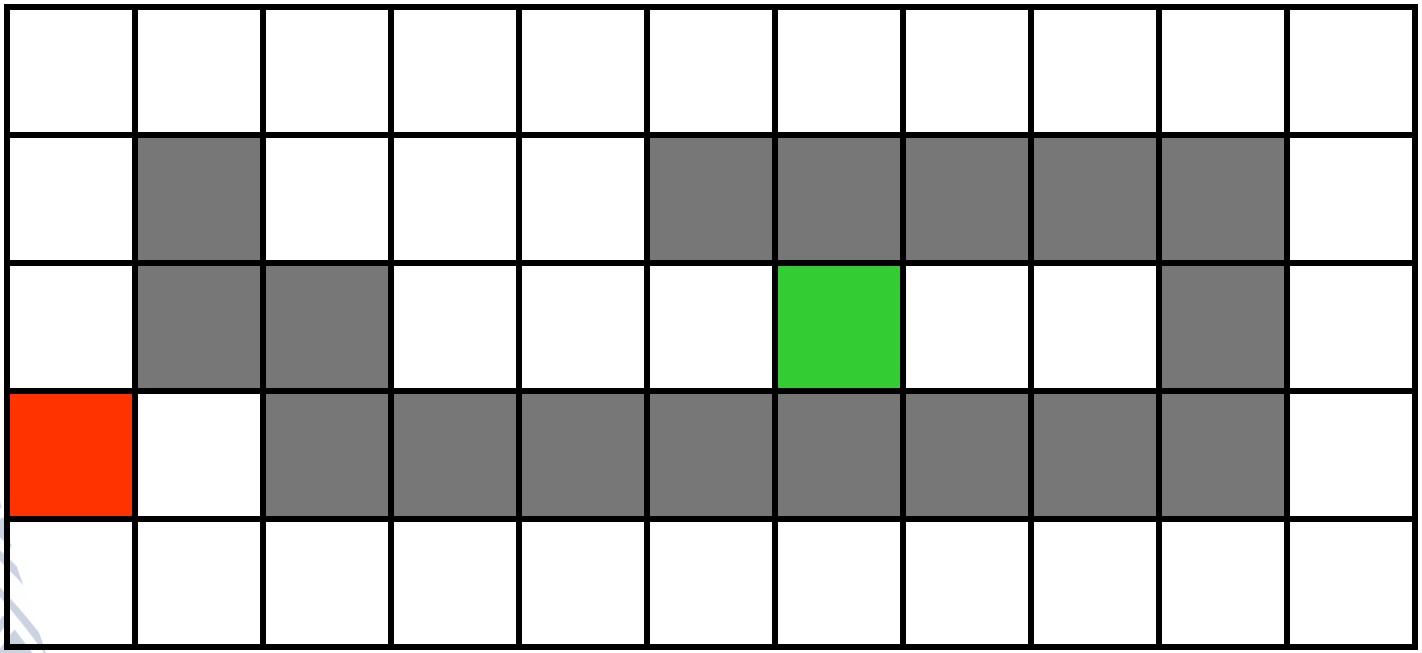
- Uma **estratégia** popular para **construir funções de avaliação** é ponderar a soma de vários fatores através de sua influência no valor da posição.
- Exemplos de funções de avaliação:
  - Por exemplo, uma função de avaliação para o xadrez poderia ter a seguinte forma:

    $c_1 * material + c_2 * mobilidade + c_3 * segurança\text{-}rei + c_4 * controle\text{-}centro + ...$

  - Para planejar um caminho dentro de uma cidade, pegue uma *linha reta*, *i.e.*, prefira o estado sucessor que está mais perto do estado meta em uma linha reta.

  - Para reparar máquinas, considere o número de peças removidas da máquina mais o número de peças defeituosas ainda dentro da máquina.

HEURISTICA PARA NAVEGAÇÃO DE ROBÔ



HEURISTICA PARA NAVEGAÇÃO DE ROBÔ

- H1:Traduzir usando a distância de Manhattan até a meta.

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

HEURISTICA PARA NAVEGAÇÃO DE ROBÔ

- H2: Também poderia ser utilizada a distância Euclidiana.

6,3	5,4	4,5	3,6	2,8	2,2	2,0	2,2	2,8	3,6	4,5
6,1		4,1	3,2	2,2						4,1
6,0			3,0	2,0	1,0	0,0	1,0	2,0		4,0
6,1	5,1									4,1
6,3	5,4	4,5	3,6	2,8	2,2	2,0	2,2	2,8	3,6	4,5

HEURISTICA Quebra-cabeça 8 peças

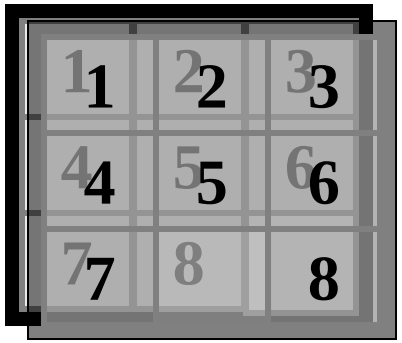
Heurística 1: **Número de peças fora do lugar** (ignorando espaço vazio)

Estado Atual

1	2	3
4	5	6
7		8

Meta

1	2	3
4	5	6
7	8	



N	N	N
N	N	N
N	Y	

Neste caso, somente "**8**" está fora do lugar, assim a função de avaliação seria igual a 1. Ou seja a **FA *traduz* a heurística *para um valor numérico***

OUTRA HEURISTICA Quebra-cabeça 8 peças

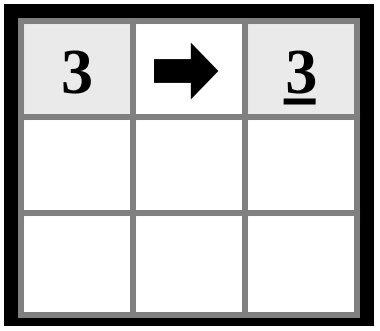
Heurística 2: **Distância de Manhattan** (sem considerar espaço vazio)

Estado Atual

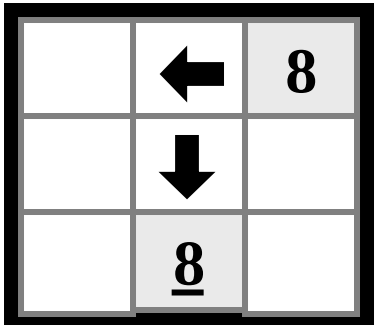
3	2	8
4	5	6
7	1	

Meta

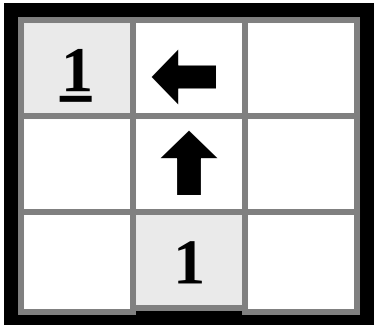
1	2	3
4	5	6
7	8	



2 espaços



3 espaços



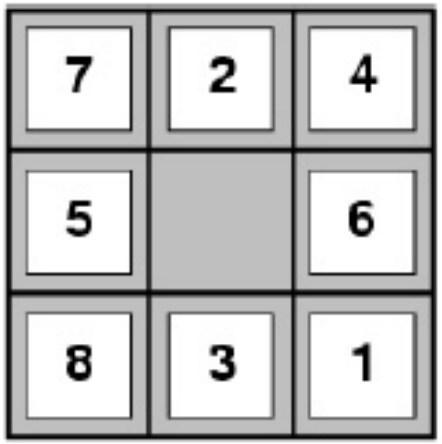
3 espaços

Total 8

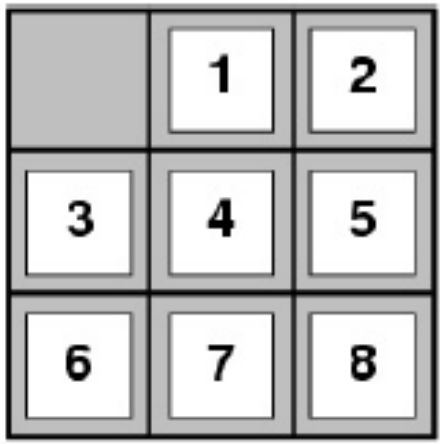
Neste caso somente "**3**", "**8**" e "**1**" estão fora do lugar por 2, 3, e 3 lugares respectivamente, assim a função de avaliação **h(n) = 8**.

Por exemplo, num plano que contem os pontos P_1 e P_2, respectivamente com as coordenadas (x_1,y_1) e (x_2,y_2), a **distância de Manhattan** é definida pela soma das diferenças absolutas entre as suas coordenadas - r: $|x_1-x_2| + |y_1-y_2|$

Heurísticas Admissíveis Fonte: Russel e Norvig



estado inicial



estado meta

- Síntese: para o quebra-cabeça de 8 peças:
  - h1(n) = número de peças fora do lugar
  - h2(n) = total da distância de Manhattan (i.e., número de quadrados da posição desejada de cada peça)
    - h1(S) = 8
    - h2(S) = 3+1+2+2+2+3+3+2 = 18

Dominância

- Se $h_2(n) >= h_1(n)$ 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	BAI	A*(h_1)	A*(h_2)
12	3.644.035	227	73
24	muitos nós	39.135	1.641

- BAI = Busca por Aprofundamento Iterativo

Problemas Relaxados

- Um problema com poucas restrições nas ações é chamado de **problema relaxado**.
- O custo de uma solução ótima para um problema relaxado é uma heurística admissível para o problema original
- Se as regras do quebra-cabeça de 8 peças são relaxadas tal que uma peça possa ser movida para qualquer lugar, então $h_1(n)$ fornece a solução mais curta
- Se as regras são relaxadas tal que uma peça possa ser movida para qualquer quadrado adjacente, então $h_2(n)$ fornece a solução mais curta

ALGORITMOS DE BUSCA LOCAL

- Em alguns **problemas de otimização**, o caminho até o objetivo é irrelevante; o **estado meta é a solução**
  - Espaço de Estados = conjunto completo de configurações
- Encontrar a configuração que satisfaz as restrições. Ex: problema *n* rainhas
  - Em tais casos, pode-se usar algoritmos de busca local
    - Manter um estado atual, tente melhorá-lo
- **Hill-Climbing** (ou otimização discreta) "parece" uma busca em profundidade usando FA.

Exemplo: *n* Rainhas

HILL-CLIMBING – Busca de Subida de Enconsta

- Os métodos de busca local, que é o caso do "***hill climbing***", são usados para resolver problemas de otimização discreta iniciados a partir de uma **configuração inicial** e **movendo-se** repetidamente **para uma melhor configuração vizinha**.
  - Uma **trajetória é gerada no espaço de busca**, que mapeia um ponto inicial **para um local ótimo**, onde a busca local é impedida de prosseguir.
  - "É como": "escalar o monte Everest em um nevoeiro denso com amnésia" ou "usar óculos que limitam sua visão a 3 metros"

HILL-CLIMBING

função Hill-Climbing(problema) **retorna** um estado que é um máximo local

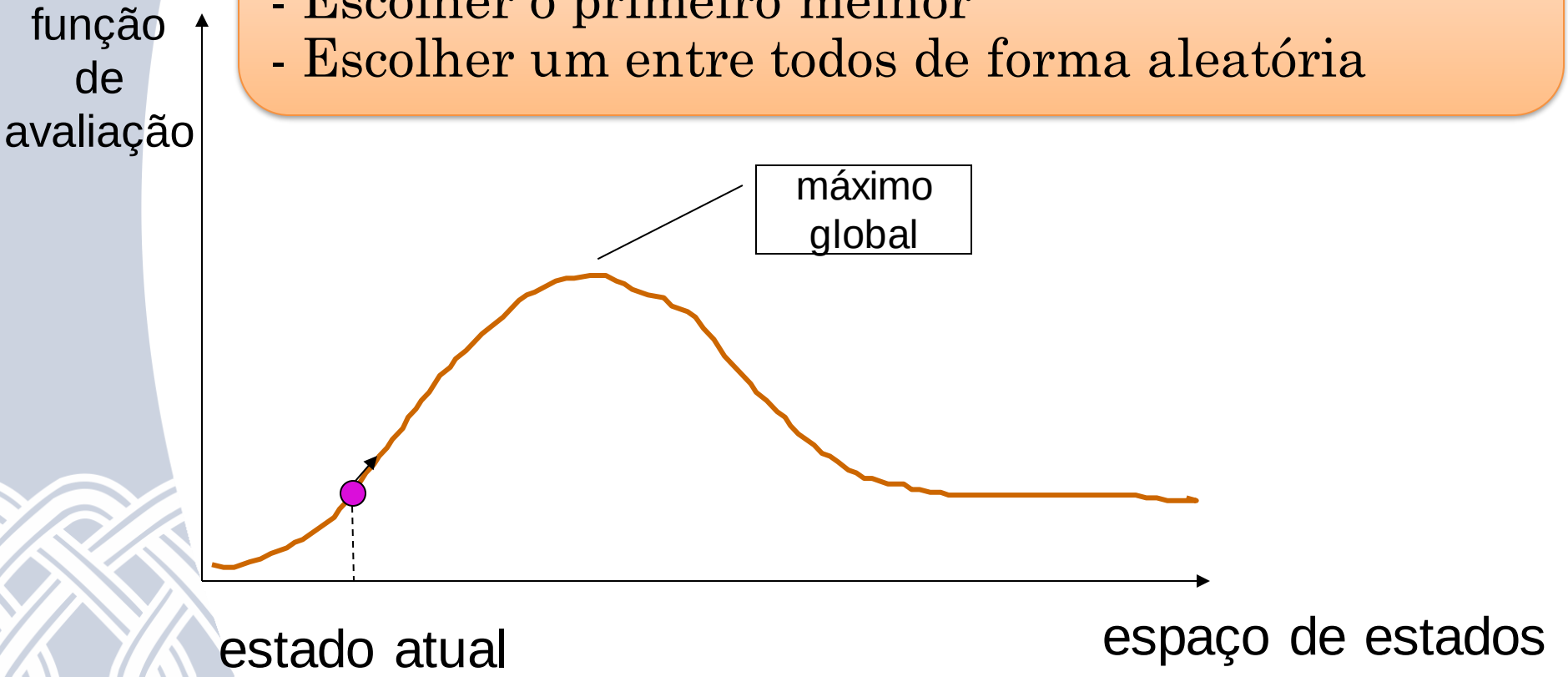
entradas: problema, um problema
variáveis locais: corrente, um no
vizinho, um nó

corrente <- CRIAR-NO(ESTADO-INICIAL[problema])
repita
vizinho <- um sucessor de corrente com valor mais alto
se VALOR[vizinho] ≤ VALOR[corrente)
então retornar ESTADO [corrente]
corrente <- vizinho

- Em cada passo do algoritmo, o nó corrente é substituído pelo **melhor vizinho** (vizinho com VALOR mais alto).
- Se for usada uma estimativa de custo de heurística h(n), o algoritmo deve ser alterado para encontrar o vizinho com o h(n) com valor mais baixo.

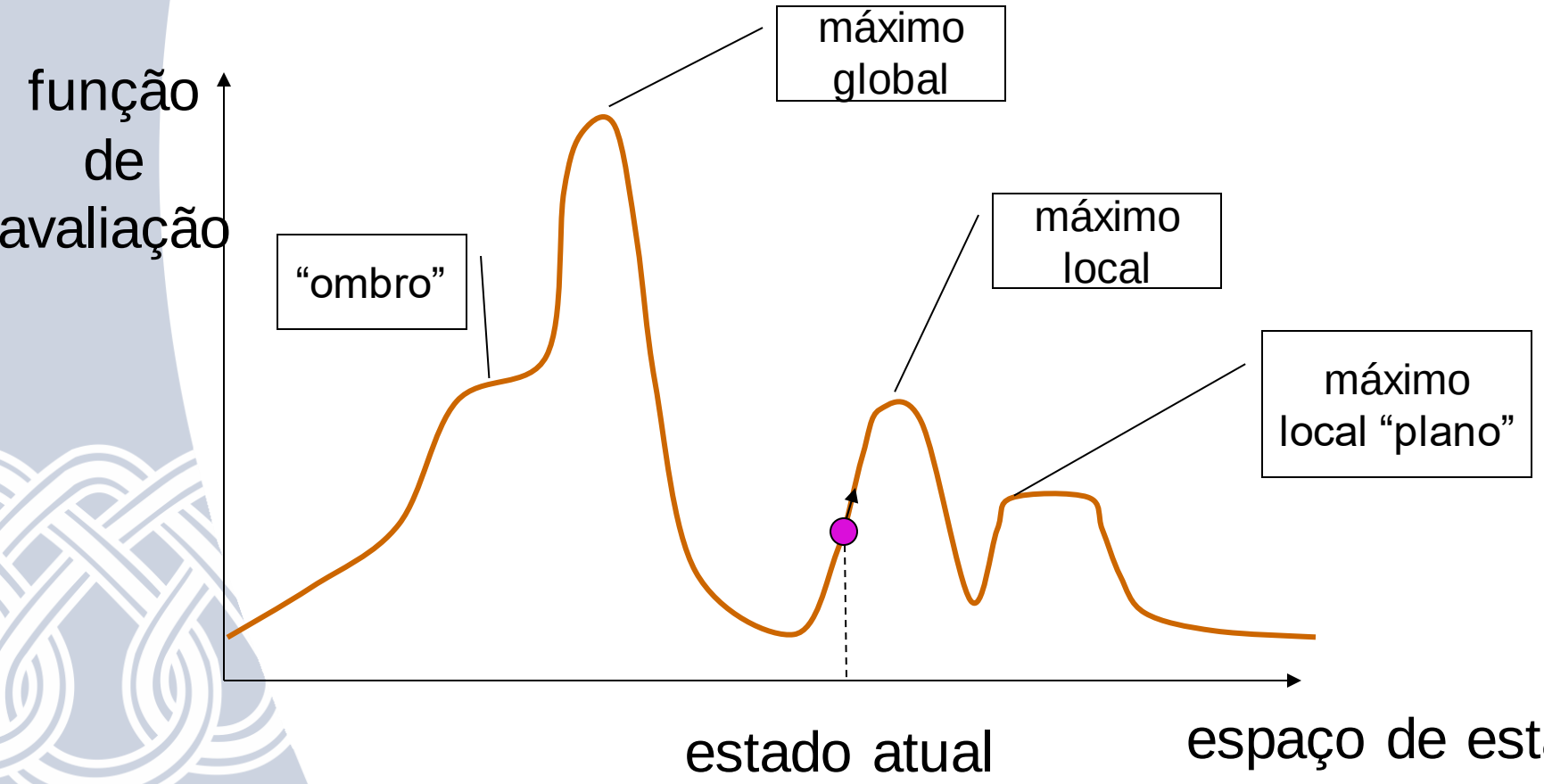
HILL-CLIMBING

Se há mais de um vizinho com a melhor qualidade:
- Escolher o primeiro melhor
- Escolher um entre todos de forma aleatória



HILL-CLIMBING

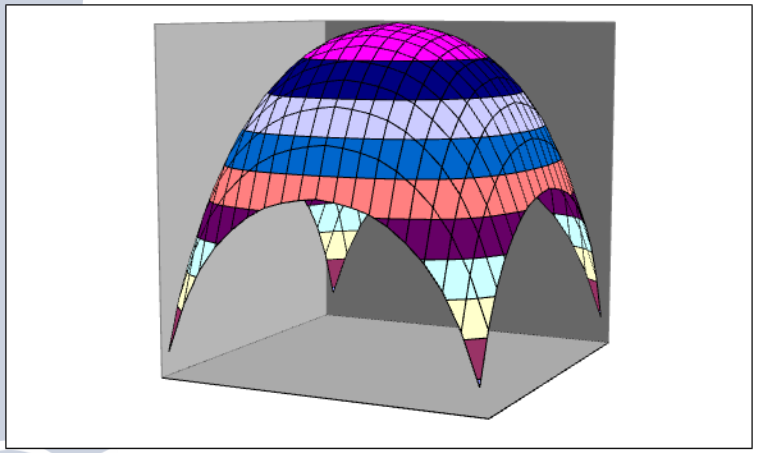
Problema: dependendo do estado inicial, pode ficar em máximos locais

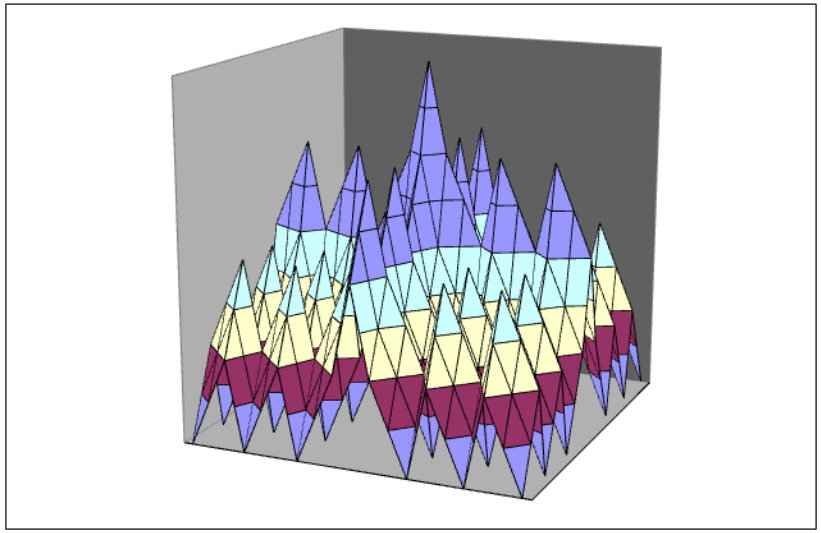


Considerações na BUSCA LOCAL

- Manter ***k*** estados como alternativa a um estado
- Comece gerando ***k*** estados aleatoriamente
  - A cada iteração, todos os sucessores de todos os ***k*** estados devem ser gerados
  - Se qualquer um deles for o objetivo, pare; caso contrário selecione ***k*** melhores sucessores da lista completa e repita

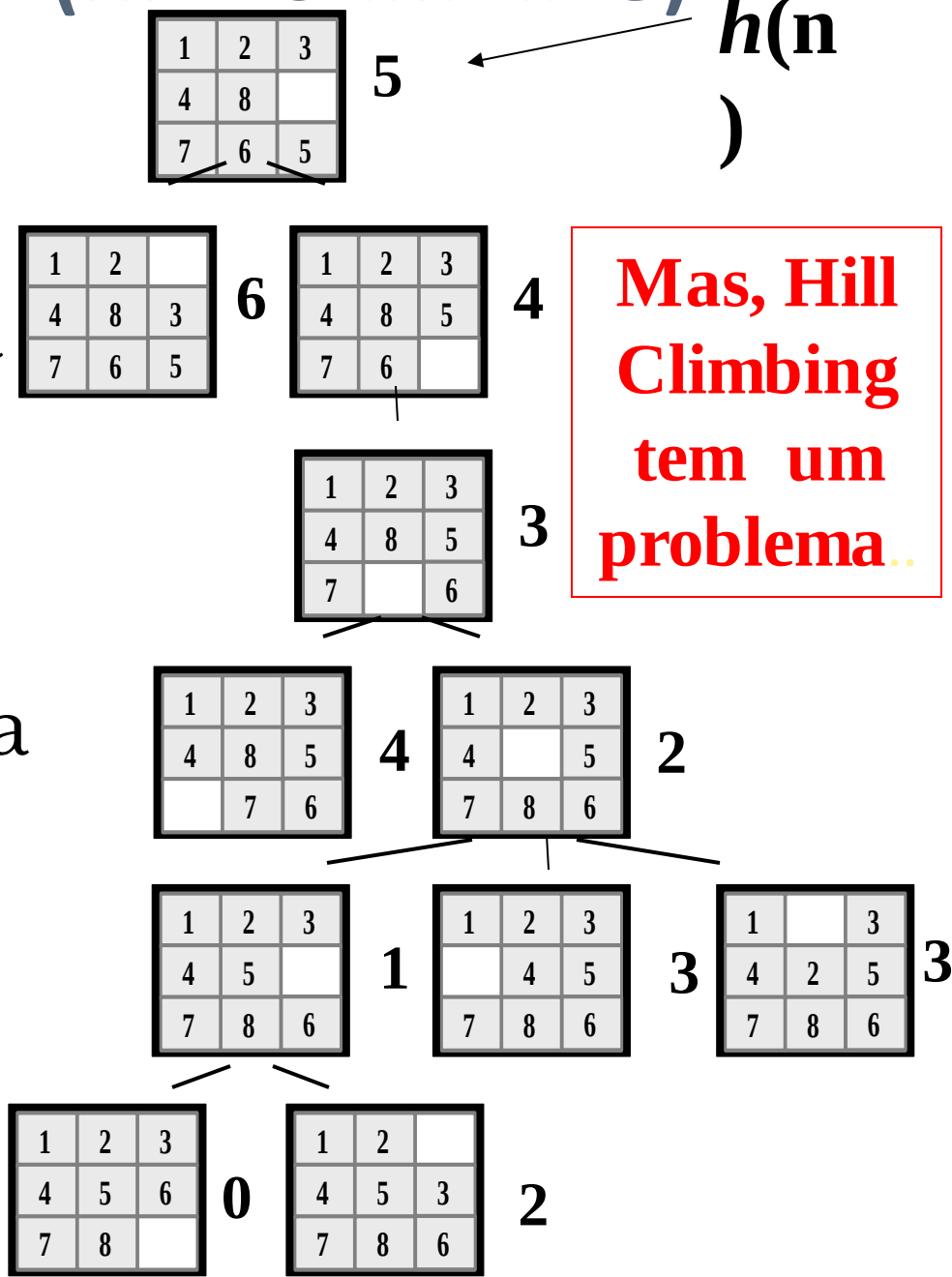
HILL-CLIMBING





8-PUZZLE (HILL-CLIMBING)

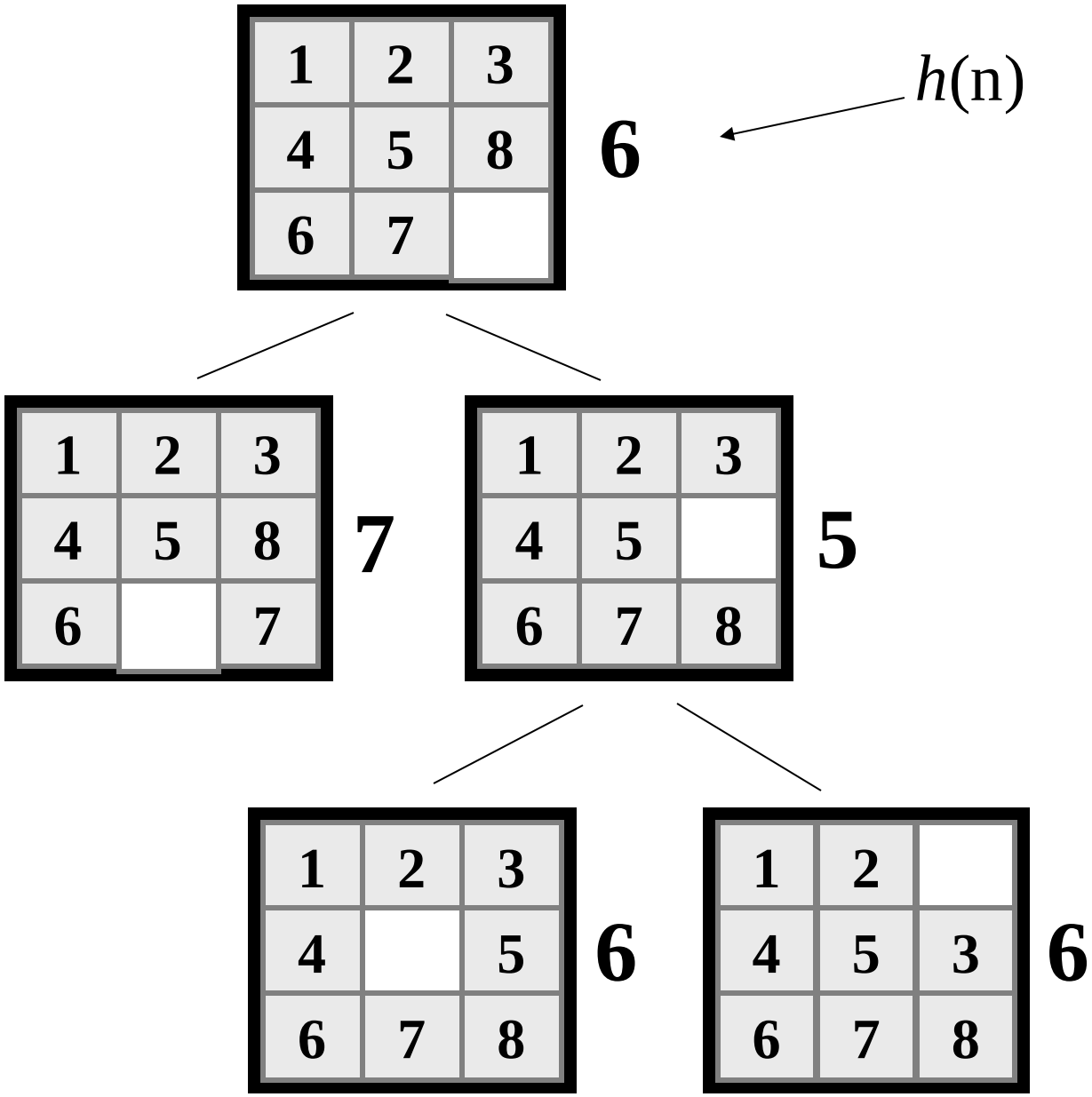
Usando como **função de avaliação h(n)** a distância de Manhattan, em alguns casos, como o mostrado a seguir, hill-climbing pode chegar rapidamente na solução.



Mas, Hill Climbing tem um problema..

Neste exemplo, **hill climbing** não encontra a solução!

Fica em um mínimo local…por ser um algoritmo **guloso** (*greedy*)



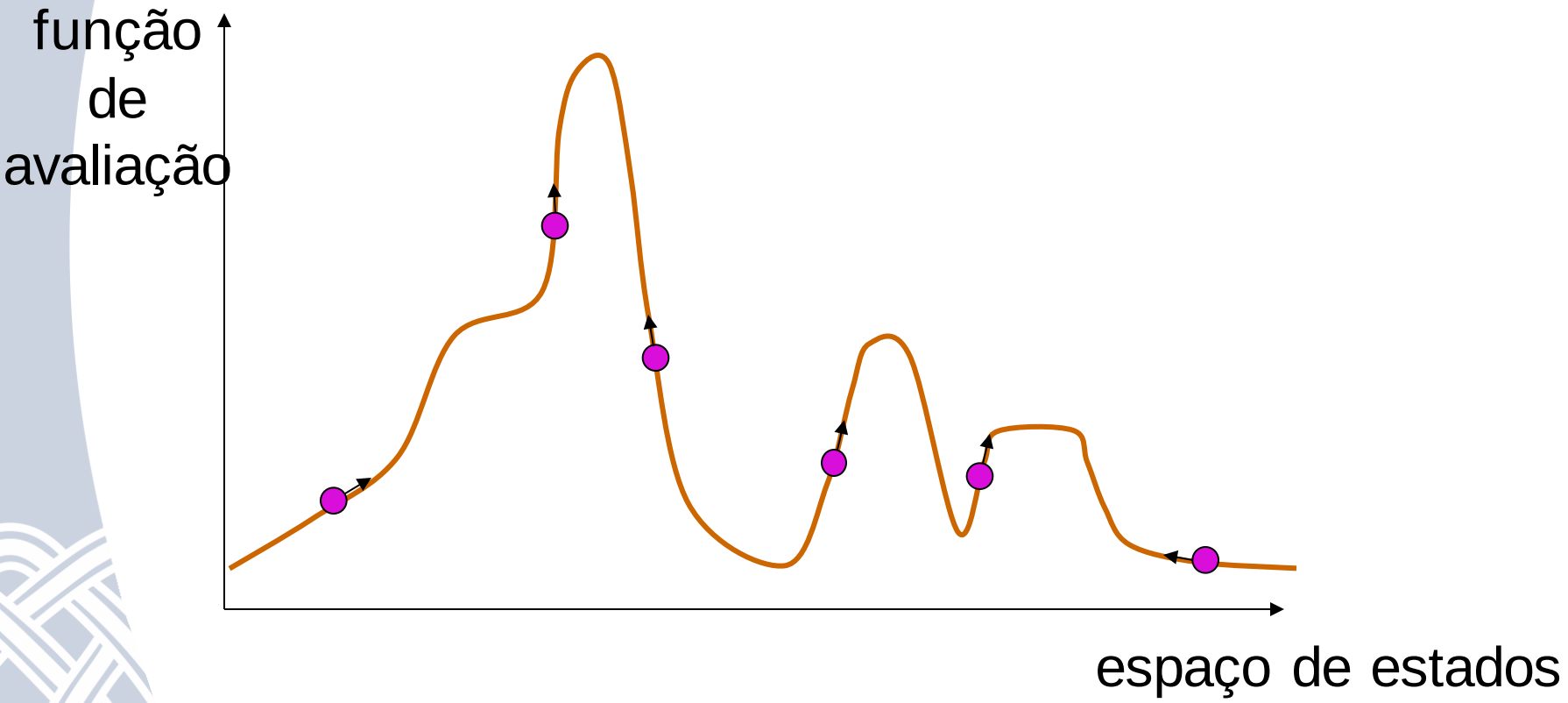
HILL-CLIMBING: PROBLEMAS

- **Máximo local**: uma vez atingido, o algoritmo termina mesmo que a solução esteja longe de ser satisfatória
- **Platôs (regiões planas)**: regiões onde a função de avaliação é essencialmente plana; a busca torna-se como uma caminhada aleatória
- **Cumes ou "ombros":** regiões que são alcançadas facilmente mas até o topo a função de avaliação cresce de forma amena; a busca pode tornar-se demorada

HILL-CLIMBING: VARIAÇÕES

- Hill-Climbing **Estocástico**
  - Nem sempre escolhe o melhor vizinho
- Hill-Climbing **Primeira Escolha**
  - Escolha o primeiro bom vizinho que encontrar
    - Útil se é grande o número de sucessores de um nó
- Hill-Climbing **Reinício Aleatório**
  - Conduz uma série de buscas hill-climbing a partir de estados iniciais gerados aleatoriamente, executando cada busca até terminar ou até que não exista progresso significativo
  - O melhor resultado de todas as buscas é armazenado

HILL-CLIMBING REINÍCIO ALEATÓRIO



Simulated Annealing
Busca de Recozimento Simulado

- Ideia: escapar dos máximos locais permitindo alguns movimentos “ruins,” mas gradualmente diminuindo sua frequência.
- Propriedades:
  - Pode-se provar: Se T diminui lentamente, então a busca de recozimento simulado (*simulated annealing*) achará um ótimo global com probabilidade de aproximação 1.
  - Muito usada em layout de VLSI, escalonamento de empresas aéreas, etc.

Busca em Feixe local

- Mantém a trilha de k estados ao invés de apenas um.
- Começa com k estados gerados aleatoriamente.
- A cada iteração, todos os sucessores de todos os k estados são gerados.
- Se um deles for o estado meta, para; caso contrário seleciona os k melhores sucessores a partir da lista completa e repete.

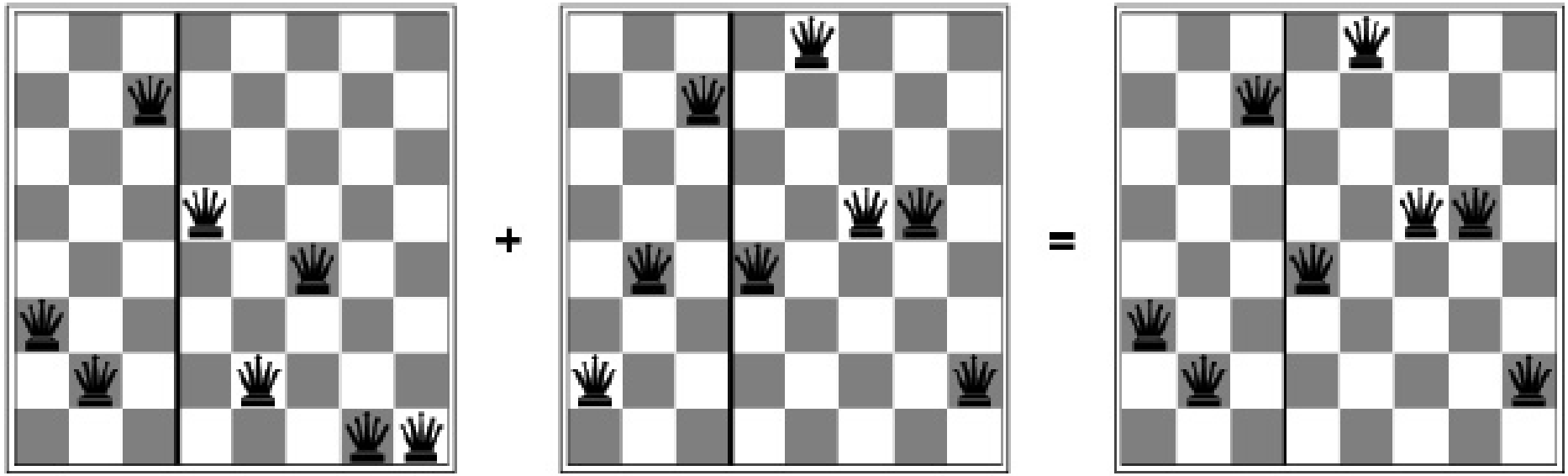
Algoritmos Genéticos

- Um estado sucessor é gerado combinando dois estados pais.
- Começa com k estados gerados aleatoriamente (população).
- Um estado é representado como uma cadeia de caracteres de um alfabeto finito (frequentemente uma cadeia de 0’s e 1’s).
- Função de avaliação (função de fitness). Valores mais altos para estados melhores.
- Produz a próxima geração de estados através de seleção, cruzamento e mutação.

Algoritmos genéticos Fonte Russel e Norvig

- Função de *fitness*: número de pares de rainhas que não se atacam (mín = 0, máx = 8 x 7/2 = 28).
  - 24/(24+23+20+11) = 31%.
  - 23/(24+23+20+11) = 29% etc.

Algoritmos Genéticos



Argumentos a favor do uso de heurística

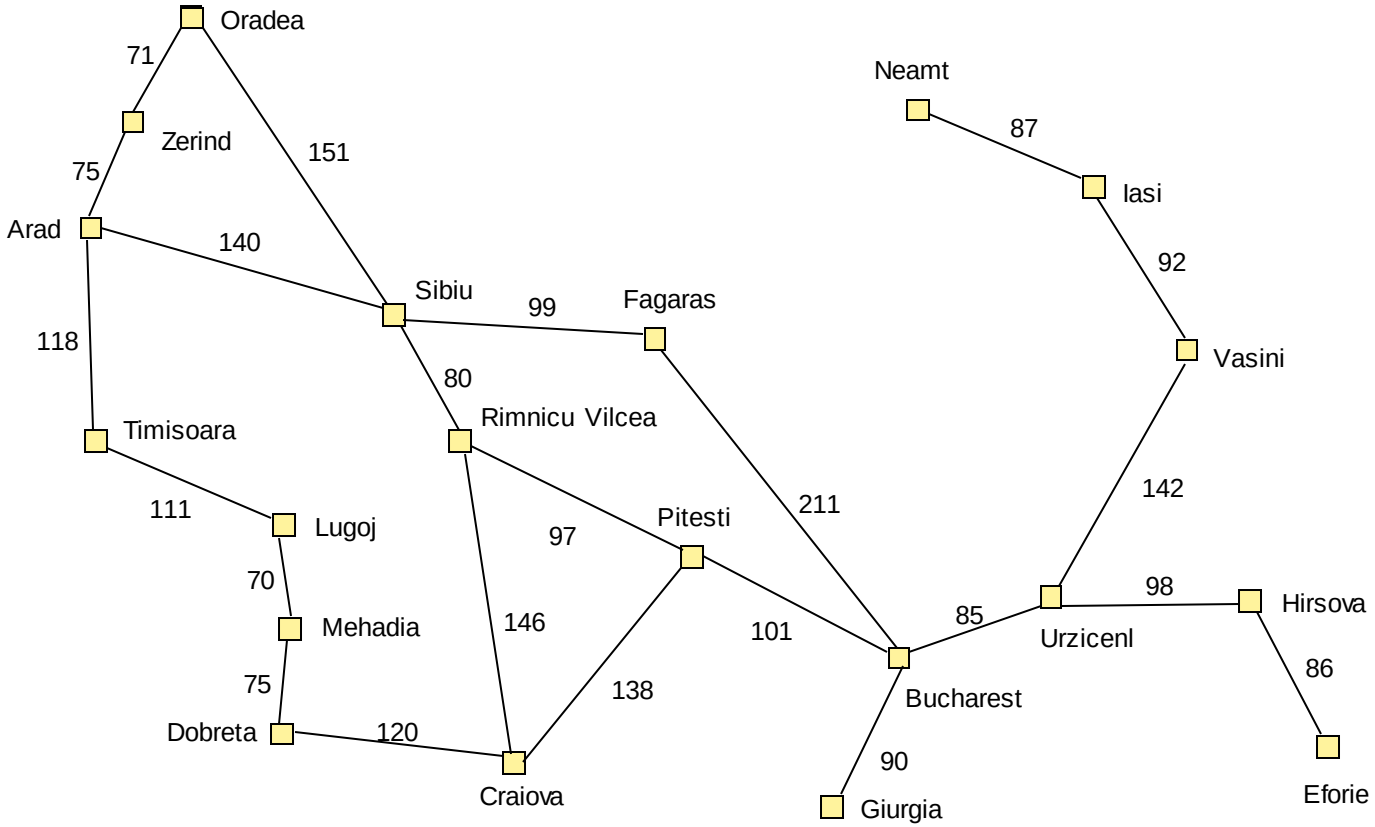
- **Raramente precisa-se da solução ótima**; uma boa aproximação normalmente serve muito bem. De fato, existem evidências de que as pessoas, quando resolvem problemas, procuram qualquer solução que satisfaça algum conjunto de requisitos, e assim que encontram, dão-se por satisfeitas.
- Ainda que as aproximações produzidas por heurísticas possam não ser muito boas no pior caso, **os piores casos raramente acontecem na vida real.**
- A tentativa de entender por que uma heurística funciona, ou por que não funciona, leva a um entendimento mais aprofundado do problema.

Exercício

- Especifique uma base de dados global, regras e uma condição de terminação para um sistema de produção para resolver o seguinte problema dos jarros de água:
  - Dado um jarro de 5 litros cheio de água e um jarro de 2 litros vazio, como se pode obter precisamente 1 litro no jarro de 2 litros? A água pode ser desperdiçada ou transferida de um jarro para outro; entretanto, só os 5 litros iniciais estão disponíveis.

Exercício

- Encontrar o caminho para ir da cidade de Arad a cidade de Bucharest utilizando os algoritmos de busca informada hill-climbing, best-first e A*

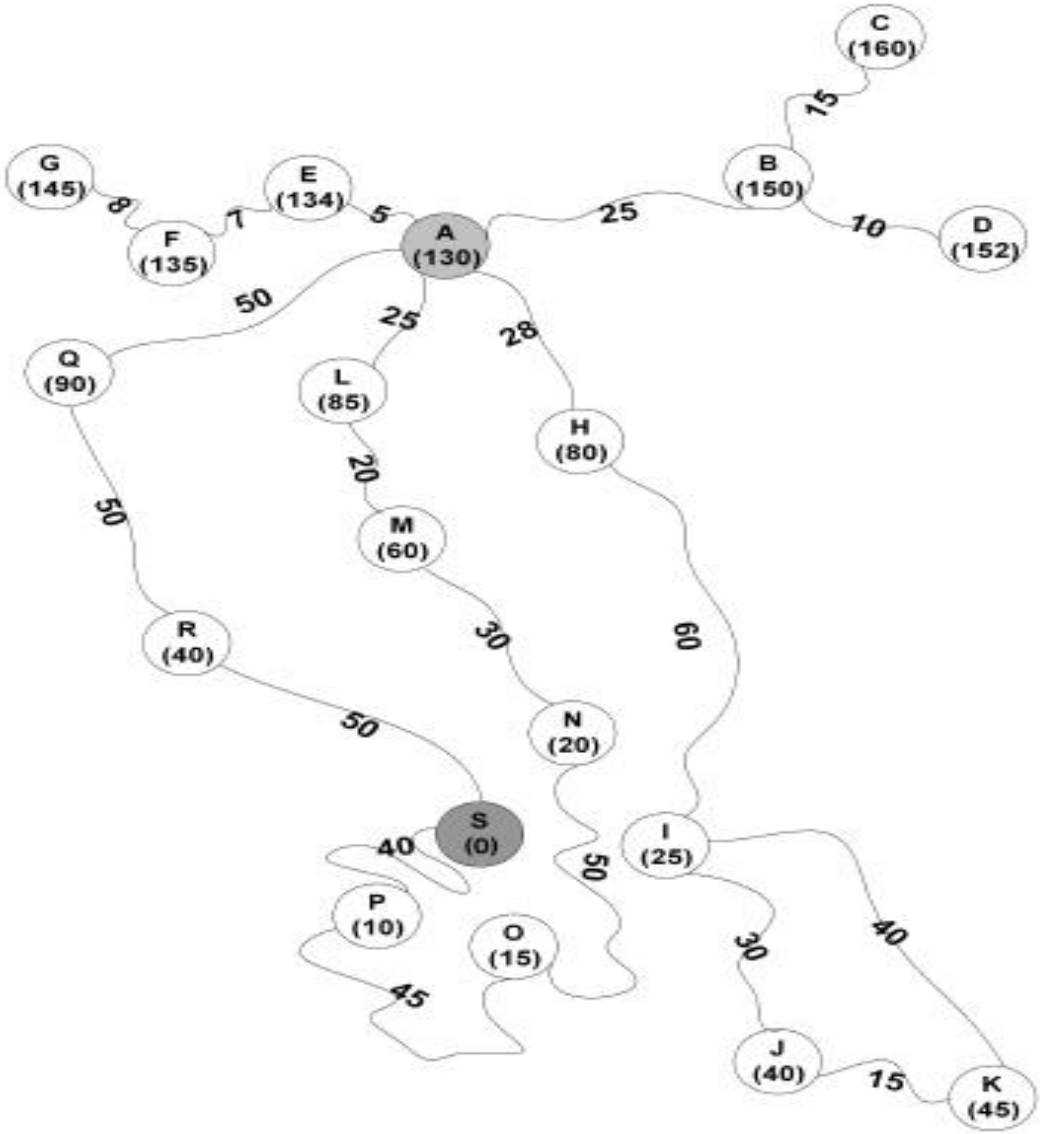


Funções de Avaliação
Para Bucharest (Romenia)

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

Exercício

Use as estratégias de busca informada e encontre os caminhos

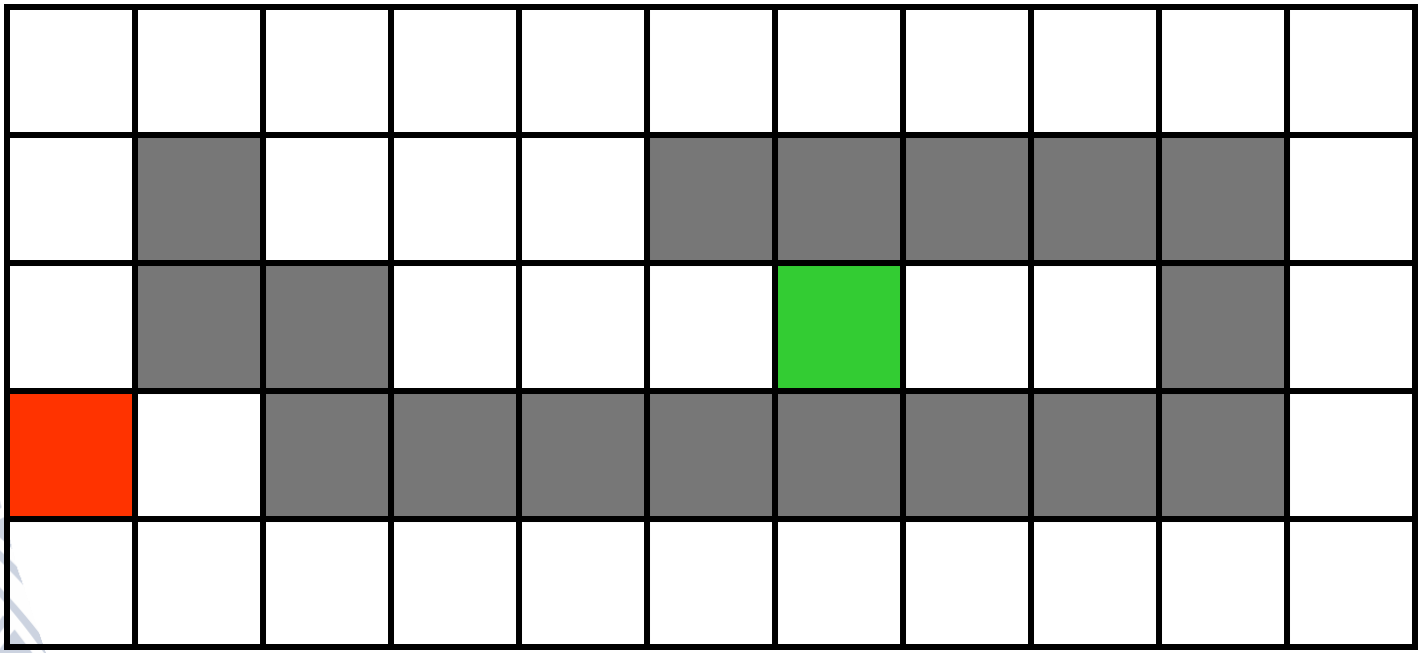


Modele um Problema como uma Árvore de Busca

- Use aplicativos para facilitar o entendimento dos algoritmos de busca
- Além dos apresentados na aula prática utilizando a Biblioteca **aima-java**, que implementa diversos algoritmos utilizados na disciplina, é parte do material de apoio livro **Artificial Intelligence - A Modern Approach**, de Stuart Russel e Peter Norvig
- Pode ser usado http://www.aispace.org/search/search.jnlp

Mais Exemplos de Funções de Avaliação h(n)

HEURISTICA PARA NAVEGAÇÃO DE ROBÔ



HEURISTICA PARA NAVEGAÇÃO DE ROBÔ

- H1:Traduzir usando a distância de Manhattan até a meta.

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

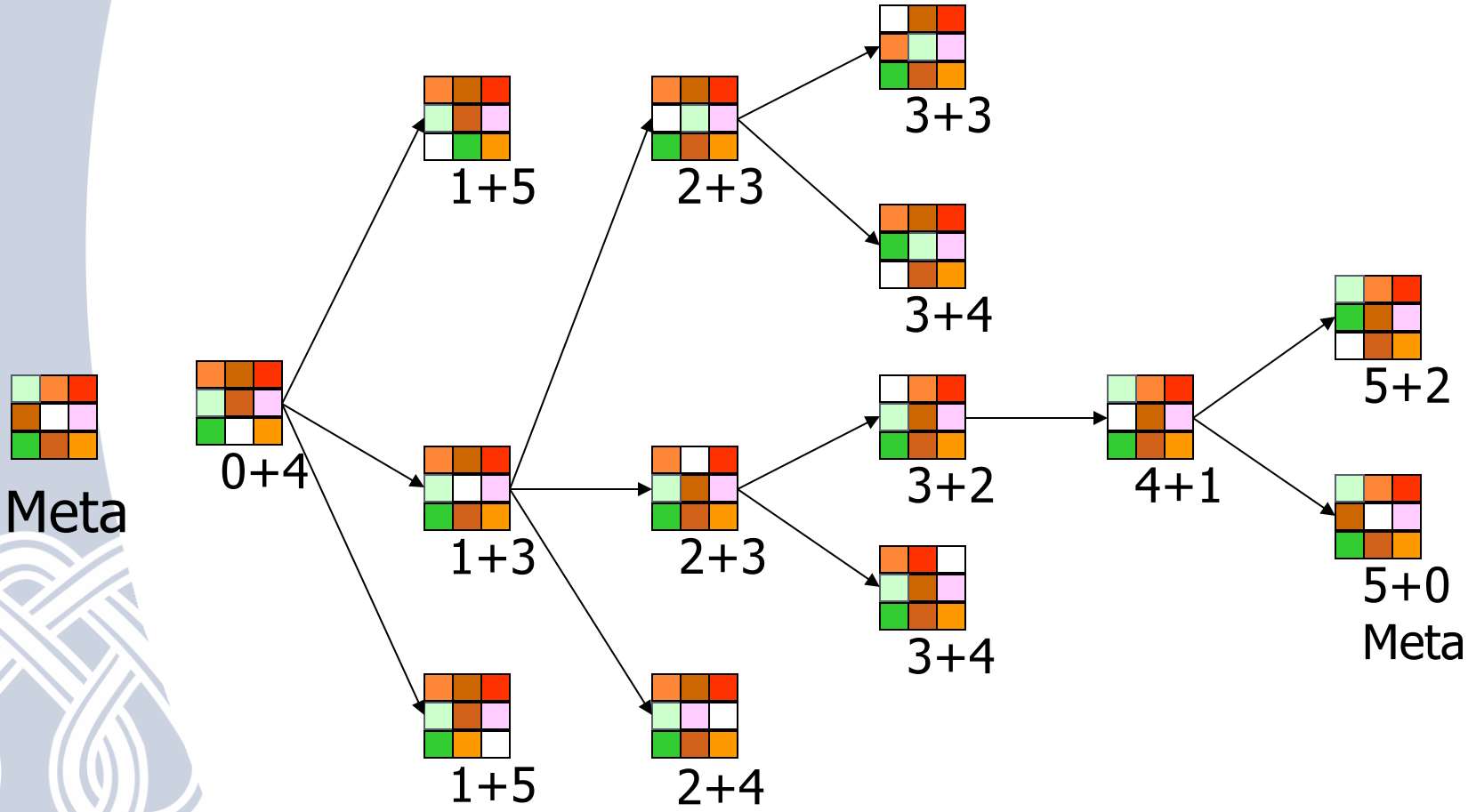
HEURISTICA PARA NAVEGAÇÃO DE ROBÔ

- H2: Também poderia ser utilizada a distância Euclidiana.

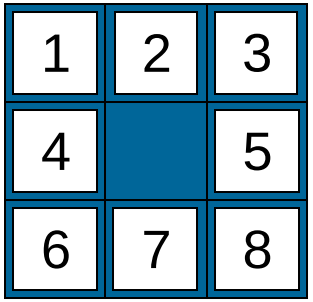
6,3	5,4	4,5	3,6	2,8	2,2	2,0	2,2	2,8	3,6	4,5
6,1		4,1	3,2	2,2						4,1
6,0			3,0	2,0	1,0	0,0	1,0	2,0		4,0
6,1	5,1									4,1
6,3	5,4	4,5	3,6	2,8	2,2	2,0	2,2	2,8	3,6	4,5

8-PUZZLE (A*)

f(N) = g(N) + h(N) , em que
g(N) = número de posições caminhadas para chegar na posição atual (N).
h(N) = número de peças fora de lugar
Custo de um movimento = 1

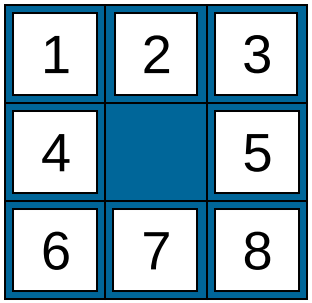


ALGUMAS FUNÇÕES HEURÍSTICAS



- Vamos olhar para algumas heurísticas para o quebra-cabeça-8
  - Soluções em média a distância 22
  - Fator de ramificação em média de 3
  - 9!/2 estados possíveis em um grafo = 181.440
- Possíveis heurísticas
  - h_1 = número de peças em posições erradas
  - h_2 = soma das distâncias das peças para as suas posições finais

ALGUMAS FUNÇÕES HEURÍSTICAS



- Comparação de e :
  - 1200 problemas aleatórios com soluções entre 2 e 24 passos
  - Solução com busca em profundidade limitada iterativa e A*
  - Comparação com número de nós gerados e com fator de ramificação
- Fator de ramificação: em média, para cada nó visitado, quantos nós são expandidos
  - Quanto mais próximo de 1, melhor é a busca

Navegação de Robô (A*)

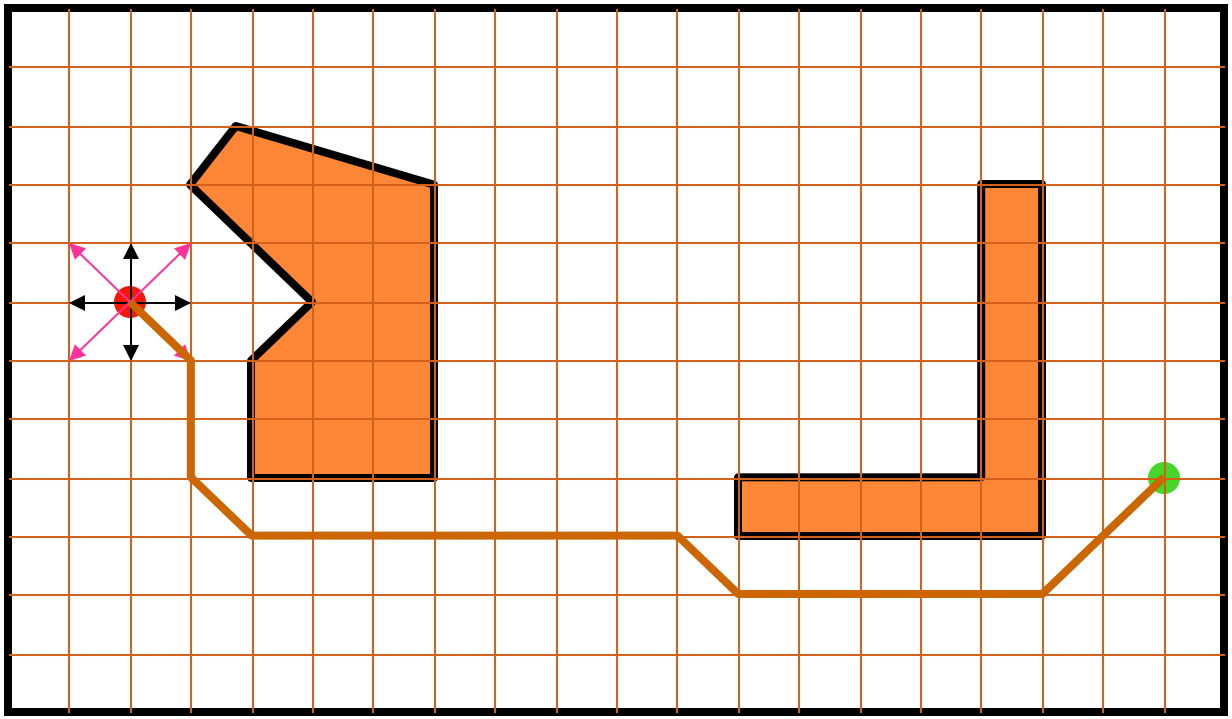
f(N) = g(N)+h(N), em que
g(N) = passos dados até estado atual (N)
h(N) = distancia de Manhattan até a meta

Custo de um passo (horizontal/vertical) = 1

8+3	7+4	6+3	5+6	4+7	3+8	2+9	3+10	4	5	6
7+2		5+6	4+7	3+8						5
6+1			3	2+9	1+10	0+11	1	2		4
7+0	6+1									5
8+1	7+2	6+3	5+4	4+5	3+6	2+7	3+8	4	5	6

Navegação de Robô (A*)

- f(N) = g(N) + h(N)
- com h(N) = distância Euclideana até a meta



Custo de um passo (horizontal/vertical) = 1

Custo de um passo diagonal = $\sqrt{2}$

Navegação de Robô (A*)

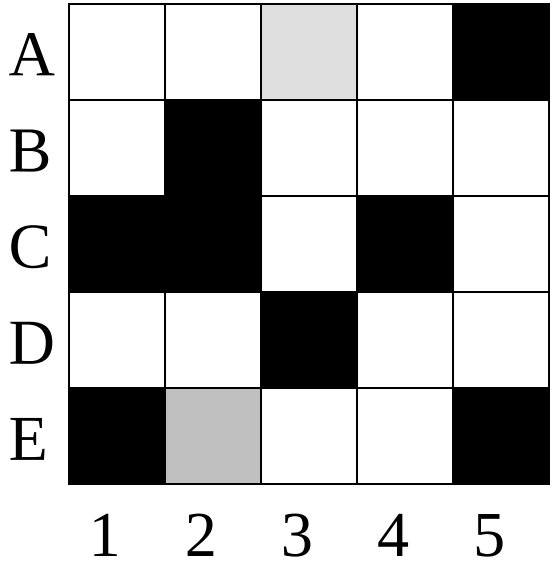
Ir de A3 pra E2, um passo por vez, evitando obstáculos (quadros pretos).

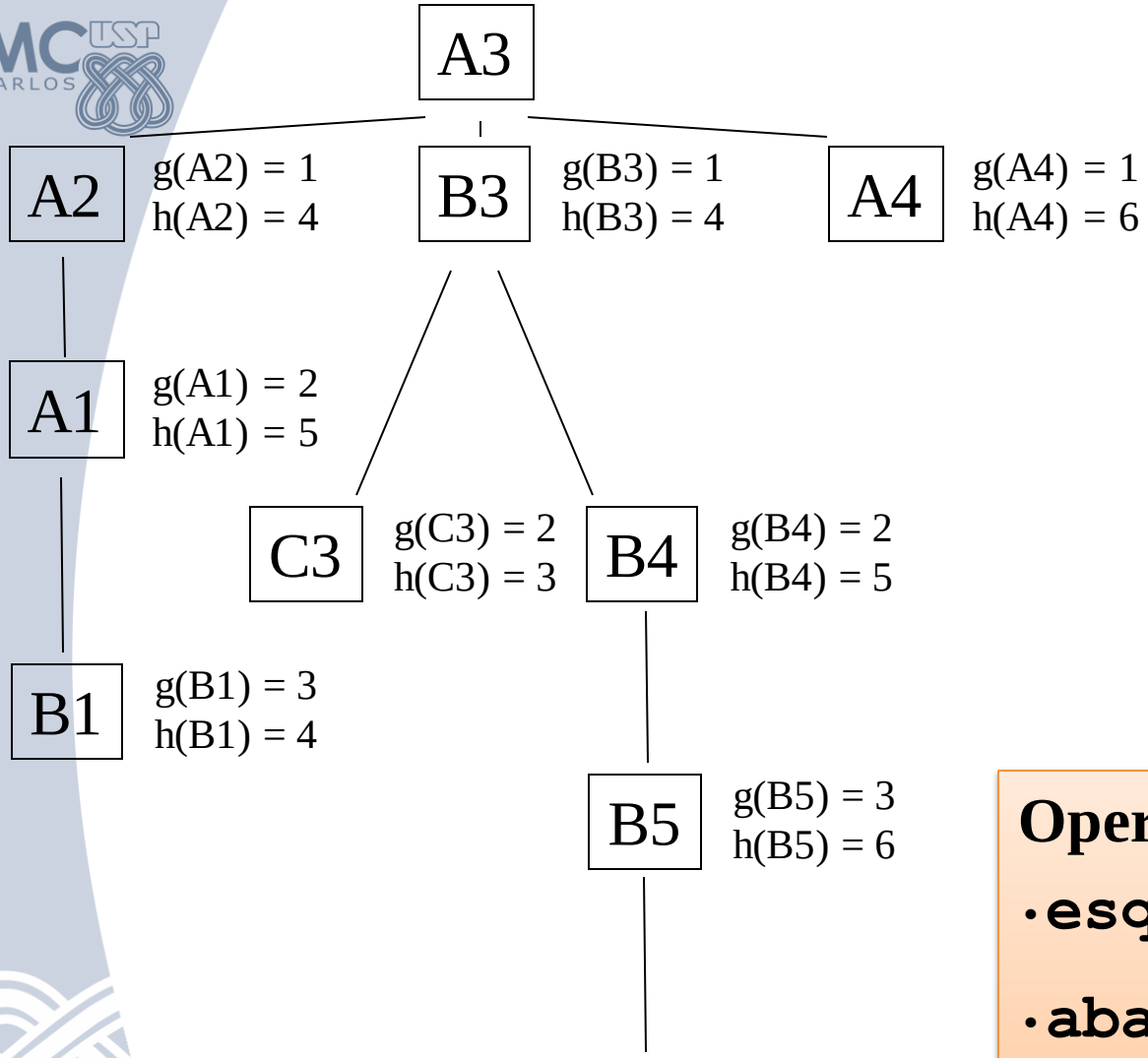
Operadores: (nessa ordem)

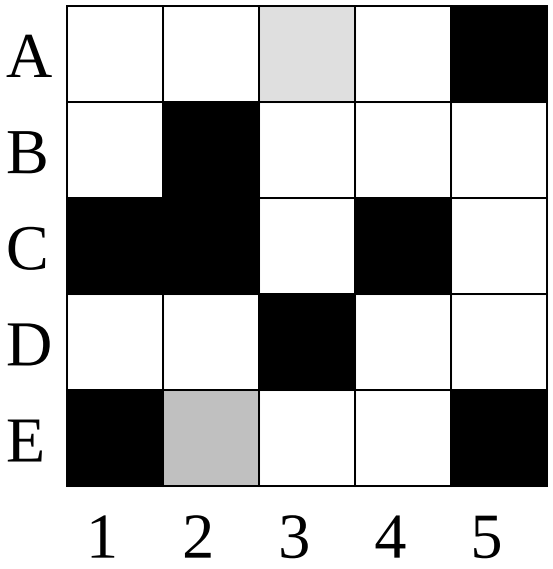
- **`esquerda`**
- **`abaixo`**
- **`direita`**
- **`acima`**

Custo unitário.

Função de avaliação h(n): distância de Manhattan







Operadores: (nessa ordem)

- **esquerda**
- **abaixo**
- **direita**
- **acima**

Referências

- RUSSEL, S.; NORVIG, P. - Artificial Intelligence: A Modern Approach. Prentice Hall; 3 edition (December 11, 2009).
- NILSSON, NILS J. - Artificial Intelligence, SAN FRANCISCO : MORGAN KAUFMANN, 1998. 513 P. IL.
- WINSTON,P.H. - Artificial Intelligence, Reading. Addison-Wesley, 1977.
- RICH, E., KNIGHT, K. - Inteligência Artificial. 2ª Ed. McGraw-Hill, Inc. 1993.
- LUGER, G. F. - Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Addison-Wesley, 4th edition, 2002.
- Rosa, J. L. G. Fundamentos da Inteligência Artificial. Editora LTC. Rio de Janeiro, 2011

- **Material preparado pelos professores:**
  - Maria Carolina Monard
  - Solange Oliveira Rezende
  - Thiago Pardo
  - Gustavo Batista
  - João Rosa
  - José Augusto Baranauskas (FFCLRP/USP)
  - + colaboradores