CC7711

Inteligência Artificial e Robótica

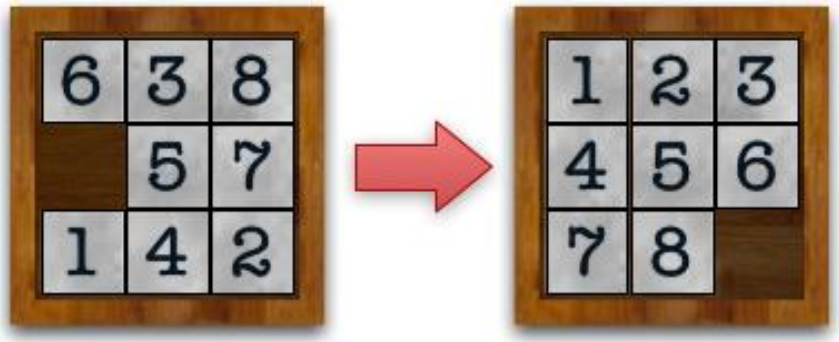
Prof. Dr. Flavio Tonidandel

Mecanismos de Busca – parte 1

CC7711 - Inteligência Artificial e Robótica

Como resolver um problemas em I.A. ?

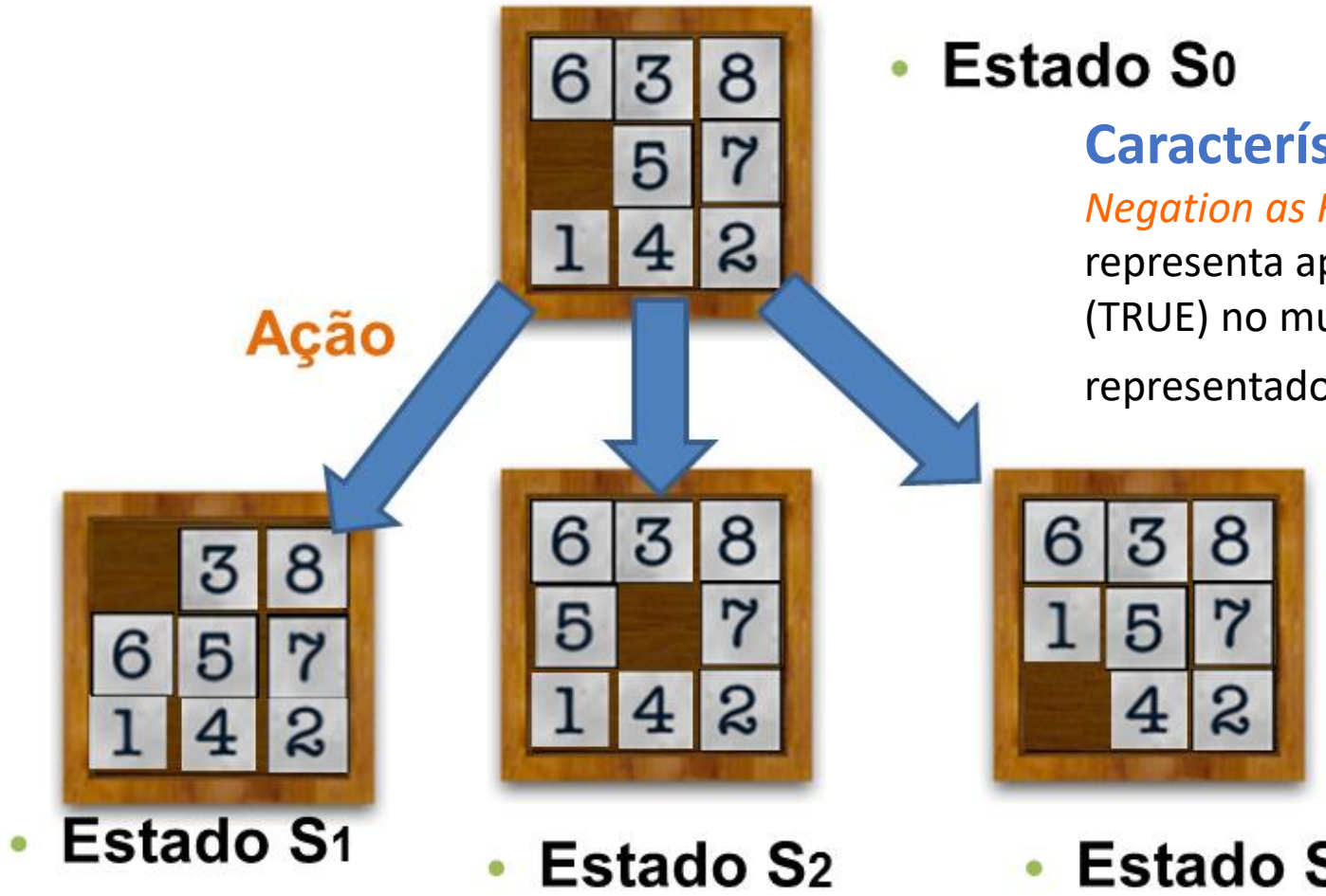
- Exemplo (Jogo dos oito)



- Qual a sequência para sair do estado inicial e chegar no estado final desejado ?

- Usaremos Mecanismos de Busca para solucionar este problema

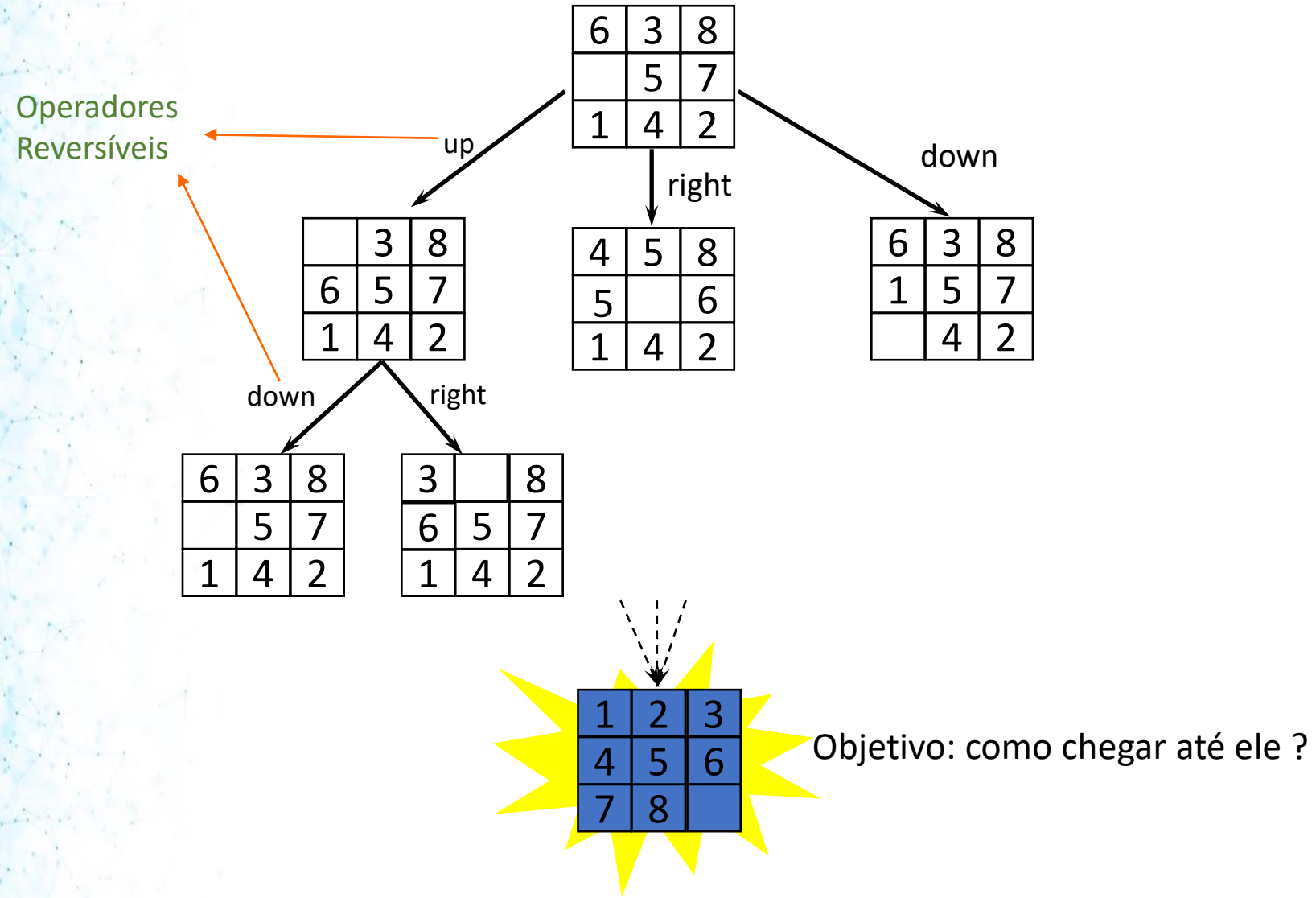
Como resolver um problemas em I.A. ?



Características dos Estados

Negation as Failure: O estado representa apenas o que é verdade (TRUE) no mundo. O que não estiver representado é considerado falso.

Ações: movimento do espaço vazio



Espaço de Estados

Um grafo pode ser usado para representar um espaço de estados, onde:

- Os nós correspondem a situações de um problema
- As arestas correspondem a movimentos permitidos ou ações ou passos da solução
- Um dado problema é solucionado encontrando-se um caminho no grafo

Um problema é definido por

- Um espaço de estados (um grafo)
- Um estado (nó) inicial (razi da árvore de busca)
- Uma condição de término ou critério de parada; estados (nós) terminais são aqueles que satisfazem a condição de término

Se não houver custos, há interesse em soluções de caminho mínimo. No caso em que custos são adicionados aos movimentos normalmente há interesse em soluções de custo mínimo da raiz até o objetivo

Mecanismos de Busca

O que é uma busca ?

- Uma busca visa encontrar uma solução para um problema através de um mecanismo algorítmico que procura no espaço de soluções.

Como é feita essa busca ?

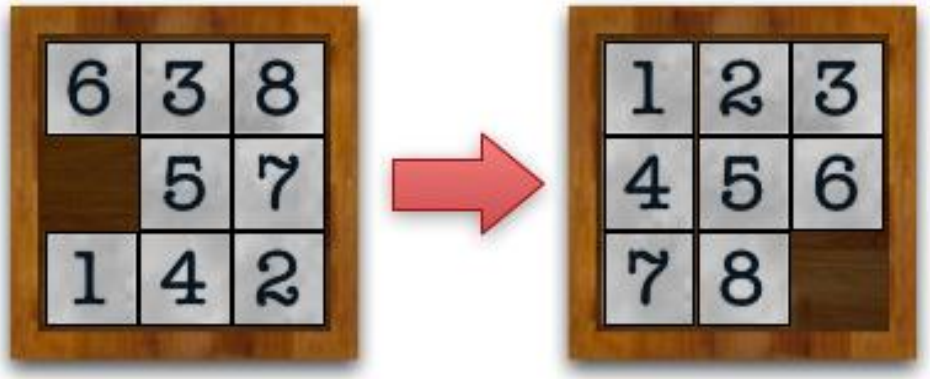
- A busca é feita em uma árvore (ou grafo) de busca. O espaço de busca é o conjunto de soluções. Essas soluções são, geralmente, representadas por estados.
- Essas soluções em forma de estados configuram um espaço de estados

O que é, então, um estado ?

- O estado é uma descrição de uma situação (solução) do domínio ao qual o mecanismo esta sendo aplicado.

Exemplos: Estados e Ações

- **JOGO DOS OITO**

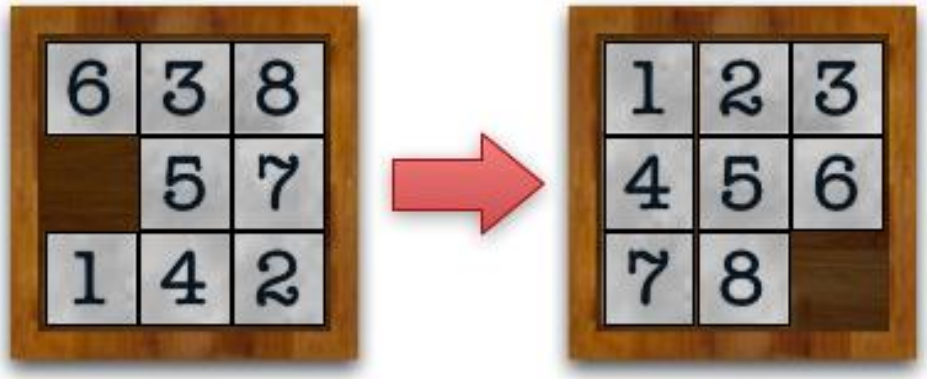


Estado Inicial Estado Final

Como representar os estados ?
Como representar as ações (operadores) ?
Qual é o estado final ?
Qual é o Custo do caminho ?

Exemplos: Estados e Ações

- **JOGO DOS OITO**



Estado Inicial Estado Final

Como representar os estados ?

Vetor : [6,3,8,*,5,7,1,4,2]
Lógica : on_place(1,6); on_place(2,3); on_place(3,8);...
Ou uma Matriz

Como representar os operadores ?

Movimenta-se o espaço vazio (*)
Vetor : Se o estado for um vetor, a mudança ocorre entre * (vazio) e os valores do vetor
Ação **subir:** [6,3,8,*,5,7,1,4,2] → [*,3,8,6,5,7,1,4,2]

Lógica : Se o estado for Lógica, precisa substituir
- Remover: On_place(1,6) e on_place(4,*)
- Inserir: on_place(1,*) e on_place(4,6)

Matriz : Se matriz os operadores funcionam como nos vetores.

Qual é o estado final? Estado é dado

Qual é o custo do caminho ? Cada ramo custa 1

Exemplos: Estados e Ações

- **Missionários & Canibais**

Como representar os estados ?
Como representar as ações (operadores) ?
Qual é o estado final ?
Qual é o Custo do caminho ?

Exemplos: Estados e Ações

- **Missionários & Canibais**

Como representar os estados ?
um estado é uma sequência ordenada de três números representando o número de missionários, canibais e botes em cada margem. Assim o estado inicial é [3,3,1,0,0,0]

Margem Esquerda | Bote | Margem Direita

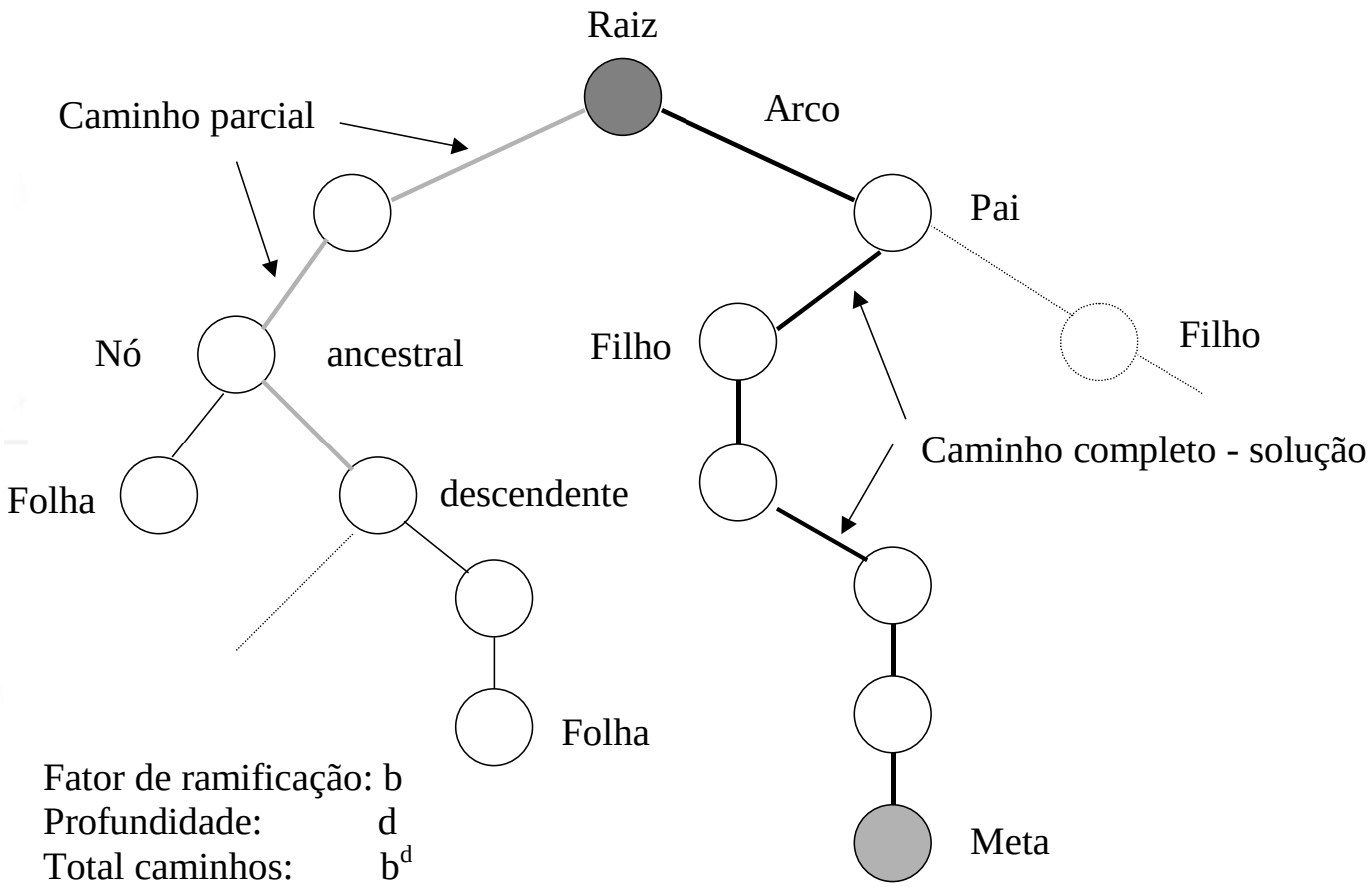
Como representar as ações (operadores) ?
São 5 operadores (M; C; M+C; 2M; 2C), que atuam de cada lado da margem.

Qual é o estado final ? [0,0,0,1,3,3]

Qual é o Custo do caminho ? Custo 1 (numero de travessias)

Árvore de Busca

Detalhamento de uma árvore



Fator de ramificação: b
Profundidade: d
Total caminhos: b^d

Avaliação de um Mecanismo de Busca

Completude(completeza) - *completeness:*

- a estratégia **sempre** encontra uma solução quando existe alguma

Custo do tempo:

- Qual a ordem de **tempo** gasto para encontrar uma solução?

Custo de memória:

- Qual a ordem de uso de **memória** para realizar a busca?

Qualidade/otimalidade (*optimality*):

- a estratégia encontra **a melhor solução** quando existem soluções diferentes?

O que difere uma busca de outra ? : a ordem de expansão dos nós

Buscas Cegas

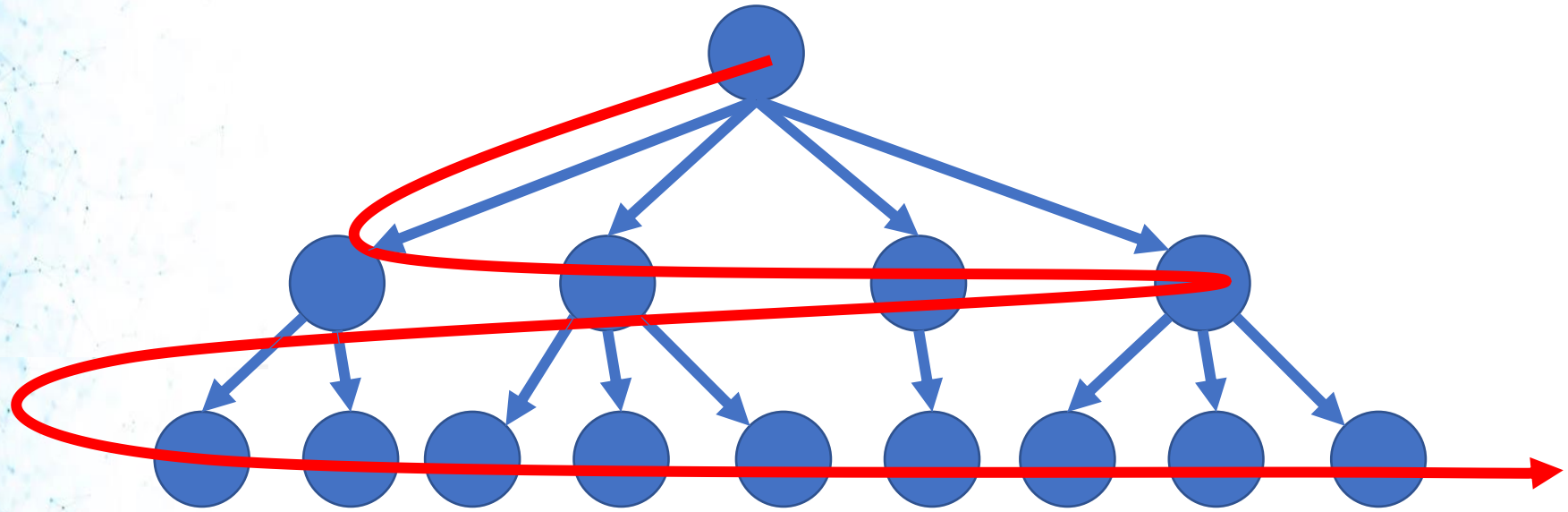
Buscas Cegas

- As buscas cegas são buscas de expansão de nós genéricas sem informações específicas de domínio.

- Exemplos:
  - Busca em Largura
  - Busca em Profundidade

- Nota de Aula:
  - Para exemplificar os algoritmos, iremos usar uma pilha ou uma fila de estados:
    - {A,B,C,D,E} é uma pilha/fila com os estados A,B,C,D e E com A no topo da lista

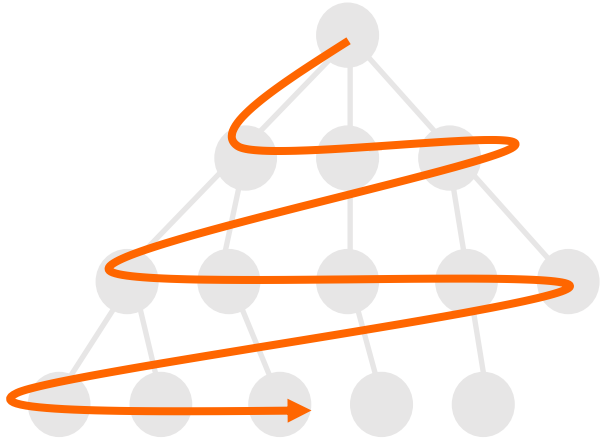
Busca em Largura

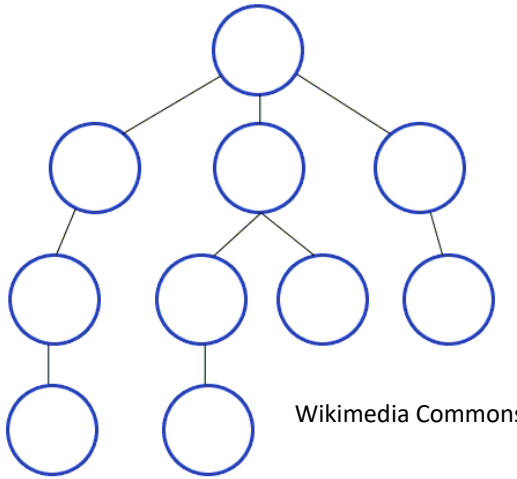


Busca em Largura

Método de Expansão do Nó:

- **A partir do Nó inicial:**
  - **Abre todos os nós da primeira profundidade**
  - **Abre todos os nós da segunda profundidade**
  - **...**





Wikimedia Commons

FILA => {A} ➔ {B,C} ➔ {C,D,E,F} ➔ {D,E,F,G}

Algoritmo: Expande o nó da cabeça da fila, removendo-o e inserindo seus nós filhos ao final da fila.

Retorna: Estado solução no topo da fila ou quando fila={} retorna falha

Características da Busca em largura

Breadth First Search (BFS):

- **É completa ?** Sim. Sempre acha uma solução se existir
- **É ótima?** Sim. Acha o menor caminho **(menor profundidade na árvore)**
  - Quando os caminhos possuem custos diferentes, a solução de menor caminho pode não ser a solução ótima de menor custo.
- **Fator de Ramificação:** todos os nós gerados a partir de um nó (b)
- **Custo de Tempo:** com fator de ramificação = b e a solução estiver no nível = d, então custo = $O(b^d)$. *EXPONENCIAL !!!*
- **Custo de Memória:** Os nós que ainda não foram expandidos devem permanecer na pilha (memória)

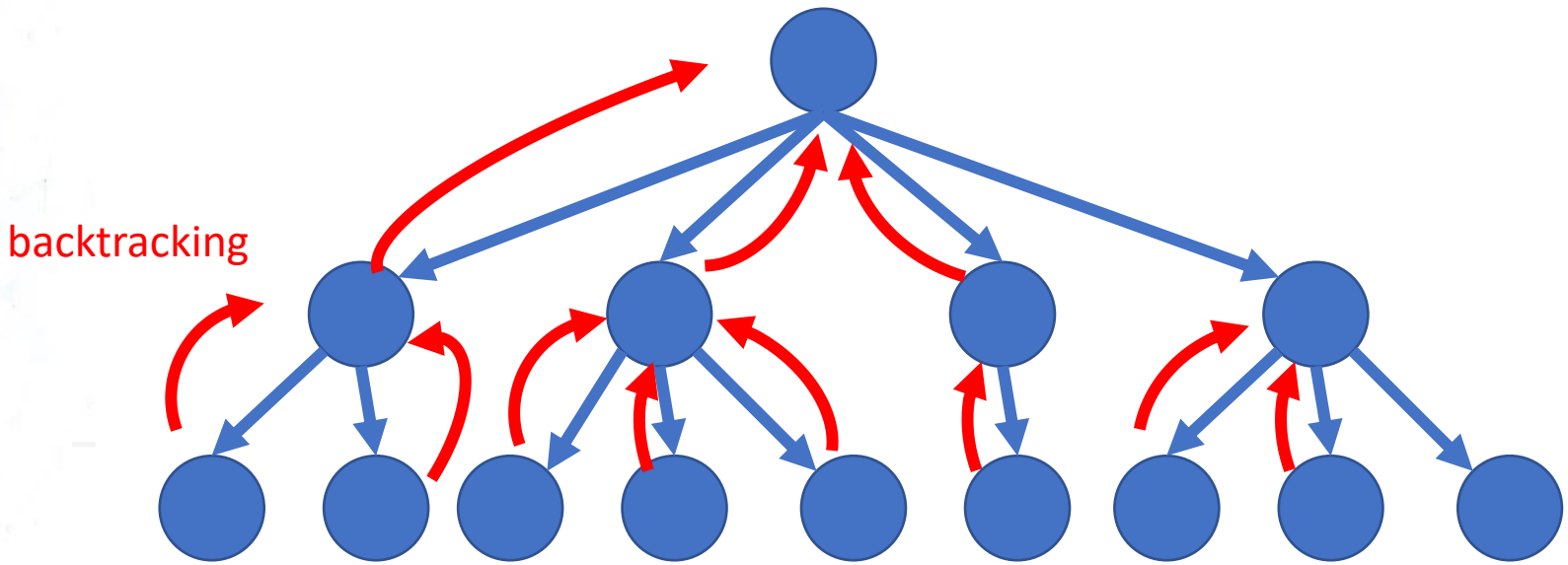
Custo do Busca em Largura (BFS)

Exemplo:

- fator de expansão b = 10
- 1.000 nós gerados por segundo
- cada nó ocupa 100 bytes

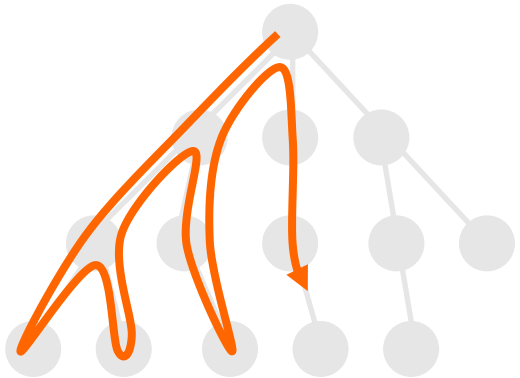
Profundidade	Nós	Tempo	Memória
0	1	1 milissegundo	100 bytes
2	111	0.1 segundo	11 quilobytes
4	11111	11 segundos	1 megabytes
6	10^6	18 minutos	111 megabytes
8	10^8	31 horas	11 gigabytes
10	10^{10}	129 dias	1 terabyte
12	10^{12}	35 anos	111 terabytes
14	10^{14}	~3500 anos	11 petabytes

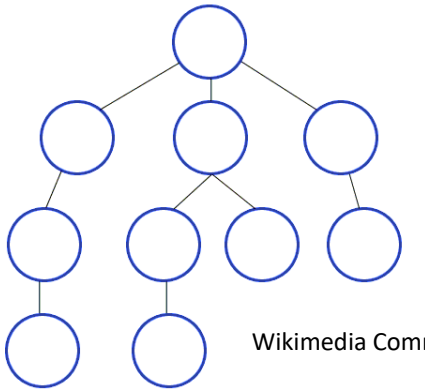
Busca em Profundidade



Busca em Profundidade

- **Método de Expansão do Nó:**
  - **A partir do Nó inicial:**
    - **Expande o primeiro nó de profundidade 1**
    - **Expande o segundo nó de profundidade 2, etc...**
    - Quando um nó final não é solução (sem filhos ou profundidade=m), o algoritmo volta para expandir os nós que ainda estão na fronteira do espaço de estados (*backtracking*)





Wikimedia Commons

Implementação Busca em Profundidade

PILHA => {A} ➔ {B,M} ➔ {C,H,L,M} ➔ {D,G,H,L,M} ➔ {E,F,G,H,L,M} ➔ {F,G,H,L,M} ➔ {G,H,L,M} ➔ {H,L,M} ➔ {I,L,M} ➔ {J,K,L,M} ➔ {K,L,M} ➔ {L,M} ➔ {M} ...

Seta ➔ indica *backtracking*

Algoritmo: Expande o nó da cabeça da pilha, removendo-o e inserindo seus nós filhos no topo.

Retorna: Estado solução no topo da pilha ou quando pilha={} retorna falha

Características da Busca em Profundidade

Depth First Search (DFS):

- **É completa ?** Não. Pois precisa-se definir um limite de profundidade m senão o algoritmo pode entrar em *looping*.
- **É ótima ?** não há garantia.
- **Fator de Ramificação: no pior caso** todos os nós gerados a partir de um nó (b)
- **Custo de Tempo: no pior caso** $O(b^m)$.
- **Custo de Memória:**

  mantém na memória o caminho que está sendo expandido no momento, e os nós irmãos dos nós no caminho (para possibilitar o *backtracking*). Isso dá b.m nós.

Para problemas com várias soluções, esta estratégia pode ser bem mais rápida do que busca em largura.

Esta estratégia deve ser evitada quando m>>d ou mesmo quando m não é conhecido.

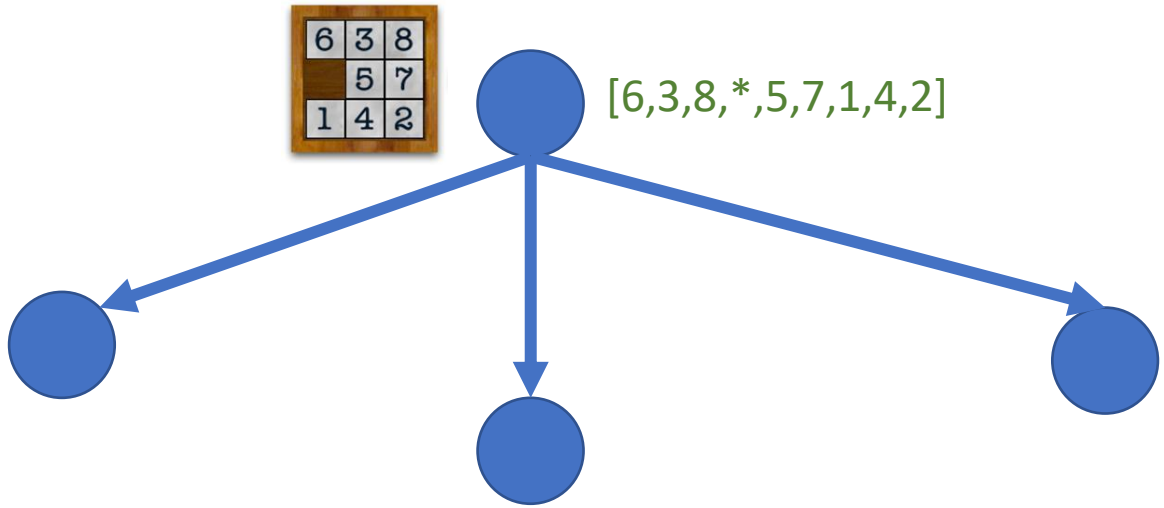
Análise DFS X BFS

- **Exemplo:**
  - fator de expansão b = 10
  - 1.000 nós gerados por segundo
  - cada nó ocupa 100 bytes

Profundidade	Nós	Tempo	Memória
DFS para m=12	Entre 120 e 10^{12}	Entre 1s e 35 anos	12 Kbytes
BFS p/ d = 12	10^{12}	35 anos	111 terabytes

  - Mas o BFS te dará a solução ótima e o DFS não necessariamente

Como implementar ?



[[*,3,8,**6**,5,7,1,4,2] , [6,3,8,**1**,5,7,*,4,2] , [6,3,8,**5**,*,7,1,4,2]]

Se implementar uma

FILA → Busca em largura
PILHA → Busca em profundidade

Até encontrar: [1,2,3,4,5,6,7,8,*]

Buscas Informadas

Loopings...

Quando ocorrem os loopings ?

- Quando um estado já visitado volta a aparecer em uma profundidade maior na árvore.
- Isto ocorre principalmente quando há operadores **reversíveis**, que podem voltar para um estado anterior. Ex. Jogo-dos-oito.

Como evitar ?

- Armazenar os estados já visitados
- Consequência: Custo de Memória *O* (b^d). *-> Armazena todos os nós*
- Geralmente implementado em *hash tables (evita gastar tempo para verificar se o estado já foi visitado ou não → eficiente).*

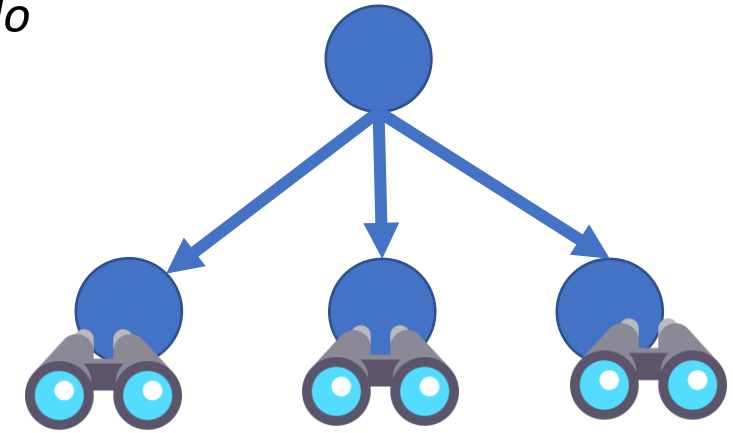
Heurísticas

- **Há como inserir conhecimento do domínio para melhorar o processo de busca ?**
  Sim. Através da definição de heurísticas.

- **O que é heurística ?** *Informação, valor e/ou quantificação de um estado quanto sua relevância ou distância de outros estados (geralmente o objetivo)*

  A heurística utiliza-se de conhecimento específico do domínio para ajudar a busca a determinar qual nó deve ser expandido

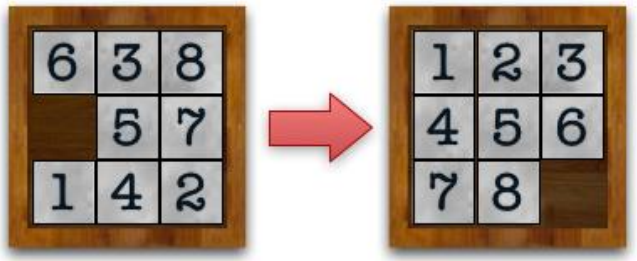
- **Métodos de Busca Heurística**
  - *Best-First Search (Melhor-Escolha)*
  - *Hill-Climbing (Subida da Colina)*
  - *A**
  - *outros*



Qual estado parece ser o mais promissor ?
O que parece levar ao objetivo ?

Buscas Heurísticas

- **Tentam evitar a explosão combinatória**
- **Pode levar a soluções "razoáveis"**
- **PROBLEMA ?**
  - **Como determinar a heurística adequada para cada problema ?**
  - A heurística deve mostrar a relevância ou o custo do nó.

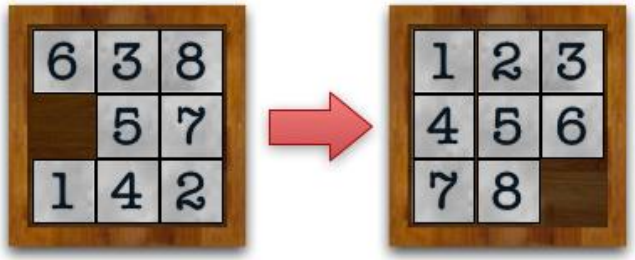


Estado Inicial Objetivo (meta)

- Não sabemos a solução. Pretendemos encontrá-la. Logo, não sabemos quantos movimentos teremos que fazer, que seria a distância real até a meta.
- Como achar uma função heurística que me fale o quanto estou próximo da meta?

Criando Heurísticas...

- Heurística 1: Número de pastilhas que estão fora de lugar: h = 8 (inclui espaço)
- Heurística 2: Número de pastilhas no lugar certo: h = 1 (inclui espaço)
- Heurística 3: Distância de Manhattan: distância horizontal + vertical de cada pastilha para o seu lugar correto: h = 16 (inclui espaço)



Estado Inicial Objetivo (meta)

CUSTO DE UM NÓ COM RELAÇÃO À META:
Custo = | valor heurístico do nó - valor heurístico da meta |

- p/ Heurística 1: Custo = |8 - 0| = 8.
- P/ Heurística 2: Custo = |1 - 9| = |-8| = 8
- p/ Heurística 3: Custo = |16 - 0| = 16

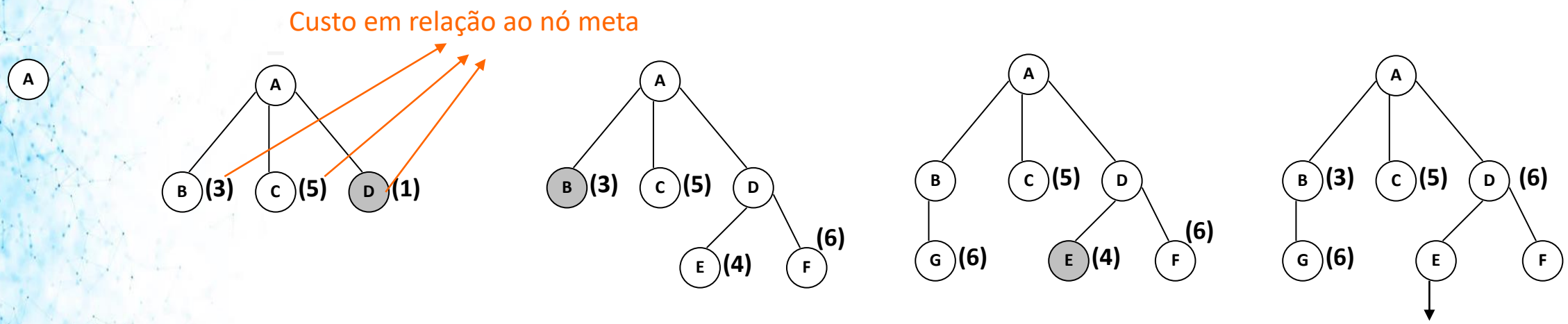
Busca Melhor-Escolha (Best First Search)

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* e suas variações

Funcionamento do Algoritmo:



Busca Melhor-Escolha

Semelhante à busca em profundidade com *backtracking*

Tenta expandir o nó mais próximo do nó final com base na estimativa de custo feita pela *função heurística h*.

- **É completa ?** Sim, desde que, como a DFS, o algoritmo controle o acesso a estados (nós) repetidos. Caso contrário pode entrar em *looping*.
- **É ótima ?** não há garantia.
- **Fator de Ramificação:** **no pior caso** todos os nós gerados a partir de um nó (b)
- **Custo de Tempo:** **no pior caso** $O(b^d)$.
- **Custo de Memória:** **no pior caso** $O(b^d)$.

Buscas mais rápidas...

Uma função heurística, quando muito bem dimensionada, permite o uso de processos de busca mais velozes.

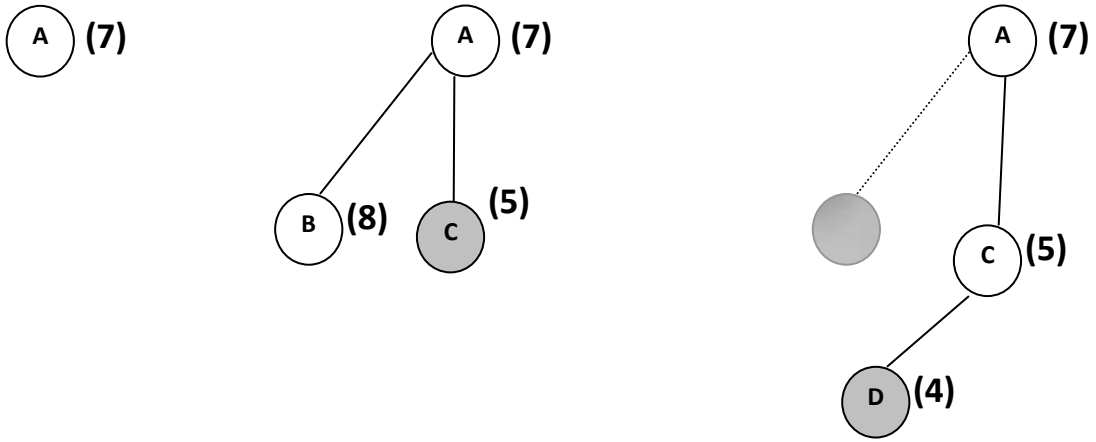
Se a heurística não for boa:

- Pode-se tomar uma decisão errada
- Fazer o sistema, na pior das hipóteses, não encontrar uma solução

- Processos de Busca Rápidos (confiam na heurística !!):
  - Subida da Encosta (*Hill-Climbing*)
  - Subida da Encosta pelo caminho mais Íngreme (*Steepest-Ascent Hill-Climbing*)
  - Subida Forçada da Encosta (*Enforced Hill-Climbing*)

Subida da Encosta (Hill-Climbing)

Funcionamento do algoritmo:

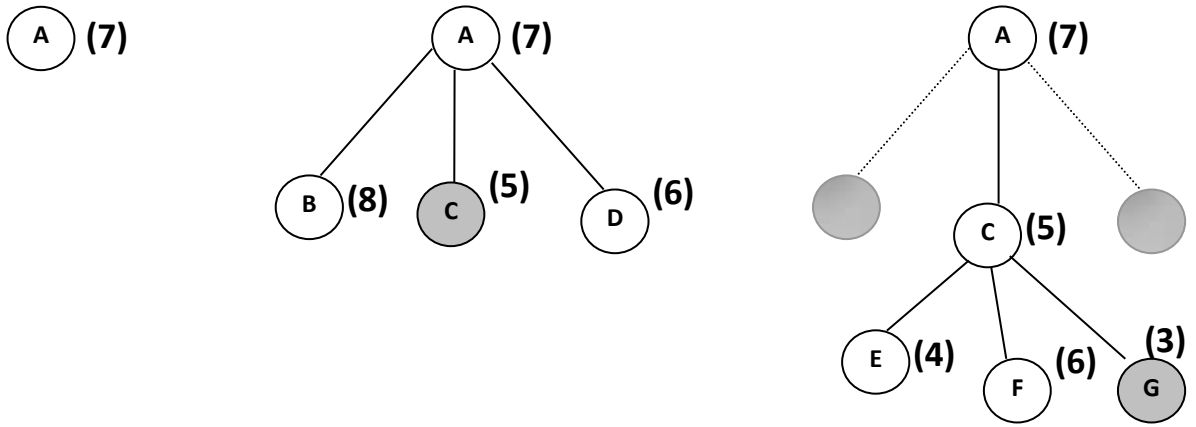


Expande o primeiro nó encontrado cuja a função heurística é melhor que a do nó-pai que o gerou. Ignora os demais nós.

- **É completa?** Não.
- **É ótima ? Não**.
- **Custo de Tempo: melhor caso:** *O(d)* **e no pior caso:** *O* (b^d).
- **Custo de Memória: não se guarda os nós**. *Usa pouca memória*

Subida Encosta pelo Caminho + Íngreme

Funcionamento do Algoritmo:



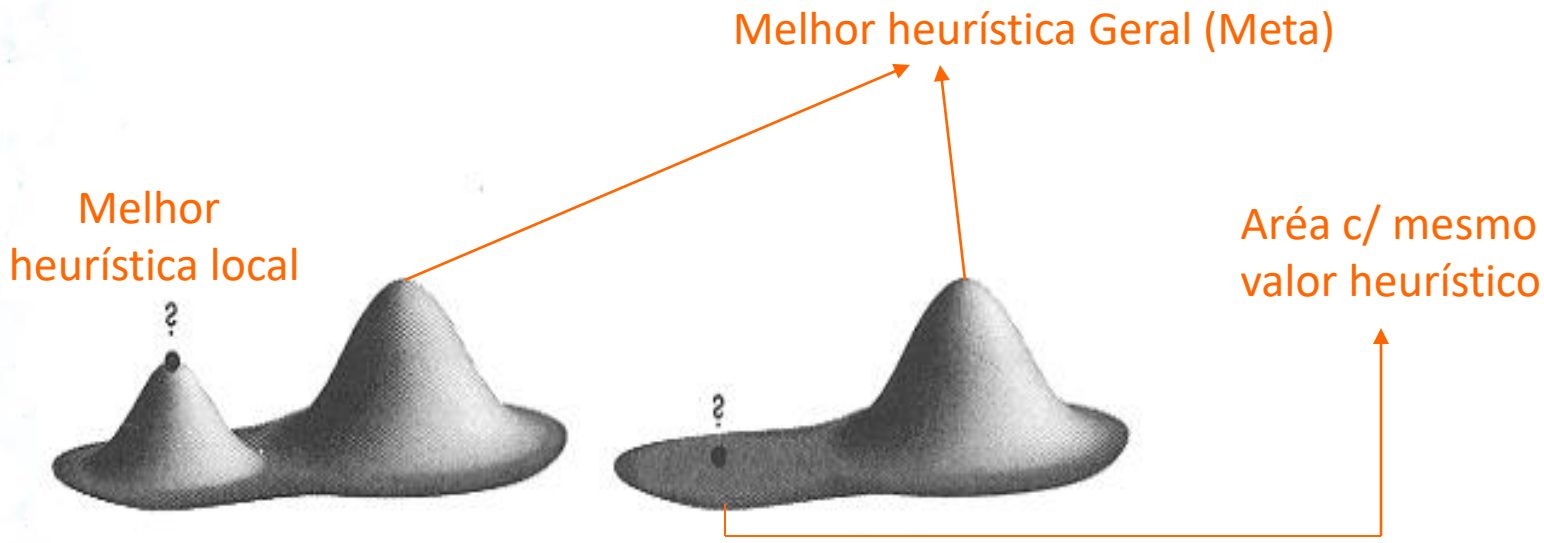
- **Dentro todos os sucessores do nó-pai, expande o melhor nó cuja a função heurística é melhor que a do nó-pai. Ignora os demais nós**
  - **É completa ?** Não.
  - **É ótima ? Não**.
  - **Custo de Tempo: melhor caso:** *O(b.d)* **e no pior caso** *O* (b^d).
  - **Custo de Memória: não se guarda os nós**. *Usa pouca memória*

Problemas com a Subida da Encosta

Os algoritmos movem-se sempre na direção que apresenta variação na função heurística.
Ele pode falhar sem encontrar uma solução.

Isso pode acarretar em 2 problemas principais:

1. Máximos locais
2. Planícies (platôs)



Problemas com a Subida da Encosta

As armadilhas:

- Acontecem pois Subida-pela-Encosta não faz backtracking
- Dependem do estado inicial

Formas de contornar tais armadilhas é:

- Recomeçar de novo com outro operador
- Aplicar uma junção de mecanismos de busca

  (Enforced Hill-Climbing - Subida Forçada da Encosta
- Permitir que a busca desça alguns passos da encosta

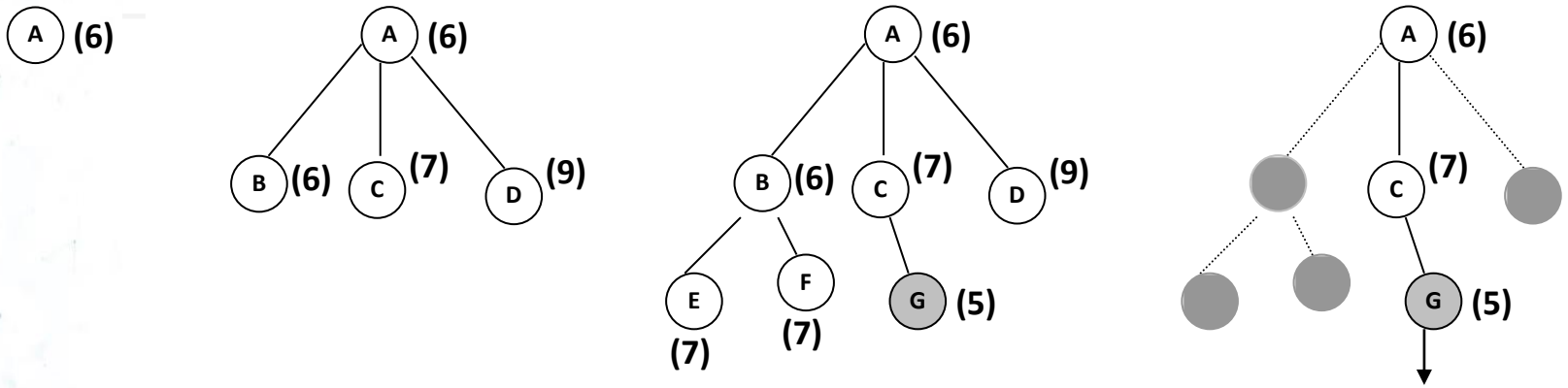
  (Simulated Annealing) – *out-of-use*

Subida Forçada da Encosta

O algoritmo Subida Forçada da Encosta é novo (1999)

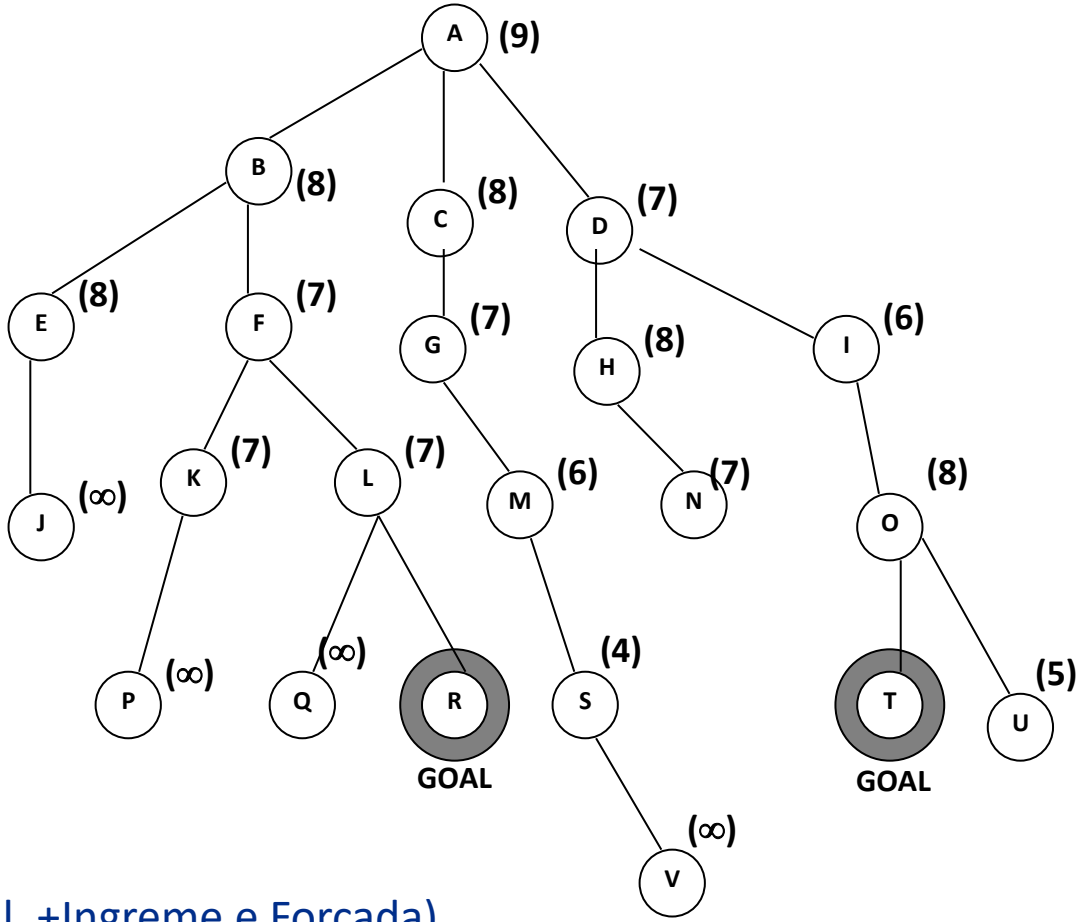
Foi inicialmente utilizada no sistema de planejamento: FF (Fast-Forward)

- **Tem como objetivo continuar procurando (expandindo) os nós sucessores do estado corrente, com a** MELHOR-ESCOLHA**, até encontrar um nó que seja melhor que o estado corrente.**



Exercício

Para a seguinte árvore:

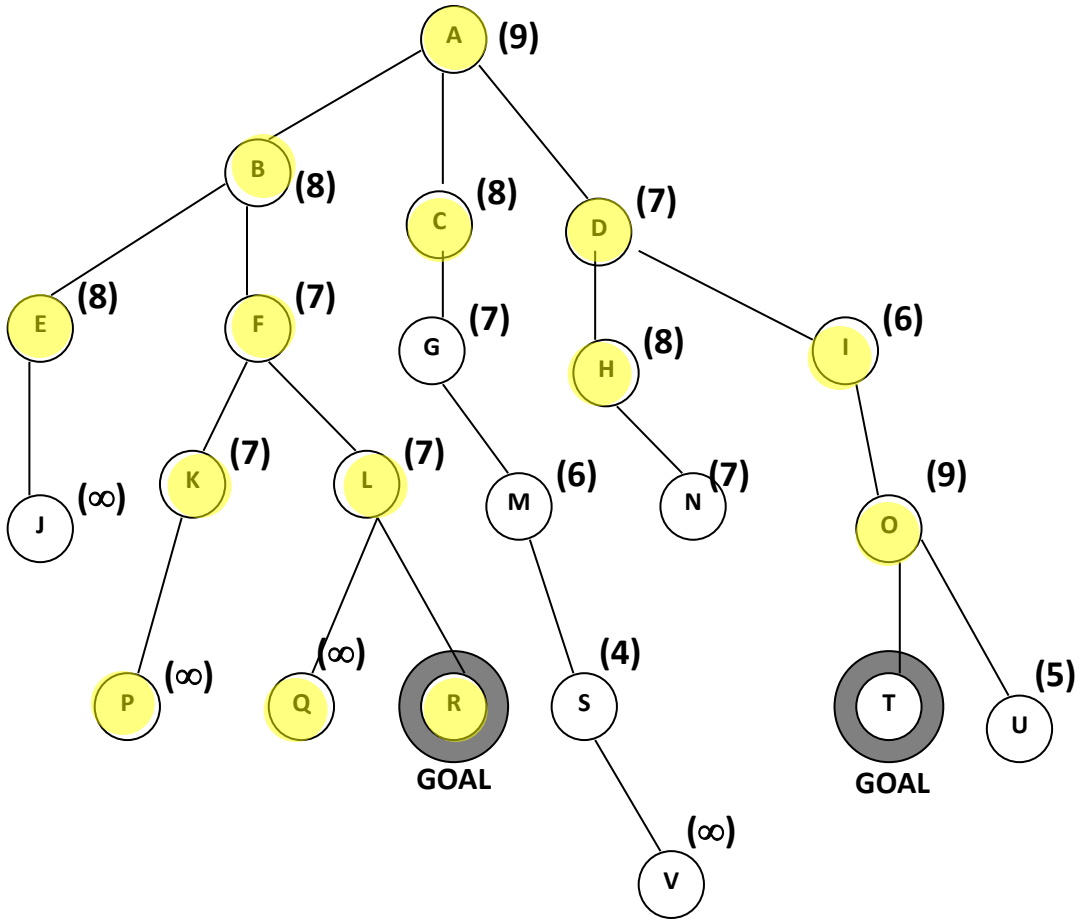


Ache a sequência de nós:

- Melhor Escolha
- Subida da Encosta (Normal, +Ingreme e Forçada)

Resposta Exercício:

- Sequência de Expansão de Nós para:
  - Melhor-Escolha

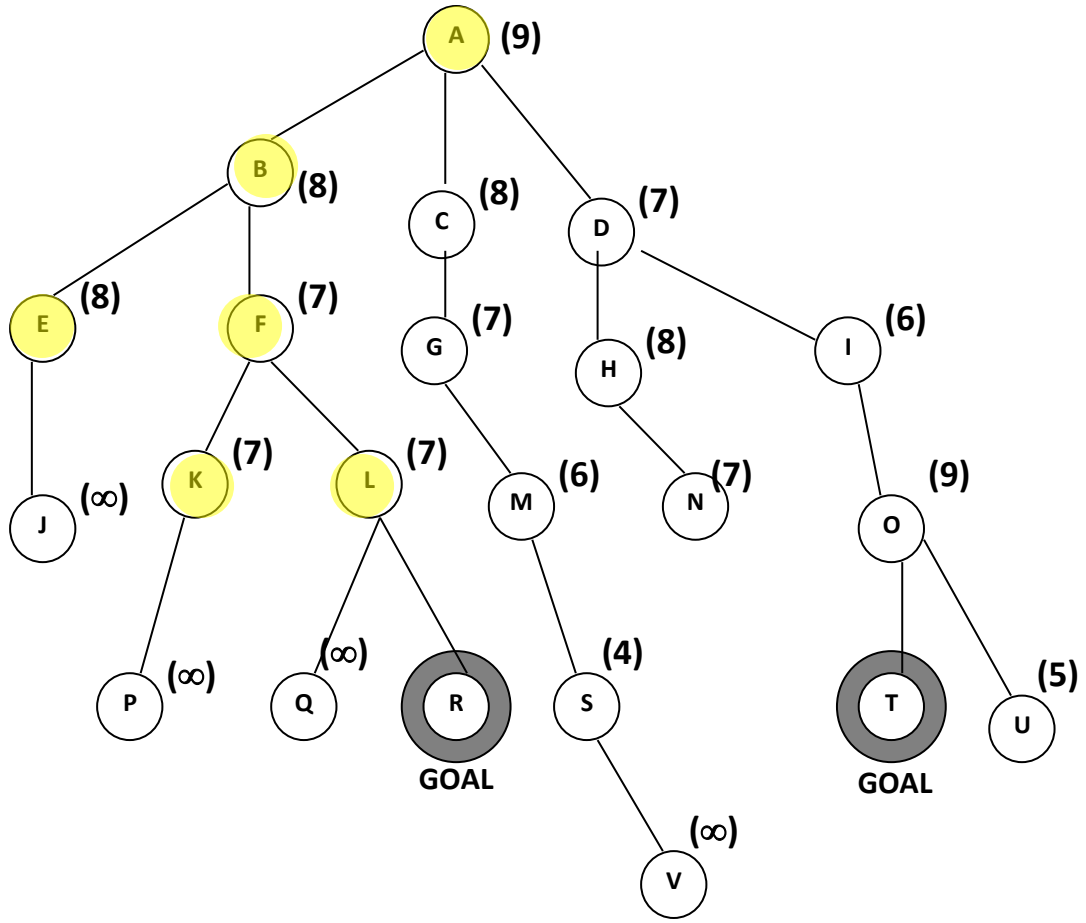


- Sequência Gerada:

  ABCDHIOEFKLPQR

Resposta Exercício:

- Sequência de Expansão de Nós para:
  - Subida Encosta

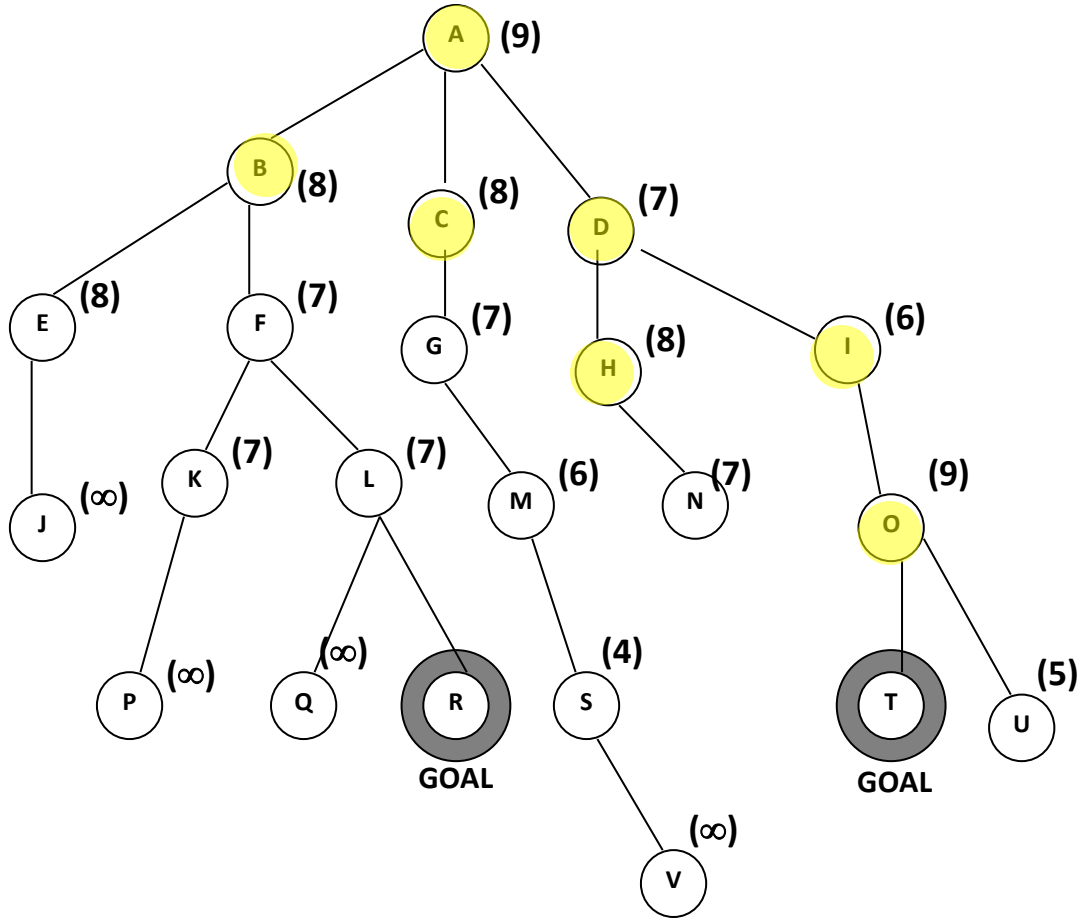


- Sequência Gerada:

ABEFKL falha !

Resposta Exercício:

- Sequência de Expansão de Nós para:
  - Subida Encosta
  caminho + Ingreme

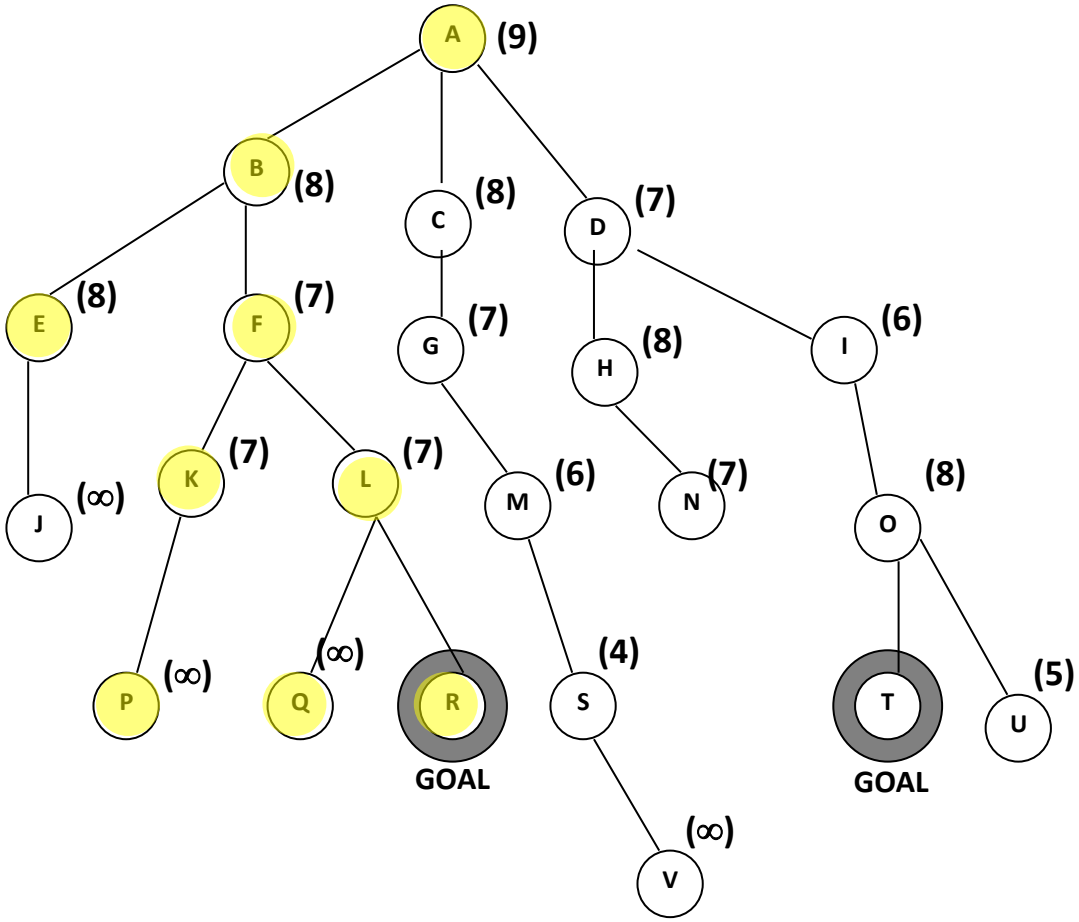


Sequência Gerada:

ABCDHIO falha !

Resposta Exercício:

- Sequência de Expansão de Nós para:
  - Subida Forçada da Encosta



- Sequência Gerada:

ABEFKLPQR

Bibliografia desta Aula

Para aprofundamento nos assuntos desta aula, segue a seguinte referência bibliográfica

- Rich, E. (Inteligência Artificial)
  - Capítulos 2, 3 e 12 (jogos)
- Russel & Norvig (Artificial Intelligence)
  - Capítulos 3,4 e 6

- Alguns slides desta aula foram baseados no slides:
- Anna Reali Costa e Geber Ramalho: “Técnicas de Busca Cega”, Poli-USP e Cin-UFPE.
- Anna Reali Costa: “Busca Informada” em Aula5e6-BuscaInformada.pdf – Poli-USP
- Geber Ramalho: Busca 2 e Busca 3, Cin-UFPE.
- José Augusto Baranauskas: Estratégias de Busca - FFCLRP-USP