U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Artificial Intelligence/ Inteligência Artificial

Lecture 4: Solving Search Problems

Luís Paulo Reis

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.
Associate Professor at DEI/FEUP – Informatics Engineering Department,
Faculty of Engineering of the University of Porto, Portugal
Vice-President of APPIA – Portuguese Association for Artificial Intelligence

Resolução de Problemas por Pesquisa

Estrutura da Apresentação:

- **Métodos de Resolução de Problemas**
- **Formulação de Problemas**
- **Espaço de Estados**
- **Pesquisa Não Informada:**
  - Primeiro em Largura, Primeiro em Profundidade, Custo Uniforme, Aprofundamento Iterativo, Pesquisa Bidirecional.
- **Pesquisa Inteligente:**
  - Pesquisa Gulosa, Algoritmo A*
- **Exemplos Práticos de Aplicação**

Pesquisa da Solução

- **Metodologia para realizar a Pesquisa da Solução:**
  1) Começar com o estado inicial
  2) Executar o teste do objetivo
  3) Se não foi encontrada a solução, usar os operadores para expandir o estado atual gerando novos estados - sucessores (expansão)
  4) Executar o teste objetivo
  5) Se não tivermos encontrado a solução, escolher qual o estado a expandir a seguir (estratégia de pesquisa) e realizar essa expansão
  6) Voltar a 4)

Pesquisa da Solução - Árvore de Pesquisa

- **Árvore de pesquisa composta por nós. Nós folhas, ou não têm sucessores ou ainda não foram expandidos!**
- **Importante distinguir entre a árvore de pesquisa e o espaço de estados!**

- **Nó da Árvore (cinco componentes):**
  - Estado a que corresponde
  - Nó que lhe deu origem (pai)
  - Operador aplicado para o gerar
  - Profundidade do nó
  - Custo do caminho desde o nó inicial

- **datatype** NODE

  components: STATE, PARENT-NODE, OPERATOR, DEPTH, PATH-COST
- Fronteira: Conjunto de nós à espera de serem expandidos
  - Representada como uma fila

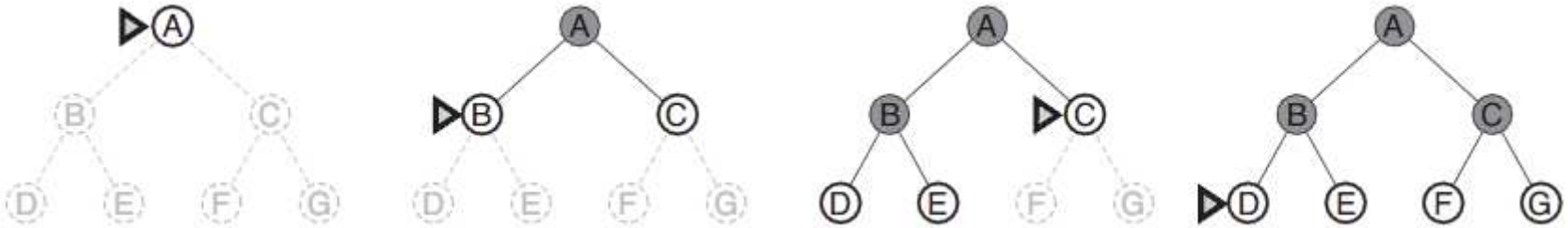
Estratégias de Pesquisa

- **Critérios de Avaliação:**
  - Complitude: Está garantido que encontra a solução?
  - Complexidade no Tempo: Quanto tempo demora a encontrar a solução?
  - Complexidade no Espaço: Quanta memória necessita para fazer a pesquisa?
  - Optimalidade: Encontra a melhor solução?
- **Tipos de Estratégias de Pesquisa:**
  - Pesquisa Não-Informada (cega)
  - Pesquisa Informada (heurística)

Pesquisa Primeiro em Largura

- **Pesquisa Primeiro em Largura (Breadth-first search)**
  - Estratégia: Todos os nós de menor profundidade são expandidos primeiro
    - Bom: Pesquisa muito sistemática
    - Mau: Normalmente demora muito tempo e sobretudo ocupa muito espaço
- **Supondo factor de ramificação b então $n=1+b+b^2+b^3+ \ldots +b^n$**
- **Complexidade exponencial no espaço e no tempo: $O(b^d)$**
- **Em geral só pequenos problemas podem ser resolvidos assim!**
- **function BREADTH-FIRST-SEARCH(problem) returns a solution or failure**

  GENERAL-SEARCH(problem,ENQUEUE-AT-END)



Pesquisa de Custo Uniforme

- **Estratégia: Expandir sempre o nó com menor custo da fronteira (medido pela função de custo da solução)**

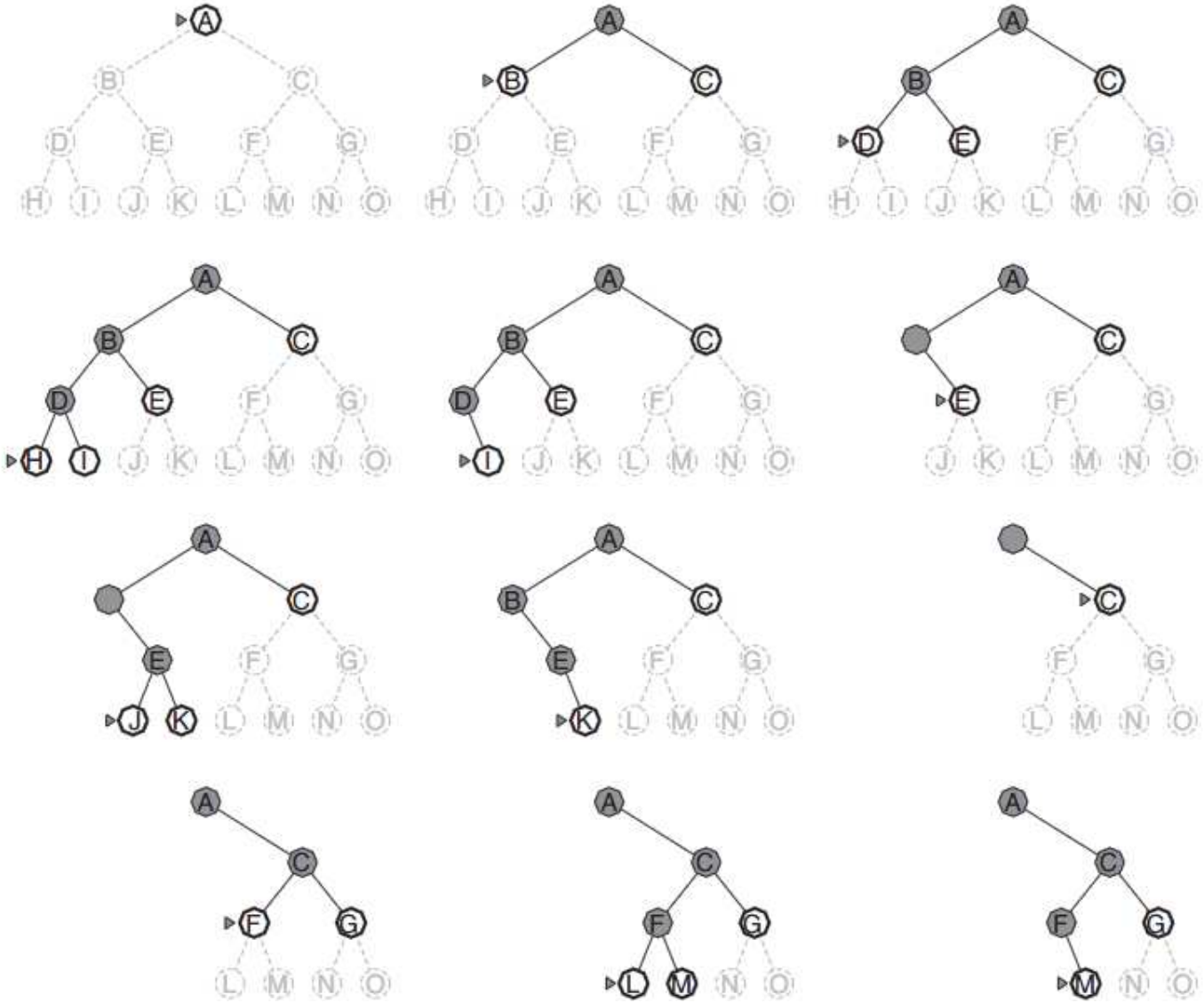
- **Pesquisa Primeiro em Largura é igual a Pesquisa de Custo Uniforme se g(n)=Depth(n)**

Pesquisa Primeiro em Profundidade

- **Pesquisa Primeiro em Profundidade (Depth-First Search)**
  - Estratégia: Expandir sempre um dos nós mais profundos da árvore
    - Bom: Muito pouca memória necessária, bom para problemas com muita soluções
    - Mau: Não pode ser usada para árvores com profundidade infinita, pode ficar presa em ramos errados
  - Complexidade no tempo $O(b^m)$ e no espaço O(bm).
  - Por vezes é definida uma profundidade limite e transforma-se em Pesquisa com Profundidade Limitada
- **function DEPTH-FIRST-SEARCH(problem) returns a solution or failure**

  GENERAL-SEARCH(problem,ENQUEUE-AT-FRONT)

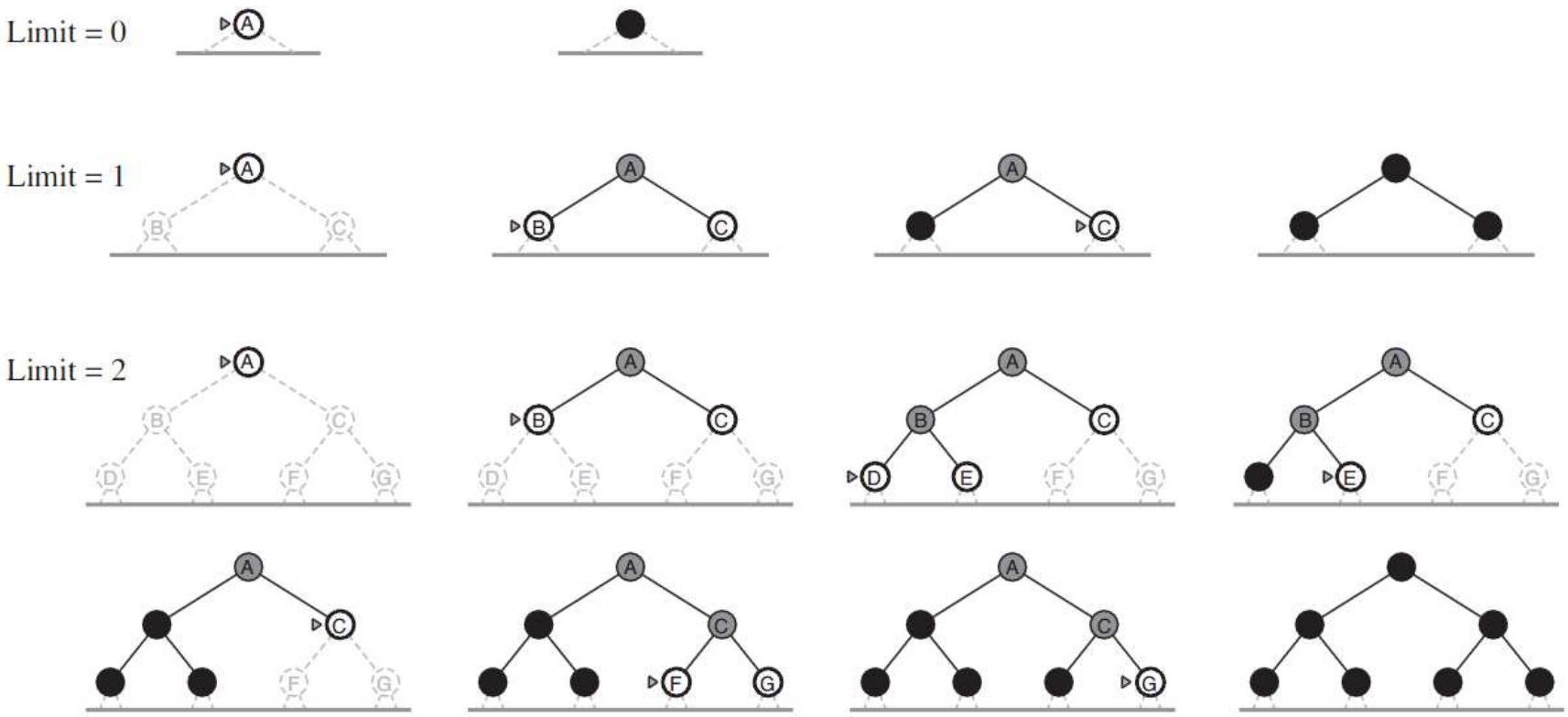
Pesquisa Primeiro em Profundidade (2)



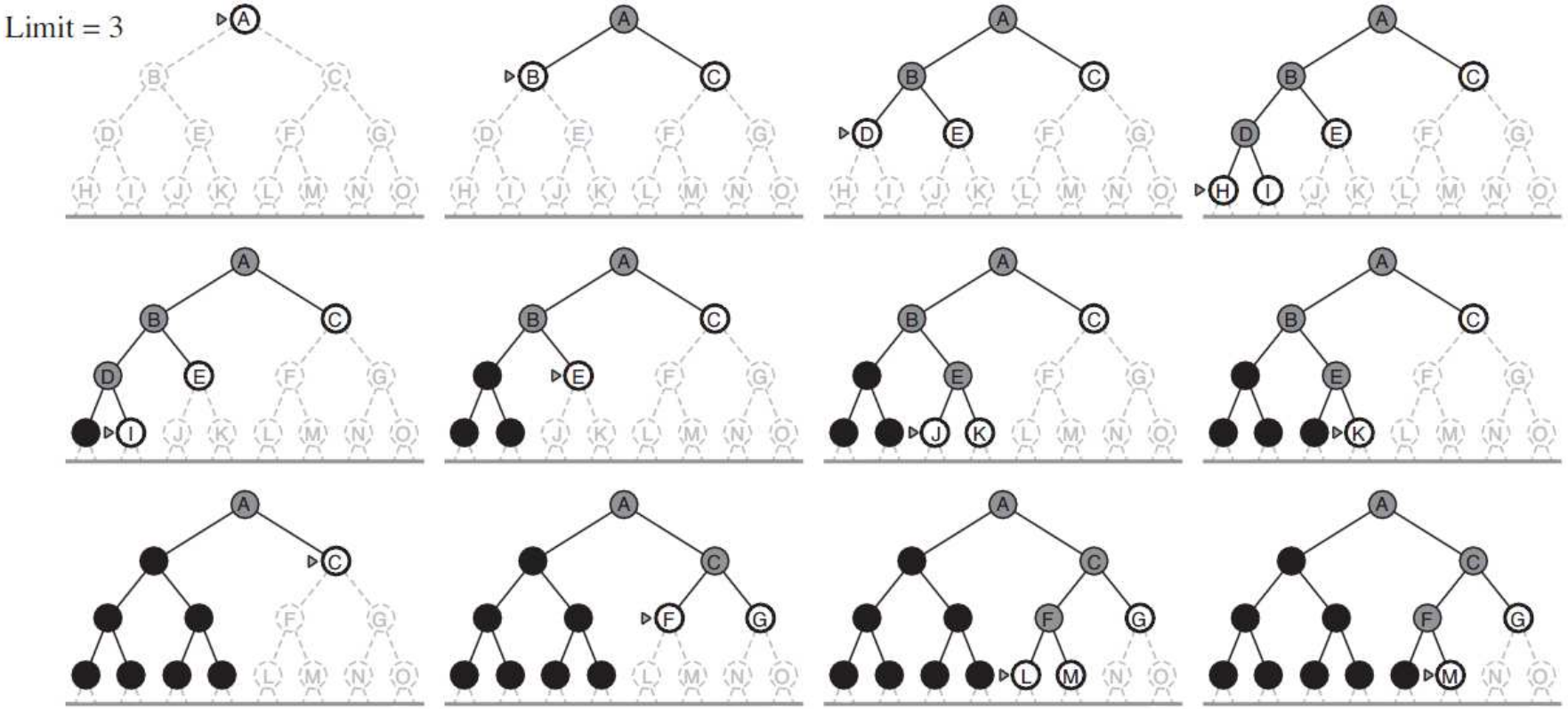
Pesquisa em Profundidade Iterativa

- Estratégia: Executar pesquisa em profundidade limitada, iterativamente, aumentando sempre o limite da profundidade
- Complexidade no tempo $O(b^d)$ e no espaço O(bd).
- Em geral é a melhor estratégia para problemas com um grande espaço de pesquisa e em que a profundidade da solução não é conhecida

Pesquisa em Profundidade Iterativa (2)

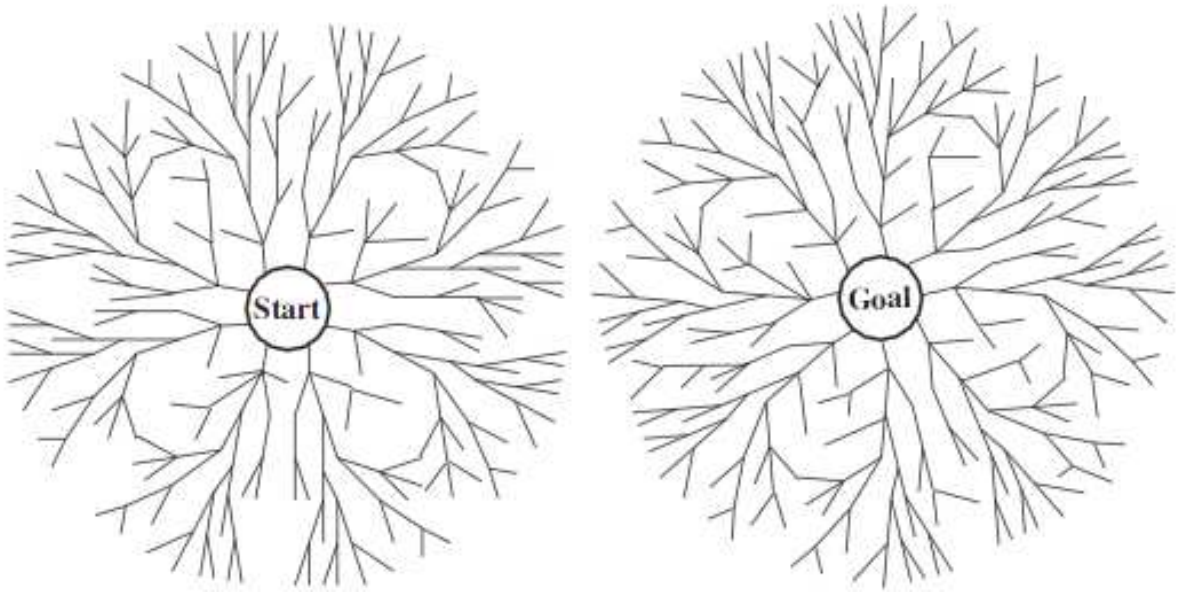


Pesquisa em Profundidade Iterativa (3)



Pesquisa Bidireccional

- **Estratégia: Executar pesquisa para a frente desde o estado inicial e para trás desde o objetivo, simultaneamente**
  - Bom: Pode reduzir enormemente a complexidade no tempo
  - Problemas: Será possível gerar os predecessores? E se existirem muitos estados objetivo? Como fazer o "matching" entre as duas pesquisas? Que tipo de pesquisa fazer nas duas metades?



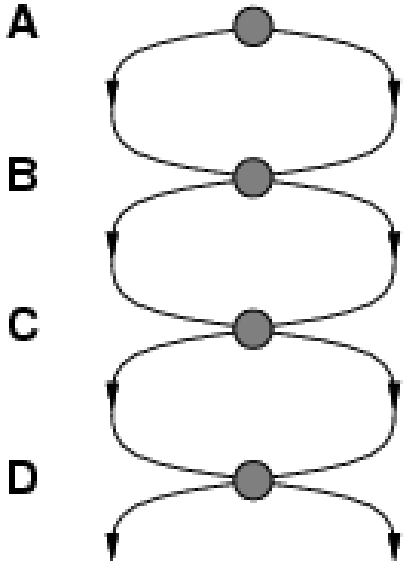
Comparação entre Estratégias de Pesquisa

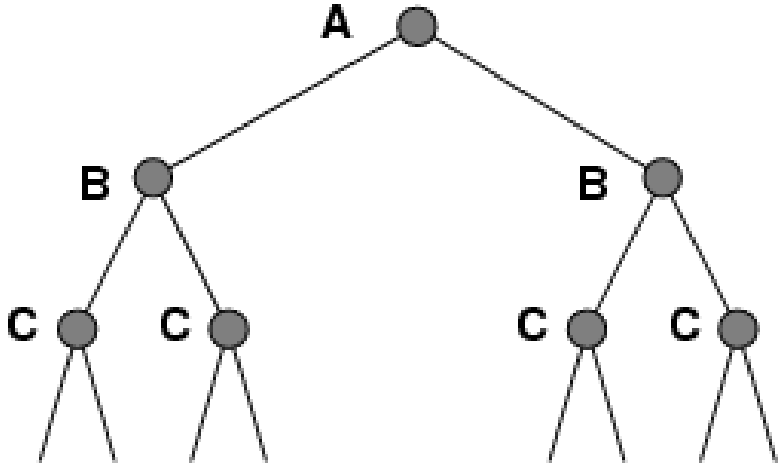
- **Avaliação das estratégias de pesquisa:**
  - B é o fator de ramificação
  - d é a profundidade da solução
  - m é a máxima profundidade da árvore
  - l é a profundidade limite de pesquisa

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Estados Repetidos

- **Falha em detetar estados repetidos pode tornar um problema linear num problema exponencial!**
- **Evitar Estados Repetidos**
  - Não voltar ao estado anterior
  - Não criar ciclos (não voltar a estados por onde passou na sequência)
  - Não usar nenhum estado repetido (será possível? Qual o custo computacional?)





Exercícios - Pesquisa

- **Formular os seguintes problemas (analisados na aula anterior) como problemas de pesquisa e resolve-los utilizando as várias estratégias de pesquisa estudadas:**
  - Missionários e Canibais (ver Aula anterior)
  - Problema dos Baldes (ver Aula anterior)
  - Torres de Hanoi (ver Aula anterior)
  - Criptogramas (ver Aula anterior)
  - 8-Rainhas e N-Rainhas (ver Aula anterior)
  - 8-Puzzle e N-Puzzle (ver Aula Anterior)

  Nota: Para todos os problemas tente que a solução seja o mais genérica possível de modo a permitir resolver versões com dados diferentes

Exercícios - Pesquisa

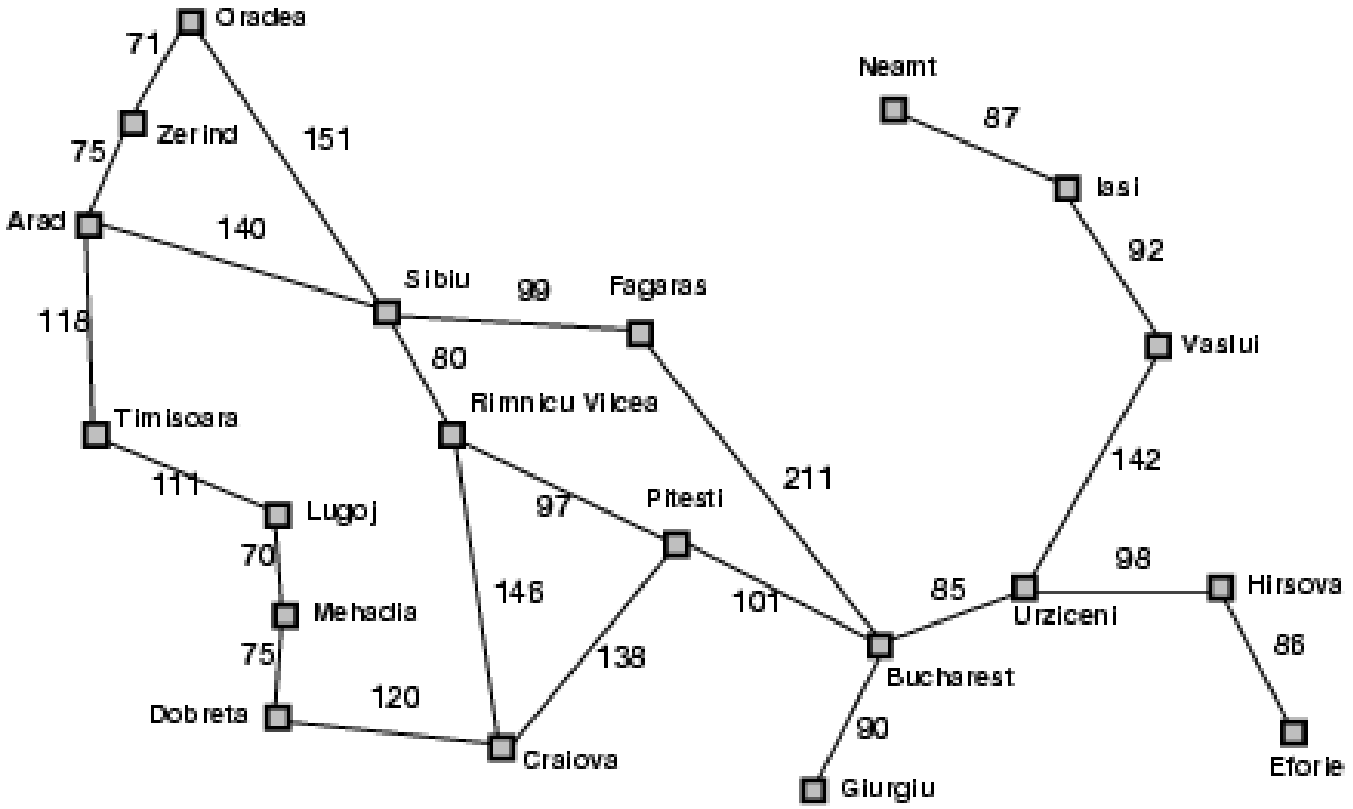
- **Formular os seguintes problemas como problemas de pesquisa e resolve-los utilizando as várias estratégias de pesquisa**
  - **Problema do Pastor:** Suponha que um Pastor leva consigo, um Lobo, um Carneiro e uma Couve. Chegado ao Rio, dispõe unicamente de um barco capaz de o transportar e transportar mais um item. O objetivo do pastor é passar para o outro lado levando os três itens mas se deixar sozinhos numa das margens, o lobo e carneiro ou o carneiro e a couve vai ter problemas!
  - **Problema do Mapa:** Colorir um mapa plano, usando 4 cores de forma a que dois vértices adjacentes não tenham a mesma cor
  - **Problema da Corrente:** O problema da corrente consiste em juntar um conjunto de elos de corrente para formar uma corrente completa. Os operadores podem abrir um elo ou fechar um elo. Na forma mais usual, o estado inicial é composto por quatro correntes com três elos e o objetivo por uma corrente circular com 12 elos.
  - **Problema do Jogo do Solitário:** Resolver o jogo solitário em que o objetivo é capturar todas as peças ficando uma peça no final no tabuleiro
  - **Problema do Solitário de Cartas:** Resolver a paciência de cartas "solitário" supondo que todas as cartas estão descobertas no início

Pesquisa Informada - Melhor Primeiro

- **Pesquisa Informada:**
  - Utiliza informação do problema para evitar que o algoritmo de pesquisa fique "perdido vagueando no escuro"!
- **Estratégia de Pesquisa:**
  - Definida escolhendo a ordem de expansão dos nós!
- **Pesquisa do Melhor Primeiro (Best-First Search)**
  - Utiliza uma função de avaliação que retorna um número indicando o interesse de expandir um nó
  - Pesquisa Gulosa (Greedy-Search) – f(n) = h(n) – estima distância à solução
  - Algoritmo A* - f(n) = g(n) + h(n) – estima custo da melhor solução que passa por n

Pesquisa Informada - Exemplo

- Estado Inicial: Arad; Objetivo: Bucharest; h(n)= distância em linha reta



Straight–line distance to Bucharest	
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	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

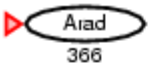
Pesquisa Gulosa (Greedy-Search)

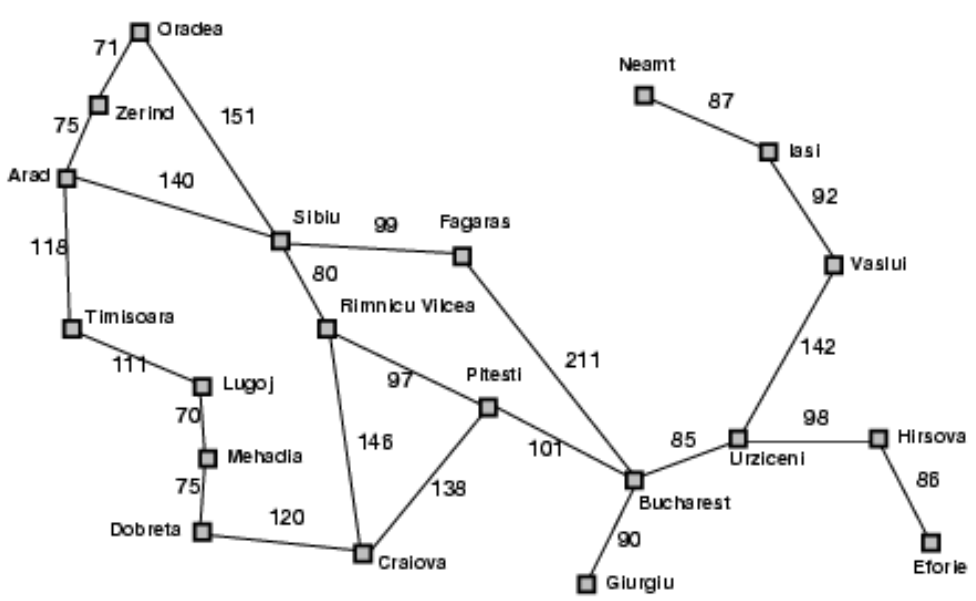
- **Estratégia:**
  - Expandir o nó que parece estar mais perto da solução
- **h(n) = custo estimado do caminho mais curto do estado n para o objetivo (função heurística)**
- **function GREEDY-SEARCH(problem) returns a solution or failure**

  return BEST-FIRST-SEARCH(problem,h)
- **Exemplo:**
  - $h_{SLD}(n)$= distância em linha reta entre n e o objetivo

Pesquisa Gulosa (Greedy-Search

- Estado Inicial: Arad; Objetivo: Bucharest; h(n)= distância em linha reta

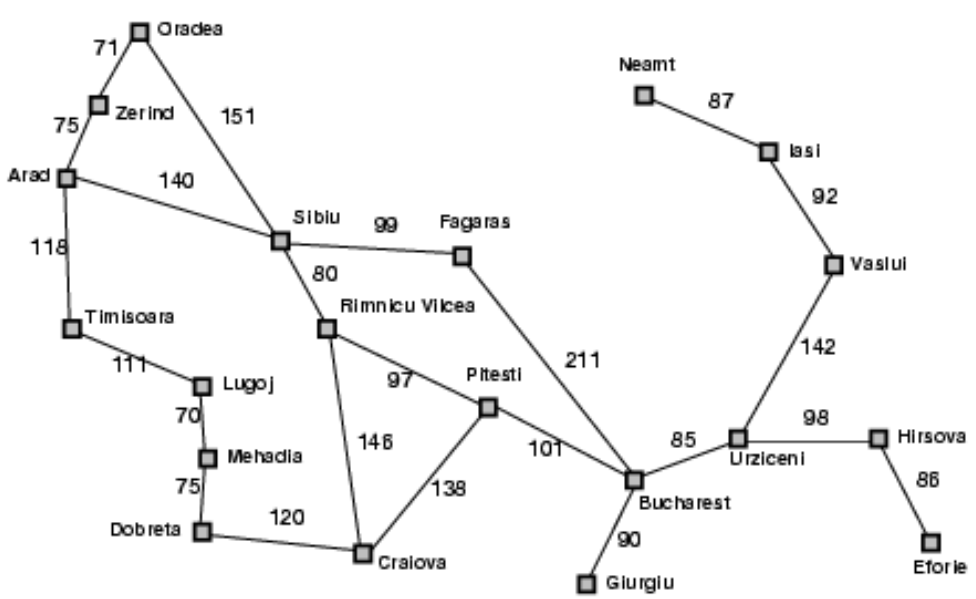




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Pesquisa Gulosa (Greedy-Search

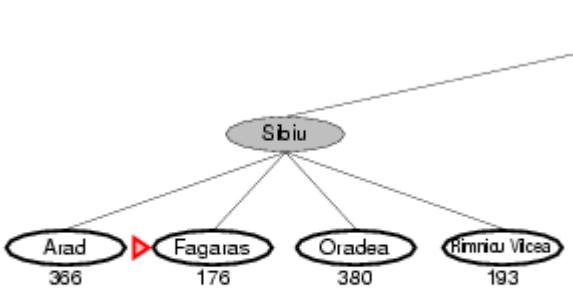
- Estado Inicial: Arad; Objetivo: Bucharest; h(n)= distância em linha reta

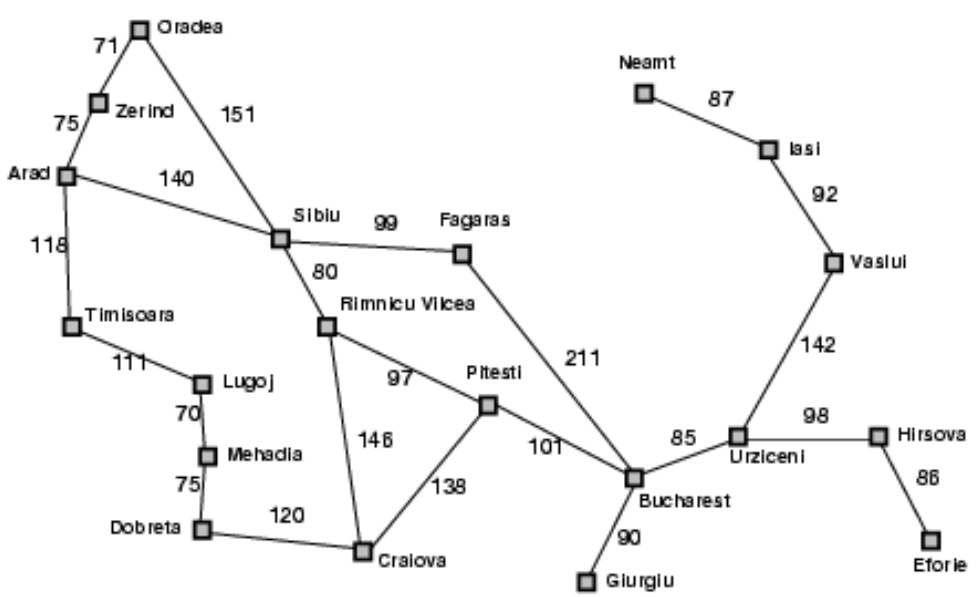


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Pesquisa Gulosa (Greedy-Search

- Estado Inicial: Arad; Objetivo: Bucharest; h(n)= distância em linha reta

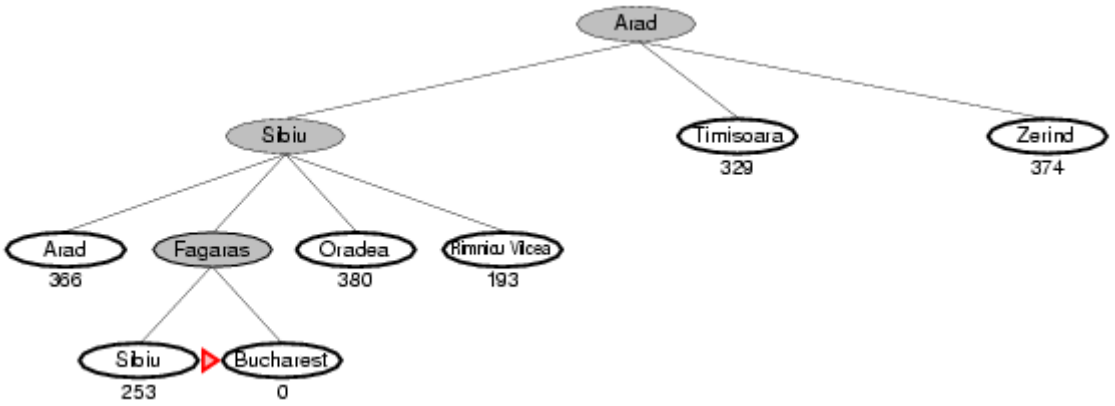




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Pesquisa Gulosa (Greedy-Search

- Estado Inicial: Arad; Objetivo: Bucharest; h(n)= distância em linha reta



Pesquisa Gulosa (Greedy-Search)

- **Propriedades:**
  - Completa? Não! Pode entrar em ciclos! (ex:, Iasi → Neamt → Iasi → Neamt → Iasi …)
  - Suscetível a falsos começos
  - Complexidade no tempo? $O(b^m)$
    - mas com uma boa função heurística pode diminuir consideravelmente
  - Complexidade no espaço? $O(b^m)$
    - Mantém todos os nós na memória
  - Ótima? Não! Não encontra sempre a solução ótima!
  - Necessário detetar estados repetidos!

Pesquisa A*

- **Estratégia:**
  - O algoritmo A* combina a pesquisa gulosa com a uniforme minimizando a soma do caminho já efetuado com o mínimo previsto que falta até a solução. Usa a função:

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

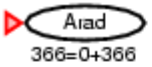
  - g(n) = custo total, até agora, para chegar ao estado n
  - h(n) = custo estimado para chegar ao objetivo (não pode sobre-estimar o custo para chegar à solução! Tem de ser otimista!)

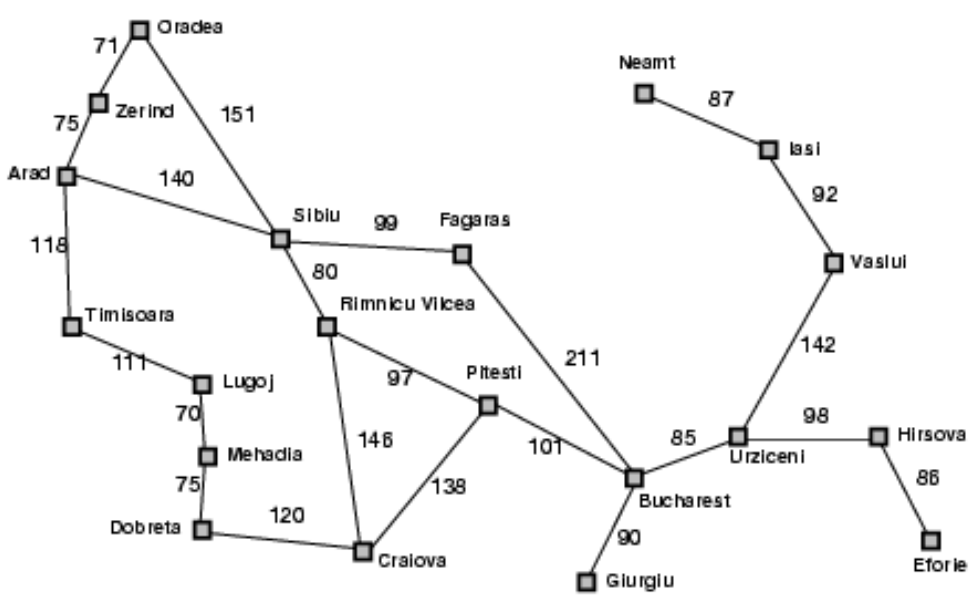
  f(n)= custo estimado da solução mais barata que passa pelo nó n
- **function A*-SEARCH(problem) returns a solution or failure**

  return BEST-FIRST-SEARCH(problem,g+h)
  - Algoritmo A* é ótimo e completo!
  - Complexidade no tempo exponencial em (erro relativo de h*comprimento da solução)
  - Complexidade no espaço: Mantém todos os nós em memória!

Pesquisa A*

- Estado Inicial: Arad; Objetivo: Bucharest; f(n) = g(n) + h(n);
- g(n) = custo até n; h(n) = distância em linha reta ao objetivo

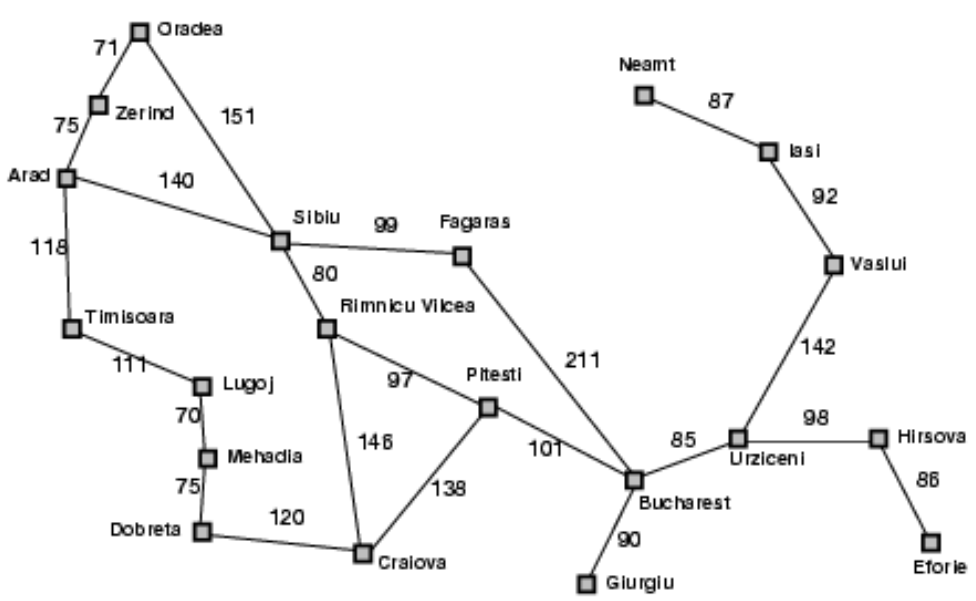




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Pesquisa A*

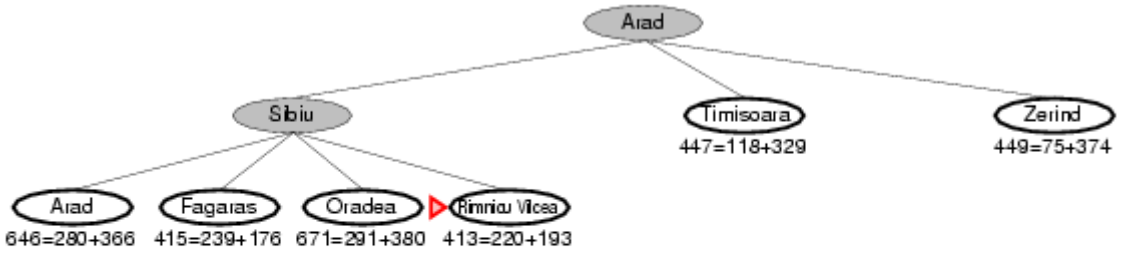
- Estado Inicial: Arad; Objetivo: Bucharest; f(n) = g(n) + h(n);
- g(n) = custo até n; h(n) = distância em linha reta ao objetivo

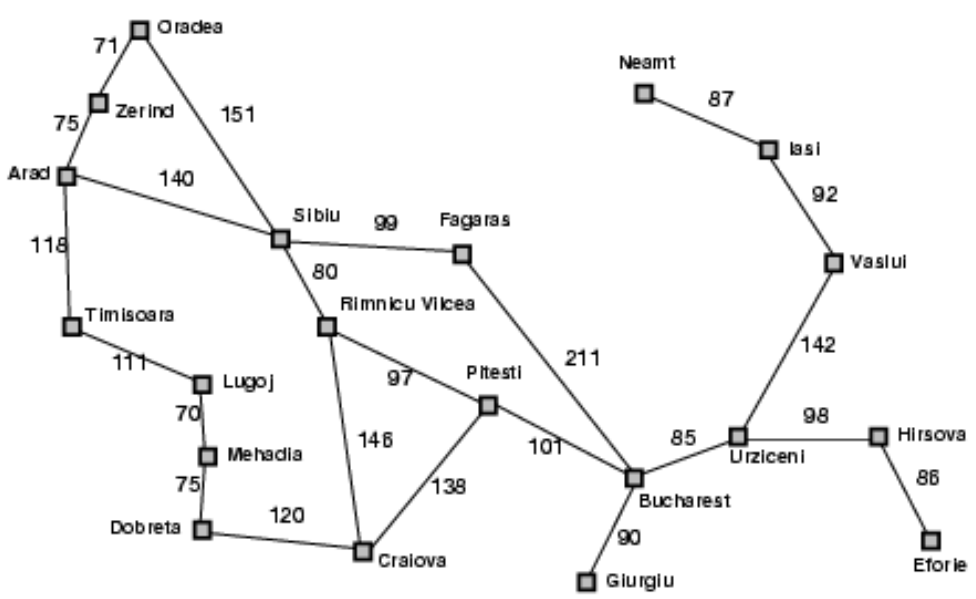


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Pesquisa A*

- Estado Inicial: Arad; Objetivo: Bucharest; f(n) = g(n) + h(n);
- g(n) = custo até n; h(n) = distância em linha reta ao objetivo

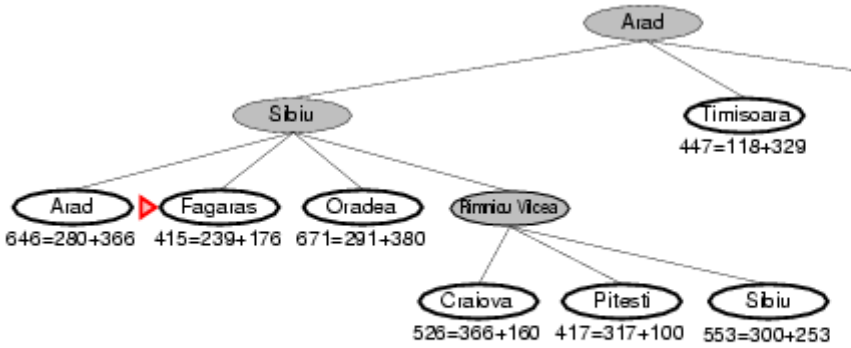


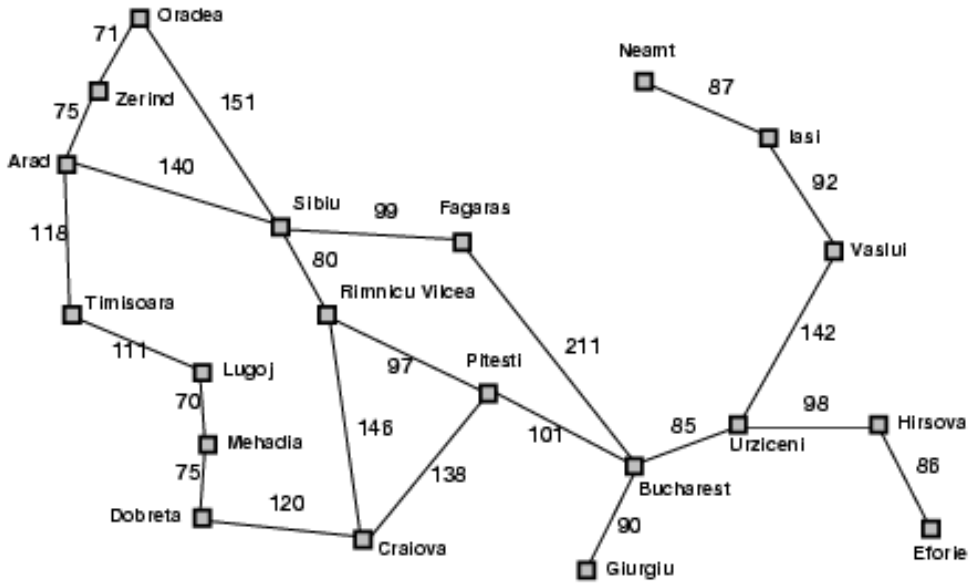


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Pesquisa A*

- Estado Inicial: Arad; Objetivo: Bucharest; f(n) = g(n) + h(n);
- g(n) = custo até n; h(n) = distância em linha reta ao objetivo

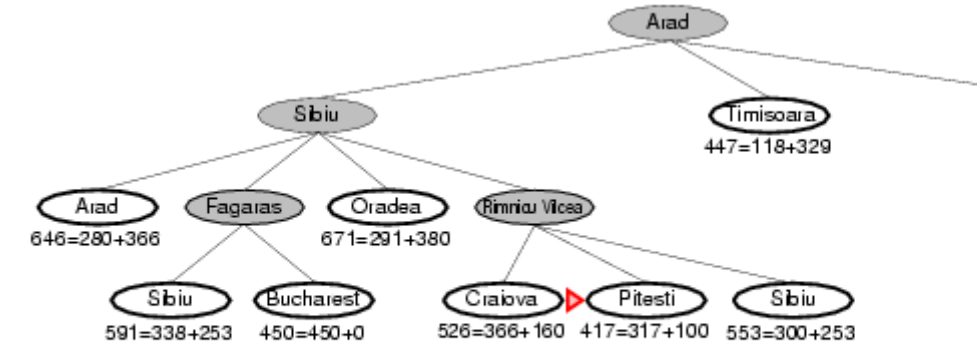


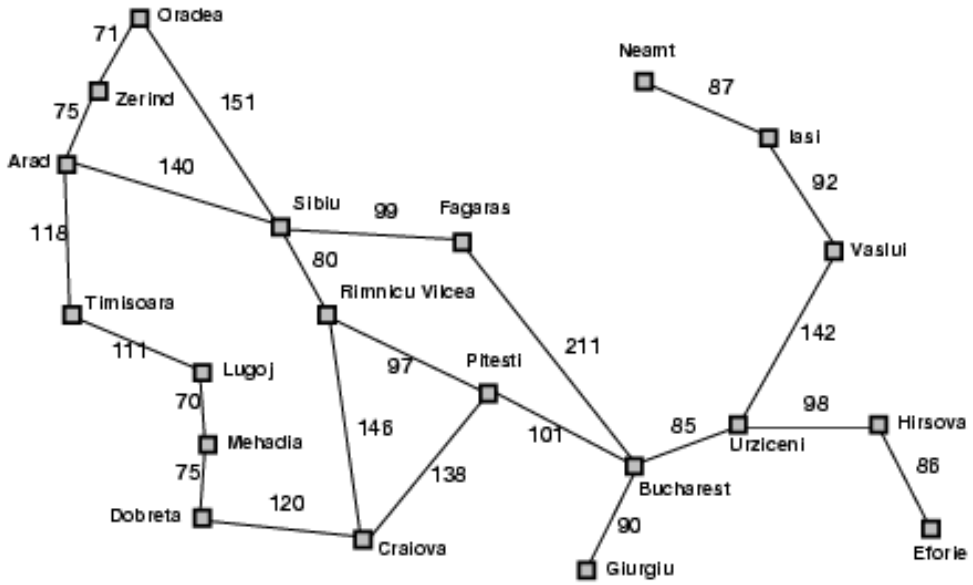


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Pesquisa A*

- Estado Inicial: Arad; Objetivo: Bucharest; f(n) = g(n) + h(n);
- g(n) = custo até n; h(n) = distância em linha reta ao objetivo

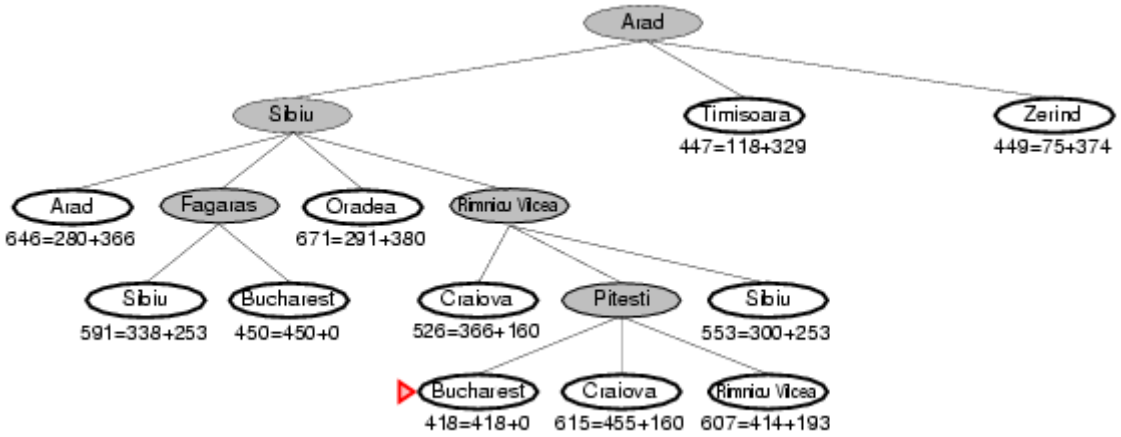




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

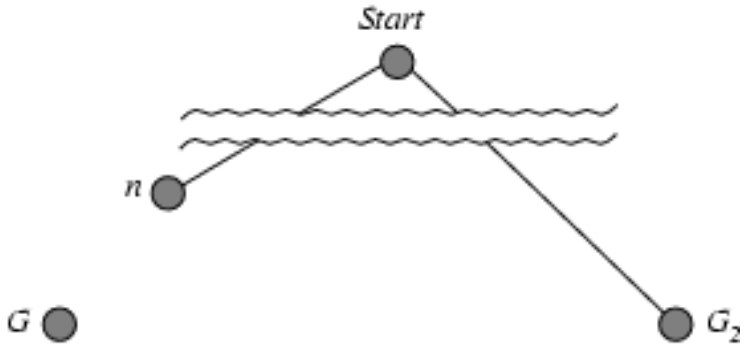
Pesquisa A*

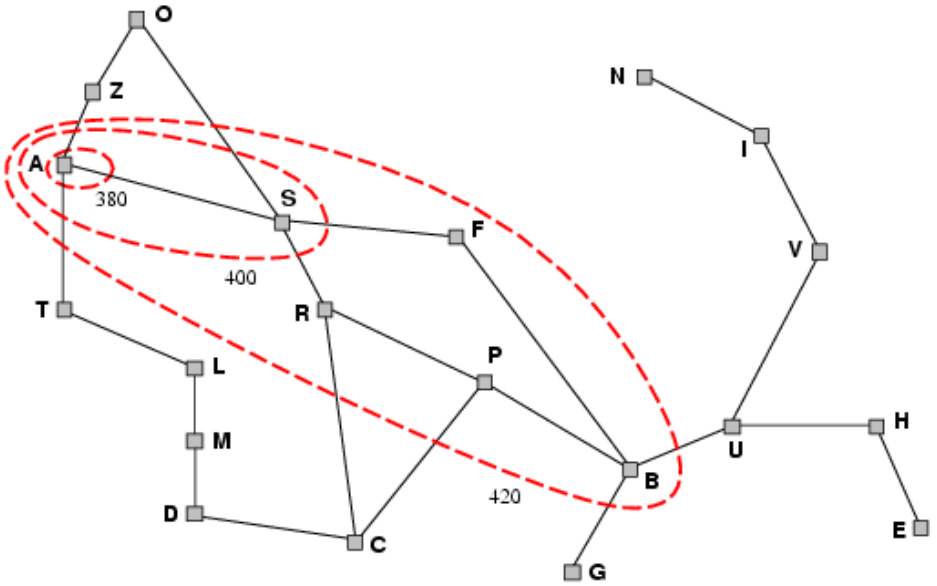
- Estado Inicial: Arad; Objetivo: Bucharest; f(n) = g(n) + h(n);
- g(n) = custo até n; h(n) = distância em linha reta ao objetivo



Prova da Optimalidade do A*

- Supondo que um objetivo sub ótimo G2 foi gerado e está na lista. Sendo n um nó ainda não expandido que leva até ao objetivo ótimo G1

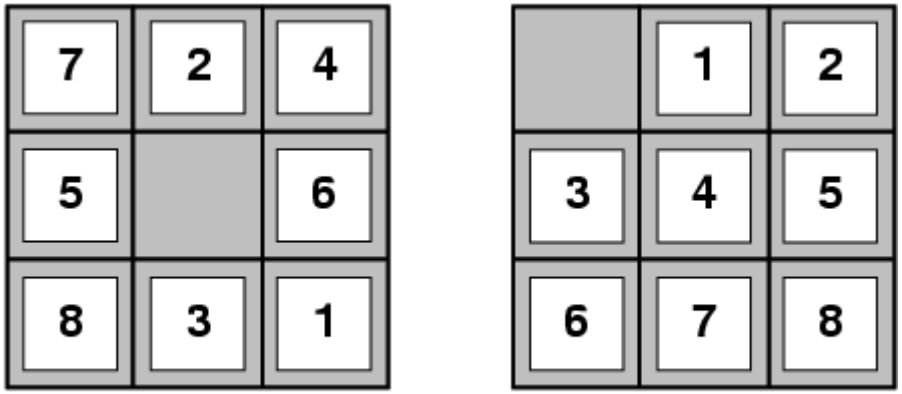




- f(G2) = g(G2) pois h(G2)=0
- f(G2) > g(G1) pois G2 é sub ótimo
- f(G2) >= f(n) pois h é uma heurística admissível
- Logo, o algoritmo A* nunca escolherá G2 para expansão!

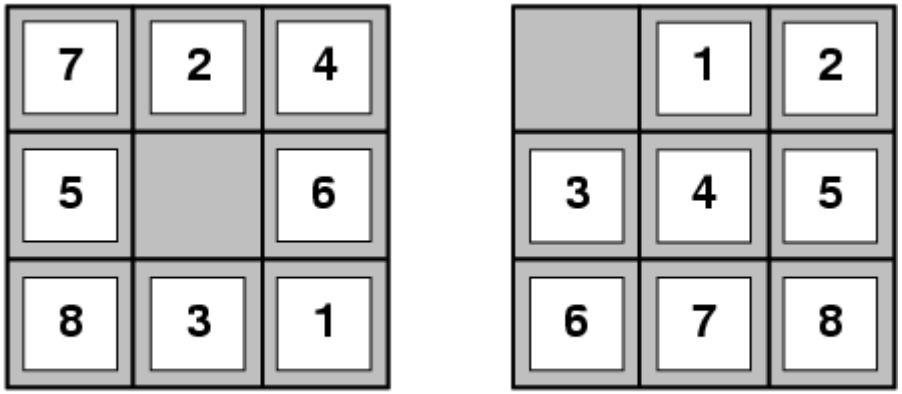
Funções Heurísticas - 8 Puzzle

- **Solução típica em 20 passos com factor de ramificação médio: 3**
- **Número de estados: $3^{20} = 3.5*10^9$**
- **Nº Estados (sem estados repetidos) = 9! = 362880**
- **Heurísticas:**
  - h1 = Nº de peças for a do sítio
  - h2 = Soma das distâncias das peças até às suas posições correctas



Funções Heurísticas - 8 Puzzle (2)

- **Relaxação de Problemas como forma de inventar heurísticas:**
  - Peça pode-se mover de A para B se A é adjacente a B e B está vazio
  - a) Peça pode-se mover de A para B se A é adjacente a B
  - b) Peça pode-se mover de A para B se B está vazio
  - c) Peça pode-se mover de A para B

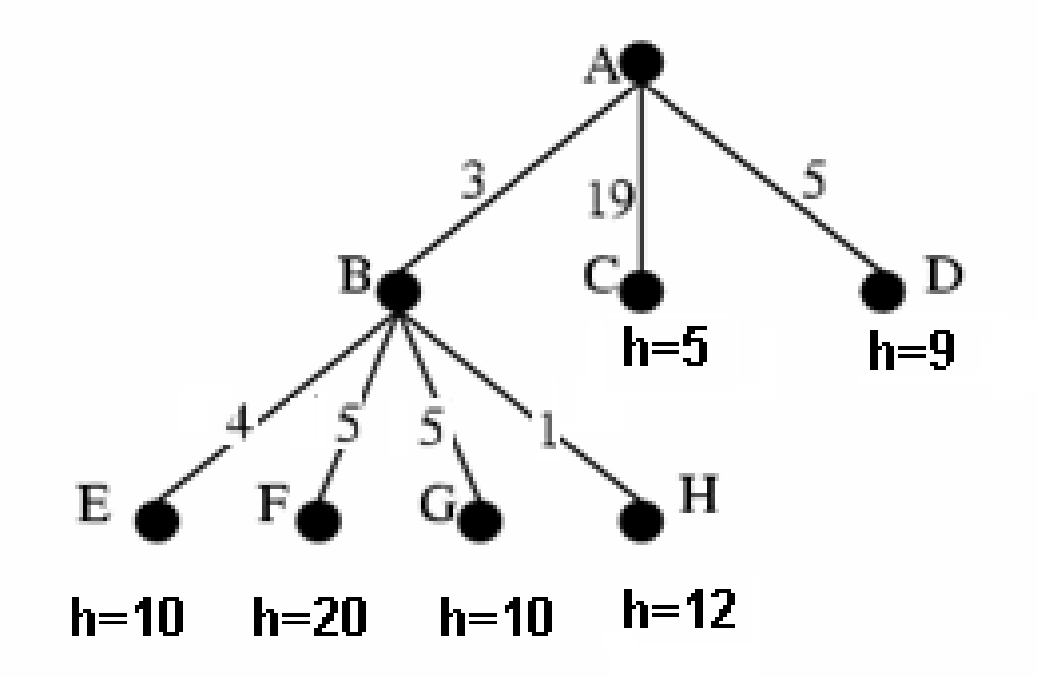


Pesquisa com Memória Limitada - IDA*/SMA*

- **IDA* - Pesquisa com Profundidade Iterativa (Iterative Deepening Search)**
  - Estratégia: Utilização de um custo limite em cada iteração e realização de pesquisa em profundidade iterativa
  - Problemas em alguns problemas reais com funções de custo com muitos valores
- **SMA* - Pesquisa Simplificada com Memória Limitada (Simplified Memory Bounded A*)**
  - IDA* de uma iteração para a seguinte só se lembra de um valor (o custo limite)
  - SMA* utiliza toda a memória disponível, evitando estados repetidos
  - Estratégia: Quando necessita de gerar um sucessor e não tem memória, esquece um nó da fila que pareça pouco prometedor (com um custo alto).

Exercício

Supondo a seguinte árvore de pesquisa em que cada arco apresenta o custo do operador correspondente, indique justificando diga qual o nó expandido em seguida utilizando cada um dos seguintes métodos: a) Pesquisa Largura; b) Pesquisa em Profundidade; c) Pesquisa de Custo Uniforme; d) Pesquisa Gulosa; e) Pesquisa A*



Exercícios

Comente a seguinte afirmação: “Uma estratégia de pesquisa óptima garante a resolução de um problema de pesquisa de modo óptimo isto é, da forma mais eficiente possível encontrando a solução no mínimo tempo possível.”

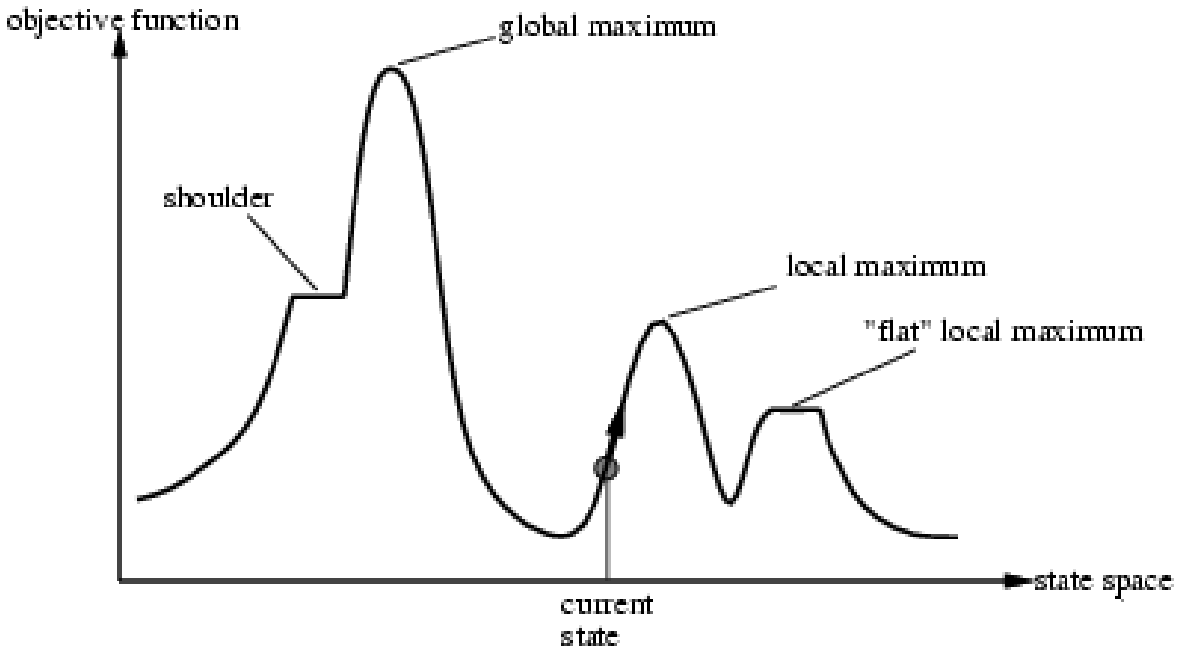
Qual o comportamento do algoritmo A* se h(n) for uma heurística perfeita que retorna sempre o custo exacto para o objectivo? Justifique.

Algoritmos de Melhoria Iterativa

- **Em muitos problemas de otimização, o caminho para o objetivo é irrelevante! O objetivo é ele mesmo a solução!**
- **Espaço de Estados = conjunto das configurações completas!**
- **Algoritmos Iterativos mantém um único estado (corrente) e tentam melhora-lo!**
- **Algoritmos de Melhoria Iterativa:**
  - Pesquisa Subida da Colina (Hill-Climbing Search)
  - Arrefecimento Simulado (Simulated Annealing)
  - Pesquisa Tabu (Tabu Search)
  - Algoritmos Genéticos (Genetic Algorithms)
  - Particle Swarm Optimization
  - Ant Colony Optimization
- **Estratégia: Começar como uma solução inicial do problema e fazer alterações de forma a melhorar a sua qualidade**

Pesquisa Subida da Colina (Hill-Climbing Search)

- **Problema: Dependendo do estado inicial pode ficar "preso" num mínimo local!**



Pesquisa Subida da Colina (Hill-Climbing Search) (2)

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Arrefecimento Simulado ("Simulated Annealing")

- **Estratégia: Escapar do mínimo local permitindo alguns "maus" movimentos mas gradualmente diminuindo a sua dimensão e frequência!**

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(Δ E/T)
```

Exercício

1) Suponha o seguinte jogo (para 1 jogador) em que o tabuleiro é constituído por 6 casas e 6 fósforos. O objetivo do jogo é colocar um fósforo em cada um dos quadrados (tal como é demonstrado na figura). Para tal, em cada jogada, o jogador pode:
   - Movimentar 1 fósforo de uma casa para outra casa que esteja à sua direita
   - Movimentar 2 fósforos de uma casa para outra casa que esteja à sua esquerda
   - Movimentar 2 ou 3 fósforos de uma casa para outra que esteja acima ou abaixo dela

a) Formule o problema como um problema de pesquisa

b) Partindo do estado representado abaixo, desenhe as árvores de pesquisa utilizando as estratégias primeiro em largura e primeiro em profundidade (considere primeiro os movimentos a partir do quadrado 1, depois a partir do 2 e assim sucessivamente).

c) Represente graficamente o espaço de estados (ignore estados em que um quadrado tenha mais de 3 fósforos).

d) Defina duas funções heurísticas que lhe permitam aplicar o algoritmo A* ao problema. Explique em que consiste o algoritmo A* e aplique-o para resolver o problema.

1 1 1		1 2 1
1 1 1	Estado Inicial	0 2 0 Alinea b)

Exercícios - Resolução de Problemas por Pesquisa

- 1) Implemente usando uma linguagem convencional ou a linguagem Prolog, os algoritmos apresentados de pesquisa geral (e as estratégias de pesquisa descritas), o algoritmo de pesquisa do melhor primeiro: pesquisa gulosa e algoritmo A*.

- 2) Implementar os algoritmo da pesquisa subida da colina, arrefecimento simulado e pesquisa tabu utilizando Prolog ou uma linguagem convencional (C++, Java ou Pascal)

- 3) Aplique os algoritmos implementados ao problema do problema das N-Rainhas e ao Puzzle-8 e compare os resultados obtidos.

U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Artificial Intelligence/ Inteligência Artificial

Lecture 3: Solving Search Problems

Luís Paulo Reis

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.
Associate Professor at DEI/FEUP – Informatics Engineering Department,
Faculty of Engineering of the University of Porto, Portugal
Vice-President of APPIA – Portuguese Association for Artificial Intelligence