Grafos: Unidade 7 – Busca em Grafos

Busca Heurística

Adaptado do material de IA do Prof. Dr. João Luis Tavares da Silva (UCS)

1

Busca Heurística

- **Como encontrar um barco perdido?... não se pode procurar no oceano inteiro...**
- Estratégias de **busca heurística** utilizam *conhecimento específico* do problema na escolha do próximo nó a ser expandido e aplicam uma *função de avaliação* a cada nó na fronteira do espaço de estados.
- Uma boa função heurística deve ser admissível i.e., nunca *superestimar* o custo real da solução
  - Distância direta (h_{dd}) é *admissível* porque o caminho mais curto entre dois pontos é sempre uma linha reta

2

Busca Heurística

- Pode-se melhorar a performance da busca através de uma função $h(n)$ que representa uma avaliação do custo para atingir o estado final, para cada nodo n do espaço de busca.
- A função $h(n)$ é chamada função heurística.
- A função heurística é dependente do problema, já que utiliza informação adicional para melhorar a busca.
- Exemplos de funções heurísticas:
  - minimizar o custo para de atingir o objetivo;
  - estimar o menor caminho do estado n até um estado objetivo.

3

Tipos de Busca Heurística

- Busca pela melhor escolha (*Best-First*)
  - Busca "gulosa" (*greedy*)
  - Busca A*
- Busca com limite de espaço
  - Busca IDA (A* interativo)
  - Subida de encosta (*HillClimbing*)

4

Busca *Best-First*

- Função de avaliação (heurística): custo para chegar até o nodo atual ($g(n)$) e o custo estimado para chegar até a solução ($h(n)$).
  - $f(n) = g(n) + h(n)$

5

Busca *Best-First*

- Admissibilidade da função heurística:
  - $h(n)$: estimativa de custo de n até a solução;
  - $g(n)$: custo real de n até a solução;
  - Se $h(n) \leq g(n)$, então h é admissível e, portanto, o algoritmo será ótimo (**sempre encontra o caminho mais curto até a solução**)
- Uma heurística é admissível se, para cada nodo, o valor retornado por esta heurística nunca ultrapassa o custo real do melhor caminho deste nodo até o objetivo

6

Busca *Best-First*

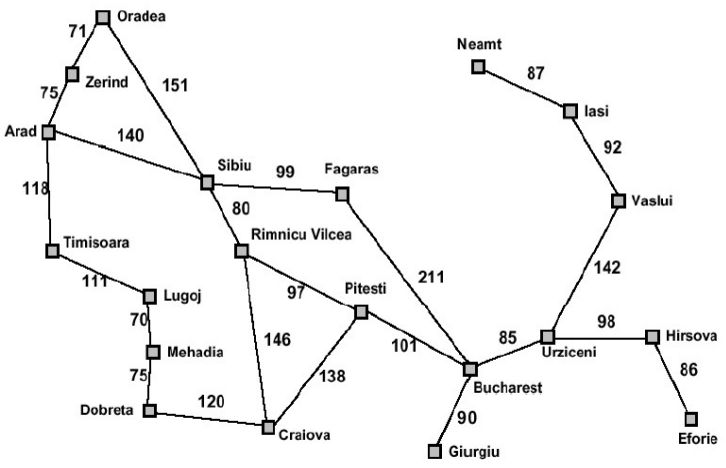
- Algoritmo:

```
BEST-FIRST (estadoInicial)
Início
    nodos ← CRIA-FILA(estadoInicial)
    loop
        se nodos é vazio então
            retorna falha
        nodo ← OBTEM-MELHOR-NODO(nodos)
        se É-OBJETIVO(nodo) então
            retorna nodo
        novos-nodos ← EXPANDE(nodo)
        nodos ← ADD-FILA(nodos,novos-nodos)
    fim_loop
Fim
```

OBTEM-MELHOR-NODO(nodos): *implementa a função f(n)*

7

Busca Heurística - *Exemplo*



h(n)

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

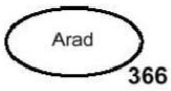
8

Busca Gulosa - *Exemplo*

- Função de avaliação:
  - $h(n)$ = estimativa do custo do nodo ao objetivo.
- *Exemplo*
  - $h(n)$ = distância em linha reta ao objetivo.
  - expande primeiro o nó que aparentemente é o mais próximo do objetivo, de acordo com $h(n)$.

9

Busca Gulosa - *Exemplo*

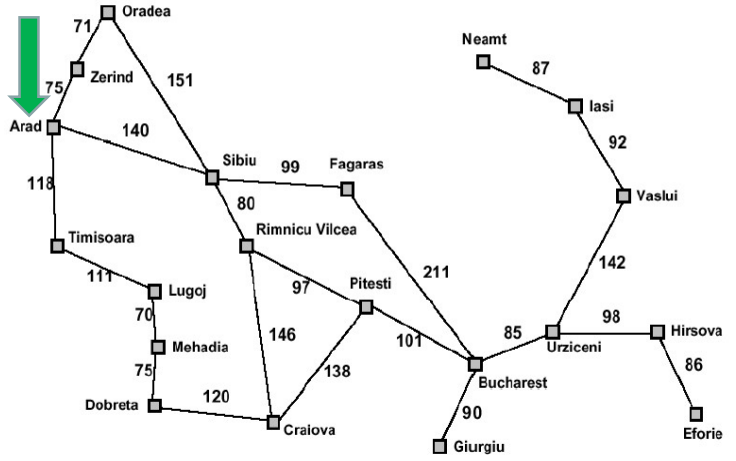


h(n)

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

10

Busca Heurística - *Exemplo*

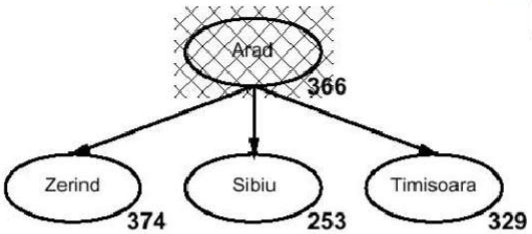


h(n)

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

11

Busca Gulosa - *Exemplo*

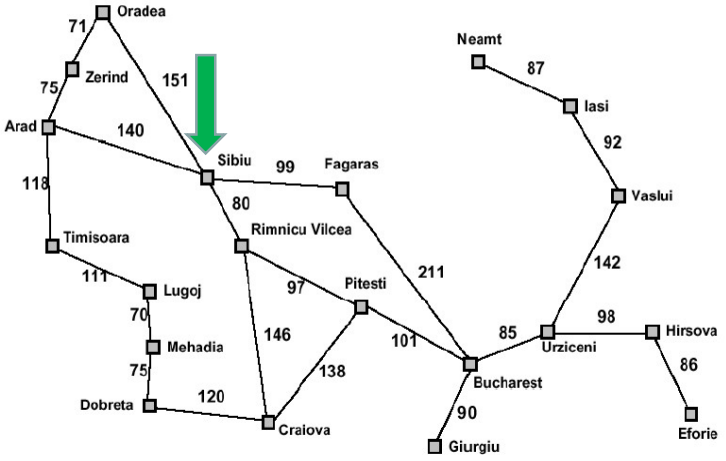


h(n)

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

12

Busca Heurística - *Exemplo*



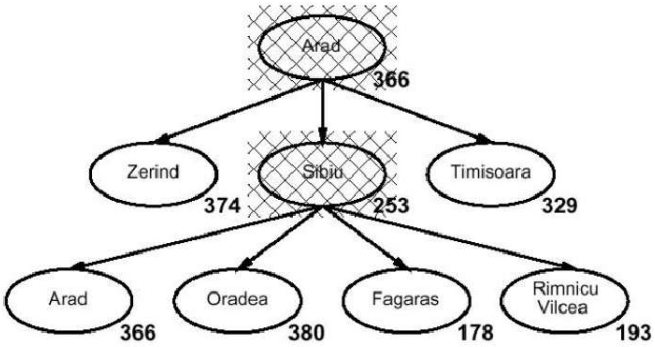
h(n)

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

13

Busca Gulosa - *Exemplo*



h(n)

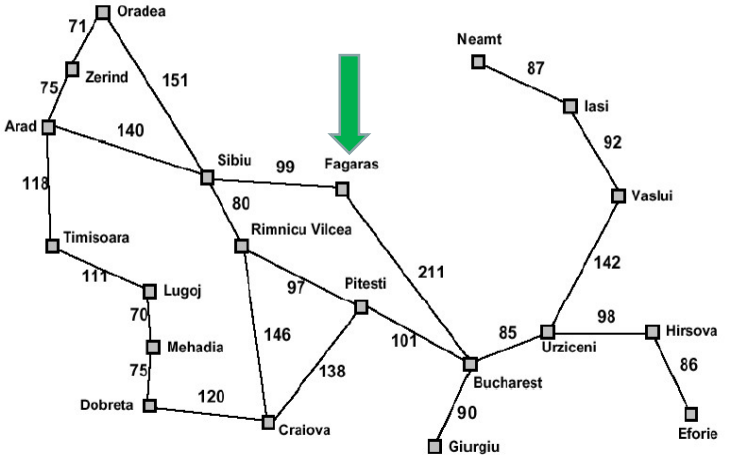
Straight–line distance to Bucharest

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

64

14

Busca Heurística - *Exemplo*



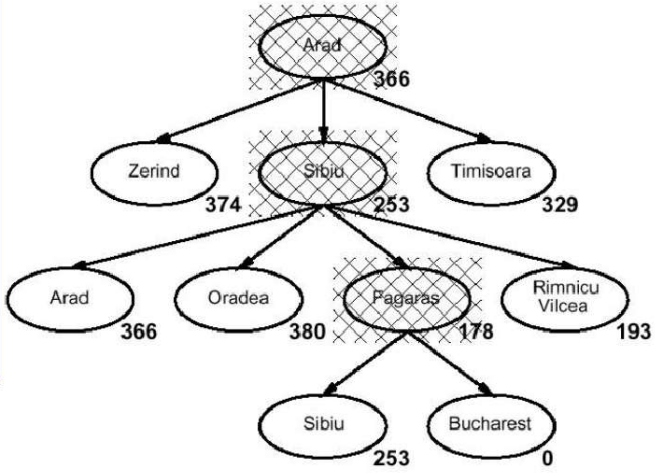
h(n)

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

15

Busca Gulosa - *Exemplo*



h(n)

Straight–line distance to Bucharest

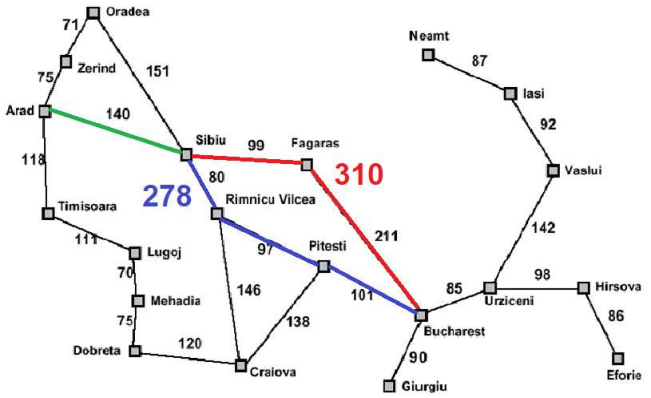
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

65

16

Busca gulosa: avaliação

- Complexidade: Não expandiu nenhum nó fora do caminho da solução: baixo custo de busca!
- Otimização: O caminho passando por Sibiu e Fagaras é 32 quilômetros mais longo do que passando por Rimnicu Vilcea e Pitesti. Não é ótimo.



17

Busca A* - *Exemplo*

- Algoritmo:

```
A-STAR (estadoInicial)
Início
    nodos ← CRIA-FILA(estadoInicial)
    loop
        se nodos é vazio então
            retorna falha
        nodo ← OBTEM-MELHOR-NODO(nodos)
        se É-OBJETIVO(nodo) então
            retorna nodo
        novos-nodos ← EXPANDE(nodo)
        nodos ← ADD-FILA(nodos,novos-nodos)
    fim_loop
Fim
```

OBTEM-MELHOR-NODO(nodos): *implementa a função* $f(n) = g(n) + h(n)$

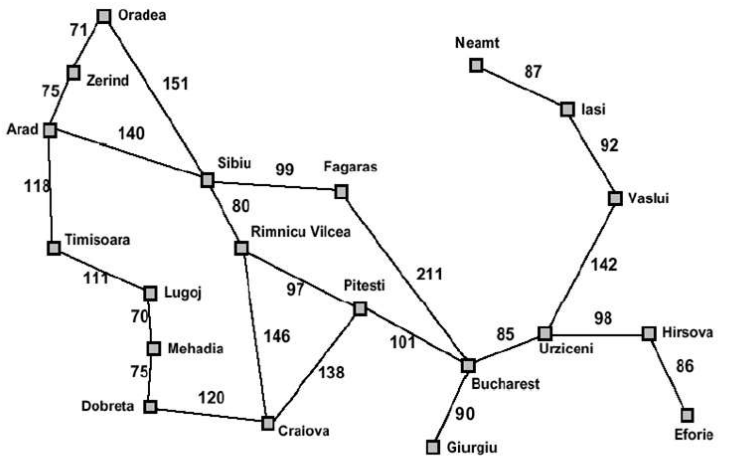
18

Busca A* - *Exemplo*

- Tenta minimizar o custo total da solução combinando:
  - *Busca Gulosa:* econômica, porém não é completa nem ótima
  - *Busca de Custo Uniforme:* ineficiente, porém completa e ótima
- Função de avaliação:
  - $f(n) = g(n) + h(n)$
  - $g(n)$ = distância de n ao nó inicial
  - $h(n)$ = distância estimada de n ao nó final
  - A* expande o nó de menor valor de f na fronteira do espaço de estados.

19

Busca A* - *Exemplo*



h(n)

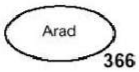
Straight–line distance to Bucharest

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

69

20

Busca A* - *Exemplo*

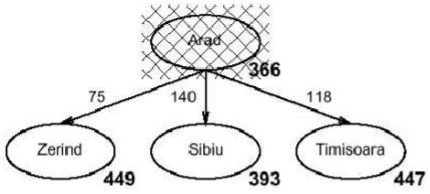


h(n)

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

21

Busca A* - *Exemplo*



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

f(Zerind) = *75* + 374 = 449

f(Sibiu) = *140* + 253 = 393

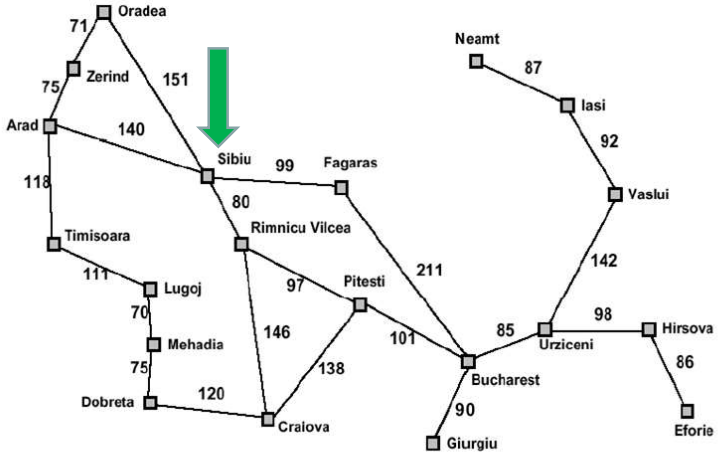
f(Timisoara) = *118* + 329 = 447

h(n)

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

22

Busca A* - *Exemplo*



h(n)

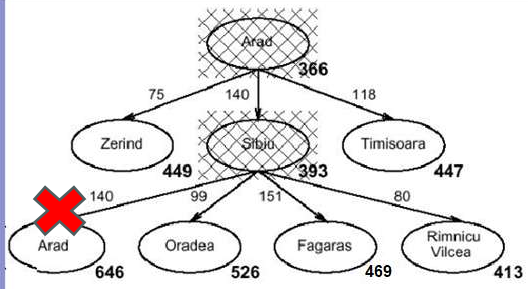
Straight–line distance to Bucharest

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

69

23

Busca A* - *Exemplo*



h(n)

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(n) = *g(n)* + h(n)

f(Zerind) = *75* + 374 = 449

f(Timisoara) = *118* + 329 = 447

f(Oradea) = (*140*+*99*) + 380 = 526

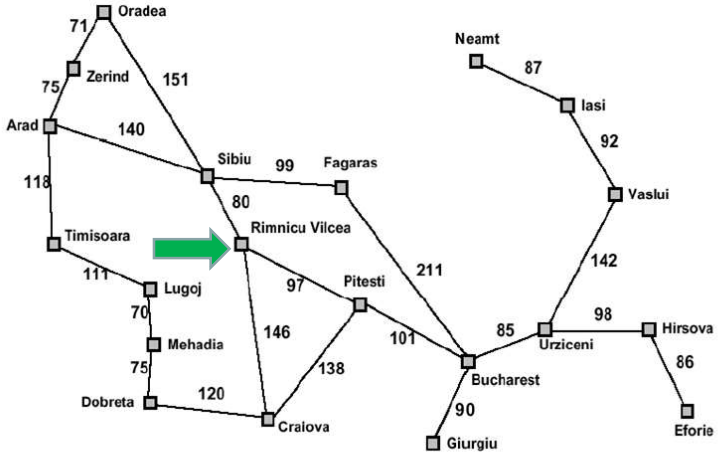
f(Fagaras) = (*140*+*151*) + 178 = 469

f(Rimnicu Vilcea) = *(140+80)* + 193 = 413

71

24

Busca A* - *Exemplo*



h(n)

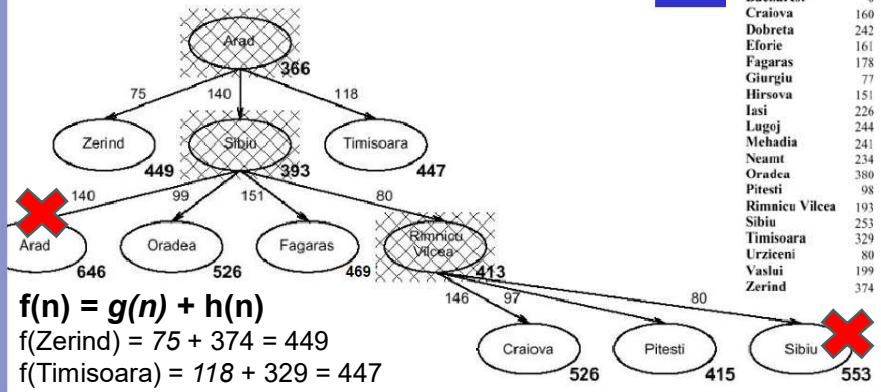
Straight–line distance to Bucharest

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

69

25

Busca A* - *Exemplo*



h(n)

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(n) = *g(n)* + h(n)

f(Zerind) = *75* + 374 = 449

f(Timisoara) = *118* + 329 = 447

f(Oradea) = (*140+99*) + 380 = 526

f(Fagaras) = (*140+151*) + 178 = 469

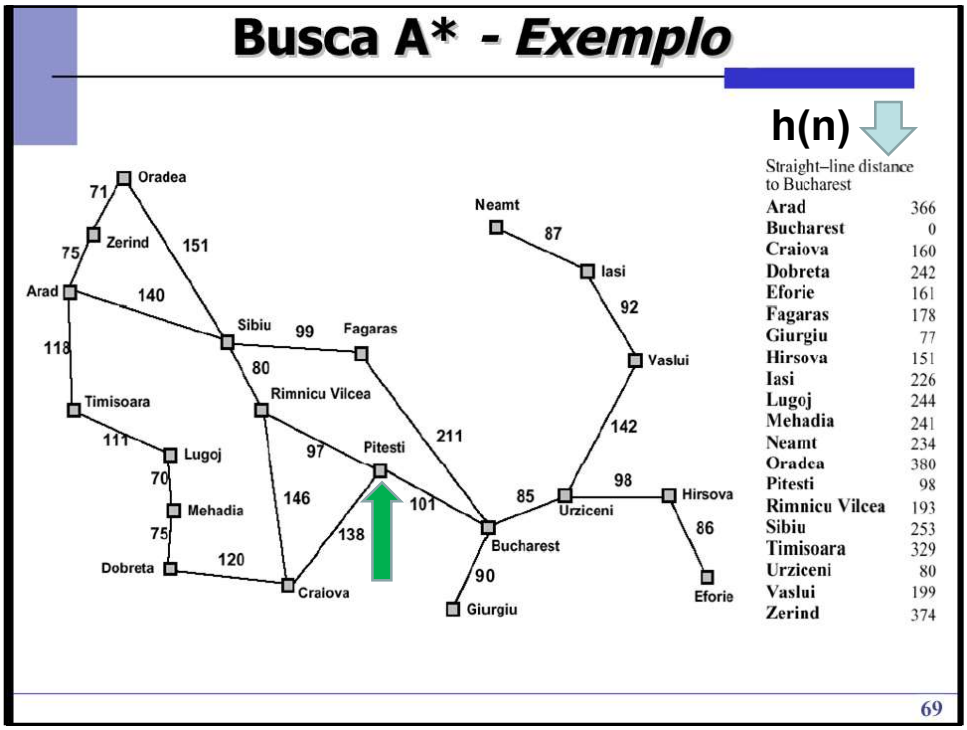
f(Craiova) = *(140+80+146)* + 160 = 526

f(Pitesti) = (*140+80+97)* + 98 = 415

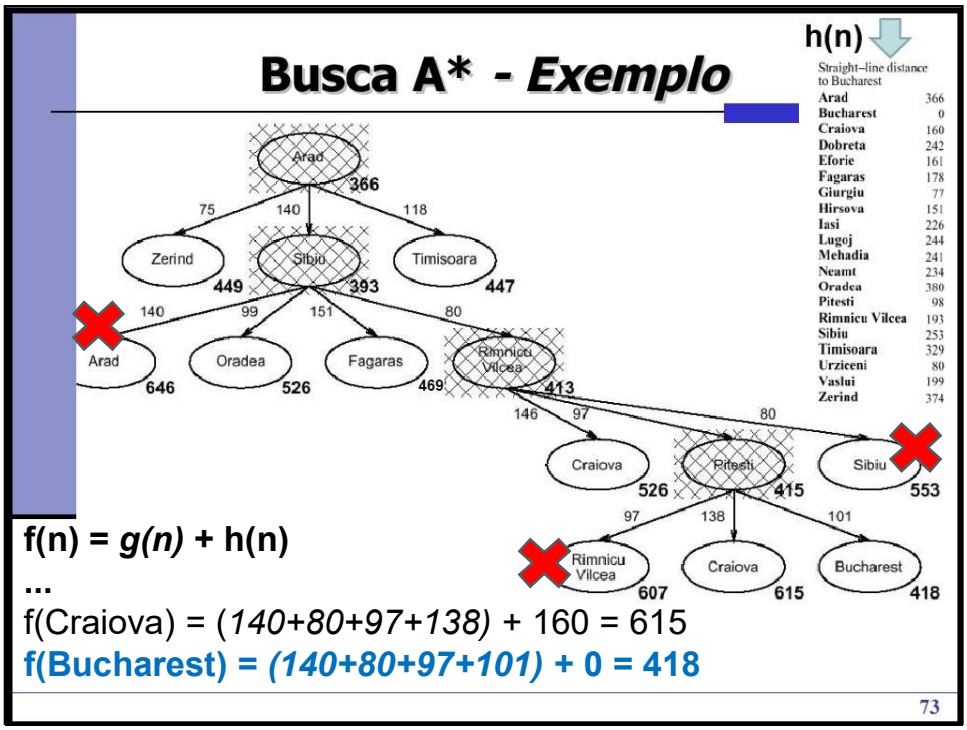
f(Sibiu) = (*140+80+80)* + 253 = 553

72

26


Busca A* - Exemplo
h(n)
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
Oradea
Neamt
Zerind
Iasi
Arad
Sibiu
Fagaras
Vaslui
Rimnicu Vilcea
Timisoara
Pitesti
Lugoj
Hirsova
Urziceni
Mehadia
Bucharest
Dobreta
Craiova
Eforie
Giurgiu
71
87
75
151
92
140
99
118
80
142
211
111
97
98
70
146
101
85
86
75
138
120
90
69

27


Busca A* - Exemplo
h(n)
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
Arad
366
75
140
118
Zerind
Sibiu
Timisoara
449
393
447
140
99
151
80
Arad
Oradea
Fagaras
Rimnicu Vilcea
646
526
469
413
146
97
80
Craiova
Pitesti
Sibiu
526
415
553
97
138
101
Rimnicu Vilcea
Craiova
Bucharest
607
615
418
f(n) = g(n) + h(n)
...
f(Craiova) = (140+80+97+138) + 160 = 615
f(Bucharest) = (140+80+97+101) + 0 = 418
73

28

Busca A*

Heurística admissível

- A* será ótima se o espaço de busca for uma árvore e *h(n)* for uma **heurística admissível**
  - Heurística admissível: nunca superestima o custo para alcançar o objetivo. É uma função otimista. Exemplo:
    - Distância em linha reta: a estrada entre duas cidades pode ser mais comprida do que a distância em linha reta, mas nunca menor (estimativa otimista).

29

Pathfinding com A*

(material adaptado prof Rudimar Dazzi)

- O que é Pathfinding?
  - Pathfinding é uma maneira de buscar a trajetória de um ponto a outro pela rota mais curta sem passar por cima de locais bloqueados, como paredes, rios e etc.
- Onde aplicar?
  - É muito usado como auxiliar na criação de jogos onde os personagens requerem uma capacidade maior de inteligência de navegação pelo mapa.

30

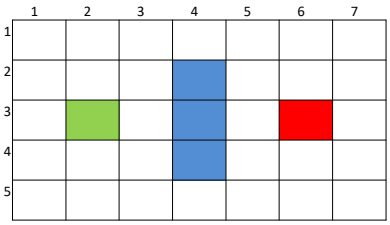
Pathfinding com A*

- O Algoritmo consiste principalmente no calculo da distância F:
  - F = G + H onde:
    - G: Custo da origem até o ponto atual
    - H: Custo estimado do ponto atual até o destino
      - Varia a heurísticas para calcular H, distância Manhattan, Euclidiana entre outras.
  - Lista aberta: Armazena os nós a serem analisados
  - Lista Fechada: Armazena os nós já analisados para evitar voltar a um nó já analisado, ou andar em circulo.

31

Pathfinding com A* (Exemplo 1)

- F = G + H
  - G=Distância percorrida
  - H=Distância do destino
- Custo G:
  - G=10 para os lados
  - G=14 para as diagonais



14	10	14
10	x	10
14	10	14

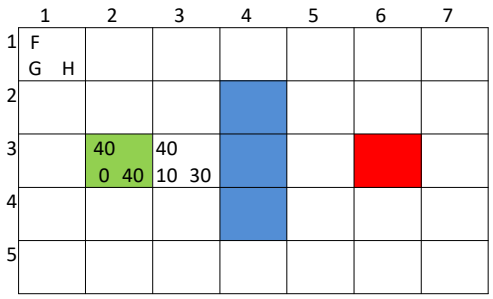
- Custo H:
  - Distância Manhattan: $H=10*(|x_1-x_2|+|y_1-y_2|)$
    - H=10*(abs(x_corrente – x_destino)+abs(y_corrente-y_destino))

32

Pathfinding com A* (Exemplo 1)

Aberta..: 32
Fechada:

Aberta..: 21,22,31,33,41,42,43
Fechada: 32



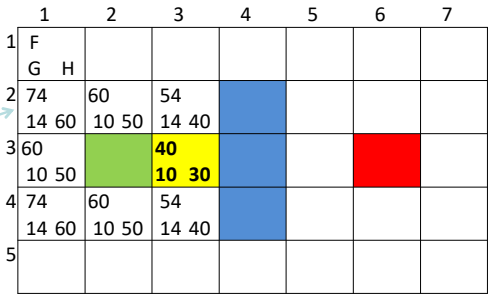
- Destino: posição 3X6
- F = G + H (para a posição 3x3)
  - G = 10 => mvto linha reta
  - H = 10*(|x_{corr}-x_{dest}|+|y_{corr}-y_{dest}|) = 10*(|3-6|+|3-3|) = 30
  - F = 10+30 = 40

33

Pathfinding com A* (Exemplo 1)

Abertos..: 21,22,31,33,41,42,43
Fechados: 32
Destino: 3X6

Exemplo:
=> para a posição 2X1
G = 14 => diagonal
H = 10*(|x_{corr}-x_{dest}|+|y_{corr}-y_{dest}|)
H = 10*(|1-6|+|2-3|) = 60
F = G + H = 14+60 = 74

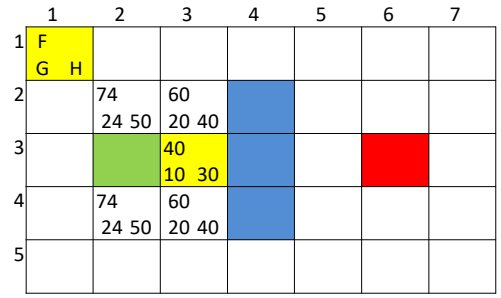


- Calcula todos as possibilidades vizinhas de 3x2
- A posição **3x3** é a que tem a menor distância
  - F = 40
  - Esse é o próximo passo (fecha 3x3)

34

Pathfinding com A* (Exemplo 1)

Abertos..: 21,22,32,31,41,42,43
Fechados: 32,33
Destino: 3X6

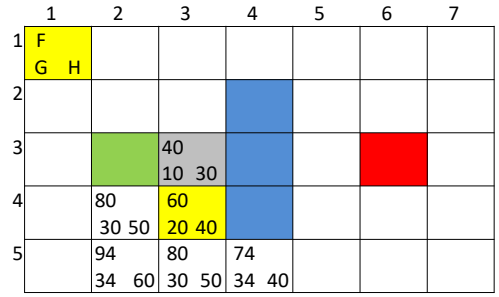


- Calcula todos as possibilidades vizinhas de 3x3
- As posições fechadas e o muro, não calcular
  - 2x3 e 4x3 tem F=54, se recalcular com referência em 3x3 F=60 (A diagonal é mais curta, fecha 4x3)
    - G em 3x3 é 10 + 10 para ir a 4x2 = 20 (+ 40 de H) =60

35

Pathfinding com A* (Exemplo 1)

Abertos..: 42,52,53,54
Fechados: 32,33,43
Destino: 3X6

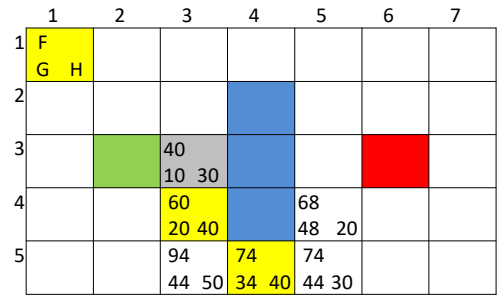


- Calcula todos as possibilidades vizinhas de 4x3
- A melhor opção é 5x4 com F=74
  - Fecha 5x4 e segue o caminho

36

Pathfinding com A* (Exemplo 1)

Abertos..: 53,55,45
Fechados: 32,33,43,54
Destino: 3X6

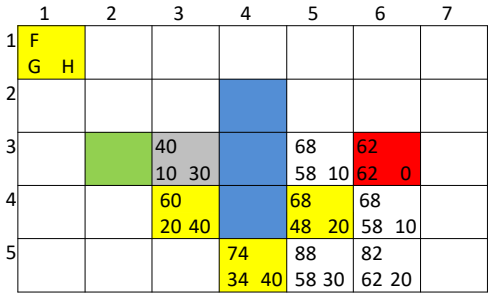


- Calcula todos as possibilidades vizinhas de 5x4
- A melhor opção é 4x5 com F=68
  - Fecha 4x5 e segue o caminho

37

Pathfinding com A* (Exemplo 1)

Abertos..: 55,56,46,35,36
Fechados: 32,33,43,54,45
Destino: 3X6



- Calcula todos as possibilidades vizinhas de 4x5
- A melhor opção é 3x6 com 62
  - Como esse é o destino processo encerrado!

38

Pathfinding com A* (Exemplo 2)

- Solução do 8 PUZZLE com A*
- F = G + H
  - G = 1 para cada passo
  - H = quantidade de peças fora do lugar
  - F da situação inicial
    - G = 0 não saiu do lugar ainda
    - H = 5 pois são 5 peças fora do lugar (6,2,3,4,5)

Situação Inicial

1	6	2
8		3
7	5	4

Situação Final

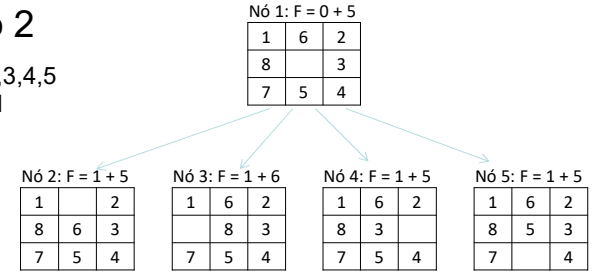
1	2	3
8		4
7	6	5

39

Pathfinding com A* (Exemplo 2)

- Fecha nó 1 e calcula todos as 4 possibilidades de movimento.
- Como 2, 4 e 5 tem F = 6, qualquer um deles pode ser escolhido para seguir.
- Fecha o 2

Abertos..: 2,3,4,5
Fechados: 1

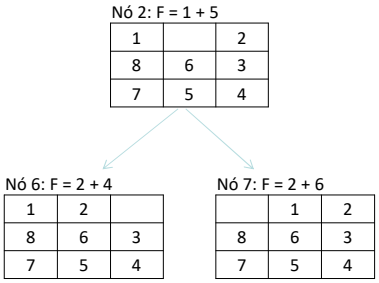


40

Pathfinding com A* (Exemplo 2)

- Fecha nó 2 e calcula todos as 2 possibilidades de movimento.
- Escolhi o nó 6 que tem o menor F = 4
- Fecha o nó 6

Abertos..: 6,7
Fechados: 1,2

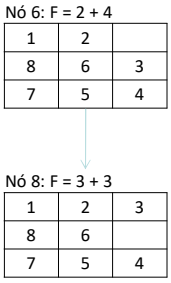


41

Pathfinding com A* (Exemplo 2)

- Fecha nó 6 e calcula todos as possibilidades de movimento. Nesse caso só temos 1
- Fecha o nó 8

Abertos..: 8
Fechados: 1,2,6

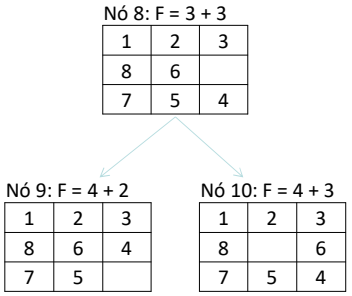


42

Pathfinding com A* (Exemplo 2)

- Fecha nó 8 e calcula todos as 2 possibilidades de movimento.
- Escolhe o nó 9 que tem o menor F = 6
- Fecha o nó 9

Abertos..: 9,10
Fechados: 1,2,6,8

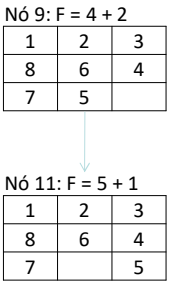


43

Pathfinding com A* (Exemplo 2)

- Fecha nó 9 e calcula todos as possibilidades de movimento. Nesse caso só temos 1
- Fecha o nó 9

Abertos..: 11
Fechados: 1,2,6,8,9

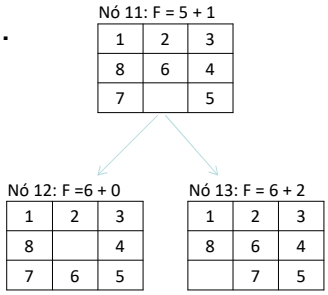


44

Pathfinding com A* (Exemplo 2)

- Fecha nó 11 e calcula todos as possibilidades de movimento.
- Nó 12 é a situação final.
- Processo encerrado.

Abertos..: 12,13
Fechados: 1,2,6,8,9,11



45