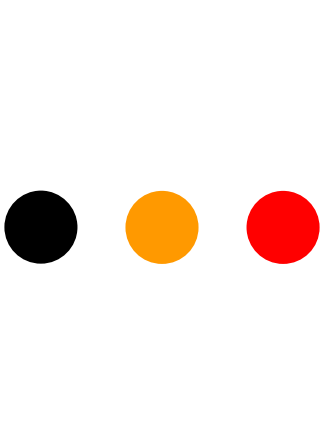


Busca Heurística

Profa. Josiane M. P. Ferreira

Texto base:

Stuart Russel e Peter Norving - “Inteligência Artificial”

David Poole, Alan Mackworth e Randy Goebel - “*Computational Intelligence – A logical approach*”

maio/2007

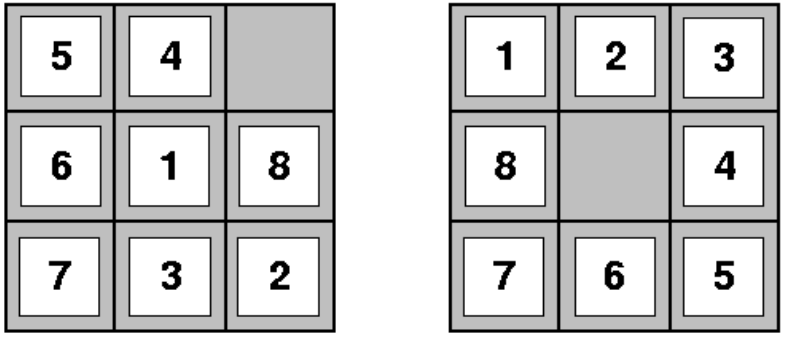
Busca heurística (busca informada)

- A idéia 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
- *h(n)* é uma estimativa do custo do caminho de um nó *n* até um nó objetivo
- *h(n)* deve ser simples de ser computada, de preferência ela deve ser derivada das propriedades do próprio nó
- *h(n)* é um subestimativa se não existe nenhum outro caminho de *n* até o objetivo que seja menor do que *h(n)*

Exemplo 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



estado inicial **estado final**

- 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

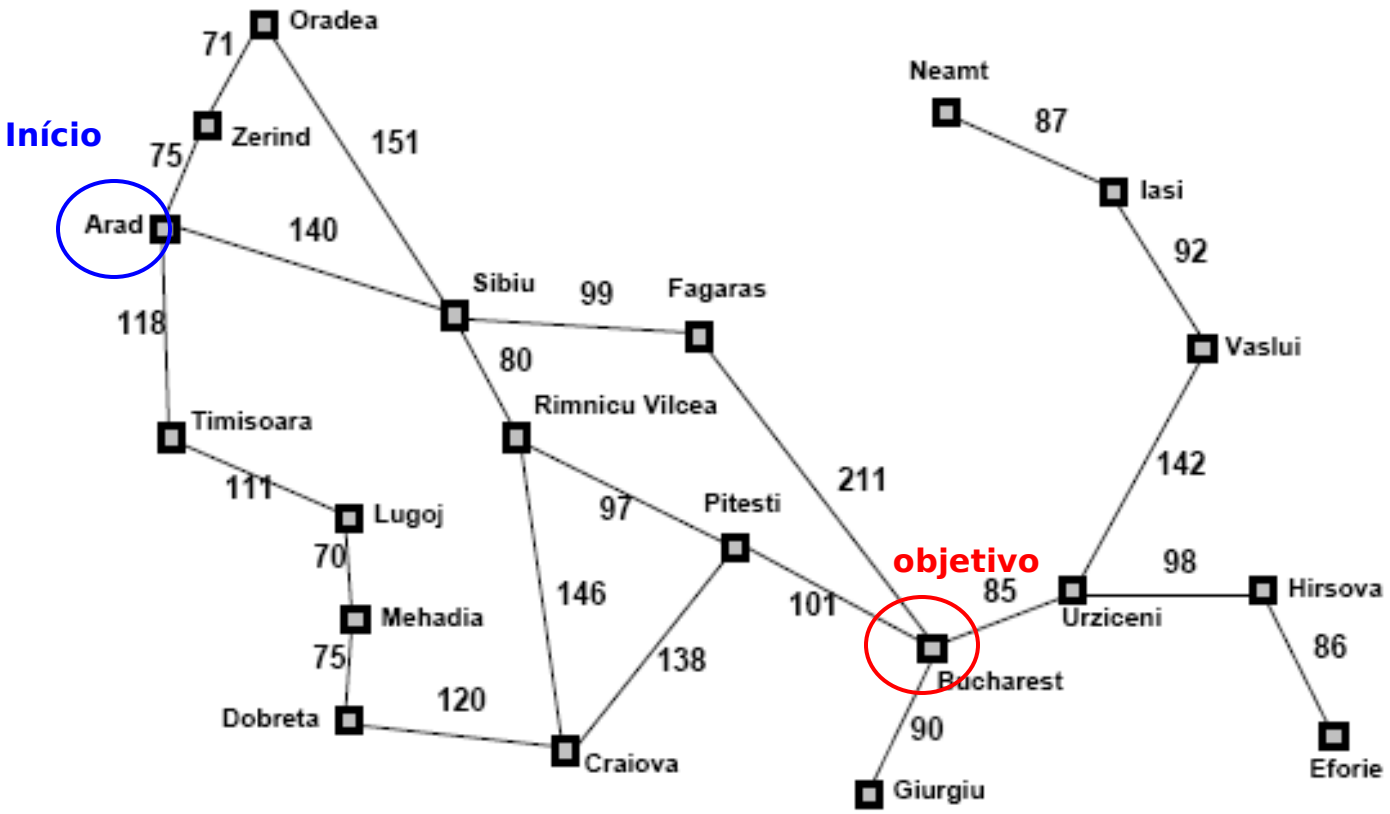
  - h1(s) = 7 (só o número 7 está no local correto)
  - h2(s) = 2+3+3+2+4+2+0+2 = 18

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

Busca pelo melhor primeiro (gulosa)

- A idéia 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*

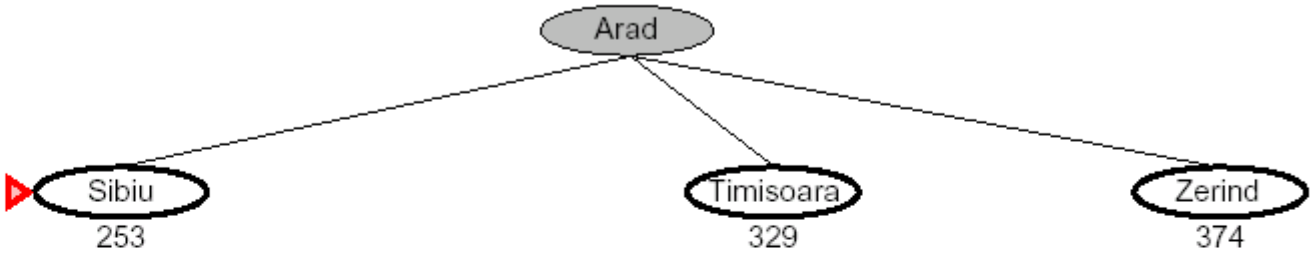
Problema: busca de caminho



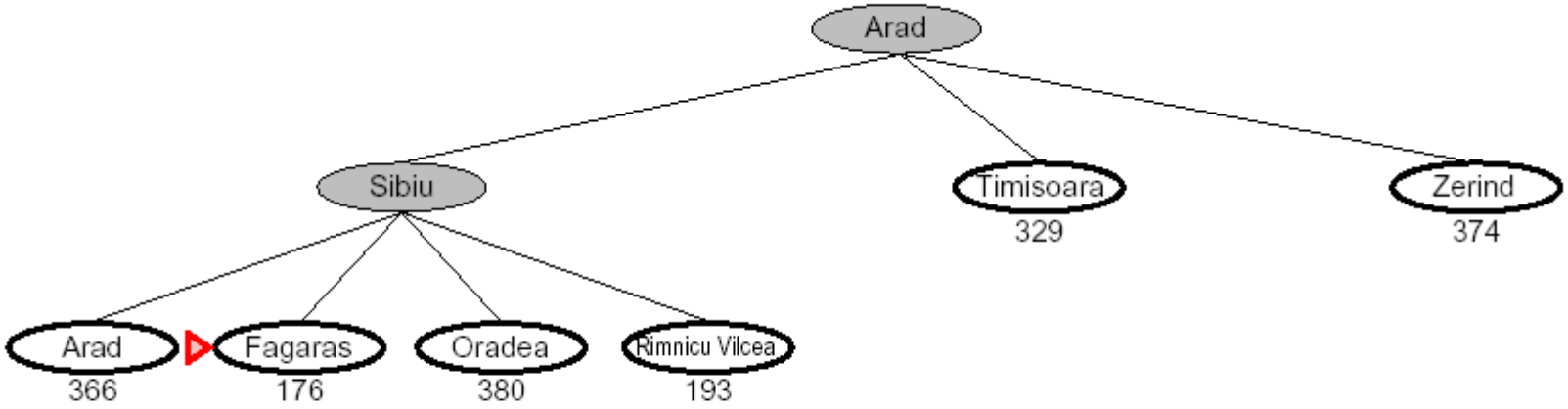
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Exemplo de busca pelo melhor primeiro (gulosa)

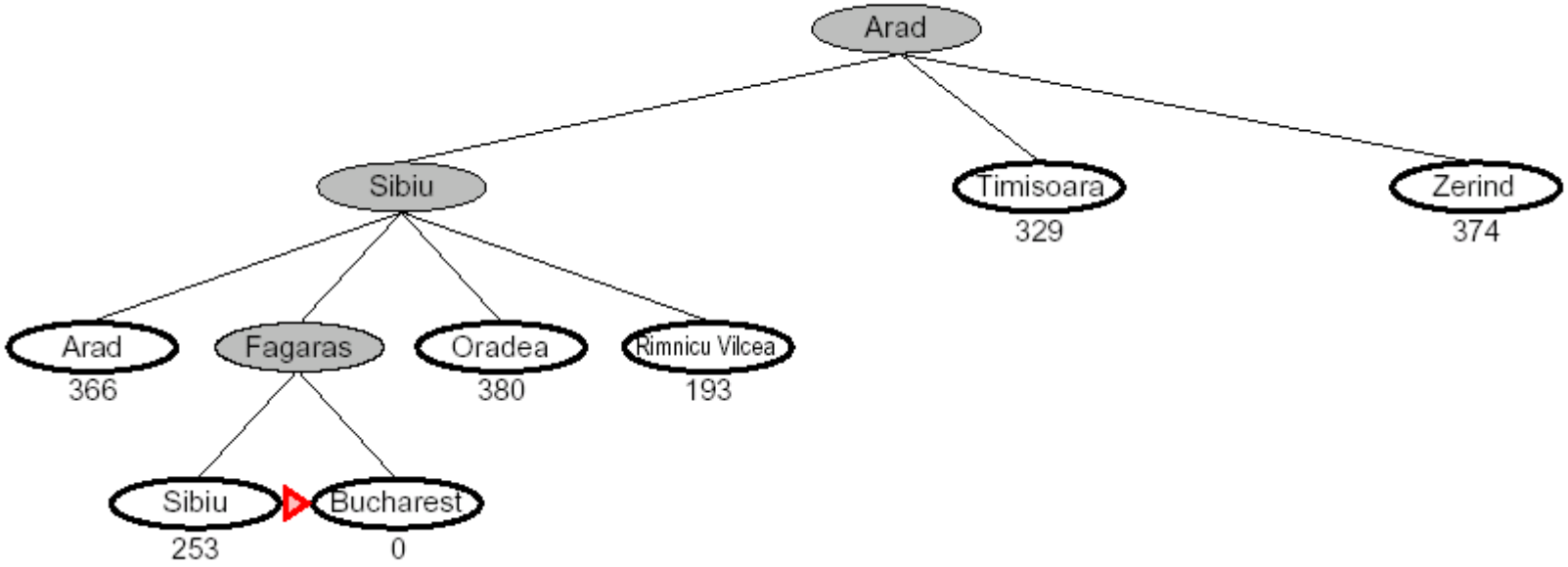
Exemplo de busca pelo melhor primeiro (gulosa)



Exemplo de busca pelo melhor primeiro (gulosa)



Exemplo de busca pelo melhor primeiro (gulosa)



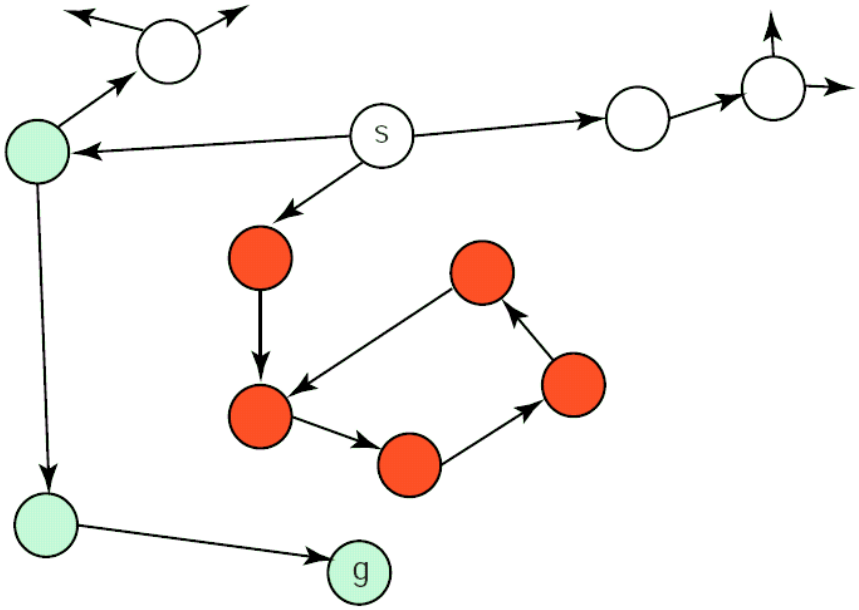
O solução encontrada não é a solução ótima.
A Solução ótima passa por Rimnicu Vilcea e Pitest e tem 32 km a menos.

Busca pelo melhor primeiro (gulosa)

```
% Usa o algoritmo genérico, mas trata a fronteira como
% uma fila de prioridade ordenada por h,
% selecionando o primeiro elemento da fila

selecione(No, [No|Fronteira], Fronteira).

adicione_a_fronteira(Vizinhos, Fronteira1, Fronteira2) :-
    append(Fronteira1, Vizinhos, Fronteira2),
    ordene_por_h(Fronteira2, Fronteira3).

% É melhor implementar a inserção na fila como uma
   inserção em um heap

busca_menor_custo(Fronteira3).
```

Complexidade da Busca pelo melhor primeiro

- Usa espaço exponencial no tamanho do caminho (como a busca em largura)
- Não garante encontrar a solução mesmo se ela existir
  - Pode ficar em um laço (exemplo ao lado)
- Quando ela encontra a solução, não pode garantir que é a melhor solução (caminho mais curto)



Busca Heurística em profundidade primeiro

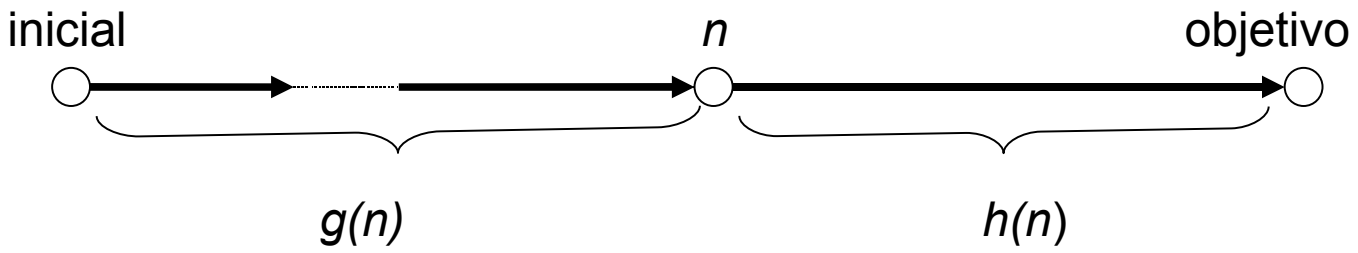
- É uma forma de utilizar o conhecimento heurístico na busca em profundidade
- Os nós vizinhos (nós gerados) são ordenados primeiro para depois serem colocados na fronteira (que ainda é uma pilha)
- Seleciona localmente qual subarvore expandir, mas continua fazendo busca em profundidade
- A complexidade de espaço é linear no tamanho do caminho
- Mas não garante encontrar uma solução (como a busca em profundidade)

Busca Heurística em profundidade primeiro

```
% Usa o algoritmo genérico, mas trata a fronteira como
% uma pilha, os vizinhos são ordenados antes de serem
% colocados na pilha

selecione(No, [No|Fronteira], Fronteira).

adicione_a_fronteira(Vizinhos, Fronteira1, Fronteira2) :-
    ordene_por_h(Vizinhos, VizinhosOrdenados).
    append(VizinhosOrdenados, Fronteira1, Fronteira2),
```

Busca A*

- Usa tanto o custo do caminho (*g(n)*) quanto o valor da heurística (*h(n)*)
- *g(n)* – custo do caminho do nó inicial até o nó *n*
- *h(n)* – valor da heurística do nó n até um nó objetivo
- *f(n)* = *g(n)* + *h(n).* Estima o custo total do caminho do nó inicial até o nó objetivo passando por *n*.



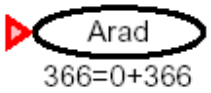
Busca A*

- Combina:
  - busca pelo menor custo primeiro (baseada em *g(n)*)
  - busca pelo melhor primeiro (baseada em *h(n)*)
- A fronteira é organizada como uma fila de prioridade ordenada por *f(n)*
- Sempre seleciona o nó da fronteira com menor custo estimado da distância entre o nó inicial e o nó objetivo passando por aquele nó

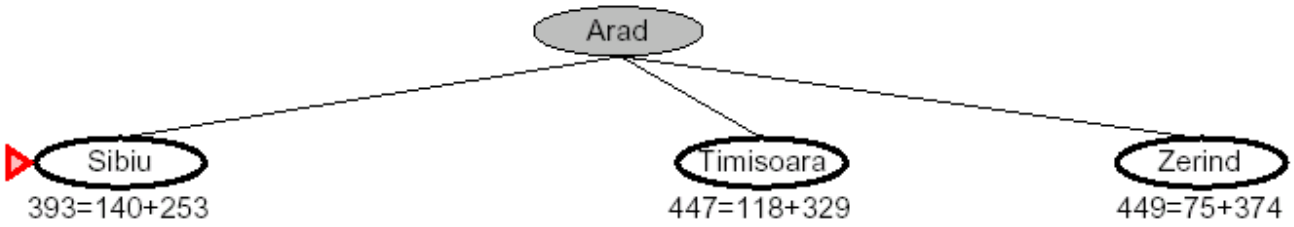
Exemplo: busca do caminho mínimo

- Problema:
  - Ir de **Arad → Bucharest**.
- Função heurística:
  - **Distância em linha reta** entre a cidade *n* e Bucharest.
  - Satisfaz a condição de admissibilidade, pois não existe distância menor entre dois pontos do que uma reta.
  - É uma boa heurística, pois induz o algoritmo a atingir o objetivo mais rapidamente.

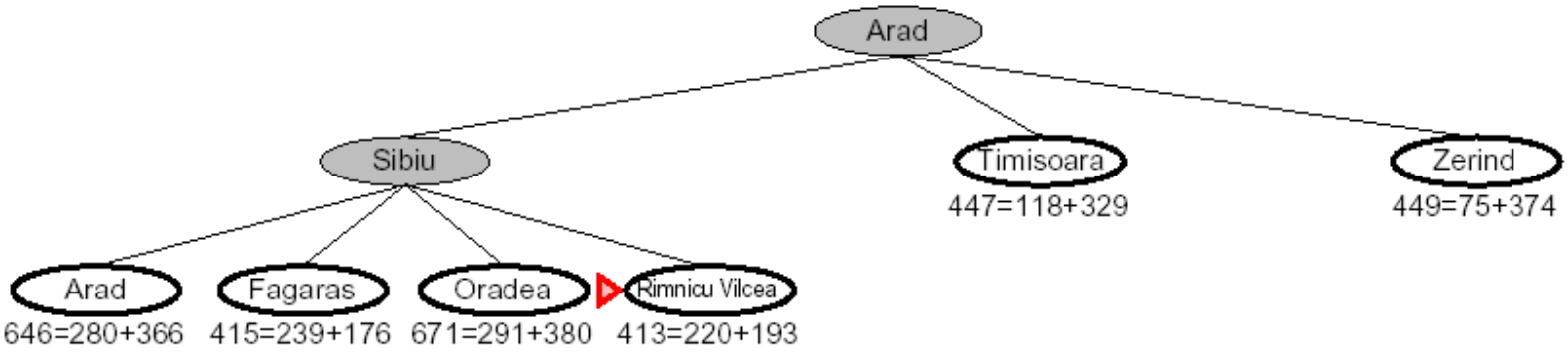
Exemplo de Busca A*



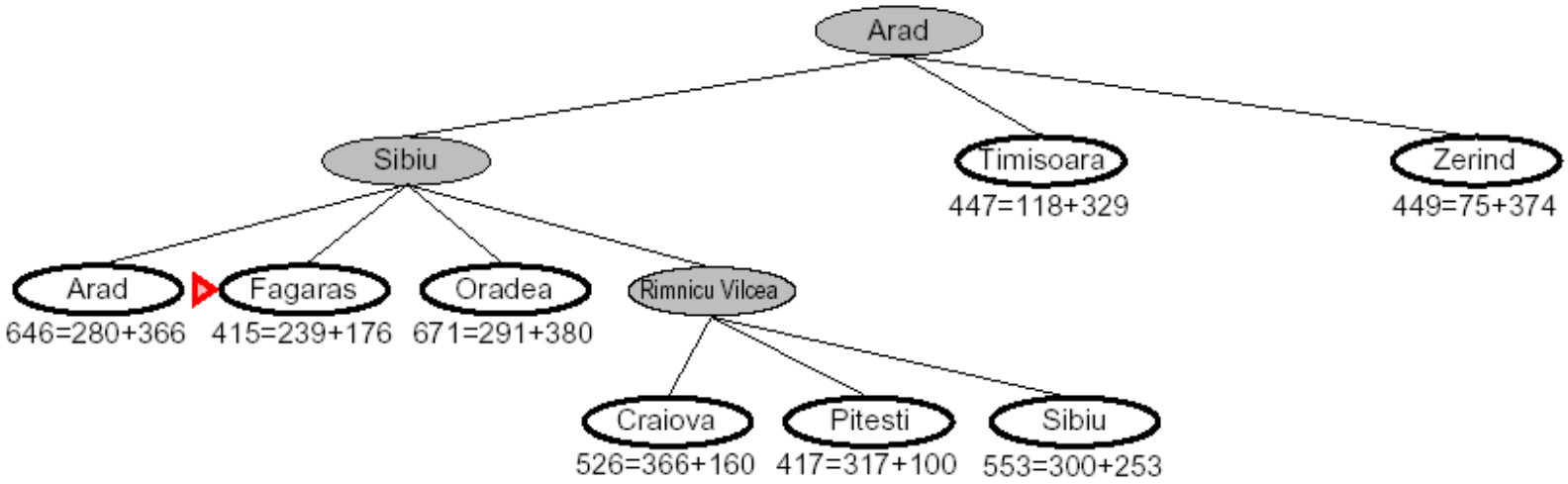
Exemplo de Busca A*



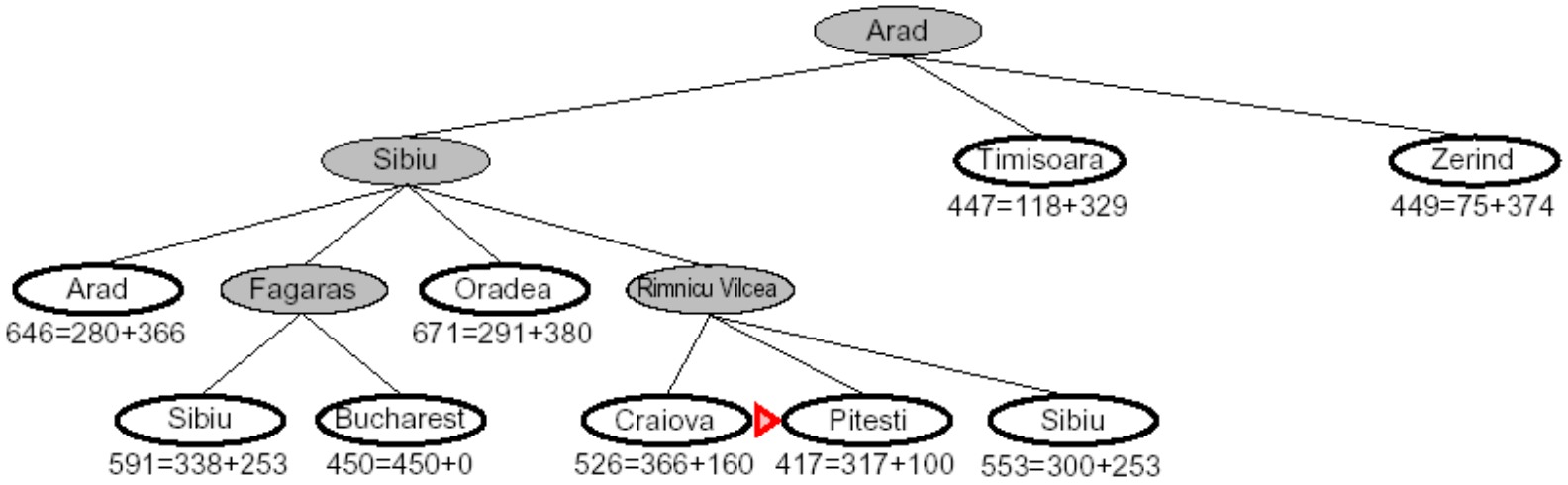
Exemplo de Busca A*



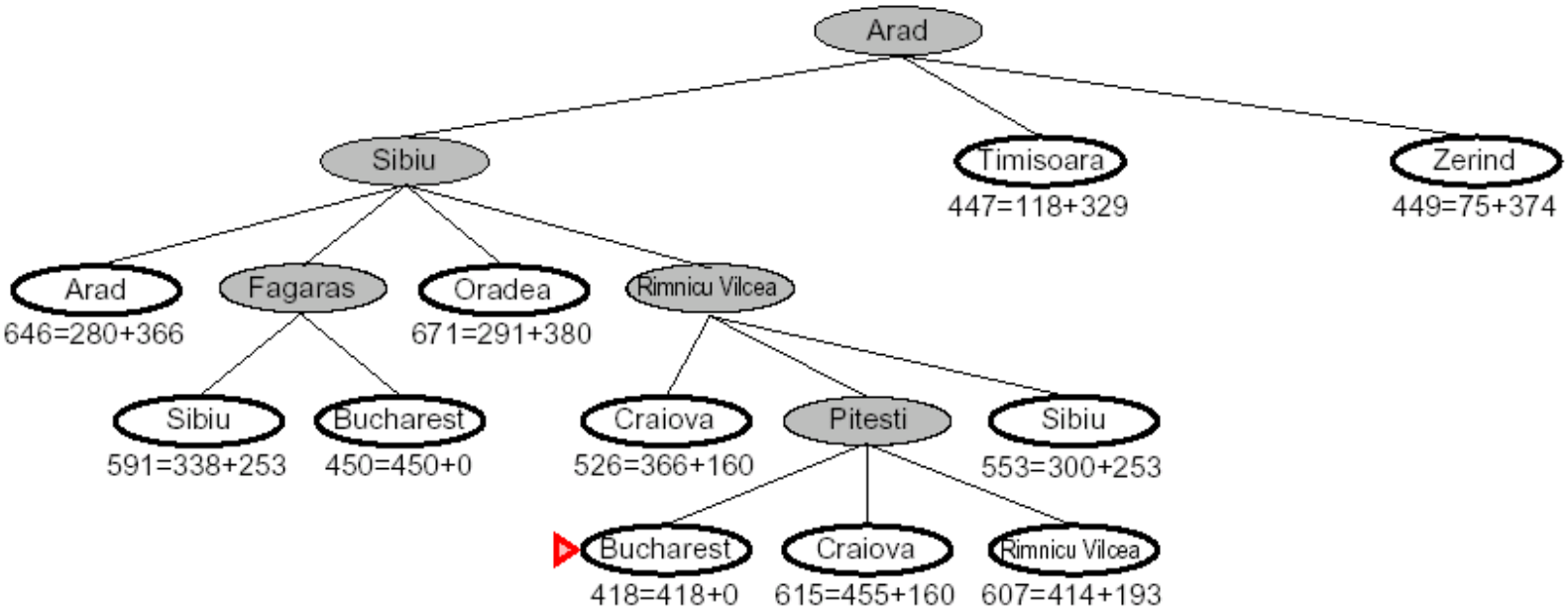
Exemplo de Busca A*



Exemplo de Busca A*



Exemplo de Busca A*



Busca A*

```
% Usa o algoritmo genérico, mas trata a fronteira como
% uma filha de prioridade ordenada por f(n)

selecione(No, [No|Fronteira], Fronteira).

adicione_a_fronteira(Vizinhos, Fronteira1, Fronteira3) :-
    append(Fronteira1, Vizinhos, Fronteira2),
    ordene_por_f(Fronteira2, Fronteira3).
```

Admissibilidade de A*

- Se existe uma solução, A* sempre encontra a solução ótima se:
  - O fator de ramificação for finito
  - O custo do caminho nunca decresce (os arcos não possuem custos negativos)
  - h(n) nunca superestima o custo do caminho
    - h(n) <= o custo real do caminho (heurística admissível)