Inteligência Artificial: Busca Heurística

Prof. Leandro Fernandes

Busca heurística (busca informada)

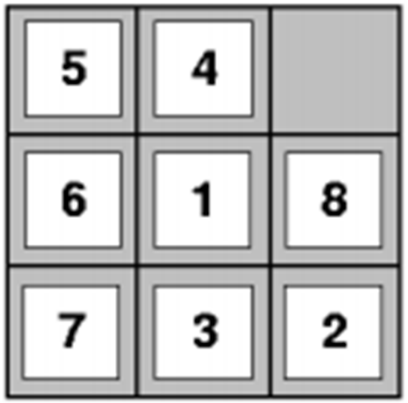
- A ideia é levar em conta o objetivo para decidir qual caminho escolher
- Pode existir conhecimento extra que pode ser utilizado para guiar o processo de busca:
- Heurística: função $h(n)$
  - é uma estimativa do custo do caminho de um nó n até um nó objetivo (futuro)
  - deve ser simples de ser computada, de preferência ela deve ser derivada das propriedades do próprio nó
  - é um subestimativa se não existe nenhum outro caminho de n até o objetivo que seja menor do que $h(n)$

Exemplos de funções heurísticas

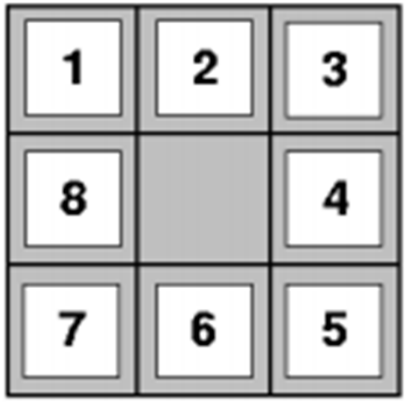
- Se os nós são pontos em um plano e o custo é a distância entre os pontos
  - $h(n)$ = distância em linha reta de n até o objetivo mais próximo
- Se é um grafo de queries da derivação de uma base
  - $h(n)$ = número de átomos da query
- Se os nós são lugares e o custo é tempo
  - $h(n)$ = distância até o objetivo, dividida pela velocidade máxima

Construindo funções heurísticas

- Exemplo de heurísticas para o 8-puzzle
  - h1(n) = número de quadrados em locais errados
  - h2(n) = distância Manhattan total => número de espaços que deve ser movido para chegar no local correto
    - h1(s) = 7 (só o número 7 está no local correto)
    - h2(s) = 2+3+3+2+4+2+0+2 = 18



Estado inicial



Estado meta

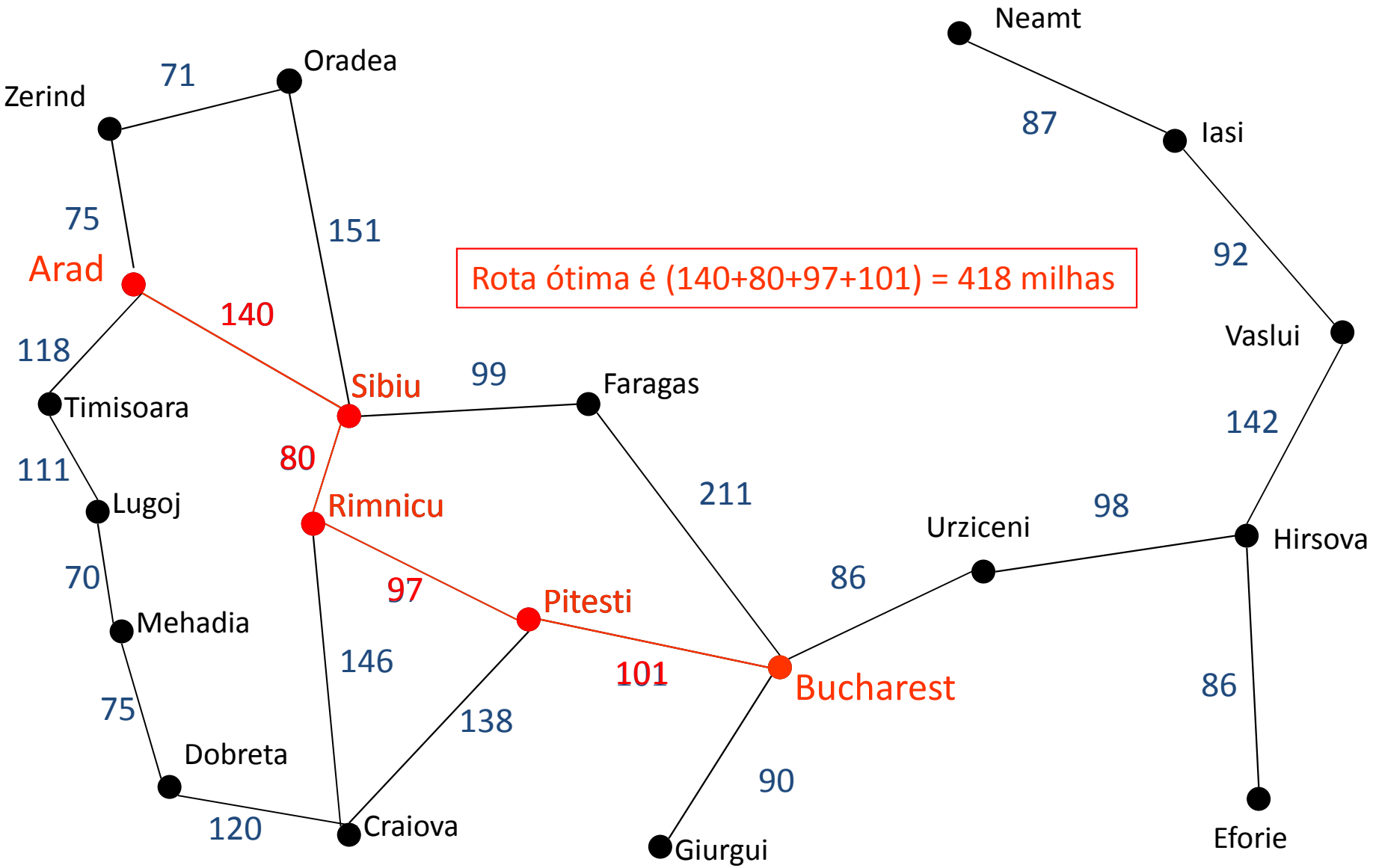
- Relaxar o problema pode ser uma boa forma de conseguir uma heurística

Best-First Search

- Agora que temos uma heurística, podemos utilizá-la diretamente na busca pelo objetivo
- A busca pelo “melhor primeiro” (*best-first search*) escolhe o nó não visitado com a melhor avaliação heurística como o próximo a ser visitado
- Pode ser implementada utilizando o mesmo algoritmo da busca em largura com menor custo (*lowest-cost Breadth First Search*)
- Entretanto, desta vez a prioridade de cada nó adicionado a fila é o valor da função heurística.

Estude o mapa da Romênia. Note as distancias entre as cidades e tente calcular a menor rota entre **Arad** e **Bucharest**. Agora pressione uma tecla para ver a rota ótima marcada em vermelho.

A rota ótima entre as duas cidades nos leva através de Sibiu, Rimnicu e Pitesti.

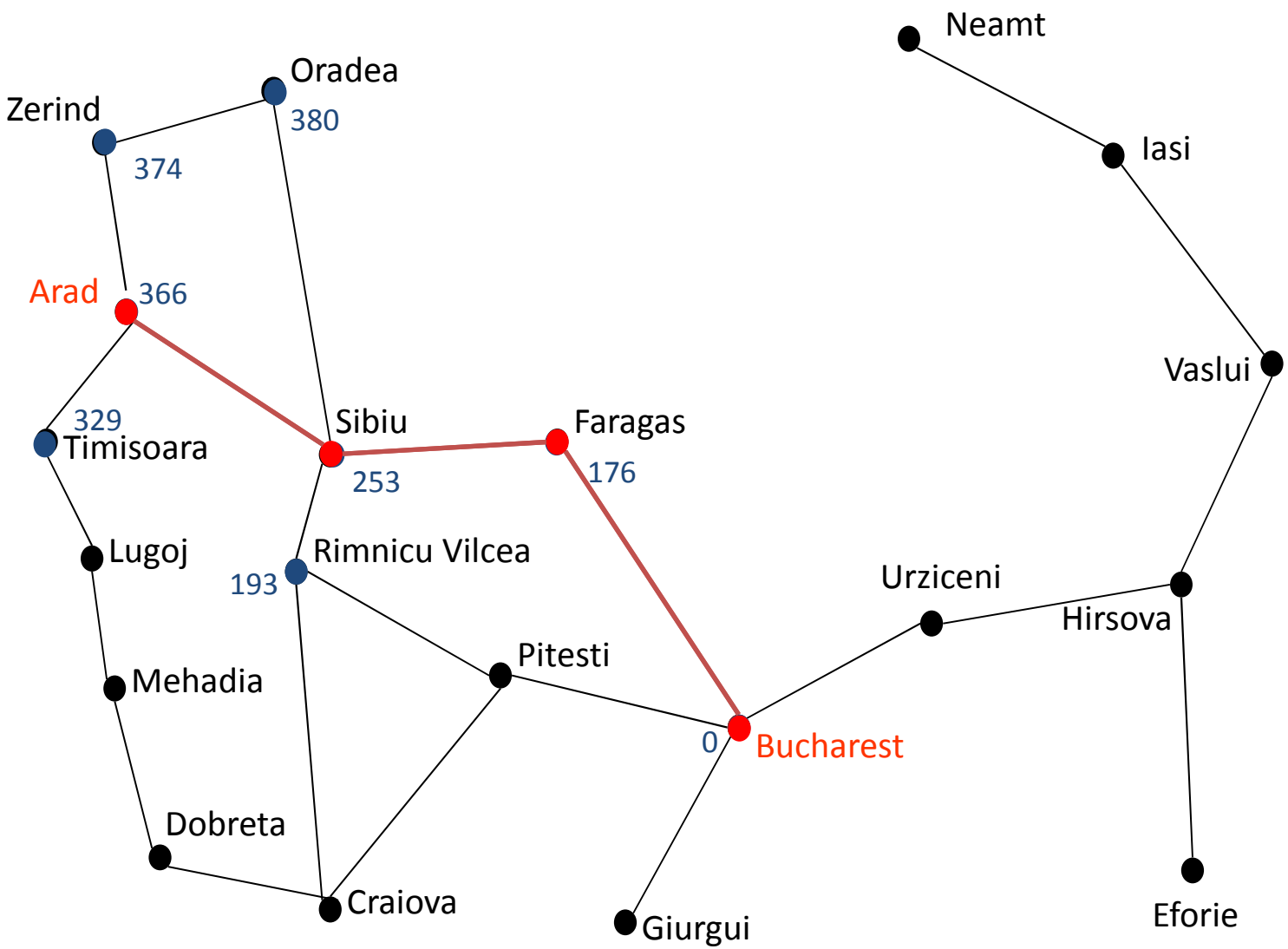


Busca Gulosa

- A ideia é selecionar o caminho no qual o último nó está mais perto do objetivo, de acordo com a função heurística.
- Seleciona o caminho na fronteira com menor valor de $h(n)$
- Trata a fronteira como uma fila de prioridade ordenada por h

- Função heurística para o exemplo: distância aérea até Bucharest

Busca Gulosa



Town	Air Dist.
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

Análise: Busca Gulosa

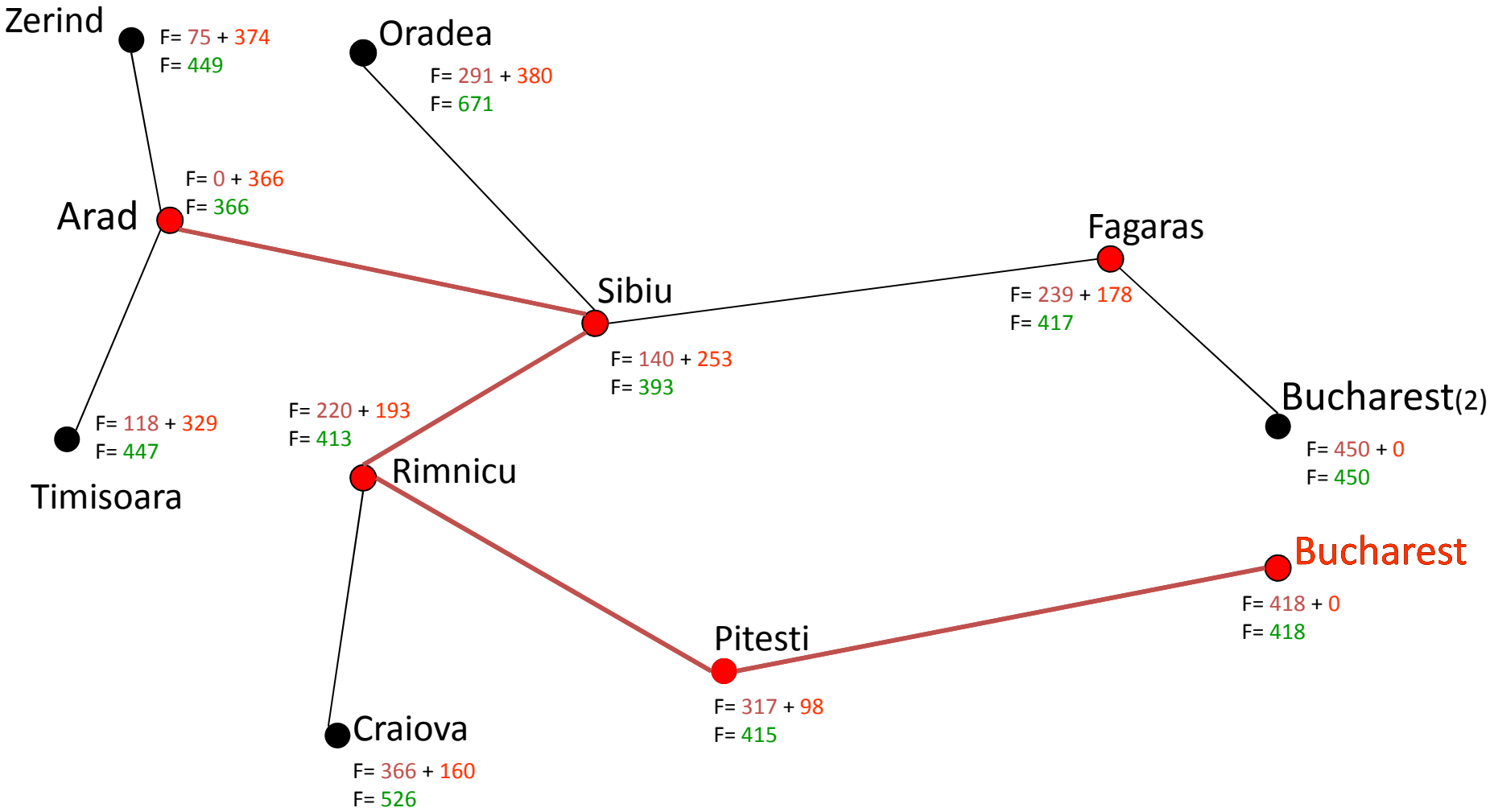
- Características:
  - Similar à Busca em Profundidade porque vai sempre na mesma direção num caminho da árvore para procurar a solução
  - Qualidade de *h* e tipo de problema podem ajudar a diminuir complexidades temporal e espacial
  - Na presença de "*dead-ends*" pode ter que escolher caminho de maior custo
- Completa? Não, pois por padrão não verifica nós repetidos no caminho
- Ótima? Não
- Complexidade temporal: $O(b^m)$
- Complexidade espacial: $O(b^m)$

A*

- Estratégia: Minimização do custo total do caminho
- Função de Avaliação: combinação de $h(n)$ com $g(n)$

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

  - $g(n)$ = custo desde o nó inicial até n
  - $h(n)$ = estimativa do custo deste n até um estado objetivo
- É completa e ótima com uma restrição na função h: nunca superestimar o custo real da melhor solução
  - neste caso, h é dita admissível
  - se h é admissível, $f(n)$ também é admissível.

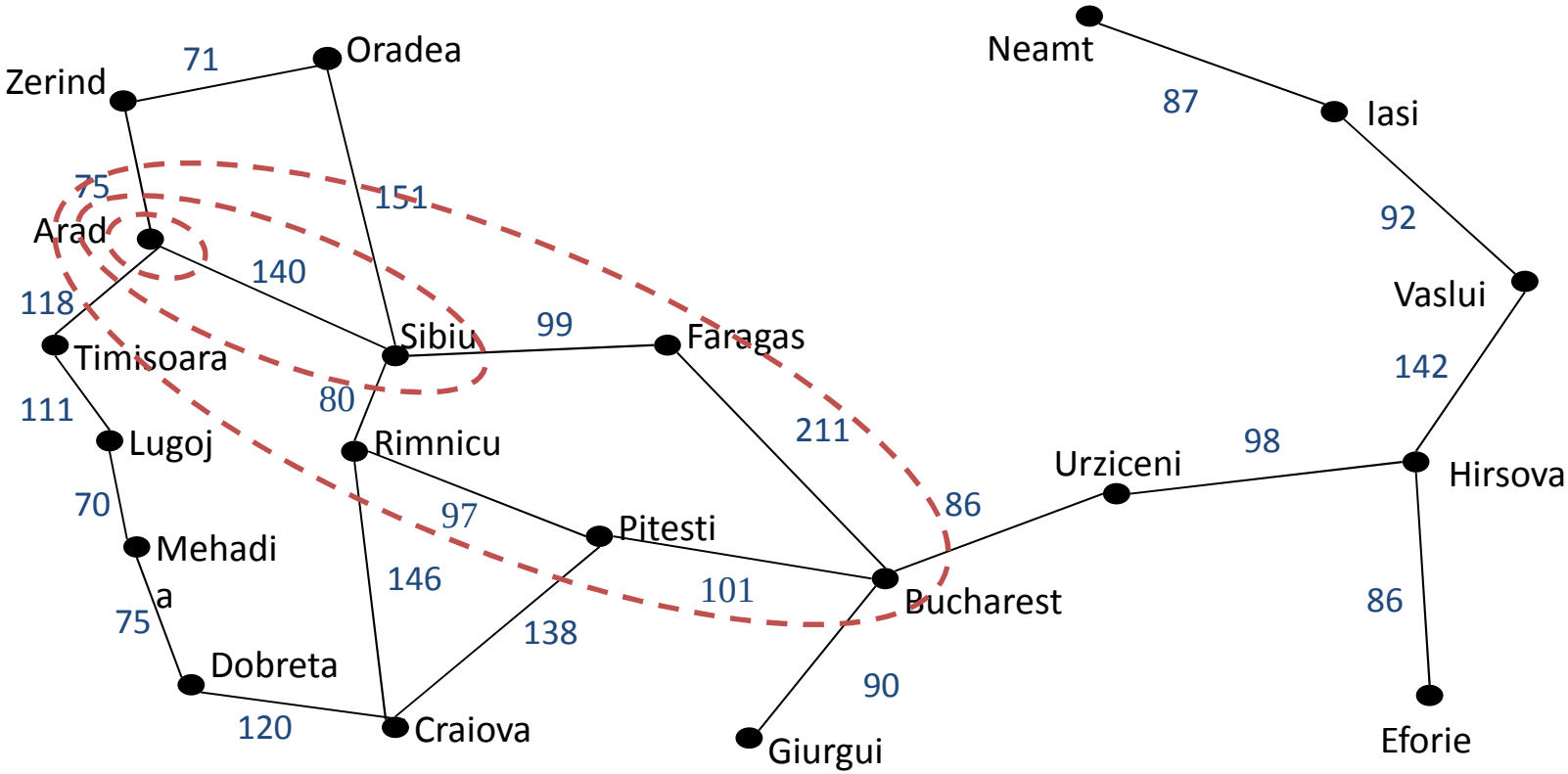

Zerind
F= 75 + 374
F= 449
Oradea
F= 291 + 380
F= 671
Arad
F= 0 + 366
F= 366
Fagaras
F= 239 + 178
F= 417
Sibiu
F= 140 + 253
F= 393
Bucharest(2)
F= 450 + 0
F= 450
F= 118 + 329
F= 447
Timisoara
F= 220 + 193
F= 413
Rimnicu
Bucharest
F= 418 + 0
F= 418
Pitesti
F= 317 + 98
F= 415
Craiova
F= 366 + 160
F= 526

Análise: A*

- Completa? Sim, somente para grafos com fator de ramificação finito
- Óptima? Sim
- Complexidade de Tempo? Exponencial
- Complexidade de Espaço? Exponencial, pois mantém todos os nós em memória (no pior caso).
- A* vs Outros:
  - **Busca gulosa** diminui o custo estimado para atingir a solução, $h(n)$, mas não é completa nem ótima
  - **Busca de custo uniforme** minimiza o custo do caminho da raiz até o nó corrente, $g(n)$. É ótima e completa, mas muito ineficiente

A* é Ótima

- A* expande os nós por ordem crescente do valor de f
- Gradualmente adiciona contornos "curvas de nível" (à semelhança dos mapas topográficos) que identificam conjuntos de nós.
- Contorno *i* tem todos os nós com $f \leq f_i$, com $f_i < f_{i+1}$



Análise: A*

- Maior problema é a complexidade espacial:
  - número de nós expandidos para chegar a um estado final cresce exponencialmente com o tamanho da entrada
  - Na prática: crescimento exponencial
    - Entretanto, crescimento exponencial não ocorre se o erro na função heurística não crescer mais rápido do que o logaritmo do custo do caminho real: $| h(n) - h^*(n) | \leq O(log(h^*(n)))$, onde $h^*(n)$ é o custo real entre n e o estado objetivo.
- Nenhum outro algoritmo ótimo garante expandir menos nós do que o A*

Heurísticas Admissíveis

- Uma heurística $h(n)$ é admissível se para cada nó n se verifica $h(n) \leq h^*(n)$, onde $h^*(n)$ é o custo real do caminho desde n até ao objetivo.
- Uma heurística admissível nunca sobrestima o custo de atingir o objetivo, i.e. é realista ou otimista.
- Teorema: se h(n) é admissível, então a procura em árvore A* é ótima.

IDA* (*Interactive Deepening A**)

- Versão iterativa em profundidade da busca A*
- Em cada iteração é incrementado o valor limite para $f(n)$; elementos com valor de $f(n)$ superior ao limite n não é analisado.
- Em cada nova iteração o valor limite é atualizado com o menor valor de $f(n)$ para os nós não explorados na iteração anterior
- Vantagem: não requer tanto espaço como A*

IDA* (*Interactive Deepening A**)

- Problema com A*: grande uso da memória.
- A ideia é:
  - Calculamos o valor $f(n)$ do estado inicial.
  - Como a solução ótima não pode ter um custo menor, realizaremos uma busca em profundidade na qual serão expandidos somente os nodos que não ultrapassam esse limite.
  - Se, de todos os nodos visitados, nenhum é a solução, recomeçamos uma nova busca, aumentando o limite.
    - E qual será o melhor valor para o novo limite? Considere os nodos que não foram expandidos. Um deles minimiza o valor $f(n)$. Como a solução ótima não pode ter um valor menor, recomeçaremos com esse novo valor. Assim por diante até que o estado final seja visitado.

RBSF (*Recursive Best-First Search*)

- A ideia consiste em memorizar para cada nodo *n* o menor valor entre o seu valor *f*(*n*) e os valores de todos os seus descendentes.
- Para realizar essa atualização, aplicamos o seguinte algoritmo:
  - Seja *n* o último nodo expandido, e $n_1, n_2, ..., n_i$, os nodos resultando dessa expansão.
  - Se o menor dos valores $f(n_1), f(n_2), ..., f(n_i)$ é maior que *f*(*n*), substituímos o valor *f*(*n*) por esse valor.
  - Repetimos esse processo, comparando os valores de *n* e seus irmãos com o valor do pai de *n*. Essa propagação pára quando encontramos um pai que não tem um valor inferior ao menor de seus filhos.
  - Depois dessa atualização, todo nodo tem um limite inferior do custo total.
- A busca tenta efetuar a expansão em profundidade. Quando existe um nodo que apresenta um valor mínimo e que não é no nível mais profundo da busca, apagamos todos os nodos que estão em um nível de profundidade maior que esse nodo mínimo, e recomeçamos a expansão a partir desse nodo.
- Apresenta um uso de memória semelhante ao da busca em profundidade.

Resumo

- A procura pela solução de um problema pode ser formulado como um espaço de possibilidades e uma busca por uma solução ou objetivo.
- Busca cega: não consideram informações adicionais
  - Busca em Profundidade (*Deep-First Search - DFS*)
  - Profundidade interativa (*Iterative Deepening*)
  - Busca em Largura (*Breadth-First Search - BFS*)
  - Busca de Custo Uniforme (*Lowest-cost BFS*)
- Busca Heurística: escolha do melhor primeiro (ou *best-first search*)
  - Busca Gulosa (*Greedy Search*)
  - A* (*A-Star Search*)
  - IDA* (*Interative Deepening A**)
  - RBSF (*Recursive Best-First Search*)