



Inteligência Artificial Ciência da Computação

Prof. Aline Paes / alinepaes@ic.uff.br

Estratégias de Busca Heurística
RN 3.5, 3.6



Universidade Federal Fluminense



Lembrando: Busca de custo uniforme

```
function UNIFORM-COST-SEARCH(problem) returns uma solução ou falha
  node = um nó com state = problem.initialState
  pathCost = 0
  frontier = lista de prioridades, ordenada por pathCost
  frontier.add(node)
  explored = emptySet
  loop do
    if frontier.empty() return falha
    node = frontier.pop()
    if problem.goalTest(node.state) return solution(node)
    explored.add(node.state)
    for each action in problem.actions(node.state) do
      child = child-node(problem, node, action) // atualiza pathCost a partir de node
      if child.state não está em exploredSet ou em frontier
        frontier.add(child)
      else if child.state está em frontier com pathCost mais alto
        frontier.replace(oldNode, child)
```

Busca Informada: Heurística

- Usa conhecimento específico sobre o problema para encontrar soluções de forma mais eficiente do que a busca cega.
- O conhecimento do problema é codificado em uma função **heurística**, $h(n)$
 - computada para cada estado
 - estima
 - o quão bom um nó é
 - o quão perto um nó está de chegar na solução

Busca informada: função de avaliação

- $h(n)$ é maior ou igual a zero para todos os nós
- $h(n) = 0$: objetivo
- A função heurística irá compor uma função de avaliação, $f(n)$

Busca Informada: Estratégia

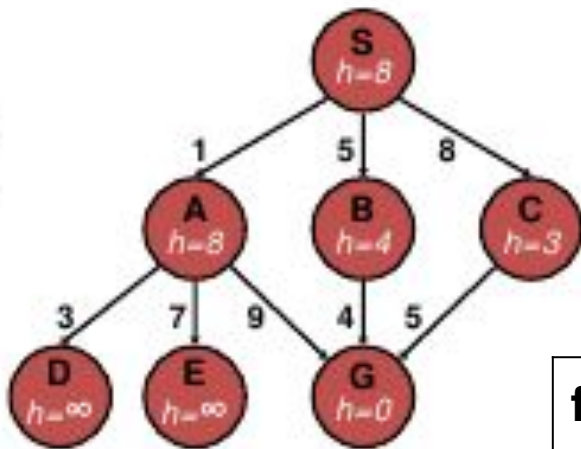
- **Estratégia geral**

- Busca pela melhor escolha
- Expandir nó mais desejável que ainda não foi expandido
- **Implementação**: Ordenar nós em *frontier* em ordem decrescente de acordo com $f(n)$
- Instancia a busca de custo uniforme
 - ao invés do custo, usamos $f(n)$ para ordenar os nós

Busca gulosa

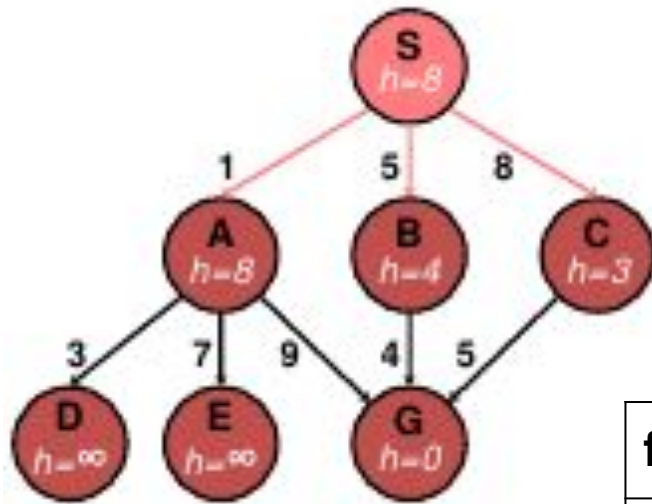
- Busca que usa somente a heurística como função de avaliação
 - $f(n) = h(n)$
- Expande o nó que parece estar mais próximo da meta, seguindo $h(n)$

Busca Gulosa: exemplo



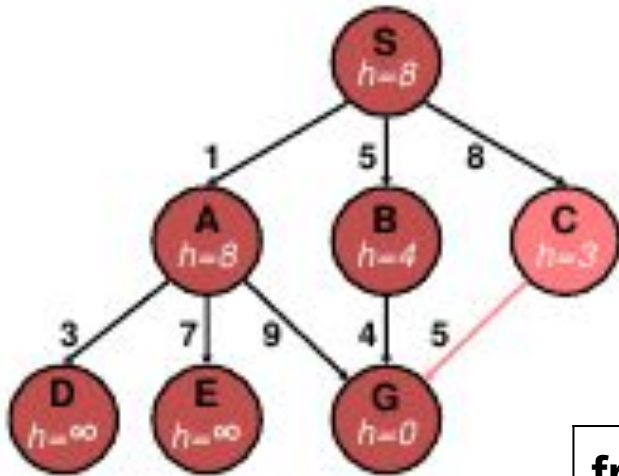
frontier	exploredSet
[S:8]	

Busca Gulosa: exemplo



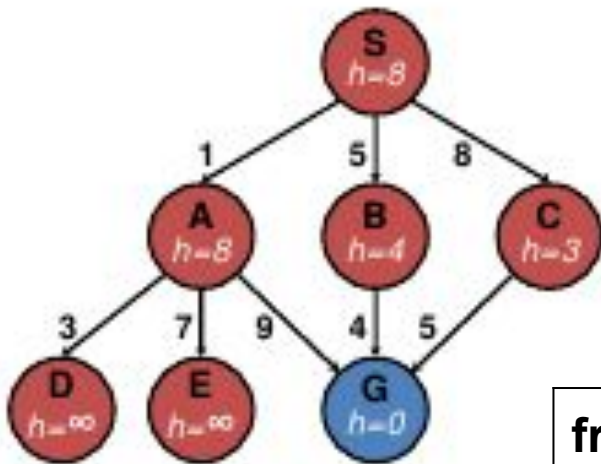
frontier	exploredSet
[S:8]	
[C:3, B:4, A:8]	S

Busca Gulosa: exemplo



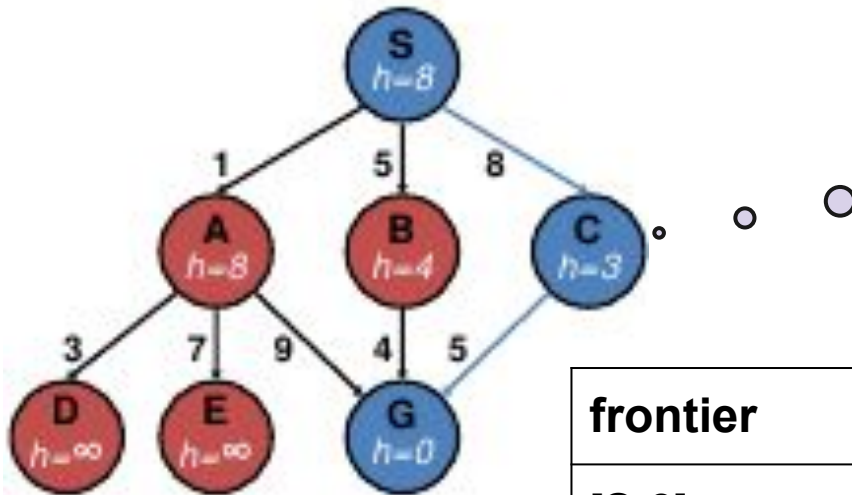
frontier	exploredSet
[S:8]	
[C:3, B:4, A:8]	S
[G:0, B:4, A:8]	S, C

Busca Gulosa: exemplo



frontier	exploredSet
[S:8]	
[C:3, B:4, A:8]	S
[G:0, B:4, A:8]	S, C
[B:4, A:8]	S, C, G

Busca Gulosa: exemplo



Custo real da
solução: 13

frontier	expanded
[S:8]	
[C:3, B:4, A:8]	S
[G:0, B:4, A:8]	S, C
[B:4, A:8]	S, C, G

Busca Gulosa

```
function GREEDY-SEARCH(problem, heuristic_function) returns uma solução ou falha
  node = make_node( problem.initialState)
  frontier = lista de prioridades, ordenada por heuristic_function(node)
  frontier.add(node)
  explored = emptySet
  loop do
    if frontier.empty() return falha
    node = frontier.pop()
    if problem.goalTest(node.state) return solution(node)
    explored.add(node.state)
    for each action in problem.actions(node.state) do
      child = child-node(problem, node, action)
      if child.state não está em exploredSet ou em frontier
        frontier.add(child)
      else if existe oldNode.state em frontier com heuristic_function(oldNode) >
        heuristic_function(child)
        frontier.replace(oldNode, child)
```

Avaliação da busca gulosa

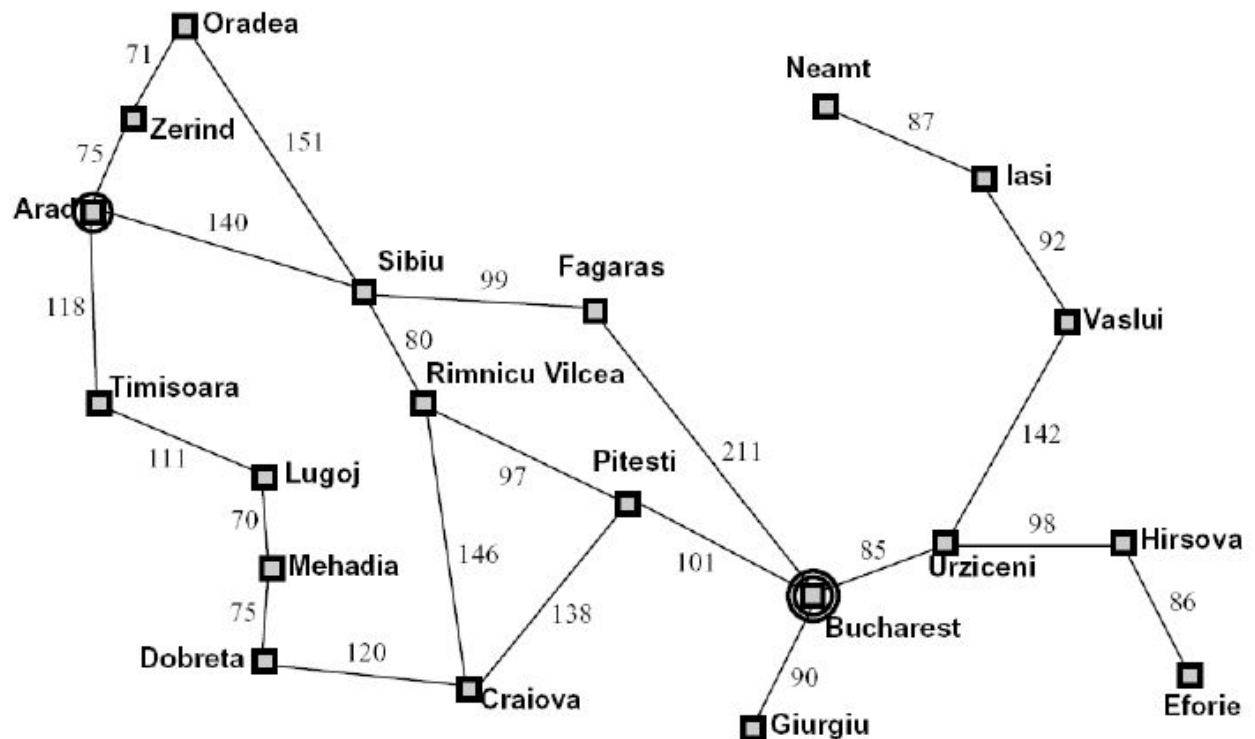
- **Completa?**
 - Sim – se espaço de estados for finito
- **Tempo?**
 - $O(b^m)$ no pior caso (uma boa heurística pode levar a uma redução substancial)
- **Espaço?**
 - $O(b^m)$ – mantém todos os nós na memória
- **Ótima?** Não

Exercício - Busca gulosa, de Arad a Bucharest

- $h(n)$ = linha reta da cidade n até a meta

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



F = [Arad:366]

S = {Arad}

F = [Sibiu: 253, Timi: 329, Zerind: 374]

S = {Arad, Sibiu}

F = [Fagaras: 178, RV: 193, Timi: 329, Zerind: 374, Oradea: 380]

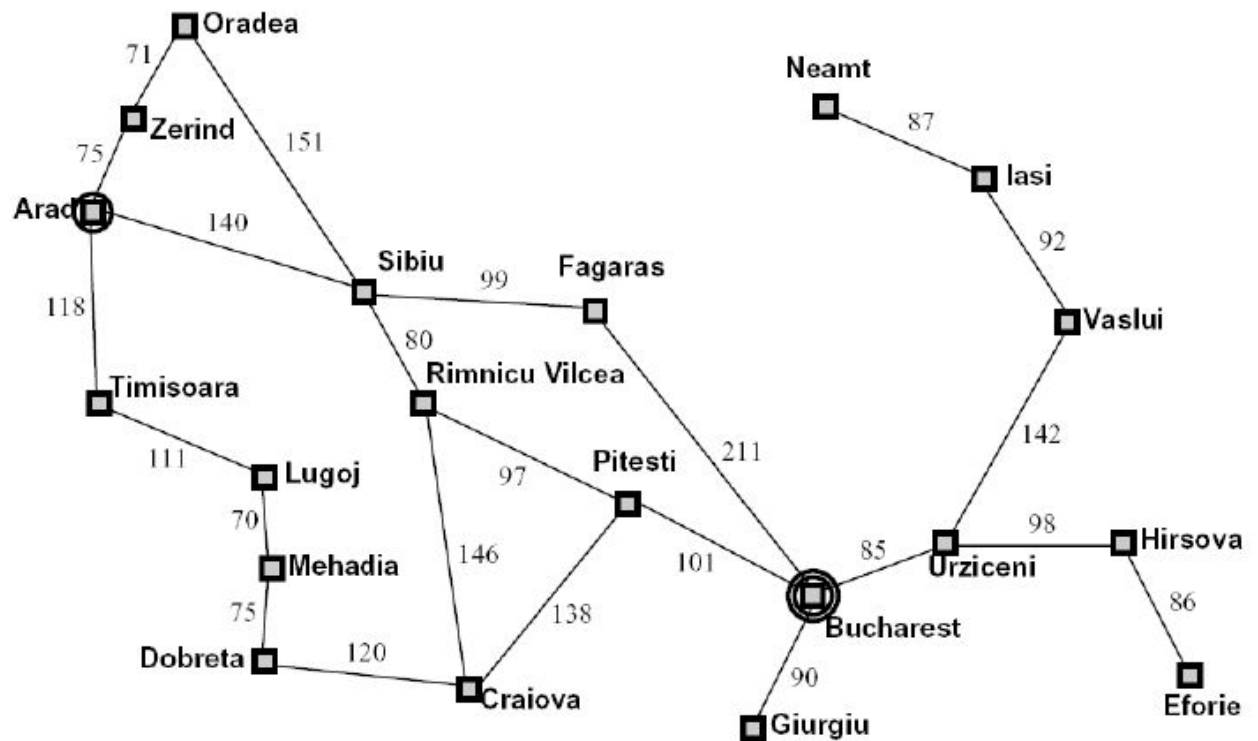
S = {Arad, Sibiu, Fagaras}

F = [Bucharest: 0, RV: 193, Timi: 329, Zerind: 374, Oradea: 380]

- $h(n)$ = linha reta da cidade n até a meta

Straight-line distance
to Bucharest

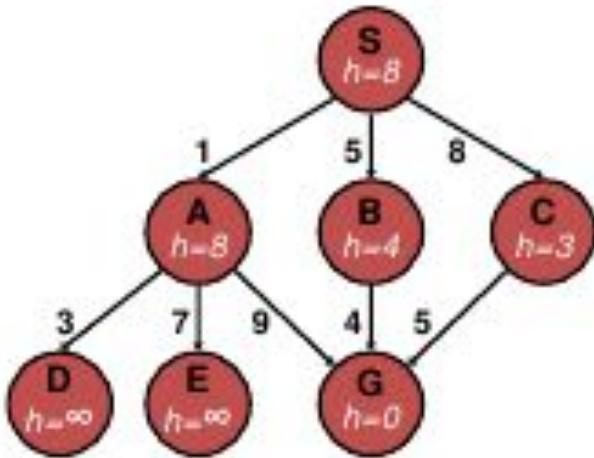
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



Busca A*

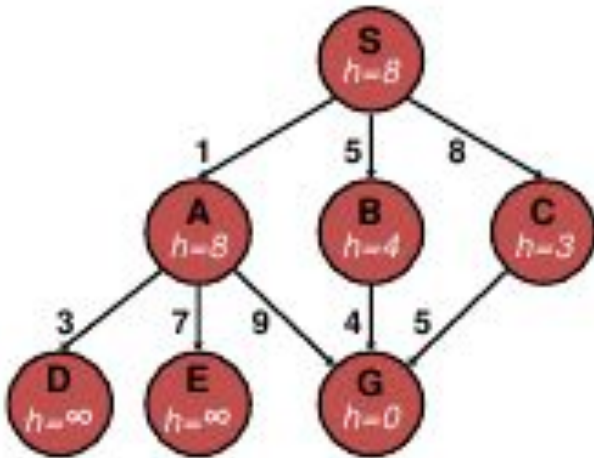
- Usa função de avaliação $f(n) = h(n) + g(n)$
 - $h(n)$: heurística, custo **estimado** do nó n até a meta
 - $g(n)$: custo do estado inicial até o nó n
 - $f(n)$: custo estimado da solução que passa por n

Busca A* - Exemplo



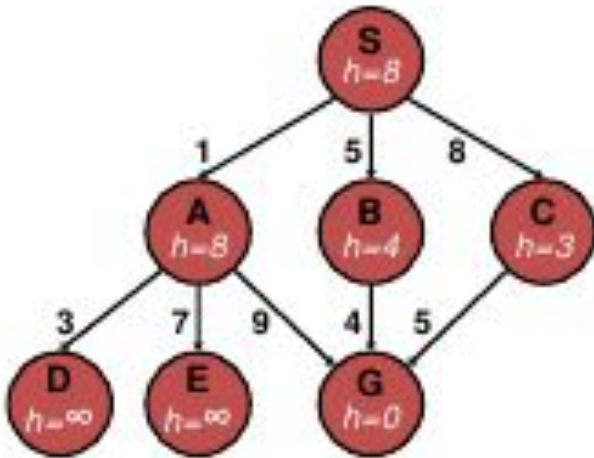
n	g(n)	h(n)	f(n)	frontier
S	0	8	8	[S:8]
A		8		
B		4		
C		3		
D		inf		
E		inf		
G		0		

Busca A* - Exemplo



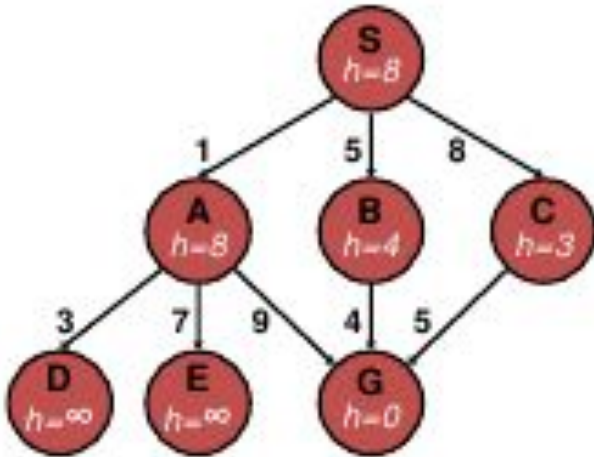
n	g(n)	h(n)	f(n)	frontier
S	0	8	8	[S:8]
A	1	8	9	[A:9]
B		4		
C		3		
D		inf		
E		inf		
G		0		

Busca A* - Exemplo



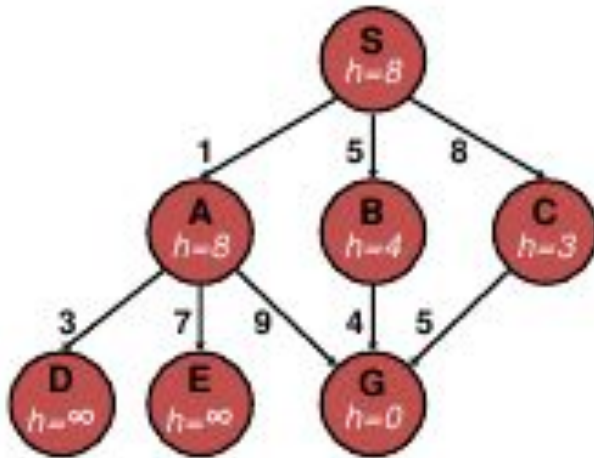
n	g(n)	h(n)	f(n)	frontier
S	0	8	8	[S:8]
A	1	8	9	[A:9]
B	5	4	9	[A:9, B:9]
C		3		
D		inf		
E		inf		
G		0		

Busca A* - Exemplo



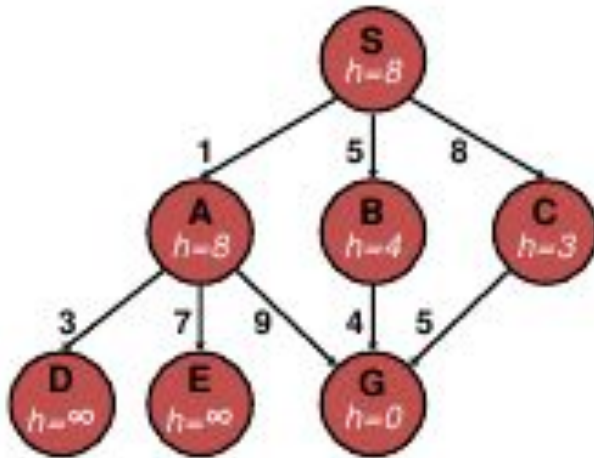
n	g(n)	h(n)	f(n)	frontier
S	0	8	8	[S:8]
A	1	8	9	[A:9]
B	5	4	9	[A:9, B:9]
C	8	3	11	[A:9, B:9, C:11]
D		inf		
E		inf		
G		0		

Busca A* - Exemplo



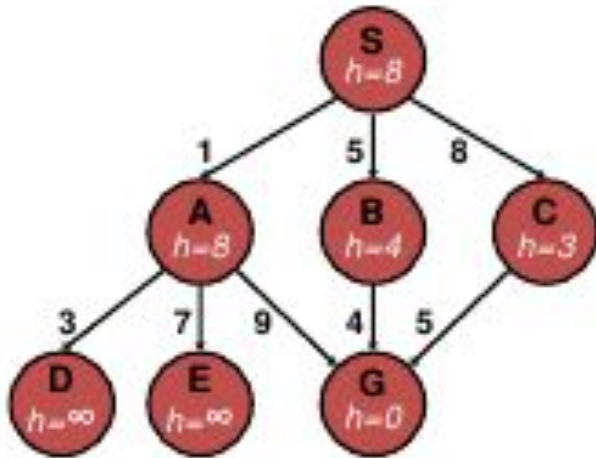
n	g(n)	h(n)	f(n)	frontier
S	0	8	8	S:8
A	1	8	9	[A:9]
B	5	4	9	[A:9, B:9]
C	8	3	11	[A:9, B:9, C:11]
D	1+3=4	inf	inf	[B:9, C:11, D:inf]
E		inf		
G		0		

Busca A* - Exemplo



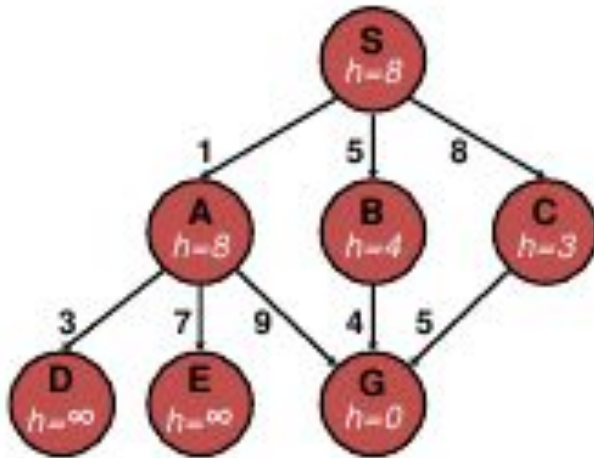
n	g(n)	h(n)	f(n)	frontier
S	0	8	8	S:8
A	1	8	9	[A:9, B:9, C:11]
B	5	4	9	[B:9, C:11, D:inf, E:inf]
C	8	3	11	
D	1+3=4	inf	inf	
E	1+7=8	inf	inf	
G		0		

Busca A* - Exemplo



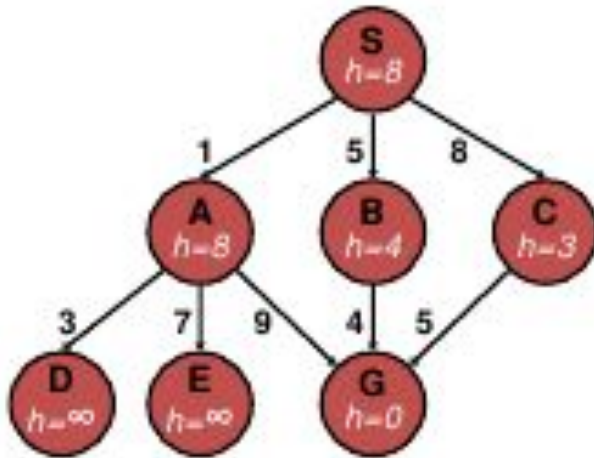
n	g(n)	h(n)	f(n)	frontier
S	0	8	8	S:8
A	1	8	9	[A:9, B:9, C:11]
B	5	4	9	[B:9, G:10, C:11, D:inf, E:inf]
C	8	3	11	
D	1+3=4	inf	inf	
E	1+7=8	inf	inf	
G	1+9=10	0	10	

Busca A* - Exemplo



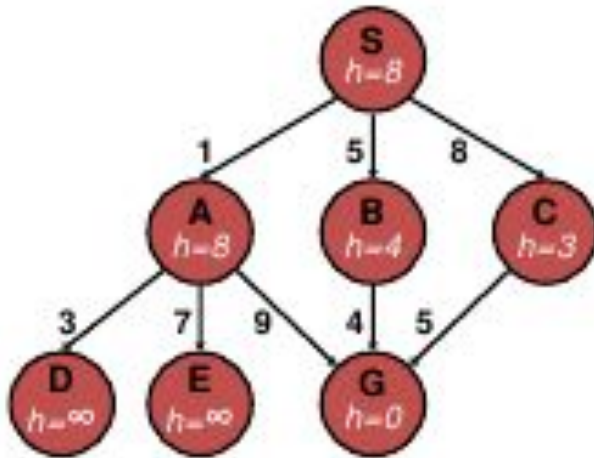
n	g(n)	h(n)	f(n)	frontier
S	0	8	8	S:8
A	1	8	9	[A:9, B:9, C:11]
B	5	4	9	[B:9 , G:10, C:11, D:inf, E:inf]
C	8	3	11	
D	1+3=4	inf	inf	
E	1+7=8	inf	inf	
G	1+9=10	0	10	

Busca A* - Exemplo



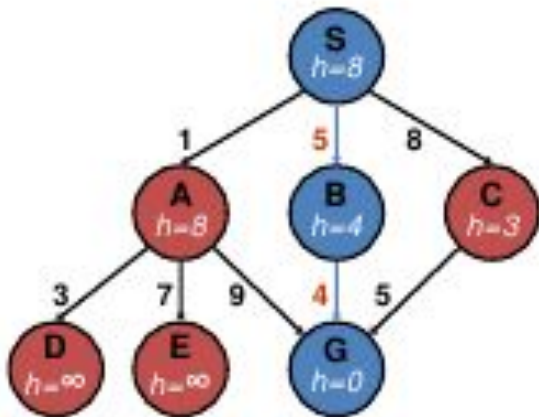
n	g(n)	h(n)	f(n)	frontier
S	0	8	8	S:8
A	1	8	9	[A:9, B:9, C:11]
B	5	4	9	[B:9 , G:10, C:11, D:inf, E:inf]
C	8	3	11	[G:9 , C:11, D:inf, E:inf]
D	1+3=4	inf	inf	
E	1+7=8	inf	inf	
G	5+4=9	0	9	

Busca A* - Exemplo



n	g(n)	h(n)	f(n)	frontier
S	0	8	8	S:8
A	1	8	9	[A:9, B:9, C:11]
B	5	4	9	[B:9, G:10, C:11, D:inf, E:inf]
C	8	3	11	[G:9 , C:11, D:inf, E:inf]
D	1+3=4	inf	inf	
E	1+7=8	inf	inf	
G	5+4=9	0	9	

Busca A* - Exemplo



Solução: S-B-G
custo = 9

n	g(n)	h(n)	f(n)	frontier
S	0	8	8	S:8
A	1	8	9	[A:9, B:9, C:11]
B	5	4	9	[B:9, G:10, C:11, D:inf, E:inf]
C	8	3	11	[G:9, C:11, D:inf, E:inf]
D	1+3=4	inf	inf	
E	1+7=8	inf	inf	
G	5+4=9	0		

Busca A* - algoritmo

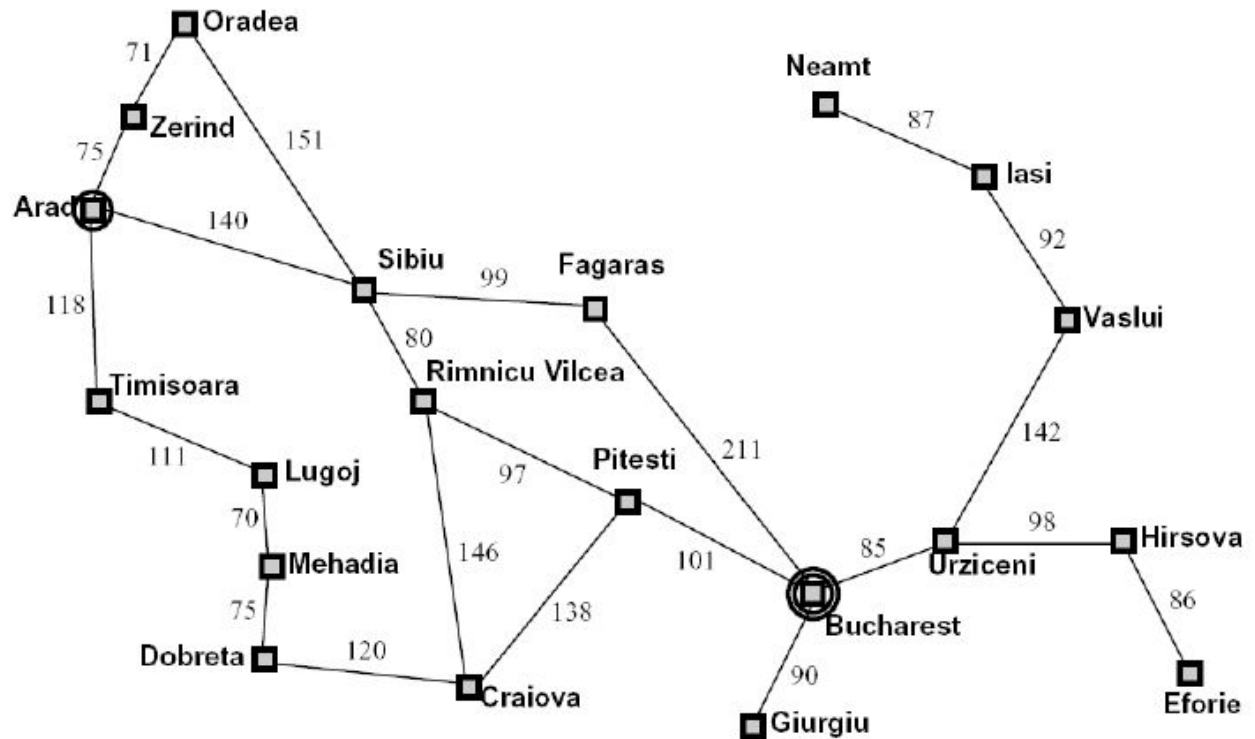
```
function A-STAR(problem, heuristic_function) returns uma solução ou falha
  node = make_node(problem.initialState)
  pathCost = 0
  frontier = lista de prioridades, ordenada por pathCost + heuristic_function(node)
  frontier.add(node)
  explored = emptySet
  loop do
    if frontier.empty() return falha
    node = frontier.pop()
    if problem.goalTest(node.state) return solution(node)
    explored.add(node.state)
    for each action in problem.actions(node.state) do
      child = child-node(problem, node, action) // atualiza pathCost a partir de node
      if child.state não está em exploredSet ou em frontier
        frontier.add(child)
      else if child.state está em frontier com f(child) menor
        frontier.replace(oldNode, child)
```

Exercício - Busca A*, de Arad a Bucharest

- $h(n)$ = linha reta da cidade n até a meta

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



F = [Arad:366]

S = {Arad}

F = [Sibiu: 393, Timi: 447, Zerind: 449]

S = {Arad, Sibiu}

F = [RV: 413, Fag: 417, Timi: 447, Zerind: 449, Oradea:671]

S = {Arad, Sibiu, RV}

F = [Pitesti: 415, Fag: 417, Timi: 447, Zerind: 449, Craiova: 526, Oradea:671]

S = {Arad, Sibiu, RV, Pitesti}

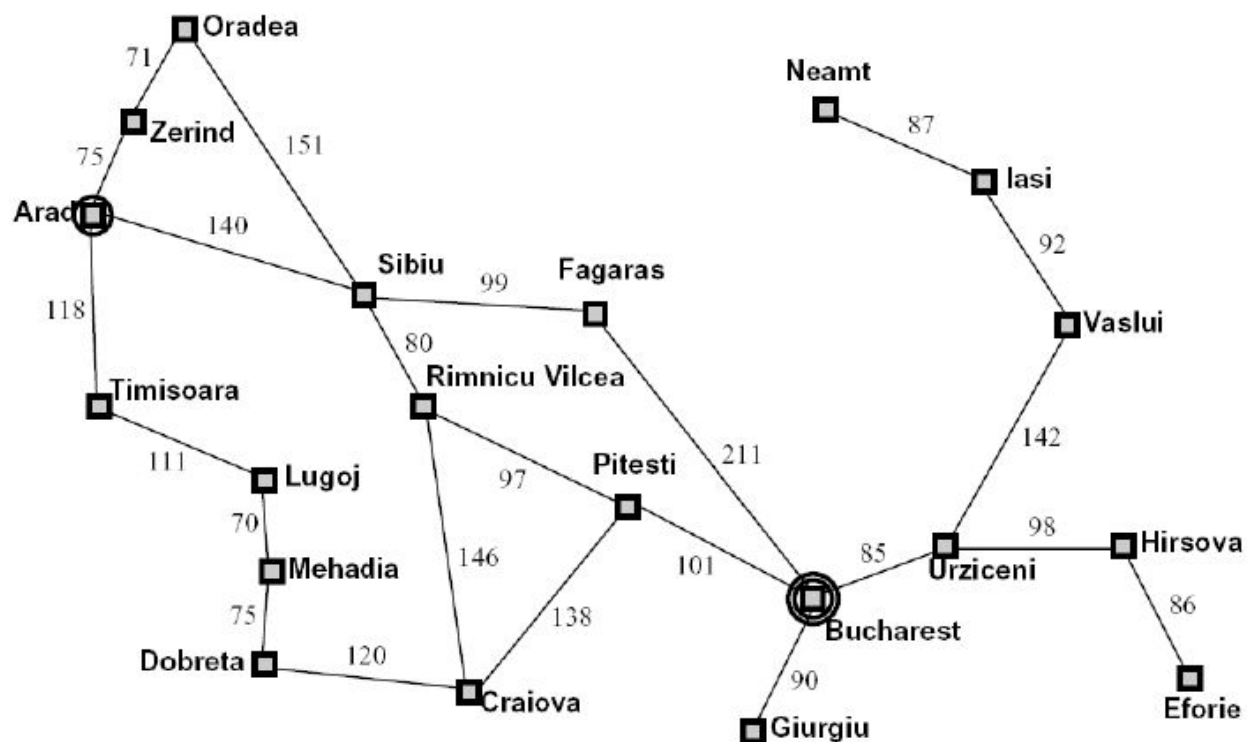
F = [Fag: 417, Buch: 418, Timi: 447, Zerind: 449, Craiova: 526, Oradea:671]

S = {Arad, Sibiu, RV, Pitesti, Fagaras}

F = [Buch: 418, Timi: 447, Zerind: 449, Craiova: 526, Oradea:671]

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



Avaliação da busca A*

- Completa?
 - Sim – se espaço de estados for finito
- Tempo?
 - Exponencial em [erro relativo de h * tamanho da solução]
- Espaço?
 - Todos os nós ficam na memória
- Ótima?
 - Sim, se
 - Versão de busca em árvore: se heurística for admissível
 - Versão de busca em grafo: se heurística for consistente
 - consistência é uma condição levemente mais forte

Busca A*

- Heurística **admissível**

- $h(n) \leq h^*(n)$, onde $h^*(n)$ é o custo mínimo real da solução que passa por n
- O custo estimado nunca é superestimado, **é otimista**
 - Ex: distância em linha reta entre dois pontos

Busca A*

- Heurística consistente (monotonicidade)
 - se para todo nó n e para todo sucessor n' de n gerado por uma ação a , o custo estimado de chegar no objetivo a partir de n não é maior do que o custo de ir para n' somado ao custo estimado de chegar no objetivo a partir de n'
 - $h(n) \leq c(n,a,n') + h(n')$
- Toda heurística consistente é admissível
- É bem difícil achar uma heurística admissível mas não consistente

Como criar heurísticas admissíveis?

- Uma possibilidade é relaxar os critérios da busca
 - idéia chave: o custo da solução ótima de um problema relaxado não é maior que o custo mínimo real

Como criar heurísticas admissíveis?

- Quebra cabeça de 8
 - Suponha que uma peça pode se mover para qualquer lugar

Como criar heurísticas admissíveis?

- Quebra cabeça de 8
 - Suponha que uma peça pode se mover para qualquer lugar
 - $h1(n)$: número de peças fora da posição

Como criar heurísticas admissíveis?

- Quebra cabeça de 8
 - Suponha que uma peça pode se mover para qualquer quadrado adjacente

Como criar heurísticas admissíveis?

- Quebra cabeça de 8
 - Suponha que uma peça pode se mover para qualquer quadrado adjacente
 - $h_2(n)$: para cada quadrado, calcular a distância em quadrados até a sua posição: distância de Manhattan

Heurísticas admissíveis

- Quebra cabeça de 8
 - $h1(n)$: número de peças fora da posição
 - $h2(n)$: distância de Manhattan

7	2	4
5		6
8	3	1

1	2	3
4	5	6
7	8	

- $h1(n)$?
- $h2(n)$?

Heurísticas admissíveis

- Quebra cabeça de 8
 - $h1(n)$: número de peças fora da posição
 - $h2(n)$: distância de Manhattan

7	2	4
5		6
8	3	1

1	2	3
4	5	6
7	8	

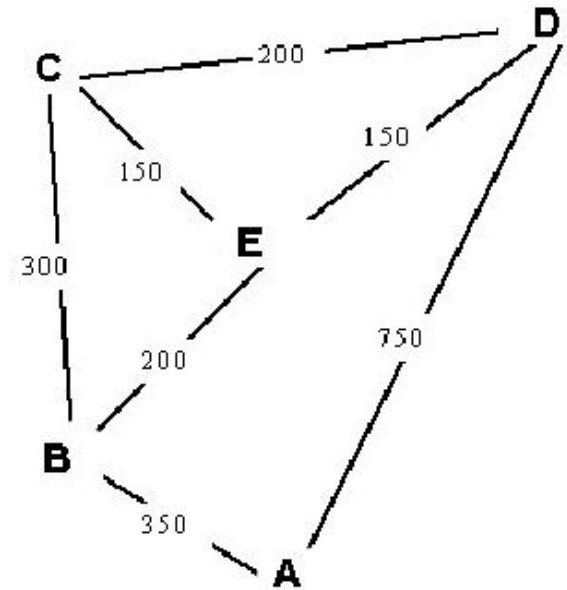
- $h1(S)$: 6
- $h2(S)$: $4+0+3+3+1+0+2+1 = 14$

Exercício

Um vendedor quer visitar as seguintes localidades do mapa, representadas por letras do alfabeto, sem repetir a visita e sem voltar ao local de origem (A).

Resolva este problema usando busca gulosa e A^* , considerando a seguinte heurística:

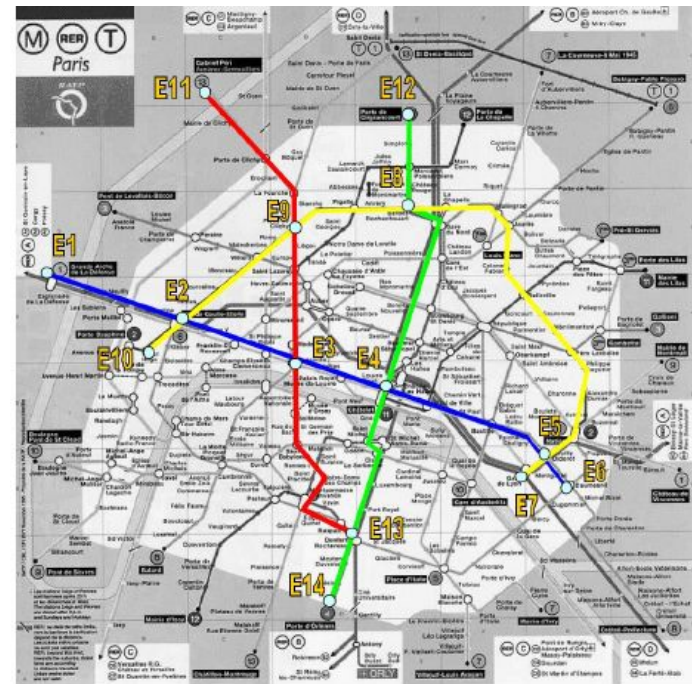
O número de quilômetros a percorrer é igual ao comprimento médio dos caminhos (200km), vezes o número de localidades que faltam percorrer. Em caso de empate entre dois nós, escolha aquele que esteja alfabeticamente primeiro.



Exercício (Puc-rio)

Considere o mapa das estações de metrô de Paris.
Descubra qual a rota que um usuário deveria seguir para sair de E1 e chegar em E12, ao usar as buscas gulosa e A*.

As tabelas de heurística e distância estão no slide seguinte



	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14
E1	-	10												
E2		-	8,5						10	3,5				
E3			-	6,3					9,4				18,7	
E4				-	13			15,3					12,8	
E5					-	3	2,4	30						
E6						-								
E7							-							
E8								-	9,6			6,4		
E9									-		12,2			
E10										-				
E11											-			
E12												-		
E13													-	5,1
E14														-

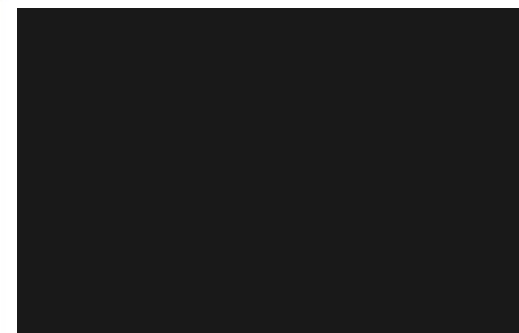




Tabela 1: Distancias reais entre as estações do metro de Paris.

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14
E1	-	10	18,5	24,8	36,4	38,8	35,8	25,4	17,6	9,1	16,7	27,3	27,6	29,8
E2		-	8,5	14,8	26,6	29,1	26,1	17,3	10	3,5	15,5	20,9	19,1	21,8
E3			-	6,3	18,2	20,6	17,6	13,6	9,4	10,3	19,5	19,1	12,1	16,6
E4				-	12	14,4	11,5	12,4	12,6	16,7	23,6	18,6	10,6	15,4
E5					-	3	2,4	19,4	23,3	28,2	34,2	24,8	14,5	17,9
E6						-	3,3	22,3	25,7	30,3	36,7	27,6	15,2	18,2
E7							-	20	23	27,3	34,2	25,7	12,4	15,6
E8								-	8,2	20,3	16,1	6,4	22,7	27,6
E9									-	13,5	11,2	10,9	21,2	26,6
E10										-	17,6	24,2	18,7	21,2
E11											-	14,2	31,5	35,5
E12												-	28,8	33,6
E13													-	5,1
E14														-

Tabela 2: Distancias em linha reta entre as estações do metro de Paris.

Exercício

A figura ao lado é um tabuleiro de xadrez modificado 5X5, sem o quadrado no topo da direita. A tarefa é capturar o rei preto (em G), movendo o cavalo branco (em V) e usando apenas os movimentos padrão definidos no xadrez.

A	B	C	D	
E	F	G 	H	I
J	K	L	M	N
O	P	Q	R	S
T	U	V 	W	X

Assuma que o modelo de transição expande os estados na seguinte ordem: (1 à direita, 2 acima); (2 à direita, 1 acima); (2 à direita, 1 abaixo); (1 à direita, 2 abaixo); (1 à esquerda, 2 abaixo); (2 à esquerda, 1 abaixo); (2 à esquerda, 1 acima); (1 à esquerda, 2 acima).

(a) Considere que o movimento do cavalo custa 1. Considere a seguinte heurística: $h(n) = |u - p| + |v - q|$, onde o quadrado associado a cada nó está na coordenada (u,v), e o quadrado do objetivo está em (p, q). $h(n)$ é admissível?

(b) Execute as buscas gulosa e A* nesse ambiente.

Exercícios

Visite o site <http://qiao.github.io/PathFinding.js/visual/> e descreva as diferenças entre as buscas realizadas para o problema de *PathFinding*, considerando as buscas A^* , gulosa e em largura. Considere diferentes posições de início e fim.

Exercícios do livro texto

Capítulo 3, 3rd edition:

- 3.2; 3.3; 3.8; 3.14; 3.15 (a-c); 3.21; 3.23