# Busca Heurística (Informada)

Ricardo Prudêncio

# Relembrando....
# Algoritmo de Geração e Teste

- **Fronteira** do espaço de estados
  - Lista contendo os nós (estados) a serem expandidos
  - Inicialmente, a fronteira contém apenas o estado inicial do problema
- **Algoritmo:**
  1. Selecionar o primeiro nó (estado) da fronteira do espaço de estados;
     - se a fronteira está vazia, o algoritmo termina com falha.
  2. Testar se o nó selecionado é um estado final (objetivo):
     - se "sim", então retornar nó - a busca termina com sucesso.
  3. Gerar um novo conjunto de estados aplicando ações ao estado selecionado;
  4. Inserir os nós gerados na fronteira, de acordo com a estratégia de busca usada, e voltar para o passo (1).

# Estratégias de Busca Exaustiva (Cega)

- Encontram soluções para problemas pela geração *sistemática* de novos estados, que são comparados ao objetivo;
- São *ineficientes* na maioria dos casos:
  - utilizam apenas o *custo de caminho* do nó atual ao nó inicial (função *g*) para decidir qual o próximo nó da fronteira a ser expandido.
  - essa medida nem sempre conduz a busca na direção do objetivo.
- Como encontrar um barco perdido?
  - não podemos procurar no oceano inteiro...
  - observamos as correntes marítimas, o vento, etc...

# Estratégias Busca Heurística (Informada)

- Utilizam conhecimento específico do problema na escolha do próximo nó a ser expandido
  - barco perdido
    - correntes marítimas, vento, etc...
- Aplicam de uma função de avaliação a cada nó na fronteira do espaço de estados
  - essa função estima o custo de caminho do nó atual até o objetivo mais próximo utilizando uma função heurística.
  - Função heurística
    - estima o custo do caminho mais barato do estado atual até o estado final mais próximo.

# Busca Heurística

- Busca pela melhor escolha
  - Busca genérica onde o nó de menor custo "aparente" na fronteira do espaço de estados é expandido primeiro
- Duas abordagens básicas:
  - 1. Busca Gulosa (Greedy search)
  - 2. Algoritmo A*

# Busca Heurística
## Algoritmo geral

- Função-Insere
  - insere novos nós na fronteira ordenados com base na Função-Avaliação
    - Que está baseada na função heurística

função Busca-Melhor-Escolha (*problema*, *Função-Avaliação*)

retorna **uma solução**

Busca-Genérica (*problema, Função-Insere*)

# Busca Gulosa

- Semelhante à busca em profundidade com *backtracking*
- Tenta expandir o nó mais próximo do nó final com base na estimativa feita pela função heurística *h*
- *Função-Avaliação*
  - função heurística *h*

# Funções Heurísticas

- Função heurística - *h*
  - estima o custo do caminho mais barato do estado atual até o estado final mais próximo.
- Funções heurísticas são específicas para cada problema
- Exemplo:
  - encontrar a rota mais barata de Recife a Juazeiro
  - $h_{dd}(n)$ = distância direta entre o nó *n* e o nó final.

# Funções Heurísticas

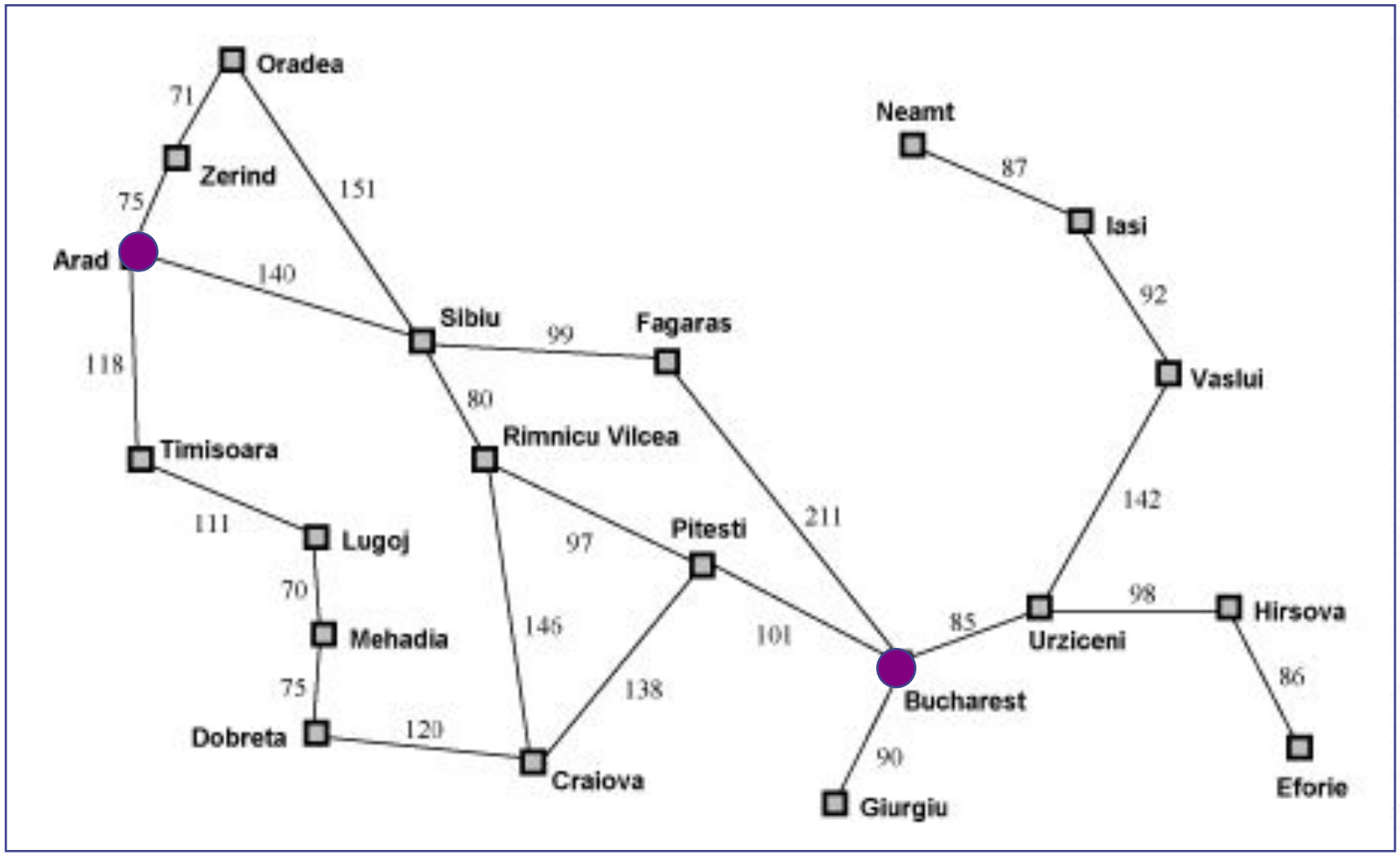
- Como escolher uma boa função heurística?
  - ela deve ser admissível
  - i.e., nunca *superestimar* o custo real da solução
- Distância direta ($h_{dd}$) é *admissível* porque o caminho mais curto entre dois pontos é sempre uma linha reta

# Busca Gulosa

- Geralmente é uma busca muito rápida
- Porém *não* é ótima:
  - escolhe o caminho que é mais econômico à primeira vista desconsiderando o custo já percorrido desde o estado inicial
- Não é completa:
  - pode entrar em *looping* se não detectar a expansão de estados repetidos
  - pode tentar desenvolver um caminho infinito
- Custo de tempo e memória: $O(b^d)$
  - guarda todos os nós expandidos na memória

# Busca Gulosa: ver exemplo

## Viajar de Arad a Bucharest




| 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 |

# Algoritmo A*

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

# Algoritmo A*

- Se *h* é *admissível*, então *f (n)* é admissível também
  - i.e., *f* nunca irá superestimar o custo real da melhor solução através de *n*
  - pois *g* guarda o valor exato do caminho já percorrido.

# Algoritmo A*
## Análise do comportamento

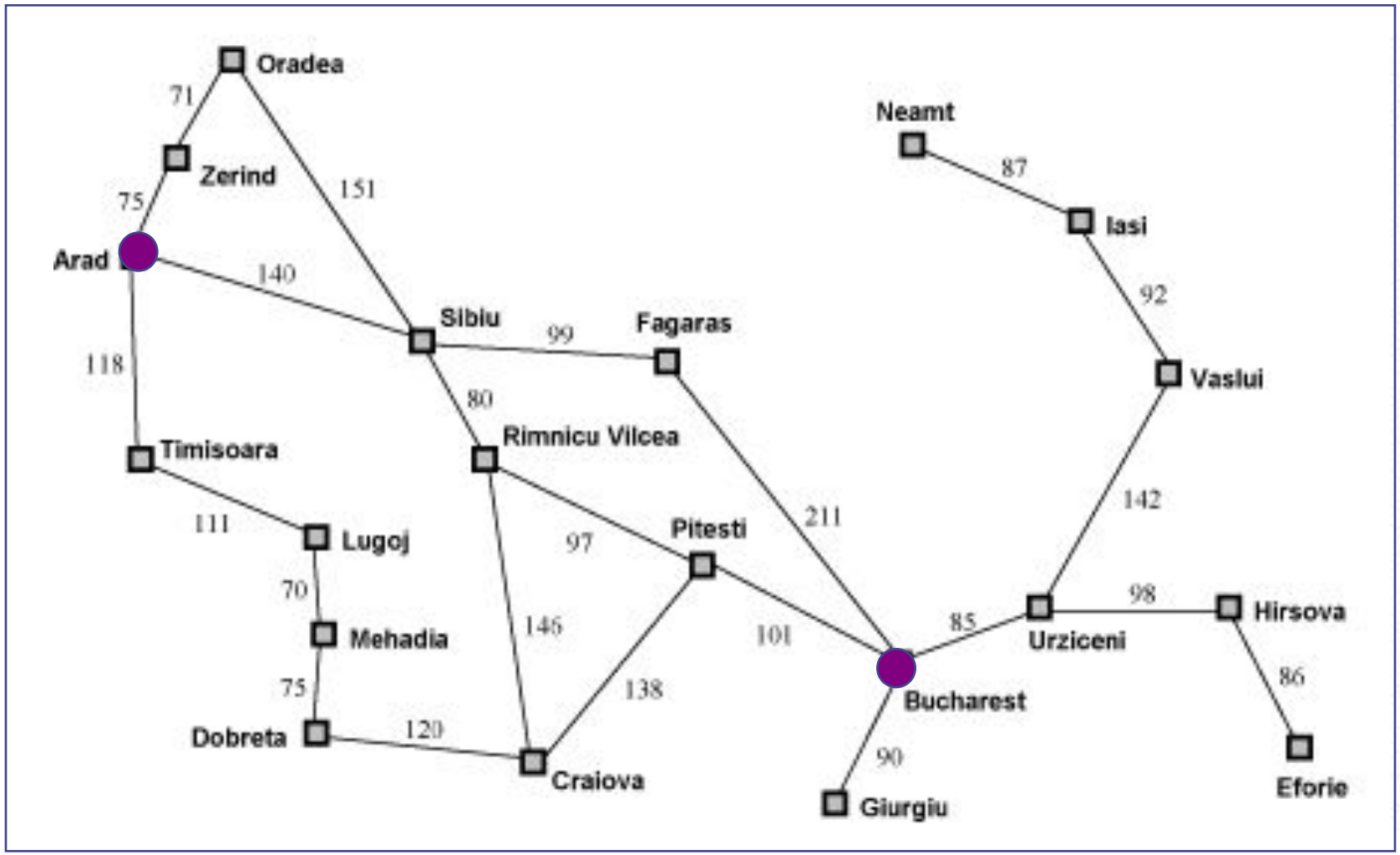
- A estratégia apresenta eficiência ótima
  - nenhum outro algoritmo ótimo garante expandir menos nós
- A* só expande nós com $f(n) \leq C^*$, onde $C^*$ é o custo do caminho ótimo
- Para se garantir otimalidade do A*, o valor de $f$ em um caminho particular deve ser não decrescente!!!
  - $f(sucessor(n)) \geq f(n)$
  - i.e., o custo de cada nó gerado no mesmo caminho nunca é menor do que o custo de seus antecessores

# Algoritmo A*:
## Análise do comportamento

- A estratégia é completa e ótima
- Custo de tempo:
  - exponencial com o comprimento da solução, porém boas funções heurísticas diminuem significativamente esse custo
    - o fator de expansão fica próximo de 1
- Custo memória: $O(b^d)$
  - guarda todos os nós expandidos na memória, para possibilitar o *backtracking*

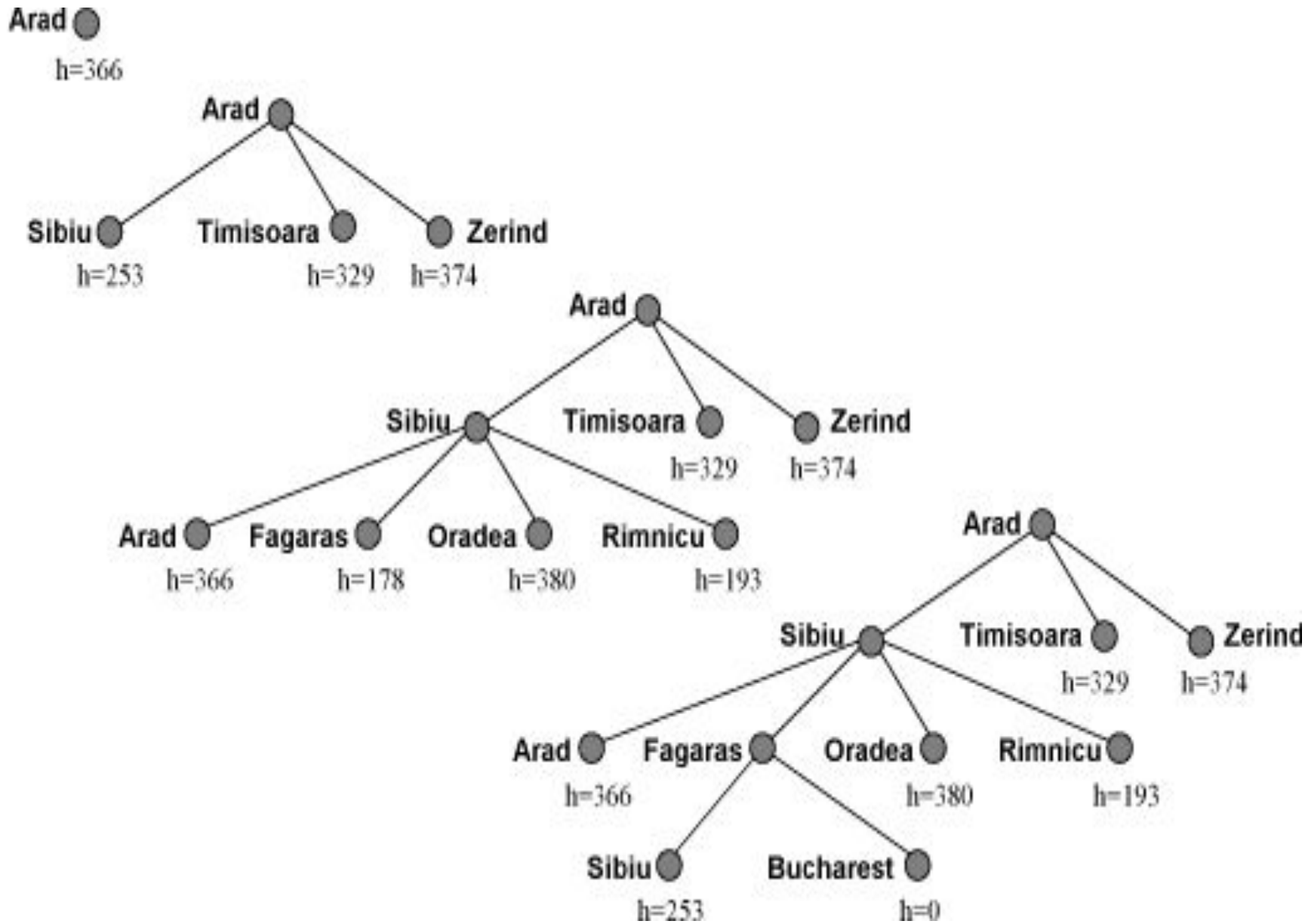
# Algoritmo A*: ver exemplo

## Viajar de Arad a Bucharest

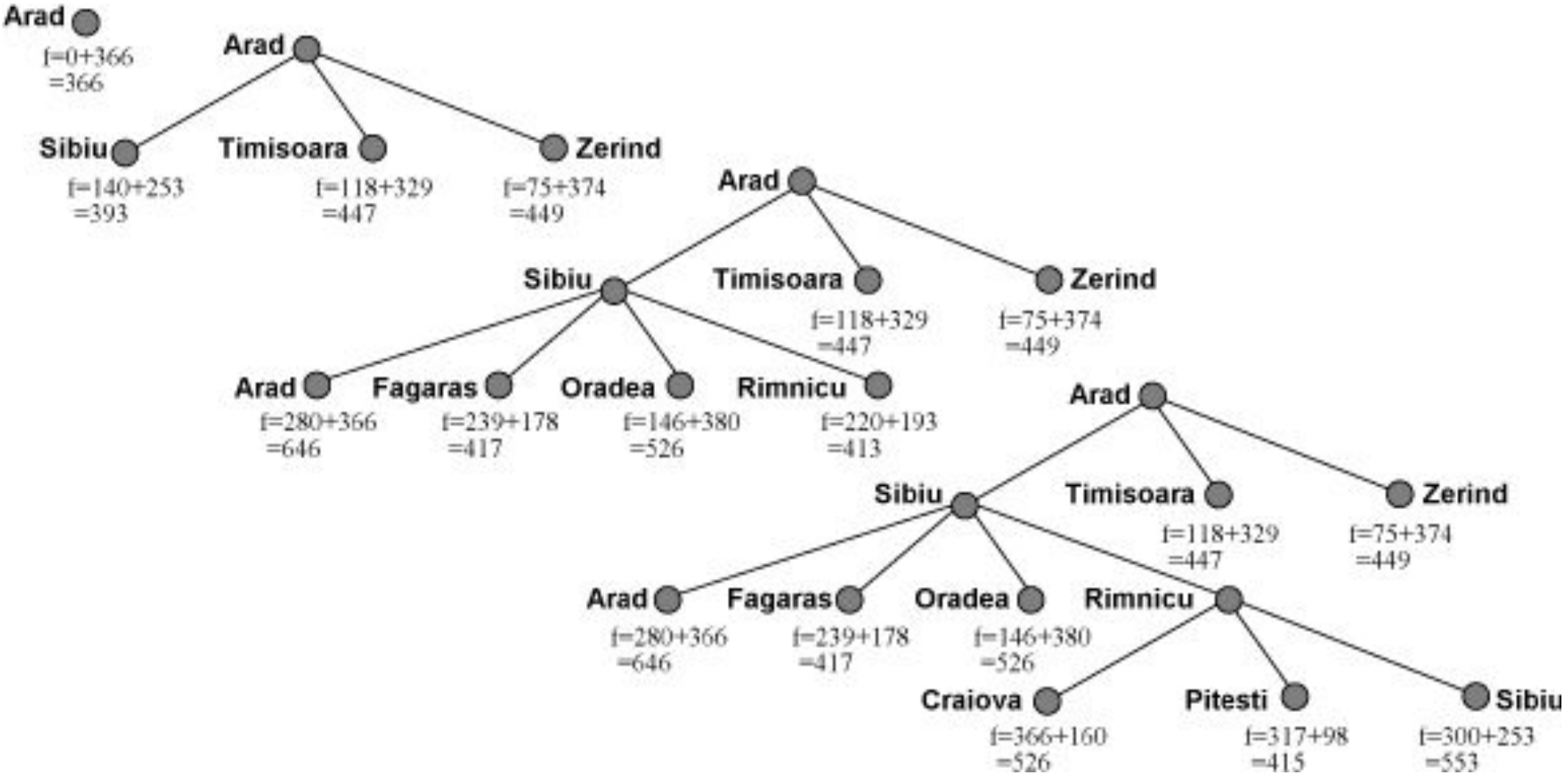



| Straight line distance to Bucharest | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 178 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 98 |
| Rimnicu Vilcea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

# Se fosse pela Busca Gulosa...



# Usando A*


Arad
f=0+366
=366
Arad
Sibiu
f=140+253
=393
Timisoara
f=118+329
=447
Zerind
f=75+374
=449
Arad
Sibiu
Timisoara
f=118+329
=447
Zerind
f=75+374
=449
Arad
f=280+366
=646
Fagaras
f=239+178
=417
Oradea
f=146+380
=526
Rimnicu
f=220+193
=413
Arad
Sibiu
Timisoara
f=118+329
=447
Zerind
f=75+374
=449
Arad
f=280+366
=646
Fagaras
f=239+178
=417
Oradea
f=146+380
=526
Rimnicu
Craiova
f=366+160
=526
Pitesti
f=317+98
=415
Sibiu
f=300+253
=553

# Inventando Funções Heurísticas

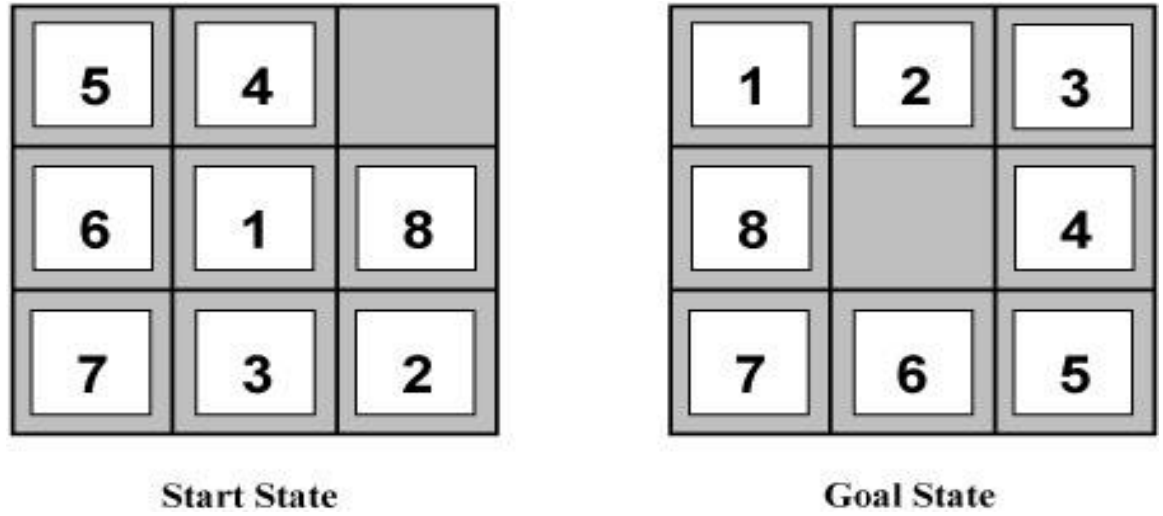
# Inventando Funções Heurísticas

- Como escolher uma boa função heurística *h* ?
  - *h* depende de cada problema particular
  - *h* deve ser *admissível*
    - i.e., não superestimar o custo real da solução
- Existem estratégias genéricas para definir *h* :
  1) Relaxar restrições do problema
  2) "Aprender" a heurística a partir de exemplos
     - Aprendizagem de Máquina

# Estratégias para definir *h*
# (1) Relaxando o problema

- Problema Relaxado
  - versão simplificada do problema original onde os operadores são menos restritivos
- Exemplo: jogo dos 8 números
  - Operador original
    - um número pode mover-se de A para B se A é adjacente a B e B está vazio
    - busca exaustiva ≈ $3^{22}$ estados possíveis
  - Operadores relaxados:
    1. um número pode mover-se de A para B se A é adjacente a B (*h2*)
    2. um número pode mover-se de A para B se B está vazio
    3. um número pode mover-se de A para B (*h1*)

# Estratégias para definir *h*
## (1) Relaxando o problema




Heurísticas para o jogo dos 8 números

***h1*** = no. de elementos fora do lugar (*h1*=7)
***h2*** = soma das distâncias de cada número à posição final
(*h2* = 2+3+3+2+4+2+0+2=18)

# Estratégias para definir *h*
## (1) Relaxando o problema

- O custo de uma solução ótima para um problema relaxado é sempre uma heurística admissível para o problema original.
- Existem softwares capazes de gerar automaticamente problemas relaxados
  - Se o problema for definido em uma linguagem formal
- Existem também softwares capazes de gerar automaticamente funções heurísticas para problemas relaxados

# Escolhendo Funções Heurísticas

- É sempre melhor usar uma função heurística com valores mais altos
  - i.e., mais próximos do valor real do custo de caminho
  - ** contanto que ela seja admissível **
- No exemplo anterior, $h_2$ é melhor que $h_1$
  - $\forall n, h_2(n) \geq h_1(n)$
  - A* com $h_2$ expande menos nós do que com $h_1$
- $h_i$ domina $h_k \Rightarrow h_i(n) \geq h_k(n)$ $\forall n$ no espaço de estados
  - $h_2$ domina $h_1$

# Escolhendo Funções Heurísticas

- Caso existam muitas funções heurísticas para o mesmo problema
  - e nenhuma delas domine as outras
  - usa-se uma heurística composta:
    - $h(n) = \max (h_1(n), h_2(n),\ldots,h_m(n))$ para cada nó n

- Assim definida, h é admissível e domina cada função $h_i$ individualmente para cada nó n do espaço de estados.

# Estratégias para definir *h*
# (2) Aprendendo a heurística

- Definindo *h* com aprendizagem de máquina
  - Esse assunto não faz parte da nossa disciplina...

**1.** Criar um corpus de exemplos de treinamento
  - Resolver um conjunto grande de problemas
    - e.g., 100 configurações diferentes do jogo dos 8 números
  - Cada solução ótima para um problema provê exemplos
    - Cada exemplo consiste em um par

      (estado no caminho "solução", custo real da solução a partir daquele ponto)

# Estratégias para definir *h*
# (2) Aprendendo a heurística

**1.** Criar um corpus de exemplos de treinamento

**2.** Usar esse corpus para treinar um algoritmo de aprendizagem indutiva

- Que será capaz de prever o custo de outros estados gerados durante a execução do algoritmo de busca