CTC-17 Inteligência Artificial
Problemas de Busca

Prof. Paulo André Castro
pauloac@ita.br
www.comp.ita.br/~pauloac
Sala 110,
IEC-ITA

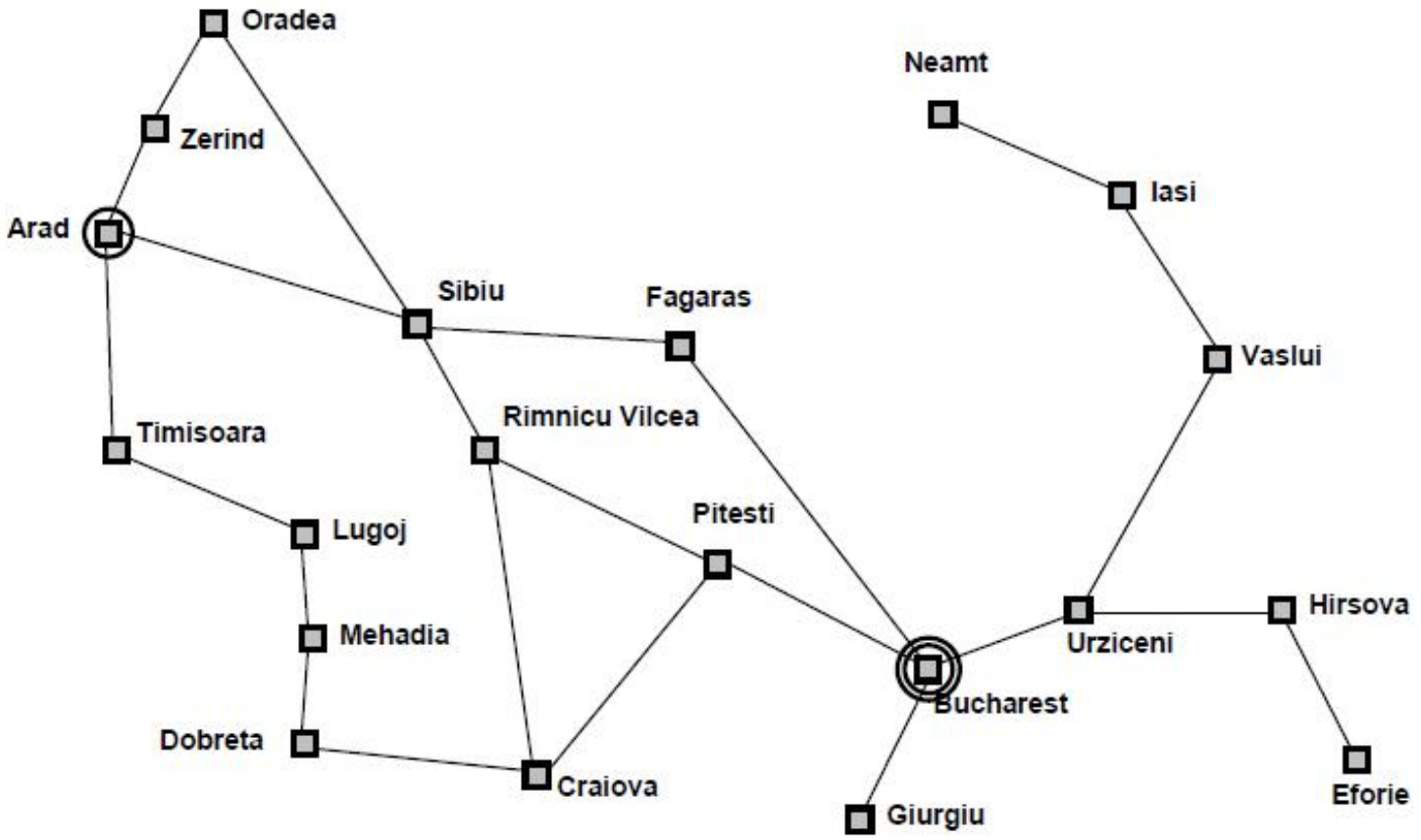
Sumário

- Agentes que buscam soluções para problemas:
  Exemplo
- Tipos de Ambiente e Busca
- Formulação de problemas
  - Exemplos de problemas
- Busca Desinformada
- Busca Informada

Exemplo: Férias na Romênia

- Em férias na Romênia; atualmente em Arad.
- Formular objetivo:
  - Estar em Bucareste
- Formular problema:
  - Estados: várias cidades
  - Operadores: dirigir entre cidades
- Achar solução:
  - seqüência de cidades, e.g., Arad, Sibiu, Fagaras, Bucareste

Exemplo: Romênia



Tipos de Ambiente e Busca

- Determinístico, observável: Problema de estados simples
- Determinístico, parcialmente observável: problema de estados múltiplos (impossível determinar qual o verdadeiro estado atual)
- Não-determinístico, parcialmente observável : problema de contingência (talvez)
  - sensores devem ser usados durante execução
  - solução é uma árvore ou política
  - muitas vezes alterna entre busca e execução
- Espaço de estados desconhecido: problema de exploração online

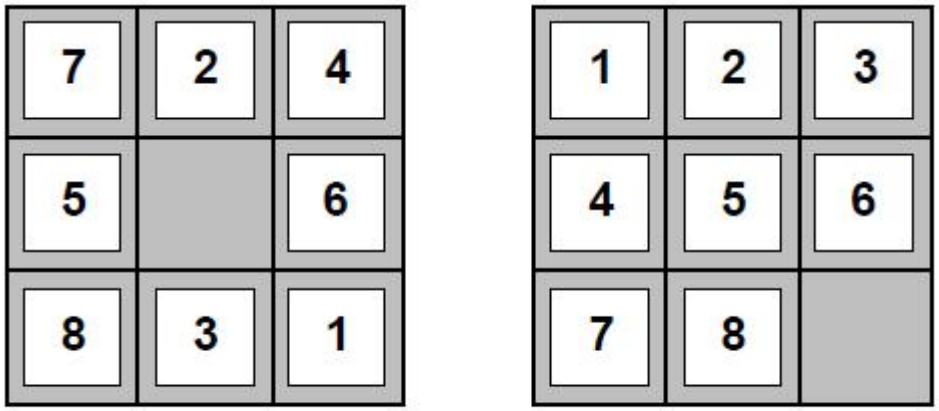
Formulação do problema de estados simples

- Um problema é definido por:
  - **Estado inicial** --- e.g., ``em Arad'' ‘
  - **Operadores**
  - (ou **função sucessor** S(x)
    - e.g., Arad → Zerind, Arad →Sibiu, etc.
  - **Teste de objetivo**: que pode ser
    - Explícito: e.g., x = ``em Bucareste''\\
    - Implícito: e.g. NoDirt(x)
  - **Custo da trajetória** (aditiva)
  - e.g., soma das distâncias, número de operadores executados, etc.
- **Solução**: é uma sequência de operadores que leva do estado inicial ao estado-objetivo

Selecionando um espaço de estados

- O mundo real é extremamente complexo
- → espaço de estados deve ser **abstraído** do processo de solução
- estado abstrato = conjunto de estados reais
- operador abstrato = combinação de ações reais
- e.g., ``Arad → $ Zerind'' representa um conjunto complexo de possíveis rotas, retornos, paradas para descanso, etc.
- Para realizabilidade garantida, qualquer estado real "em Arad" deve levar a algum estado real ``em Zerind'‘
- Solução (abstrata)
  - conjunto de trajetória reais que são soluções no mundo real
  - Cada ação abstrata deve ser mais ``fácil'' do que o problema original!

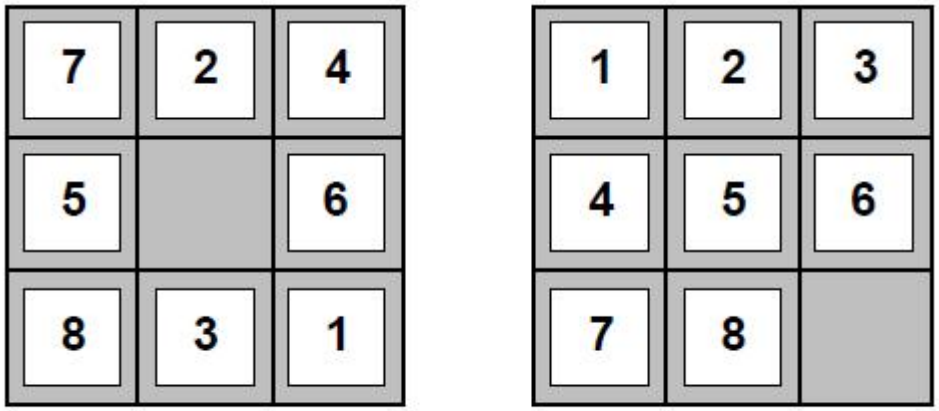
Exemplo: Um quebra-cabeças



Start State | Goal State

- Estados?
- Operadores?
- Teste do objetivo?
- Custo da trajetória?

Exemplo: Um quebra-cabeças

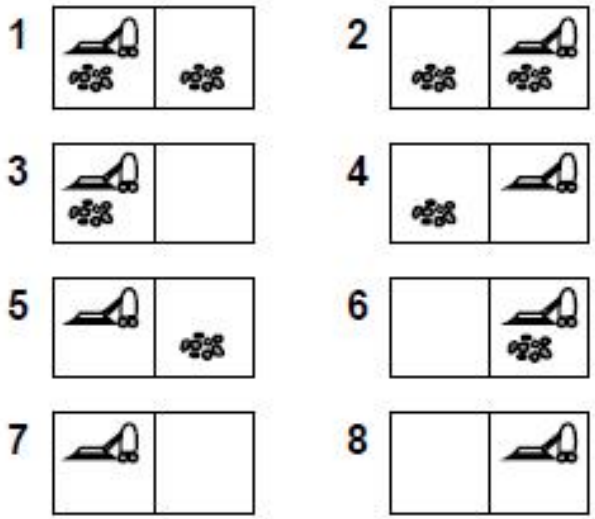


Start State Goal State

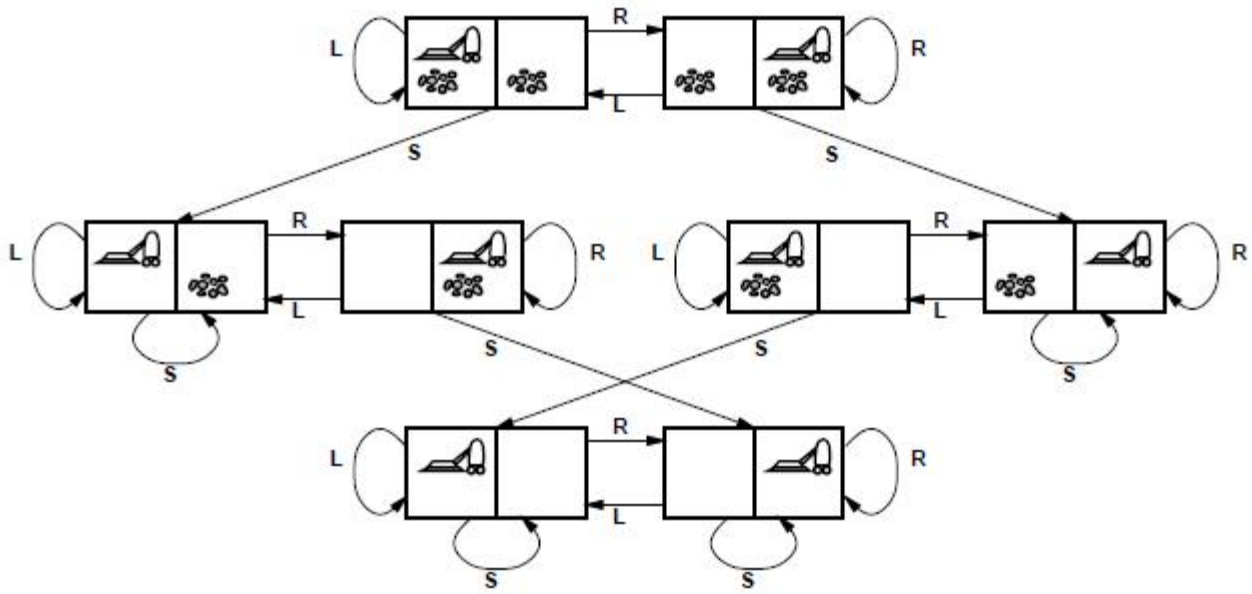
- Estados? localização dos blocos móveis
- Operadores? `mover espaço' pra esquerda, pra direita, pra cima, pra baixo
- Teste do objetivo? = atingir *estado-objetivo* (dado)
- Custo da trajetória? 1 por movimento
- Obs: achar solução ótima para n casas é NP-completo

Exemplo: problema do aspirador

- Problema do aspirador: Um robô aspirador deve limpar duas salas contíguas. Modelar o problema significa abstrair as carcterísticas relevantes para o problema e desprezar as demais.
- Possíveis estados?

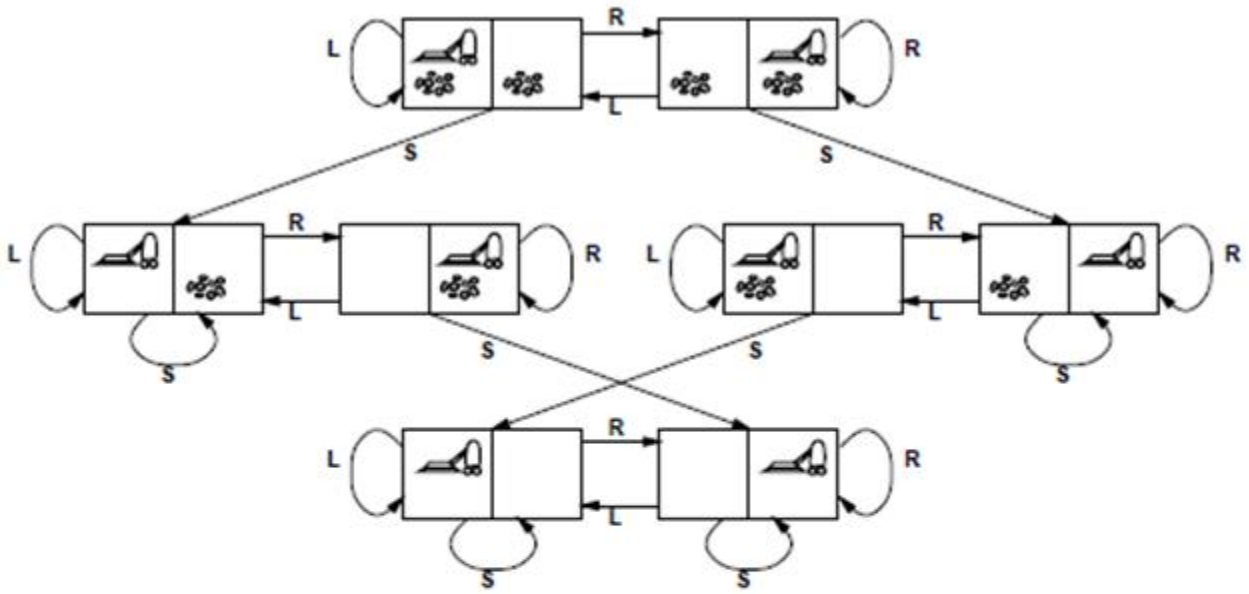


Exemplo: grafo do problema do aspirador



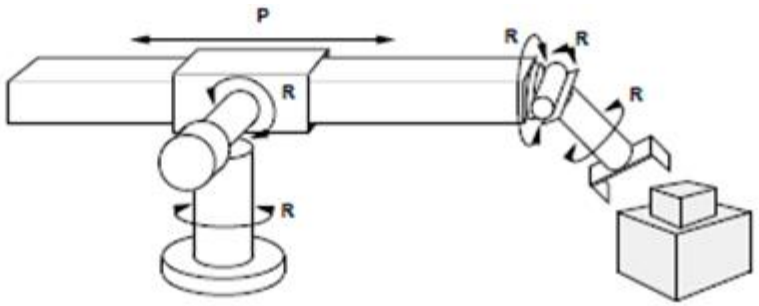
- Estados?
- Operadores?
- Teste do objetivo?
- Custo da trajetória?

Exemplo: grafo do problema do aspirador



estados??:	localização do aspirador e da sujeira (sujeira é variável binária)
operadores??:	*Esquerda*, *Direita*, *Aspira*
teste do objetivo??:	limpeza total
custo da trajetória??:	1 por operador

Exemplo: Linha de montagem automatizada



estados??: coordenadas reais dos ângulos das juntas do robô
partes do objeto por montar

operadores??: movimentos contínuos das juntas do robô

teste do objetivo??: montagem completa do objeto

custo da trajetória??:

Busca Desinformada e Busca Informada

- Busca desinformada utiliza apenas a formulação do problema no processo de busca
  - (Estados, Operadores, Teste do objetivo, Custo da trajetória)

- Busca informada é qualquer uma que usa informação além daquela disponibilizada pela definição do problema. A solução fica dependente do domínio do problema

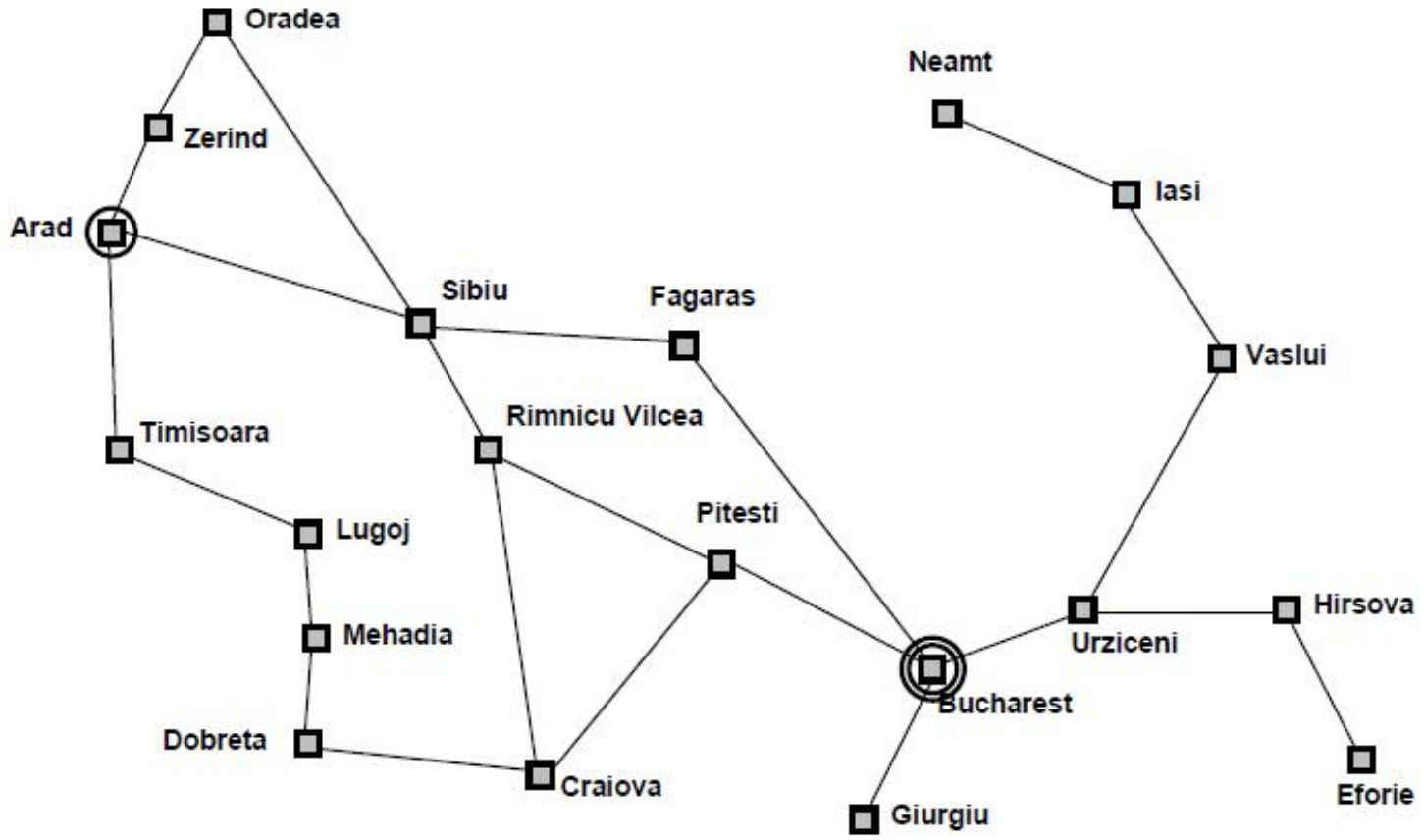
Algoritmo Básico

- Idéia: exploração simulada do espaço de estados via geração dos sucessores dos estados já explorados

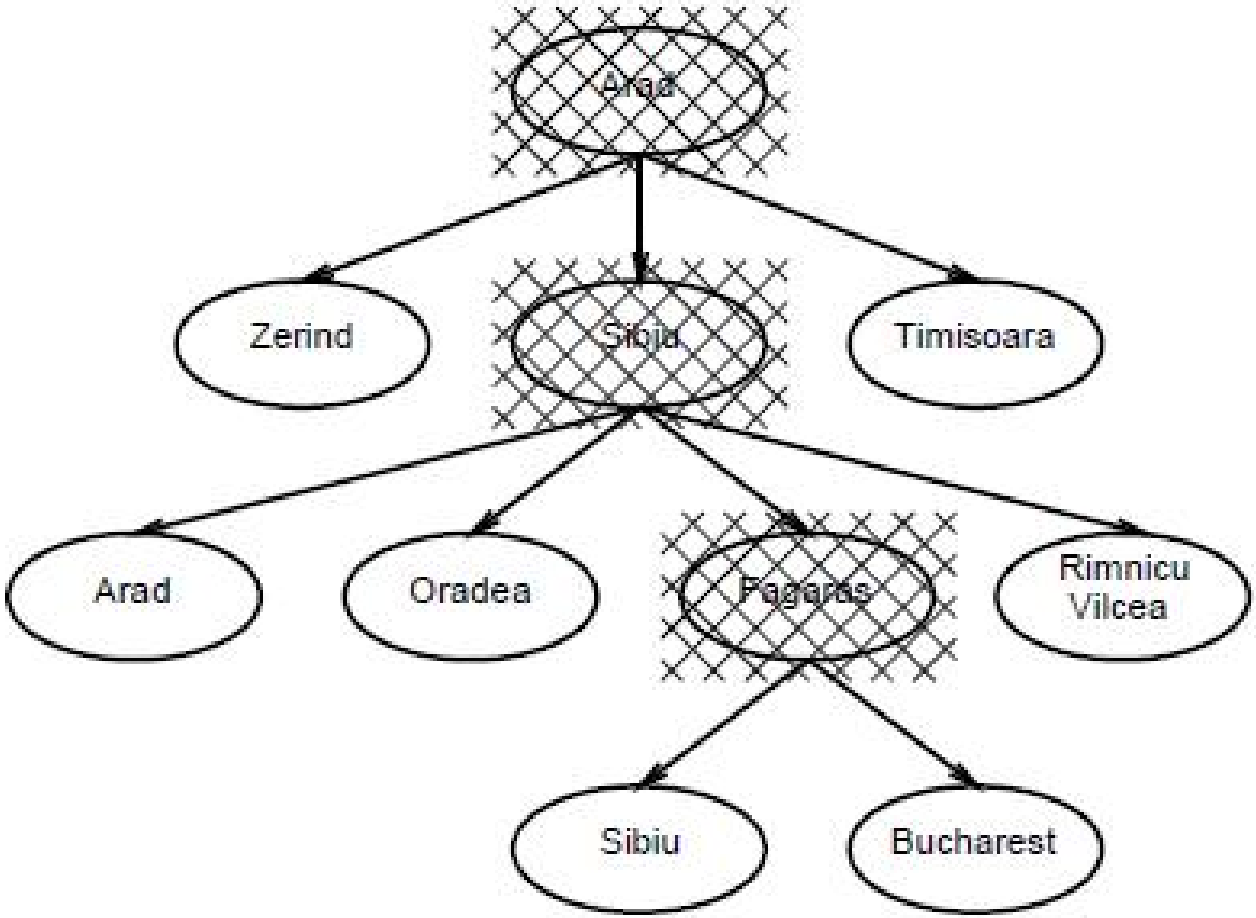
```
function GENERAL-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

Busca genérica: exemplo

- Caminho de Arad para Bucareste?

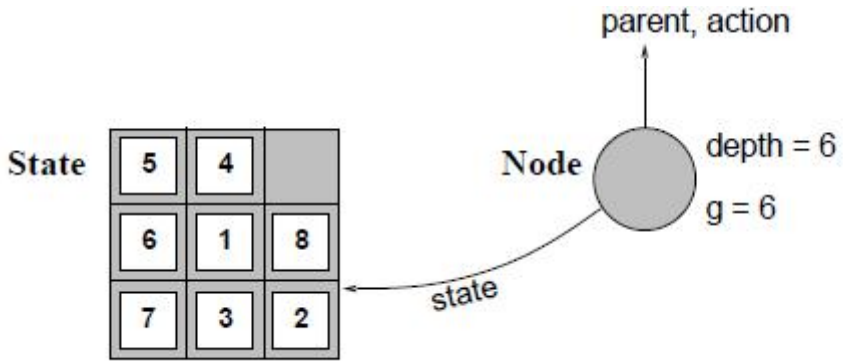


Busca genérica: exemplo



Estados vs. Nós

- Um estado é uma (representação de) uma configuração física
- Um nó é uma estrutura de dado constituinte de uma árvore de busca e inclui pai, filhos, profundidade, custo da trajetória
- Estados não têm pai, filhos, profundidade, ou custo da trajetória!



- Expandir significa criar novos nós, usando os Operadores (ou SuccessorFn) do problema para determinar os estados associados aos nós.

Estratégias de busca

- Uma estratégia de busca é uma escolha da *ordem de expansão dos nós*
- Estratégias são avaliadas pelos seguintes parâmetros:
- **Completeza:** a solução (quando existe) é sempre encontrada?
- **Complexidade no tempo**: número de nós gerados/expandidos
- **Complexidade no espaço**: número máximo de nós na memória
- **Otimalidade**: a solução encontrada é de custo mínimo?

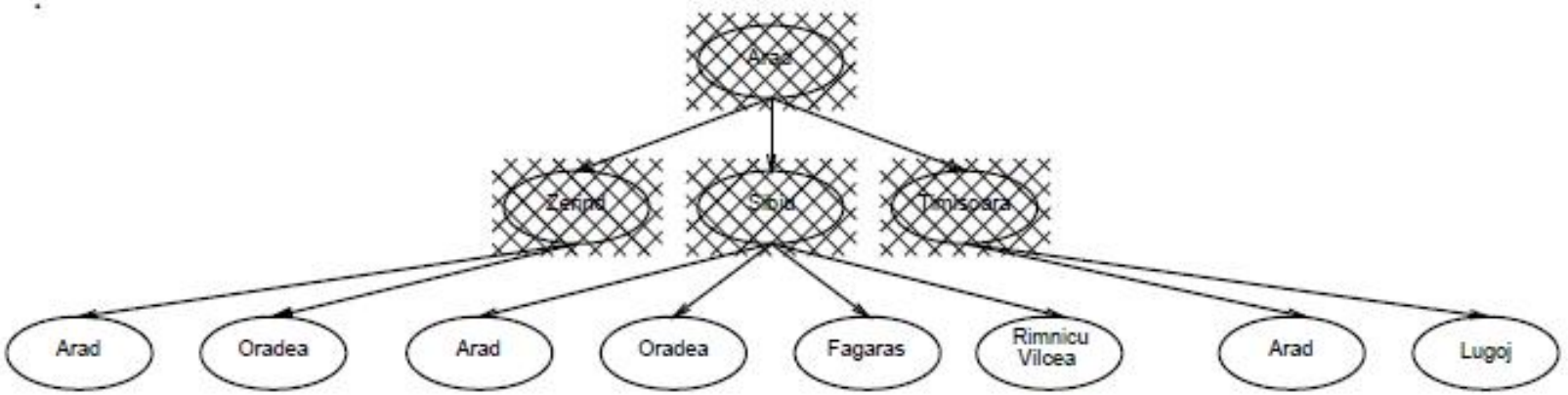
- Complexidades no tempo e espaço são medidas em termos de
- b---fator de ramificação máximo da árvore de busca
- d---profundidade da solução de custo mínimo
- m---profundidade máxima do espaço de busca (pode ser infinito)

Estratégias de busca desinformada

- Estratégias desinformadas usam apenas a informação disponibilizada pela definição do problema
  - Busca em largura
  - Busca de custo uniforme
  - Busca em profundidade
  - Busca em profundidade limitada
  - Busca de aprofundamento iterativo

Busca em largura

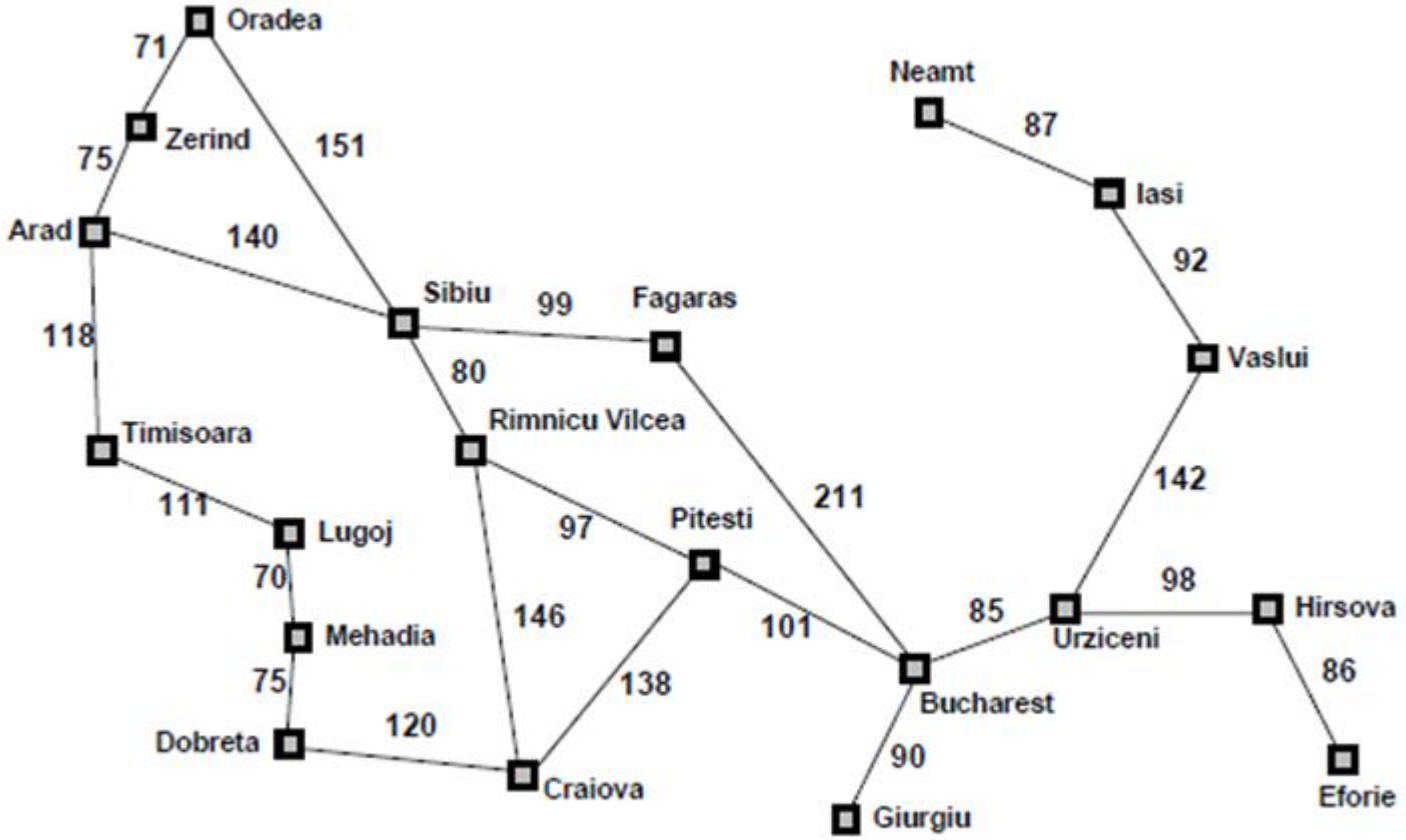
- Expande o nó menos profundo
- Implementação:
- QueueingFn = coloca sucessores no final da fila



Propriedades da busca em largura

- **Completa**: Sim (se b for finito)
- **Tempo**: 1+b+b^2+b^3+.... +b^d = O(b^d)$, i.e., exponencial em d
- **Espaço**: O(b^d) (mantém todos os nós na memória)
- **Ótima**: Sim (se custo por passo = 1); em geral não-ótima
- Espaço é o grande problema; pode facilmente gerar nós à razão de 1MB/seg (24hrs = 86GB !!!...considerando 1KB/nó)

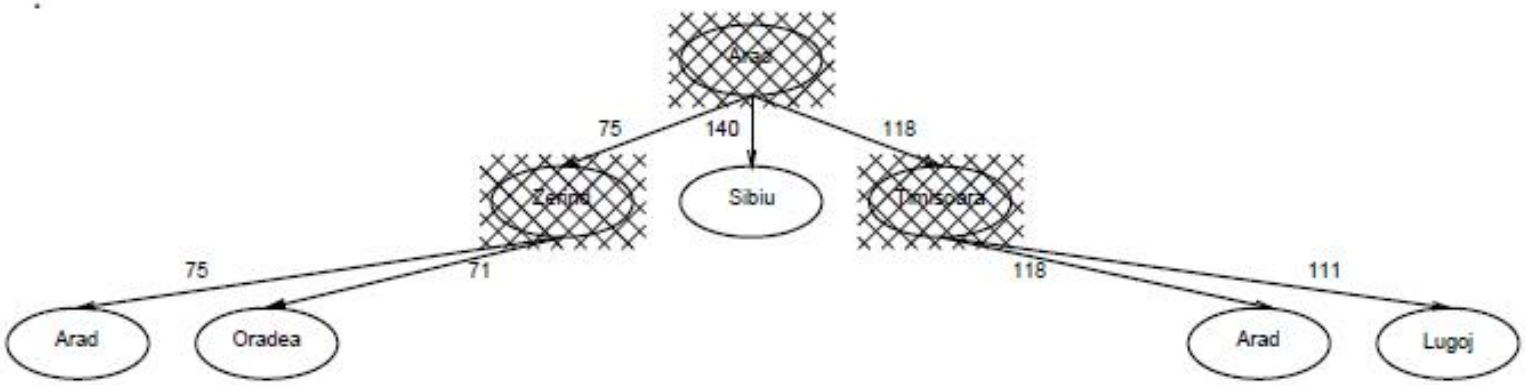
Romênia com custos por passo em km



Busca de custo uniforme

- Idéia: Expandir nó de custo (g(n)) mínimo

- Implementação: QueueingFn = inserir em ordem de custo crescente

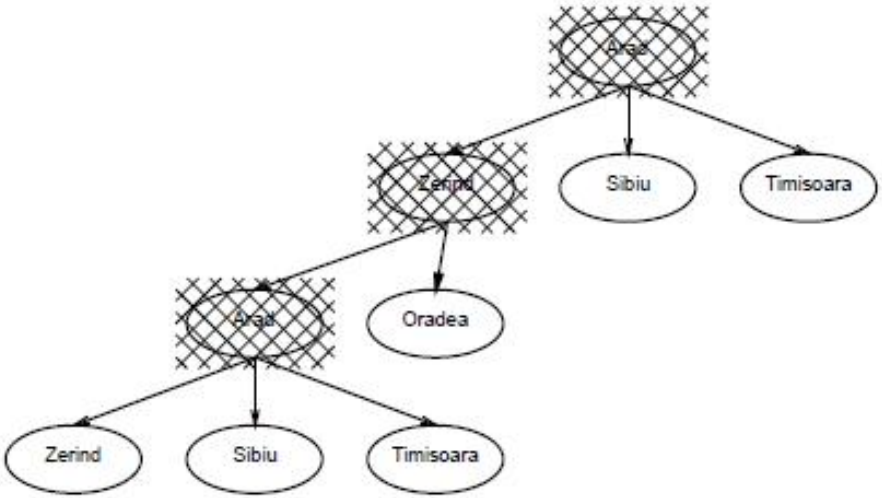


Propriedades da busca de custo uniforme

- **Completa**: Sim, se custos maiores que zero
- **Tempo e Espaço**: Pode-se demonstrar que a complexidade em tempo e espaço é $O(b^{C/i})$. Onde C é o custo da solução ótima e $i>0$ é o menor custo de ação.
  - A busca de custo uniforme frequentemente explora grandes sub-árvores com pequenos passos antes de explorar caminhos com grandes passos e talvez mais úteis
- **Ótima**: Sim, se o custo por passo maior que zero

Busca em profundidade

- Expandir o nó mais profundo
- Implementação: QueueingFn = insere sucessores na frente da fila



- I.e., busca em profundidade pode executar ciclos infinitos
- O espaço de busca deve ser **finito e acíclico** (ou deve haver checagem de estados repetidos)

Propriedades da busca em profundidade

- **Completa**: Não – deficiente em espaços de profundidade infinita ou cíclicos
  - Modificado para evitar estados repetidos na trajetória
  - Completa em espaços finitos
- **Tempo**: $O(b^m)$: péssimo se m for muito maior do que d
  - mas se soluções são densas, pode ser muito mais rápido do que em largura
- **Espaço**: O(bm), i.e., linear no espaço!
- **Ótima**: Não

Limitando a profundidade

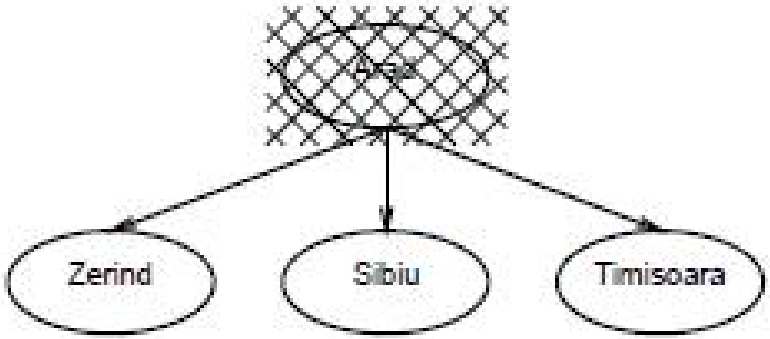
- Pode-se minimizar o problema de árvores profundas (m muito maior que d) limitando a profundidade
- = busca em profundidade com limite de profundidade igual a L

- Implementação: Nós à profundidade L não têm sucessores

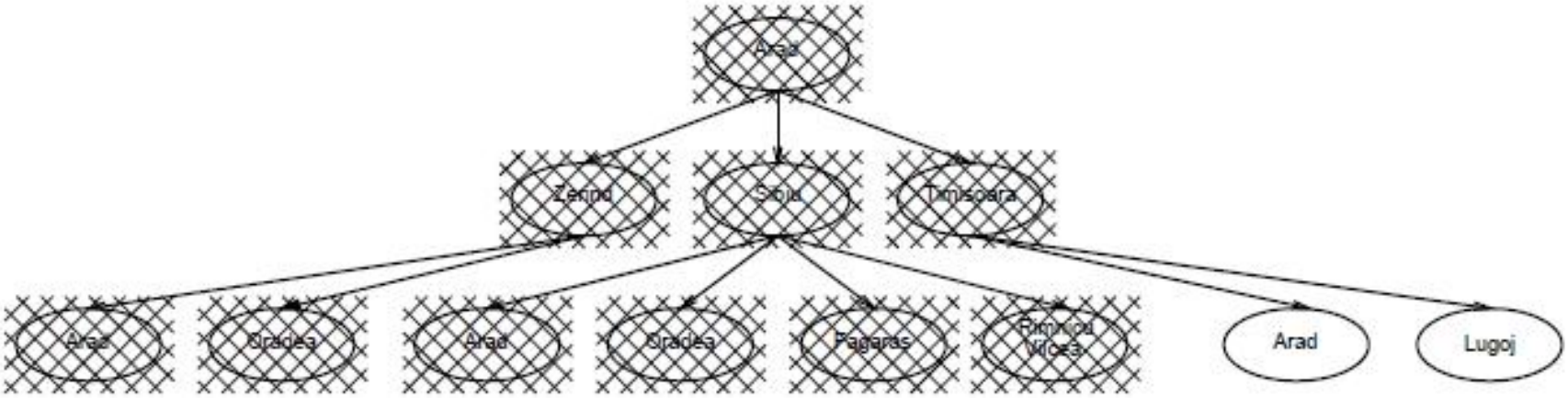
- Problema: não saber a profundidade da solução!

Busca de aprofundamento iterativo l=0

Busca de aprofundamento iterativo l=1



Busca de aprofundamento iterativo l=2



Propriedades da busca de aprofund. Iterativo

- **Completa:** Sim
- **Tempo:** $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \cdots + b^d = O(b^d)$
- **Espaço:** $O(bd)$
- **Ótima:** Sim, se custo por passo = 1

Resumo Buscas Desinformadas

- A formulação de um problema requer abstração para evitar detalhes irrelevantes do mundo real, de modo a se definir um espaço de estados que possa ser explorado.
- Existem várias estratégias de busca não-informada.
- Busca de aprofundamento iterativo é linear no espaço e não requer muito mais tempo do que os outros algoritmos.
- Com informação adicional, o desempenho pode ser bem melhor….

Estratégias de Busca Informada

- Estratégias informadas podem usar informação além daquela disponibilizada pela definição do problema
  - (Estados, Operadores, Teste do objetivo, Custo da trajetória)
- Buscas Best-First (Best-First Search, BFS)
  - Busca gananciosa ou gulosa (greedy)
  - Busca A*

Buscas Best-First

- Idéia: Estimar a desejabilidade de expansão de cada nó através de uma função de avaliação

- Expandir o nó mais desejável

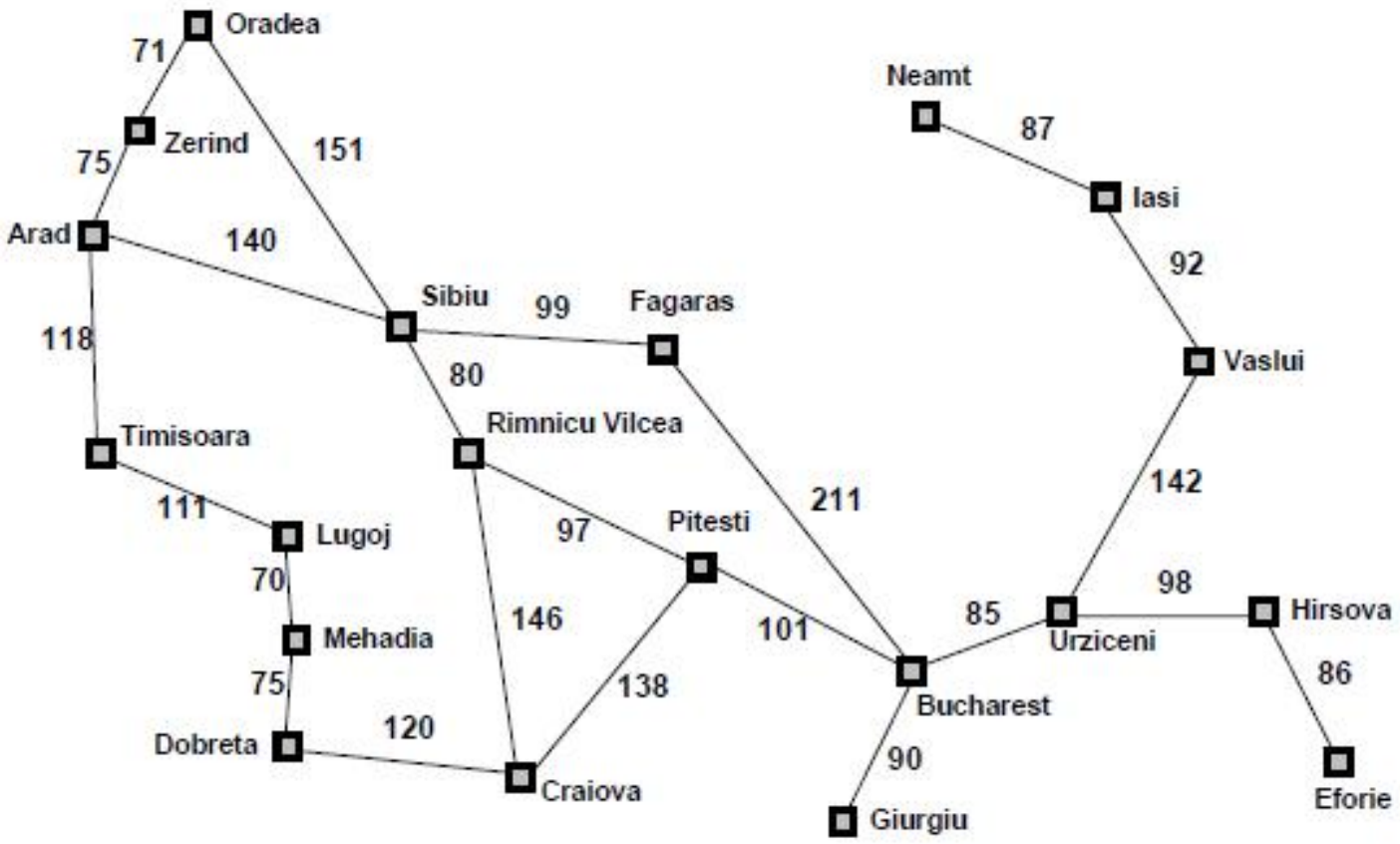
- Implementação
  - Introduzir em QueueingFn sucessores na ordem de desejabilidade
  - Casos Particulares de Busca
    - Busca Greedy
    - Busca A*

Best-First Search (Greedy)

- Define-se uma função que estima o menor custo possível de atingir um estado objetivo a partir de um determinado nó n.
- Geralmente chamada de função de avaliação: h’ (n). É um função que estima o custo real h(n) até um estado objetivo a partir do nó n
- No caso do problema Férias na Romênia. h’ (n) poderia ser a distância em linha reta

- A busca greedy expande o nó que parece mais próximo do objetivo

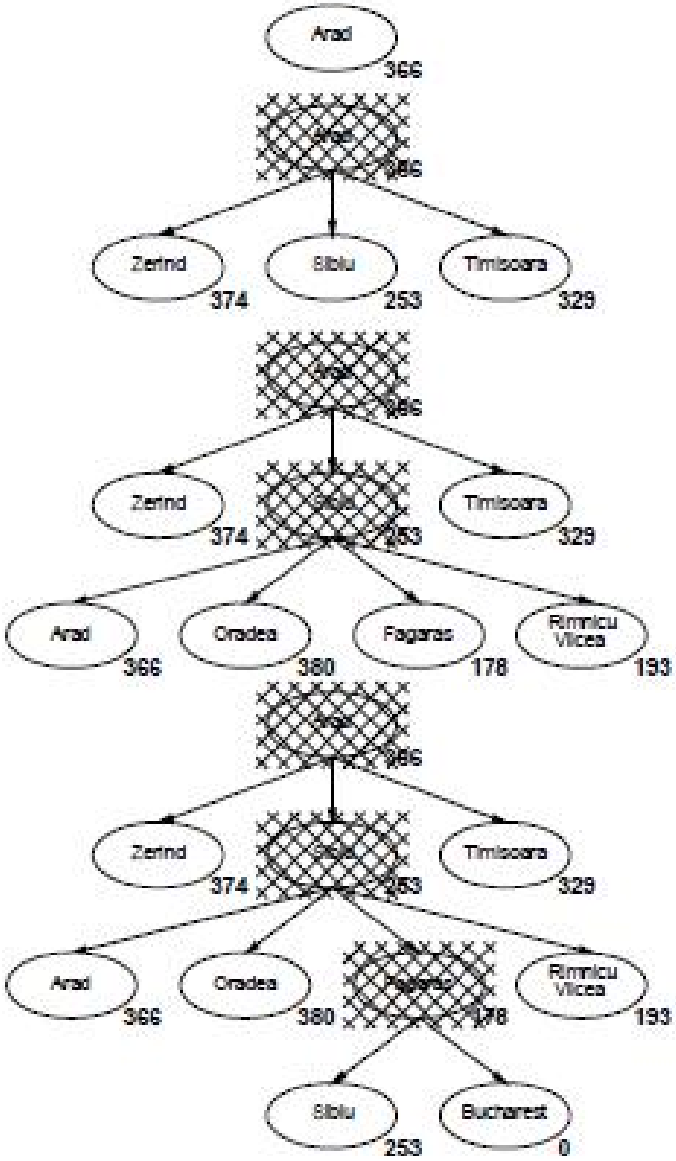
Romênia com custos de passo em km



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

Busca Greedy

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



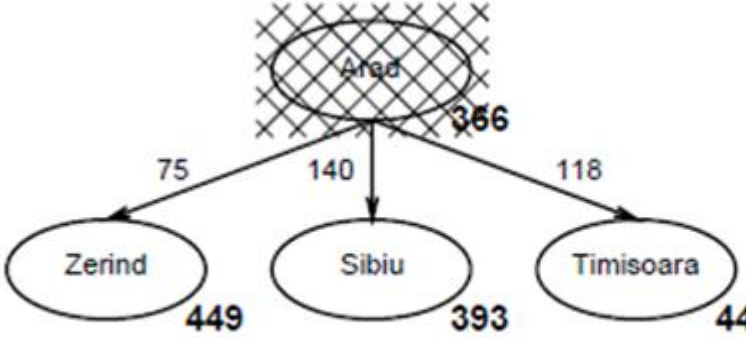
- Custo = 140+99+211=450

Avaliação da Busca Greedy

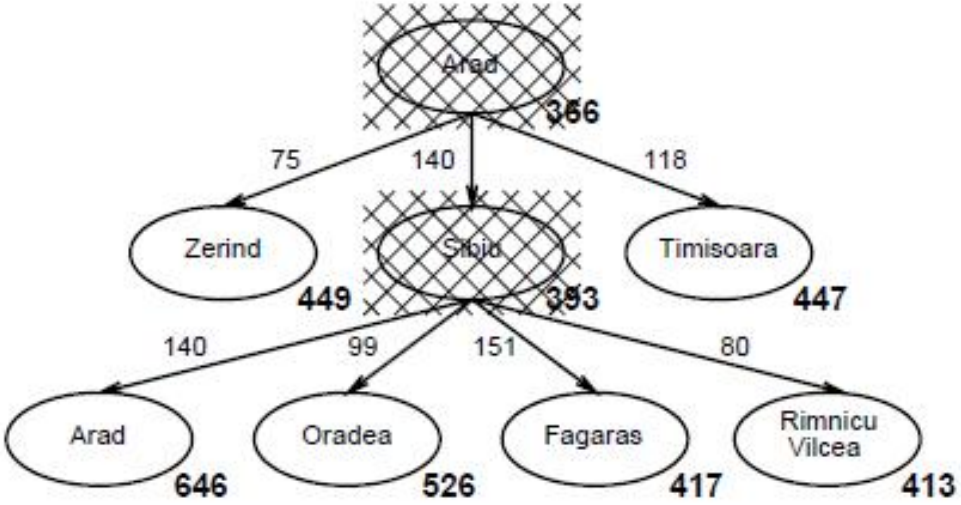
- **Completa:** Não – deficiente em espaços de profundidade infinita ou cíclicos
  - Modificada para evitar estados repetidos na trajetória, é Completa em espaços finitos
- **Tempo:** $O(b^d)$: Boas heurísticas podem levar a uma grande melhoria
- **Espaço:** $O(b^d)$ mantem todos os nós em memória
- **Ótima:** Não

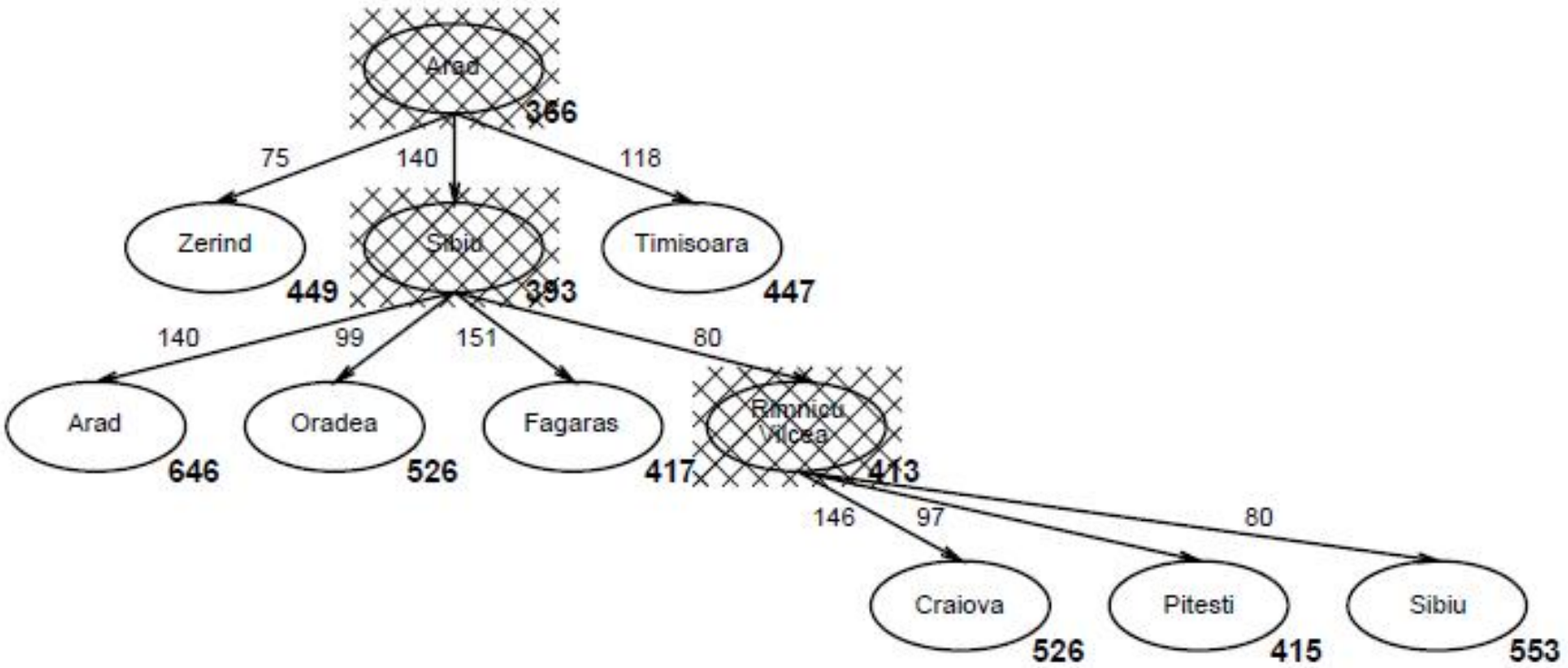
Busca A*

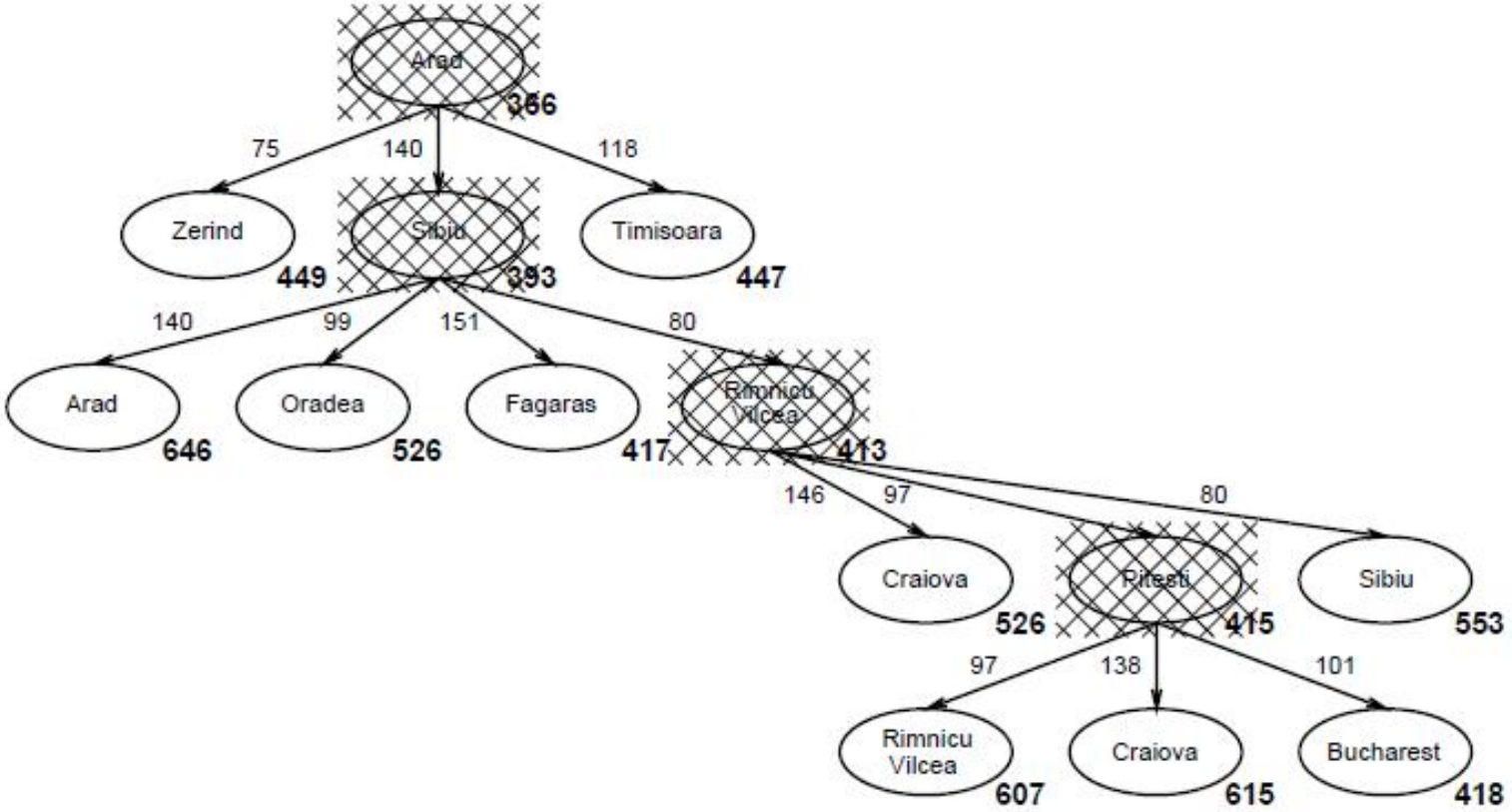
- Idéia: Evitar expandir caminhos que já ficaram caros
- Define-se uma função de avaliação que fornece para cada nó n, f(n)=g(n)+h(n). Onde:
  - f(n) - Custo real até um estado objetivo via n.
  - g(n)- Custo real da trajetória para alcançar o nó n a partir da raiz (supõe-se conhecido)
  - h(n) - Custo real mínimo de n até um estado objetivo
- Para realizar expansões em geral, precisa-se estimar h(n) e logo f(n) será também um valor estimado
  - f'(n) - Custo estimado até um estado objetivo via n.
  - h'(n) - Custo estimado mínimo de n até um estado objetivo
- A* expande o nó com menor f'(n) dentre os possíveis

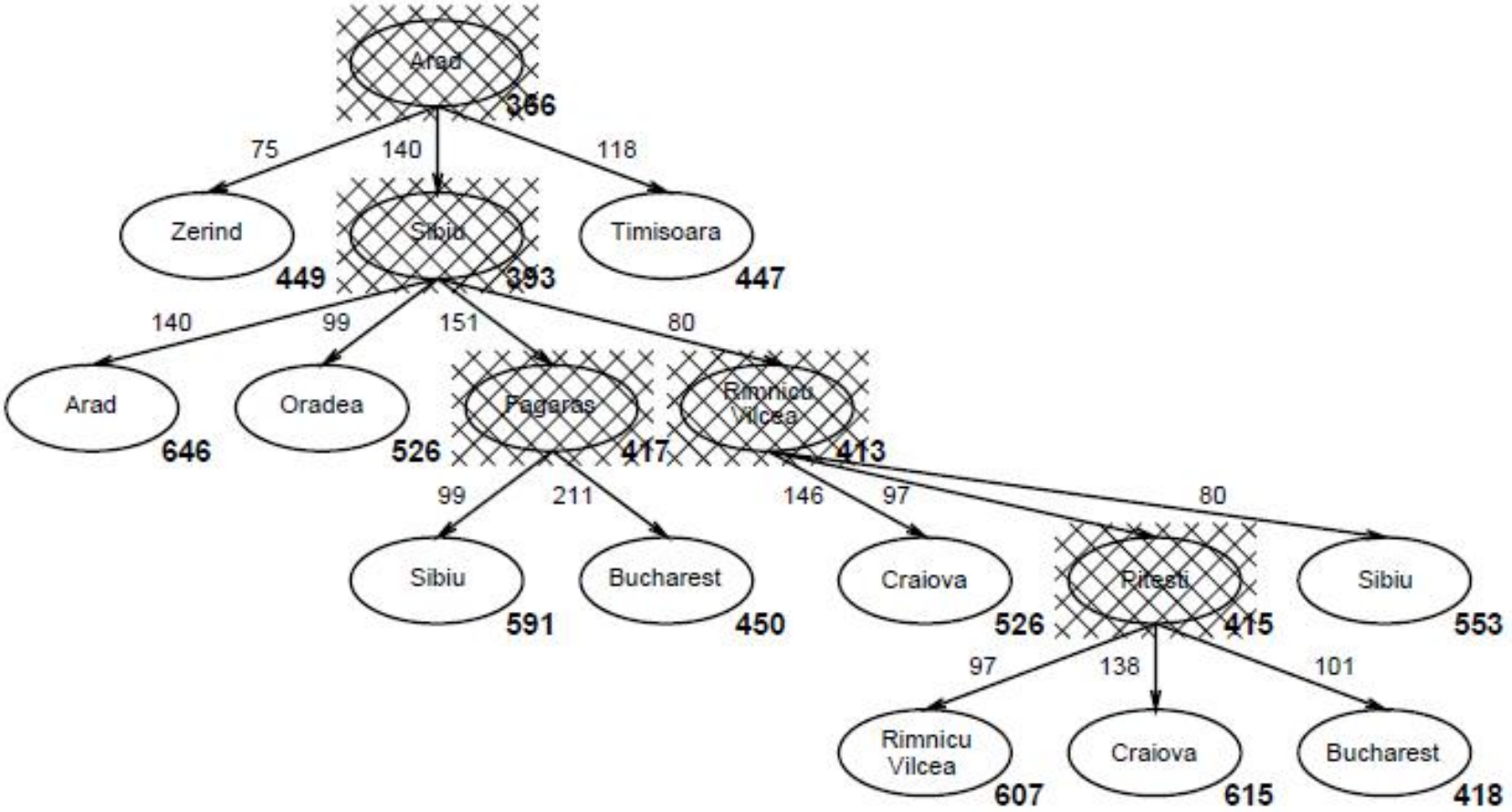


=f'(Timisoara)=
(118+329)


Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447
140
99
151
80
Arad
646
Oradea
526
Fagaras
417
Rimnicu
Vilcea
413


Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447
140
99
151
80
Arad
646
Oradea
526
Fagaras
417
Rimnicu Vilcea
413
146
97
80
Craiova
526
Pitesti
415
Sibiu
553


Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447
140
99
151
80
Arad
646
Oradea
526
Fagaras
417
Rimnicu Vilcea
413
146
97
80
Craiova
526
Pitesti
415
Sibiu
553
97
138
101
Rimnicu Vilcea
607
Craiova
615
Bucharest
418



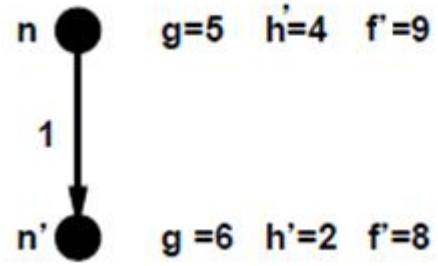
- Busca Greedy: Custo = 140+99+211=450
- Busca A*: Custo = 140+80+97+101=418

Avaliação da Busca A*

- **Completa**: Sim, a menos que haja infinitos nós
- **Tempo**: $O(b^d)$: Boas heurísticas podem levar a uma grande melhoria
- **Espaço**: $O(b^d)$ mantém todos os nós em memória
- **Ótima**?
  - Sim, Se:
  - 1.O fator de ramificação é finito
  - 2. Custos são positivos
  - 3. Utiliza uma heurística h'(n) **admissível**. Isto é $h'(n) \leqslant h(n)$. Então (pode-se demonstrar que) A* é uma busca ótima. Admissível = não superestima custo

Um problema

- A situação abaixo pode ocorrer em uma heurística admissível, suponha que n' é um sucessor de n



- Mas isto joga fora informação, pois com f'(n)⩾9, então f'(n') ⩾9
- Manter f'(n) não-descrescente ajudaria a aumentar a velocidade de busca
- Ao invés de f'(n')=g(n')+h'(n'), pode-se usar
  - f'(n') =max{g(n')+h'(n'); g(n)+h'(n)}
- Com pathmax, sempre não-decrescente ao longo de qualquer caminho …. a complexidade no tempo diminui

Heurísticas Consistentes

- Pode-se obter o efeito de f' não-descrente com a seleção de heurísticas apropriadas:
  - Heurísticas que garantam que para cada nó **n** e para todo sucessor **o** de n gerado por uma ação **a** o custo estimado a partir de **n** é menor igual ao custo estimado de **o** mais o custo de alcançar o. Ou seja:
  - $h'(n) \leq h'(o) + c(n,a,o)$
- Chama-se esse tipo de heurística de heurística consistente
- Pode-se provar que isto também garante f' não-decrescente
- Também é possível provar que toda heurística consistente é admissível....mas o contrário não é verdade

Prova: Se Heurística Consistente- Então é Admissível

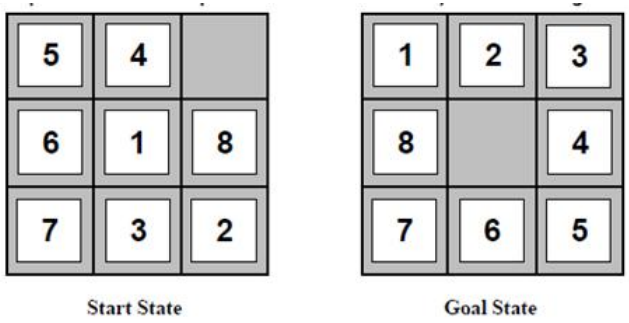
- Prova-se que uma heuristica consistente é também admissível por inducao finita.
- Se consistente é admissivel a um nó do estado final x , pois H' (final)=H(final)=0.
  - h' (x) ≤ h' (final) +c(x, a, final) -> h' (x) ≤ h(x)
- Supondo verdade para o (sucessor de n)
  - h '(o) ≤ h(o) e h' (n) ≤ h'(o)+ c(n,a,o) ->
  - h' (n) ≤ h(o)+ c(n,a,o) = h(n).
  - Logo é admissível.

Prova: Se h' é consistente então f' é não decrescente

- Seja o um sucessor de n no caminho ótimo
- Então:
  - f' (o)=g(o)+h' (o) por definição
  - Como h' é consistente então:
    - h' (o)>=h' (n)-c(n,a,o)
- Substituindo h' (o) na primeira equação:
  - f' (o)>=g(o)+h' (n)-c(n,a,o) =
  - g(o)+h' (n)-[g(o)-g(n)]=
  - g(n)+h' (n)=f' (n)
- Logo f' (o)>=f' (n) para todo n, ou seja f' é não decrescente.

Exemplo: heurística admissível mas não consistente

- Não é fácil encontrar h que seja admissível mas não consistente, mas é possível….Exemplo:



- Podemos contar quantos blocos estão na posição errada, isso seria uma heurística admissível....e também consistente
- Podemos calcular o custo apenas dos blocos 5,4,6 e 1 e usar como heurística, ainda é admissível e consistente...
- Agora selecionamos aleatoriamente o custo de [5,4,6,1] ou [7,3,8,2] ainda é admissível, mas não é mais consistente.

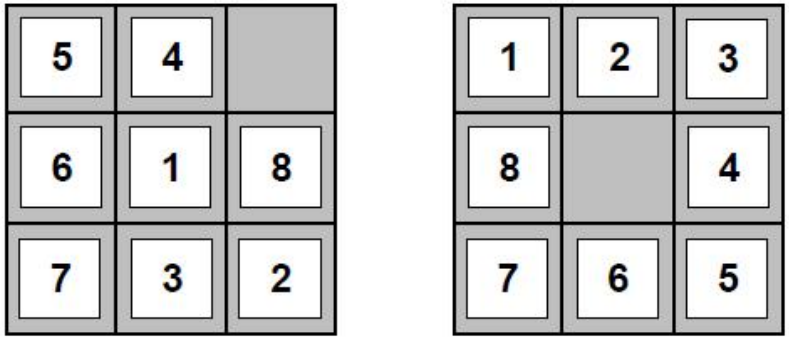
Características de A*

- É possível demonstrar que para algoritmos que estendem o caminho de busca a partir da raiz e usam a **mesma** função heurística **não há outro algoritmo** que garantidamente expanda menos nós que A*
  - exceto possivelmente através de desempate entre nós com f’ (n) de custo igual
  - Diz-se que A* é otimamente eficiente

Heurísticas Admissíveis

- Caso do problema dos blocos deslizantes:

$h_1(n)$ = número de blocos mal-posicionados
$h_2(n)$ = distância Manhattan total
(i.e., no. de quadrados para a localização desejada de cada bloco)



$h_1(S)$ =?? 7
$h_2(S)$ =?? 2+3+3+2+4+2+0+2 = 18

- Essas são heurísticas admissíveis e consistentes?
- Qual a melhor?

Obtenção de Heurísticas

Heuristica admissível pode ser derivada da solução ***exata*** do custo de uma versão ***relaxada*** do problema

Se as regras do quebra-cabeça são relaxadas de forma que um bloco possa mover-se *para qualquer lugar*, então $h_1(n)$ dá a solução mais curta.

Se as regras são relaxadas de forma que um bloco possa mover-se para *qualquer quadrado adjacente*, então $h_2(n)$ dá a solução mais curta.

Heurística Dominante

- Comparação entre IDS e A* no problema dos blocos deslizantes
- (IDS) = Iterative deepening Search=Busca de Aprofundamento Iterativo (já vista)

Se $h_2(n) \geq h_1(n)$ para todo n (ambos admissíveis) então h_2 **domina** h_1 e é melhor para busca

Custos de busca típicos:

$d = 14$ IDS $=$ 3.473.941 nós
$A^*(h_1) = 539$ nós
$A^*(h_2) = 113$ nós

$d = 24$ IDS $=$ muitos nós
$A^*(h_1) = 39.135$ nós
$A^*(h_2) = 1.641$ nós

Observe o compromisso entre dominância e admissibilidade...

Heurísticas Dominantes e Dominadas

- No exemplo, anterior IDS - Busca de aprofundamento iterativo
- A*(h1) - A* usando heuristica h1
  - h1 numero de blocos mal posicionados
- A*(h2) - A* usando heuristica h2
  - h2- numero de movimentos para colocar os blocos na posicao correta desconsiderando impedimentos por outros blocos= distancia Manhattan

- Pode-se mostrar que Heuristicas dominantes nunca irão expandir mais nós que heurísticas dominadas