A*

Busca com informação (heurística)

- ➔ Conhecimento específico do problema.
- ➔ Definição do próprio problema.

Pode encontrar soluções de forma mais eficiente que uma estratégia sem informação.

Busca pela melhor escolha

- Busca em árvore
- Busca em grafo
- Como funciona:
  - Nó é escolhido para expansão com base em f(n).
  - Normalmente o nó com a avaliação mais baixa é selecionada.

Se f(n) for inacurada, poderá levar a perda da busca.

Função Heurística - h(n)

- ➔ Componente fundamental desses algoritmos.
- ➔ h(n) = Custo estimado do caminho mais econômico do nó “n” até um nó objetivo.
- ➔ Forma mais comum de aplicar conhecimento adicional ao algoritmo de busca.
- ➔ Ex: A heurística em um problema de localização pode ser definida como uma linha reta de uma cidade para outra.

Busca gulosa pela melhor escolha

- ➔ Tenta expandir o nó mais próximo da meta, na suposição de que isso provavelmente levará a uma solução rápida.
- ➔ Avalia nós usando apenas a função heurística, f(n) = h(n).
- ➔ Custo mínimo, porém não é ótima.
- ➔ Semelhante à busca em profundidade.

Busca A*

- ➔ minimizando o custo total estimado da solução.
- ➔ f(n) = g(n)+h(n)
  - ◆ g(n) = custo do caminho desde o nó inicial até o nó “n”.
  - ◆ h(n) = custo estimado do caminho de custo mais baixo desde “n” até o objetivo.

f(n) = custo estimado da solução de custo mais baixo passando por “n”.

Obs: g(n) = 0 → Busca gulosa.

h(n) = 0 → Busca não informada.

Algoritmo: Começando a procura

- ➔ Começa do ponto A, conferindo os nós adjacentes
  - Acrescenta o ponto A a uma lista aberta de nós.
  - verifica todos os nós alcançados
  - Remove o ponto A da lista aberta e o coloca em uma lista fechada, que você não precisa procurar novamente.
  - Se escolhe o nó com menor valor de F da lista aberta.

Continuando a procura:

- ➔ A escolha do próximo nó
  - ◆ Retire-o da lista aberta e o coloque na lista fechada
  - ◆ Conferir os nós adjacentes (ignorando os da lista fechada)
    - Se eles não estiverem na lista aberta, acrescente-os.
    - Se o nó já estiver na lista aberta, confira o valor de seu caminho
  - ◆ Se o custo G do novo caminho é menor, troque o pai do nó selecionado

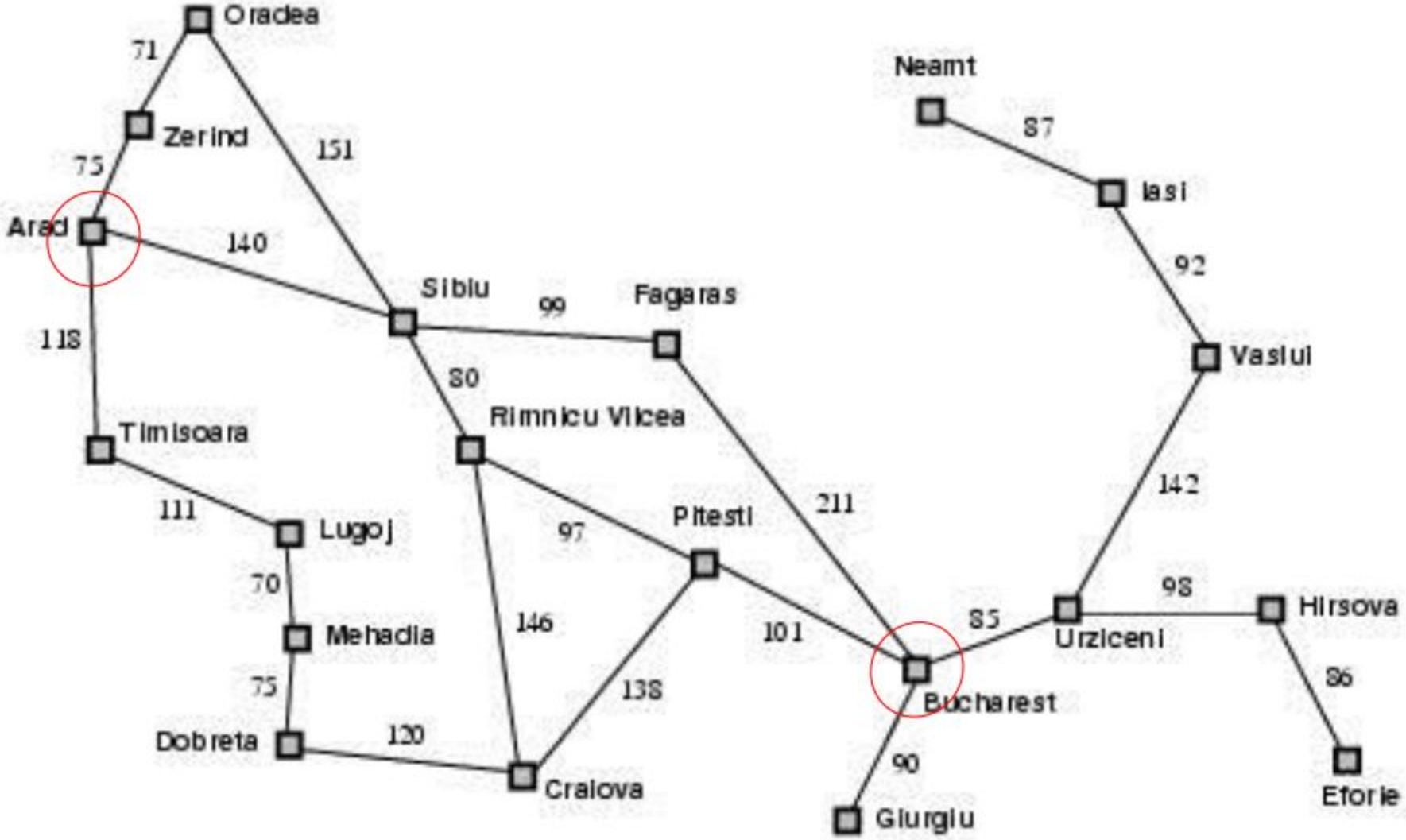
RECURSIVIDADE ATÉ ACHAR O PONTO B DESEJADO

A*

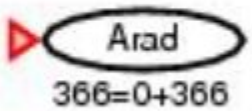
```
ABERTA                                  // Lista de nós que serão analisados
FECHADA                                 // Lista de nós que já foram analisados
Adicione o nó INICIO na lista ABERTA

Repita {
      ATUAL = nó da lista ABERTA com menor F
      remova ATUAL da lista ABERTA
      insira ATUAL na lista FECHADA

      se ( ATUAL == FIM )               // Encontramos um caminho
            retorne;
      se ( lista ABERTA estiver vazia)  // NÃO encontramos um caminho
            retorne;

      para cada VIZINHO de ATUAL
            se (VIZINHO é um bloqueio OU VIZINHO está na lista FECHADA)
                  passe para o próximo VIZINHO

            se (VIZINHO não está na lista ABERTA)
                  defina ATUAL como nó pai de VIZINHO
                  calcule G, H e F de VIZINHO
                  insira VIZINHO na lista ABERTA
            senão se ( VIZINHO está na lista ABERTA E passar por ATUAL produz um valor G
            menor)
                  recalcule G e F de VIZINHO
                  defina ATUAL como nó pai de VIZINHO
}
```

Exemplo: Distância entre as cidades

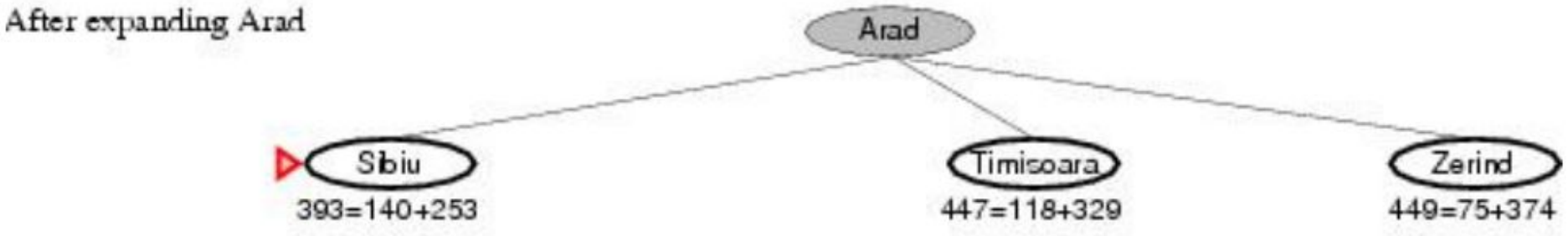
- h*(n) = é o caminho mais curto de uma cidade para outra em linha reta.
- h(n) = é o caminho pela estrada de uma cidade a outra.
  - h(n)>=h(n)*, assim uma heurística admissível.


Oradea
71
Neamt
Zerind
87
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vasiui
80
Rimnicu Vilcea
Timisoara
142
111
211
Pitesti
Lugoj
97
98
70
Hirsova
146
85
Mehadia
101
Urziceni
75
86
138
Bucharest
Dobreta
120
90
Eforie
Craiova
Giurgiu

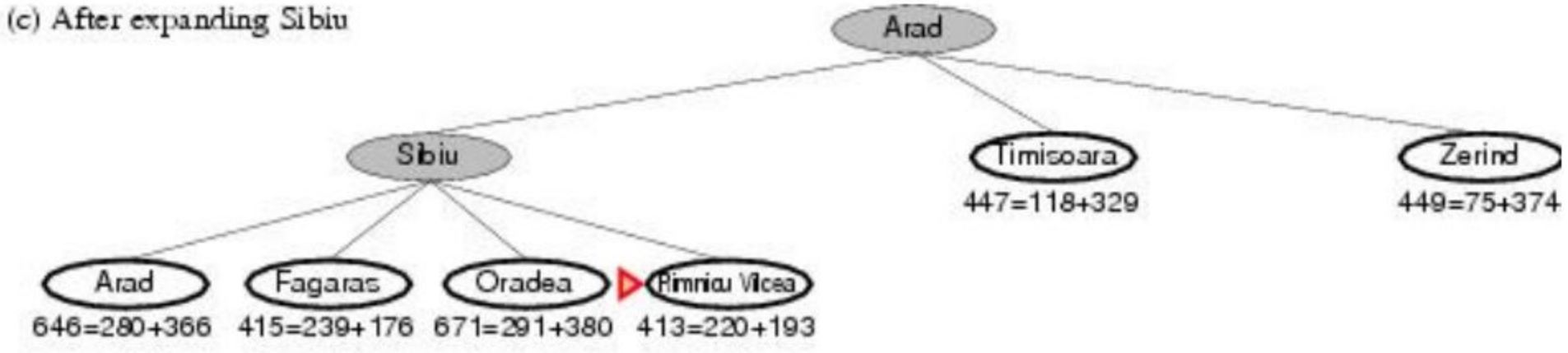
(a) The initial state



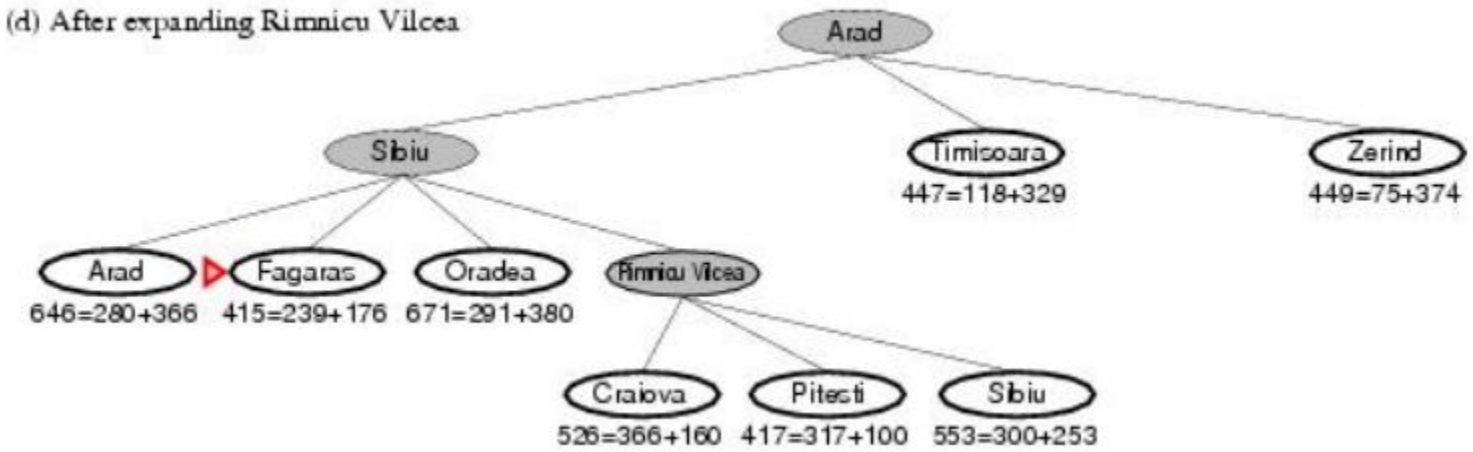
- ➔ Encontrar Bucareste partindo de Arad:
  - f(arad)= 0 + h(arad)= 0 + 366 = 366



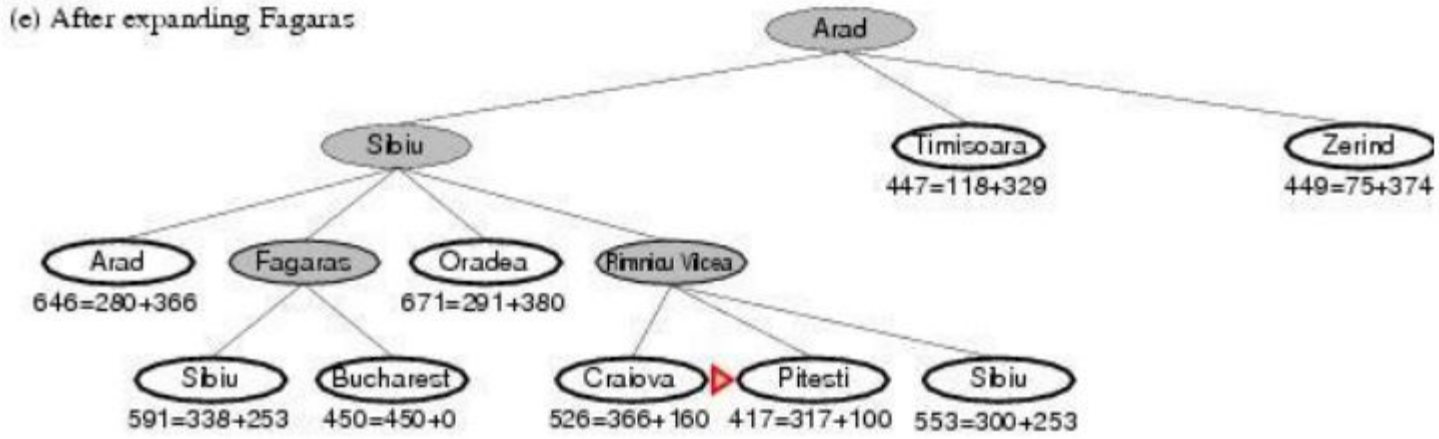
- ➔ Expandindo Arad e determinando f(n) para cada nó:
  - f(Sibiu) = c(Arad,Sibiu) + h(sibiu) = 140 + 253 = 393
  - f(Timisoara) = c(Arad,Timisoara) + h(Timisoara) = 118 + 329 = 447
  - f(Zerind) = c(Arad,Zerind) + h(Zerind) = 75 + 374 = 449
- ➔ Melhor escolha é Sibiu



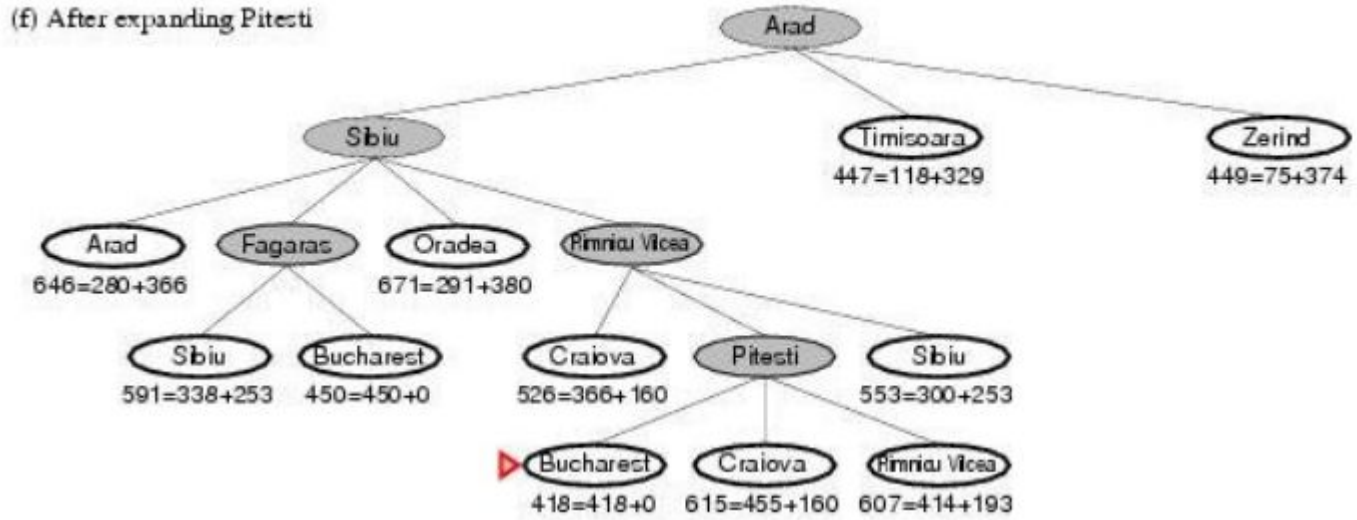
- ➔ Expandindo Sibiu e determinado f(n) para cada nodo:
  - f(Arad) = g(Arad) + h(Arad) = 280 + 366 = 646
  - f(Fagaras) = g(Fagaras) + h(Fagaras) = 239 + 176 = 415
  - f(Oradea) = g(Oradea) + h(Oradea) = 291 + 380 = 671
  - f(Riminicu Vilcea) = g(Rimnicu Vilcea) + h(Rimnicu Vilcea) = 220 + 193 = 413
- ➔ Melhor escolha é Rimnicu Vilcea



- ➔ Expandindo Rimnicu Vilcea:
  - f(Craiova) = g(Craiova) + h(Craiova) = 366 + 160 = 526
  - f(Pitesti) = g(Pitesti) + h(Pitesti) = 317 + 100 = 417
  - f(Sibiu) = g(Sibiu) + h(Sibiu) = 300 + 253 = 553
- ➔ Melhor escolha é Fagaras.



- ➔ Expandindo Fagaras:
  - f(Sibiu) = g(Sibiu) + h(Sibiu) = 338 +253 = 591
  - f(Bucareste) = g(Bucareste) + h(Bucareste) = 450 + 0 = 450
- ➔ Melhor escolha é Pitesti



- ➔ Expandindo Pitesti:
  - f(Bucareste) = g(Bucareste) + h(Bucareste) = 418 + 0 = 418
- ➔ Melhor escolha é Bucareste.
  - Solução ótima(dado que h(n) é admissível).

A* é:

- ➔ Completa.
- ➔ Ótima.

Obs: Isso não significa que A* seja a resposta para todas as necessidades de busca.

Problema em usar A*

- ➔ Maioria dos problemas o número de nós é exponencial.
- ➔ Não é prática para problemas de grande escala.
- ➔ Em geral A* esgota o espaço, antes de esgotar o tempo.

- Algoritmos já desenvolvidos superaram o problema do espaço sem sacrificar o caráter ótimo, a um custo pequeno de execução:
  - AIA*
  - BRPM
  - LMA*
  - LSMA*

Referências

- Livro de Inteligência Artificial, Russell e Norving, Cap 4.
- http://mat.uab.cat/~alseda/MasterOpt/AStar-Algorithm.pdf
- https://www.youtube.com/watch?v=s29WpBi2exw
- https://www.youtube.com/watch?v=aYKSuRTFndo&list=PLt_f2ildHl1ktUExfX4IQI9_uDyvASrHC
- http://www.policyalmanac.org/games/aStarTutorial_port.htm
- http://abrindoojogo.com.br/inteligencia-artificial-nos-games-–-pathfinding