FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO

Inteligência Artificial

Unidade III – Buscas Cegas

Profa. Vládia Pinheiro

Adaptação de conteúdo dos Prof.André Coelho e Prof.Vasco Furtado

Resolução de Problemas

- **Agentes reativos (Percepção – Regras condição/ação – Ação)**
- **Agentes de Resolução de Problemas:**
  - Baseado em objetivos
  - Decidem o que fazer encontrando sequência de ações que levam a estados desejáveis
  - Utilizam um método de resolução de problemas
  - Para algumas aplicações, podem ser utilizadas <u>estratégias de buscas em espaço de estados</u>

Caracterização de problemas

- Relembrando o que Minsky disse...
  - "Inteligência é o nome que damos a qualquer um dos processos contidos em nossas mentes que nos capacitam a solucionar problemas difíceis"
- Antes de se tentar construir uma máquina que venha a ser capaz de resolver um certo problema, é necessário identificar:
  - Quais as informações disponíveis sobre o problema;
  - Quais as formas alternativas de se representar o problema;
  - O que caracteriza uma solução (resultado) p/ o problema; e
  - Quais as soluções satisfatórias do problema.

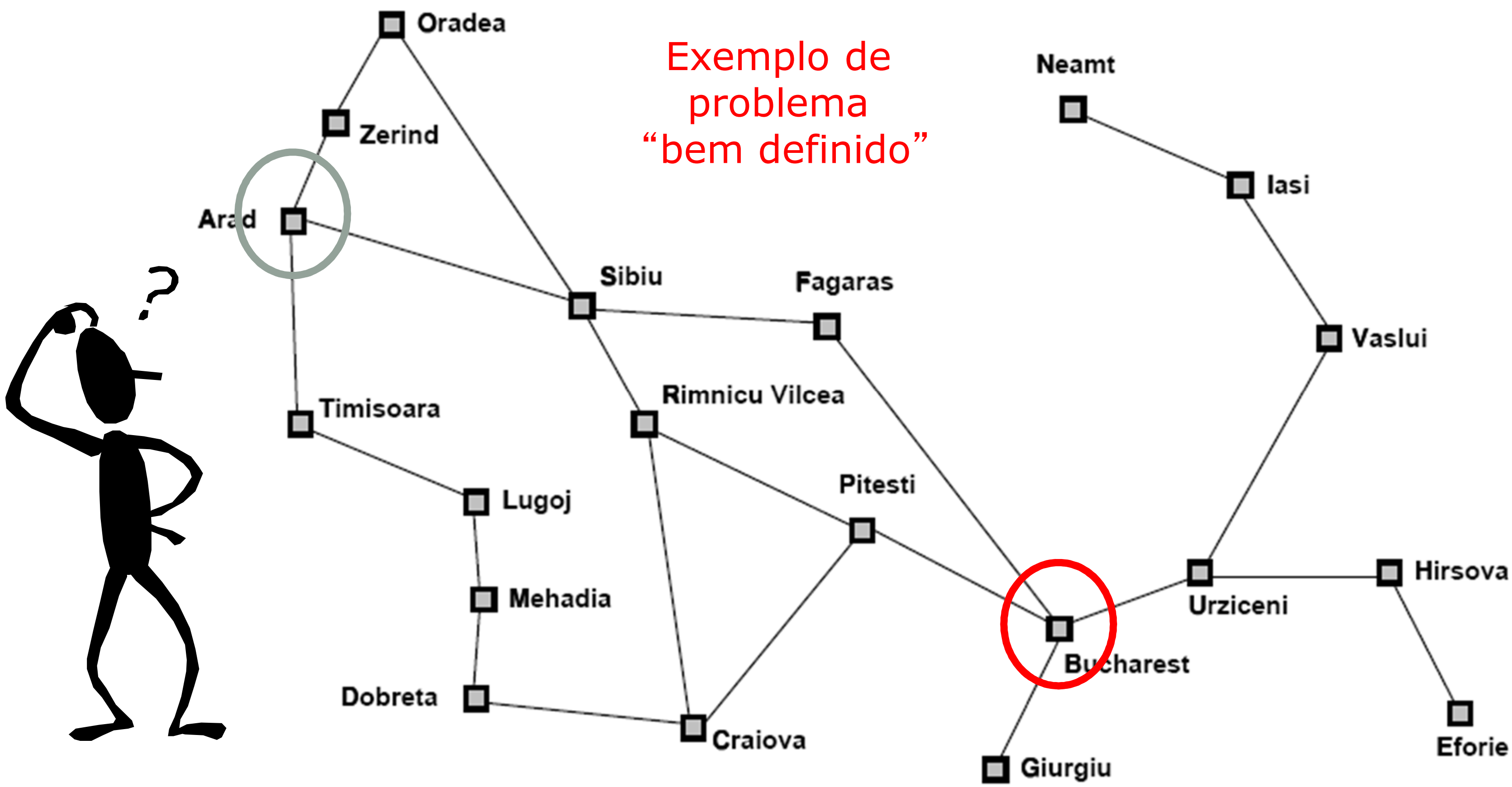
Caracterização de problemas

- Perguntas antes do tratamento de um problema qualquer:
  - Já são conhecidos todos os passos necessários para se chegar a uma solução para esse problema?
    - Se sim, então pode-se gerar um algoritmo p/ resolvê-lo
  - O problema é passível de ser decomposto?
    - Ou seja, a solução para o problema inicial pode ser obtida pela composição das soluções de alguns problemas mais elementares?
  - Uma solução satisfatória é relativa ou absoluta?
    - Ela será absoluta se não depender das condições iniciais do problema
  - O universo do problema é determinístico?
    - Ou seja, é possível planejar uma sequência de passos p/ a qual a solução encontrada será sempre a mesma?

Agentes de Resolução de Problemas

Como ir de Arad a Bucareste pelo caminho mais curto????

Exemplo de problema “bem definido”



Agentes de Resolução de Problemas

1. Formulação do objetivo
   - Ter um objetivo em mente ajuda a organizar o seu comportamento, simplificando suas decisões
2. Formulação de problemas
   - Processo de decidir que ações e estados do mundo devem ser considerados em função do objetivo estabelecido anteriormente
3. Processo de busca
   - Um algoritmo de busca recebe um problema formulado como entrada e retorna uma solução sob a forma de uma sequência de ações
4. Fase de execução
   - Cada ação da sequência gerada na busca é executada pelos atuadores do agente

Agentes de Resolução de Problemas

função AGENTE-RESOL-PROB(*percepção*) **retorna** ação
entradas: *percepção* % uma percepção
variáveis estáticas: *seq* %seq. de ações
estado %descrição do estado atual do mundo
objetivo %pode ser que inicialmente nulo
problema %formulação do problema

estado ← ATUALIZAR-ESTADO(*estado*, *percepção*);
se *seq* está vazia **então faça**
objetivo ← FORMULAR-OBJETIVO(*estado*); 1
problema ← FORMULAR-PROB(*estado*, *objetivo*); 2
seq ← BUSCA(*problema*); 3
ação ← PRIMEIRO(*seq*); 4
seq ← RESTO(*seq*)
retornar *ação*

Ciclo de “formular, buscar e executar”

Formulação de problemas via espaço de estados

- Componentes-chave:
  - Estado inicial, situação (e.g., config. de variáveis) em que o agente começa a resolver o problema
  - Função-sucessor, que define, p/ o estado atual de resolução, quais as ações (operações) válidas e quais os estados vizinhos gerados por tais ações
    - SUCESSOR(*x*) retorna conj. de pares <operação, sucessor>
  - Espaço de estados, definido implicitamente pelo estado inicial + função-sucessor → grafo
    - Todos os estados acessíveis a partir do estado inicial
  - Caminho é uma sequência de estados conectados por uma sequência de operações (transições) no grafo de estados possíveis

Formulação de problemas via espaço de estados

- Componentes-chave (cont.):
  - Teste de objetivo é que determina se um dado estado é um estado-objetivo (final)
    - Definido explicitamente ou via propriedade abstrata
  - A função custo de caminho atribui um custo numérico a cada caminho do espaço de estados
    - Deve refletir a medida de desempenho do agente
    - Definida como a soma dos custos das ações individuais (ou seja, os custos de passo) ao longo do caminho
    - $c(x,a,y)$: custo do passo (ação, transição) de ir do estado x p/ o seu sucessor y (<u>geralmente</u>, é não-negativo)
  - Solução do problema é o caminho do estado inicial ao estado-objetivo
  - Solução ótima é a de <u>menor</u> custo de caminho

Formulação de problemas via espaço de estados

- Por qual motivo adotar o conceito de estado?
  - As informações do mundo real são absurdamente complexas, sendo praticamente impossível modelá-las completamente
    - No exemplo do aspirador, o mundo dele tem várias outras informações: a cor do tapete, se é dia, de que material o aspirador é feito, quanto ele tem de energia, etc.
  - A noção de estado é utilizada para abstrair esses detalhes e considerar somente o que é relevante para a solução do problema
  - O mesmo se dá com as operações modeladas pela função-sucessor: são abstrações de opers. reais
    - Ir para a posição da direita, p. ex., implica, na verdade, em várias outras operações de baixo nível

Formulação do problema do <u>Viajante Romeno</u>

- **Estados**: cada um representa a presença do agente em uma cidade: "Em(*cidade*)"
- **Estado inicial**: Em(arad)
- **Função-sucessor**: gera o estado-sucessor de cada ação de dirigir: Sucessor(Em(arad)) → {<Ir(sibiu), Em(sibiu)>, <Ir(timisoara), Em(timisoara)>, <Ir(zerind), Em(zerind)>}
- **Teste de objetivo**: Em(buscareste)?
- **Custo de caminho**: como o tempo é essencial, o custo de cada caminho pode ser seu comprimento em quilômetros (soma das distâncias entre as cidades do caminho)

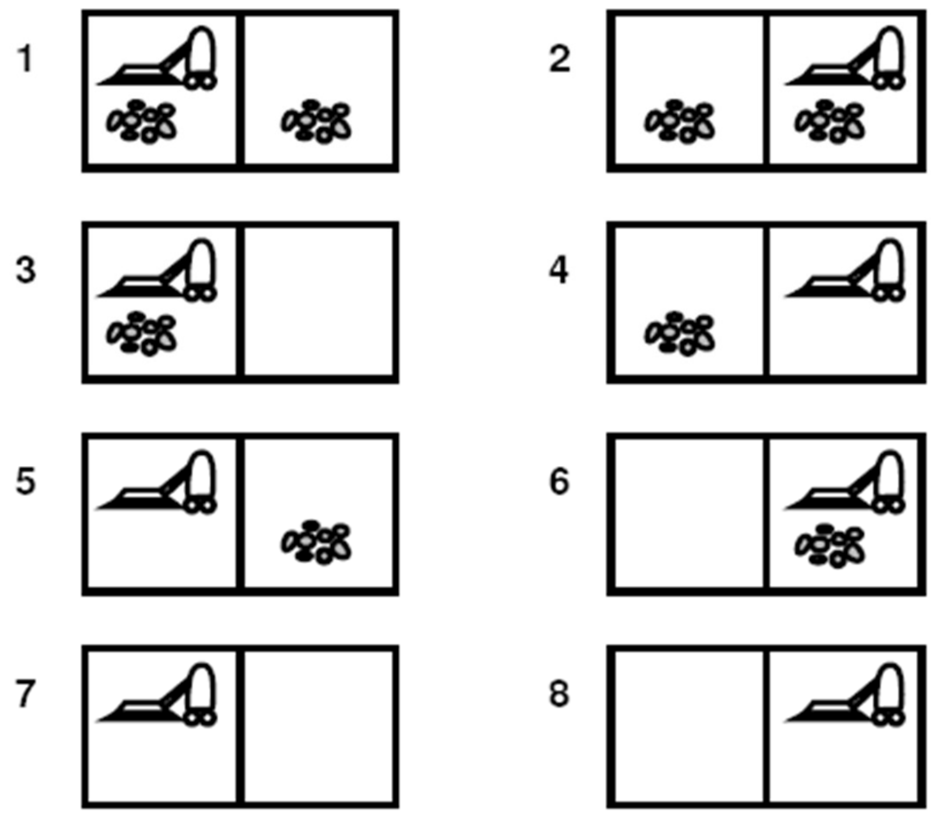
Formulação do problema do Aspirador de Pó

- **Estados**: são duas as posições físicas que o aspirador pode ocupar em cada momento; porém, são quatro as possibilidades de sujeira ou limpeza dos dois quadrados ao mesmo tempo → $2 \times 2^2 = 8$
- **Estado inicial**: qualquer um
- **Função-sucessor**: gera os estados válidos que resultam da tentativa de executar as ações de Ir p/ Esquerda, Ir p/ Direita e Aspirar
- **Teste de objetivo**: ambos os quadrados estão limpos no menor tempo possível?
- **Custo de caminho**: cada passo custa 1 (já que o tempo de limpeza é importante) e assim o custo do caminho é o número de passos do caminho

Formulação de problemas via espaço de estados

- Exemplo do aspirador de pó

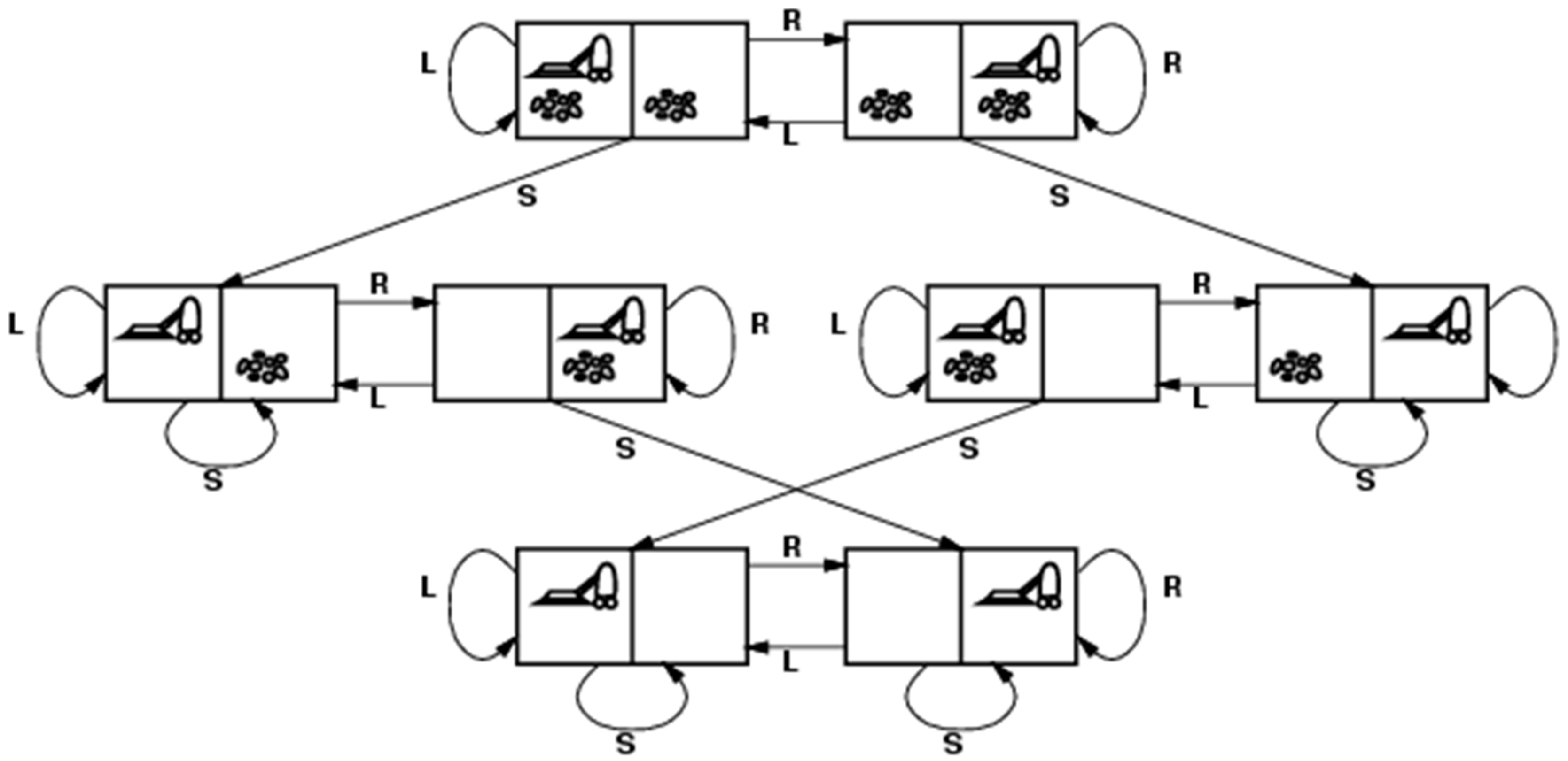
Estados possíveis:



Formulação de problemas via espaço de estados

- Exemplo do aspirador de pó

Espaço de estados



Problema Missionários e Canibais

- Transportar, de uma margem à outra de um rio, um grupo de 3 missionários e 3 canibais, de forma que, em nenhum momento, o número de canibais seja maior que o número de missionários. Existe um único barco que pode ser usado para transportar no máximo dois passageiros.
- Experimente em: http://www.plastelina.net/games/game1.html

- Exercício: Formular este problema

Buscas em Espaços de Estados

- **Definição:**
  - Processo de encontrar uma sequência de ações possíveis que levam a estados de valor conhecido, dentro de um universo denominado espaço de estados.
- **Estratégia de Busca**
  - Indica qual nó-folha escolher a cada expansão (geração de um novo conj. de nós pela aplicação da função-sucessor)
  - De uma forma geral, os passos a seguir são:
    - Coloca-se o estado inicial como nó-raiz da árvore;
    - Cada operação sobre o nó-raiz gera um nó sucessor; e
    - Repete-se esse processo para novos nós até se encontrar aquele que representa o estado-objetivo.

Buscas em Espaços de Estados

Algoritmo de busca geral

função BUSCA-EM-ÁRVORE(*problema*, *estratégia*)
retorna uma solução ou falha
  iniciar a árvore de busca usando o estado inicial de *problema*;
  repita
    se não existe nenhum candidato p/ expansão **então retornar** falha;
    escolher um nó-folha p/ expansão de acordo com *estratégia*;
    se o nó contém um estado-objetivo **então retornar** a solução correspondente;
    senão expandir o nó e adicionar os nós resultantes (folhas) à árvore de busca;

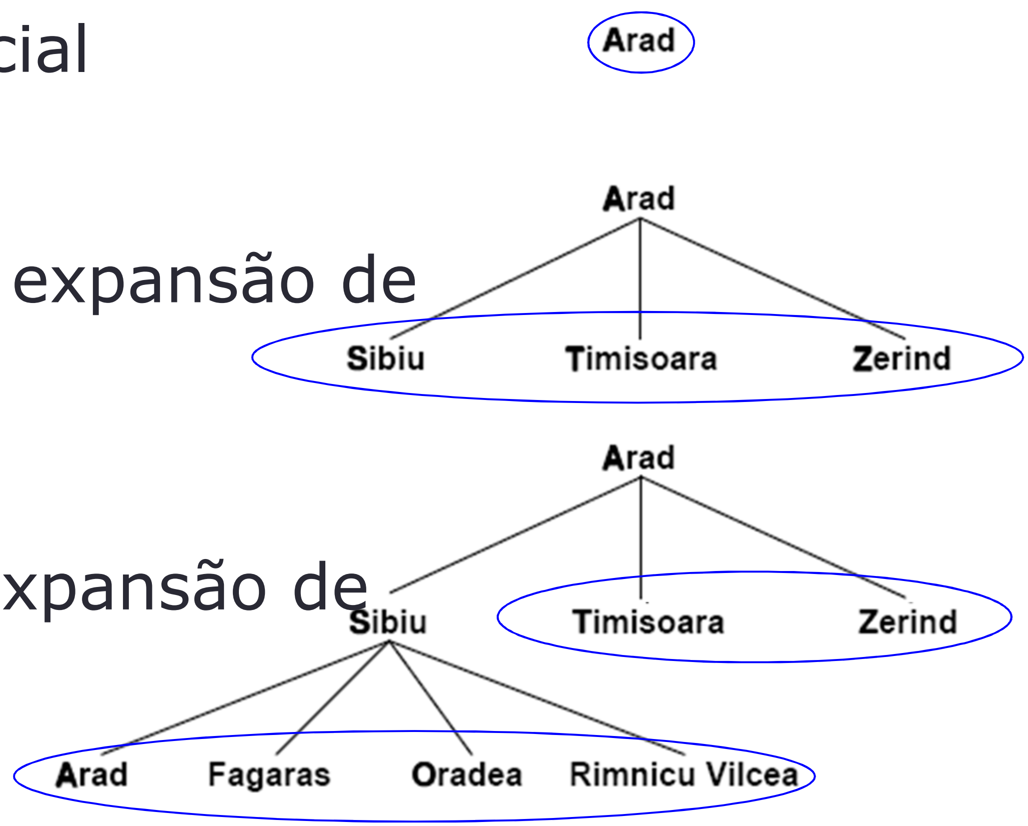
Buscas em Espaços de Estados

- Exemplo do viajante romeno

(a) Estado inicial

(b) Depois da expansão de Arad

c) Depois da expansão de Sibiu

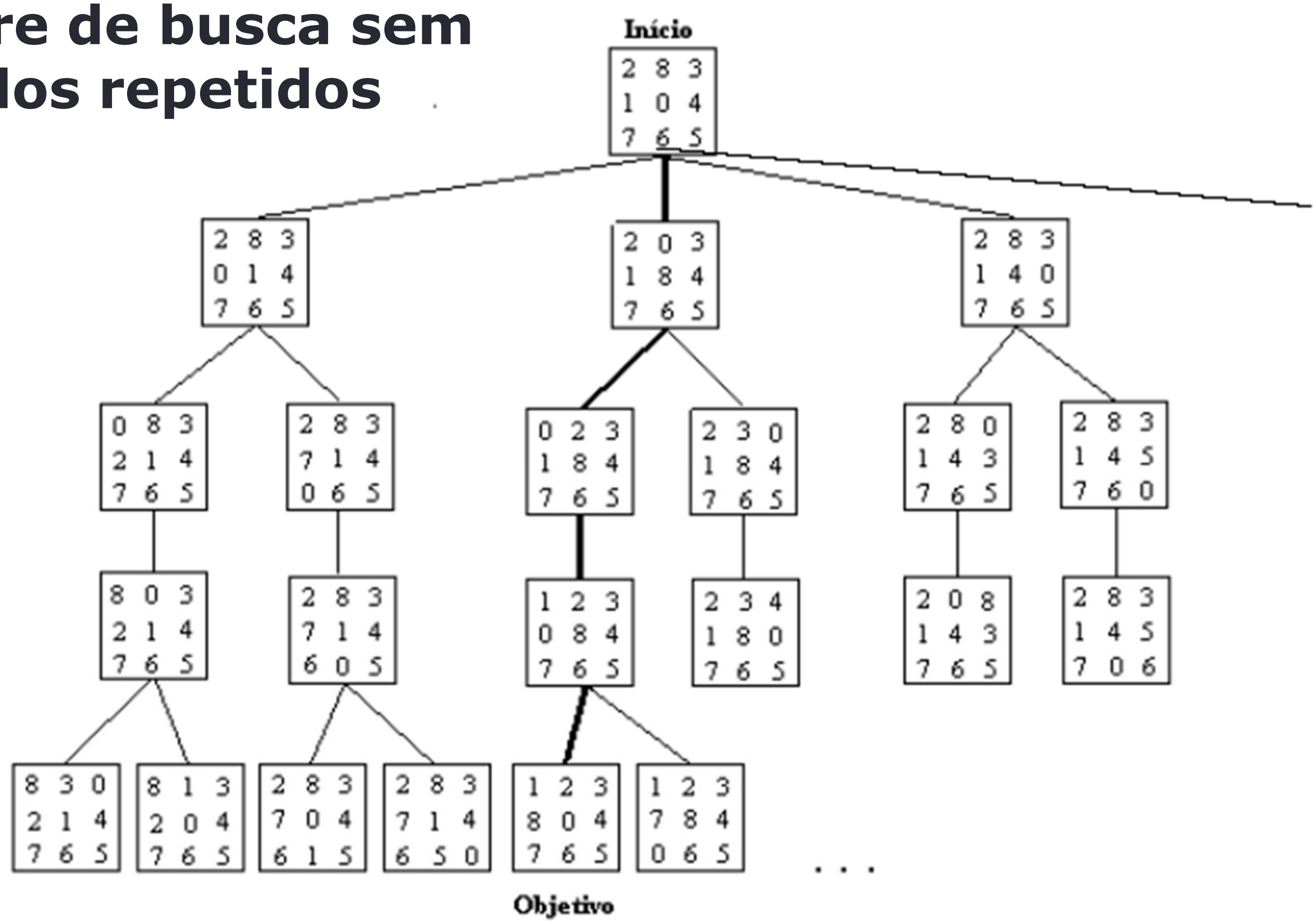


: Candidatos à expansão

Buscas em Espaços de Estados

- Exemplo do Puzzle-8

Árvore de busca sem estados repetidos

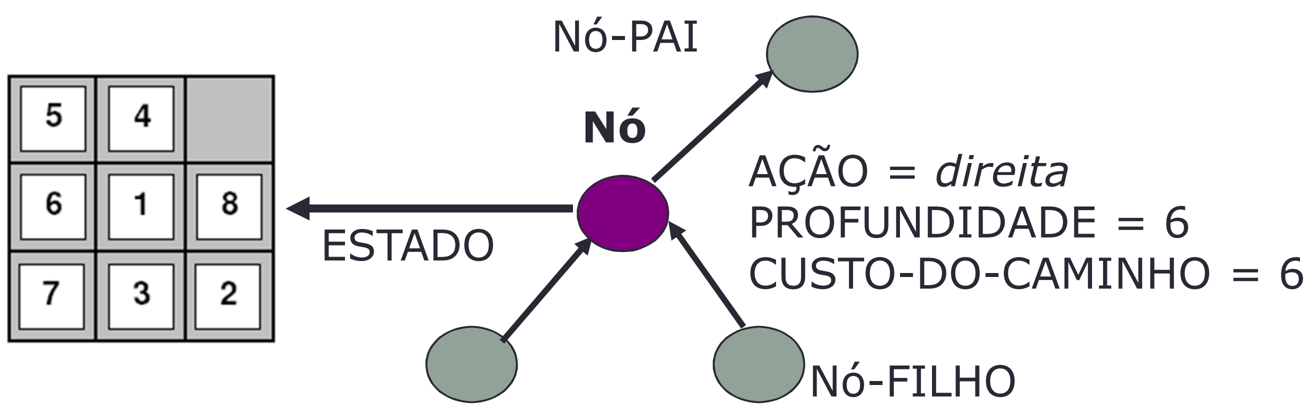


Buscas em Espaços de Estados

- Cada nó da árvore é uma estrutura de dados com cinco componentes:
  - ESTADO: o estado que representa no espaço de estados;
  - NÓ-PAI: o nó da árvore de busca que gerou o atual;
  - AÇÃO: a operação aplicada ao pai que o gerou;
  - CUSTO-DO-CAMINHO: o custo acumulado desde o estado inicial (tradicionalmente denotado por $g(n)$); e
  - PROFUNDIDADE: o número de passos desde o início.

Buscas em Espaços de Estados

- Diferença entre nó e estado:
  - Um nó é apenas uma representação de um certo estado em um caminho específico da árvore.
  - Dois nós diferentes podem referenciar o mesmo estado abstrato do mundo, pois foram gerados por dois caminhos (ou nós-pai) de busca diferentes
- Fronteira ou borda:
  - Conjunto de nós-folhas gerados à espera de expansão
  - Implementada por uma fila ➔ diferentes estratégias de busca implementam diferentes tipos de fila



Buscas em Espaços de Estados

- Como evitar a geração de estados repetidos?
  - Usar estratégias de poda, em que nós representando estados repetidos não são gerados/expandidos:
    - Um nó igual a seu pai não é gerado/expandido;
    - Um nó igual a qualquer um de seus antecessores do caminho não é gerado/expandido; e
    - Um nó igual a qualquer outro da árvore que já tenha sido gerado/expandido não pode ser gerado/expandido.

Exemplos

  - Dificuldade: tais estratégias podem consumir tempo!
    - Saída: tabelas *hash* (c/ tempo ótimo de consulta)

Estratégias de Busca

- Tipos de estratégias de busca:
  - Busca cega (exaustiva ou não-informada): não recebe informação adicional sobre o problema além da sua definição via espaço de estados
  - Busca heurística (informada): dispõe de informações adicionais que facilitam a procura por boas soluções
- Estratégias de busca cega mais comuns:
  - Busca em largura (ou em extensão)
  - Busca de custo uniforme
  - Busca em profundidade (ilimitada ou limitada)
  - Busca com aprofundamento iterativo
- Direção da expansão:
  - Do estado inicial para um estado final
  - De um estado final p/ o estado inicial (menos comum)
  - Busca bidirecional

Estratégias de Busca

- Critérios de avaliação das estratégias
  - Completeza: a estratégia **sempre** garante encontrar uma solução quando ela existir?
  - Otimalidade: a estratégia encontra a **solução ótima** (i.e., de menor custo de caminho)?
  - Complexidade temporal: quanto tempo ela leva para encontrar uma solução?
  - Complexidade espacial: quanta memória é necessária para se executar a busca?

Estratégias de Busca

Busca em largura (*breadth-first search*)

Estratégia: o nó-folha de **menor** profundidade e mais à **esquerda** da árvore é escolhido para ser testado e expandido primeiro

Algoritmo:

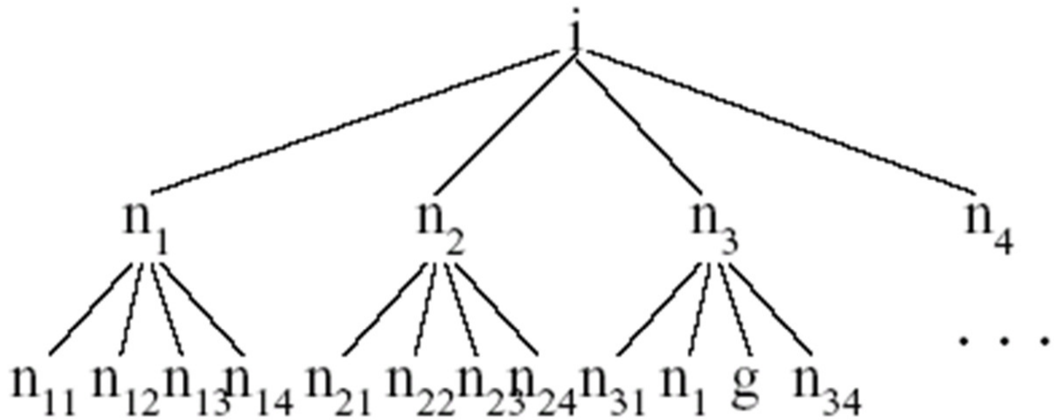
função BUSCA-EM-LARGURA(*problema*)
retorna uma solução ou falha
BUSCA-EM-ÁRVORE(*problema*, INSERE_NO_FIM);

Busca em Largura

- **Algoritmo tem como base uma fila FIFO**

1. Seja *L* uma lista de nós não examinados.
2. Se *L* é vazia, fim da busca, senão, pegue um nó de *L*
3. Se *n* é o objetivo, fim e retorne o caminho percorrido
4. Caso contrário, remova *n* de *L* e some ao fim de *L* todos os filhos de *n,* retorne ao passo 2

Busca em Largura



$$\{i\}$$

$$\{n_1, n_2, n_3, n_4\}$$

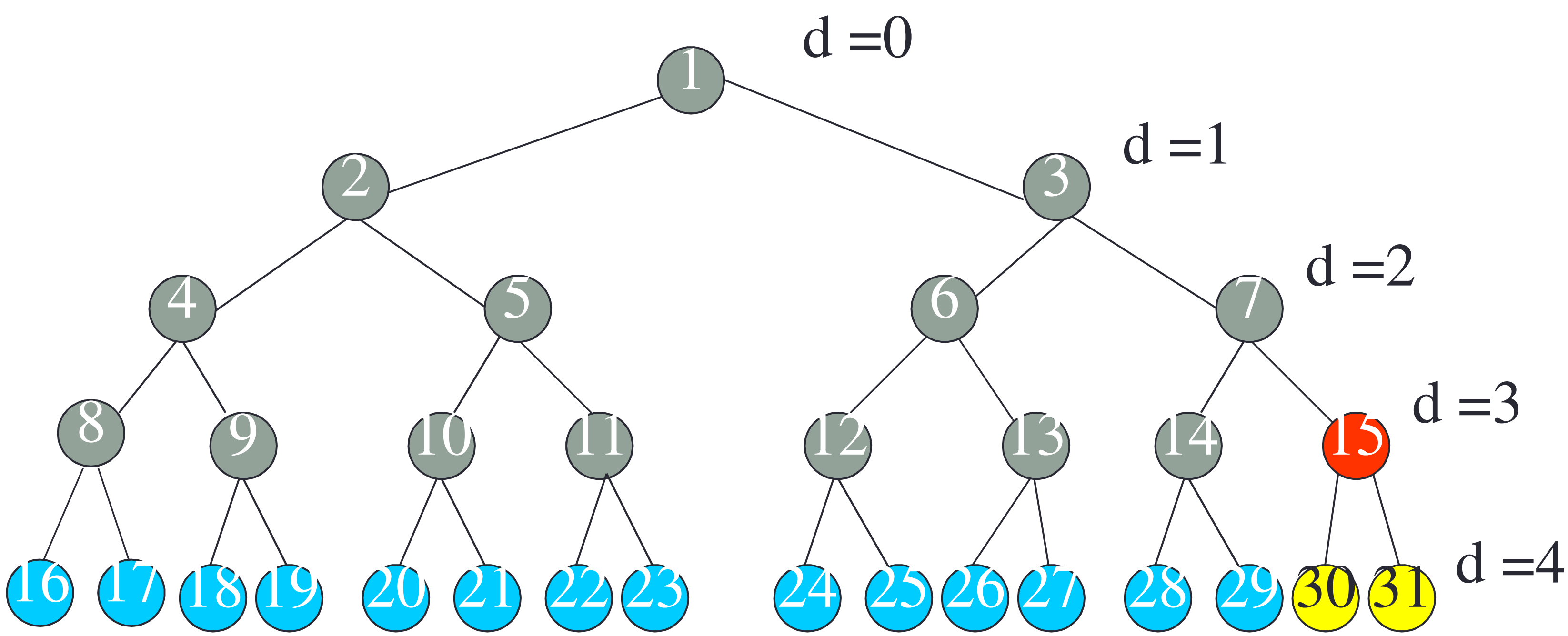
$$\{n_2, n_3, n_4, n_{11}, n_{12}, n_{13}, n_{14}\}$$

$$\{n_3, n_4, n_{11}, n_{12}, n_{13}, n_{14}, n_{21}, n_{22}, n_{32}, n_{24}\}$$

Busca em Largura

- Exemplo

Profundidade = 4



Na profundidade 3 considerando que a solução é o último nó (pior caso) quantos nós ficam na memória?

$b^{d+1} - b \Rightarrow 2^{3+1} - 2 = \mathbf{14}$

= Nó já visitado (tirado da memória)

= Nó por visitar (em memória)

= Nó resposta (visitado mas não expandido)

= Nó ainda não criado ou expandido

Busca em Largura

- **Análise de Complexidade Temporal e Espacial:**
  - Cada estado tem ***b*** sucessores e cada sucessor gera outros ***b*** sucessores. Logo,
    - Número de nós na profundidade d = b^d
  - Supondo que o estado-objetivo esteja na profundidade ***d***. No pior caso teríamos que explorar todos os nós da profundidade ***d*** produzindo (b^{d+1}- b) nós no nível (d+1). Como todo nó gerado deve permanecer na memória, então
    - O espaço necessário é de pelo menos= b^{d+1}- b nós
    - Complexidade Temporal é exponencial = $O(b^{d+1})$
    - Complexidade Espacial (memória) = Complexidade Temporal
  - Obs: **b** (branches) representa as ramificações ou galhos, e **d** (deep) a profundidade.

Busca em Largura

- Esta estratégia é completa (se $b < \infty$, $d < \infty$) ↑
- É ótima?
  - Sempre encontra o nó-objetivo mais "raso", mas que nem sempre é aquele de menor **custo de caminho**, caso as operações tenham custos de passo diferentes
  - Se o custo de caminho é uma função crescente conforme a profundidade do nó, então é ótima
    - P. ex., isso ocorre quando todas as operações têm o mesmo custo de passo (=1) → $g(n)$ = PROFUNDIDADE(n)
- Complexidade temporal: ↓
  - Número total de nós gerados até se encontrar a solução é, no pior caso, igual a $b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b)$
  - Custo exponencial: $O(b^{d+1})$
- Complexidade espacial: ↓↓
  - Todo nó gerado ao longo da busca deve ser mantido sempre na memória, levando a se guardar $O(b^{d+1})$ nós

Busca em Largura

- Embora seja simples e sistemática, só dá bons resultados quando a profundidade da árvore de busca é bem pequena!

Profundidade	Nós	Tempo	Memória
2	1100	0,11 segundo	1 megabyte
4	111.100	11 segundos	106 megabytes
6	10^7	19 minutos	10 gigabytes
8	10^9	31 horas	1 terabyte
10	10^{11}	129 dias	101 terabytes
12	10^{13}	35 anos	10 petabytes
14	10^{15}	3.523 anos	1 exabyte

Figura 3.11 Requisitos de tempo e memória para a busca em extensão. Os números mostrados pressupõem um fator de ramificação $b = 10$, 10.000 nós/segundo, 1.000 bytes/nó.

Busca de Custo Uniforme

Estratégia: expande o nó da fronteira com o menor custo de caminho ($g(n)$) associado

Cada operação pode ter um custo de passo diferente

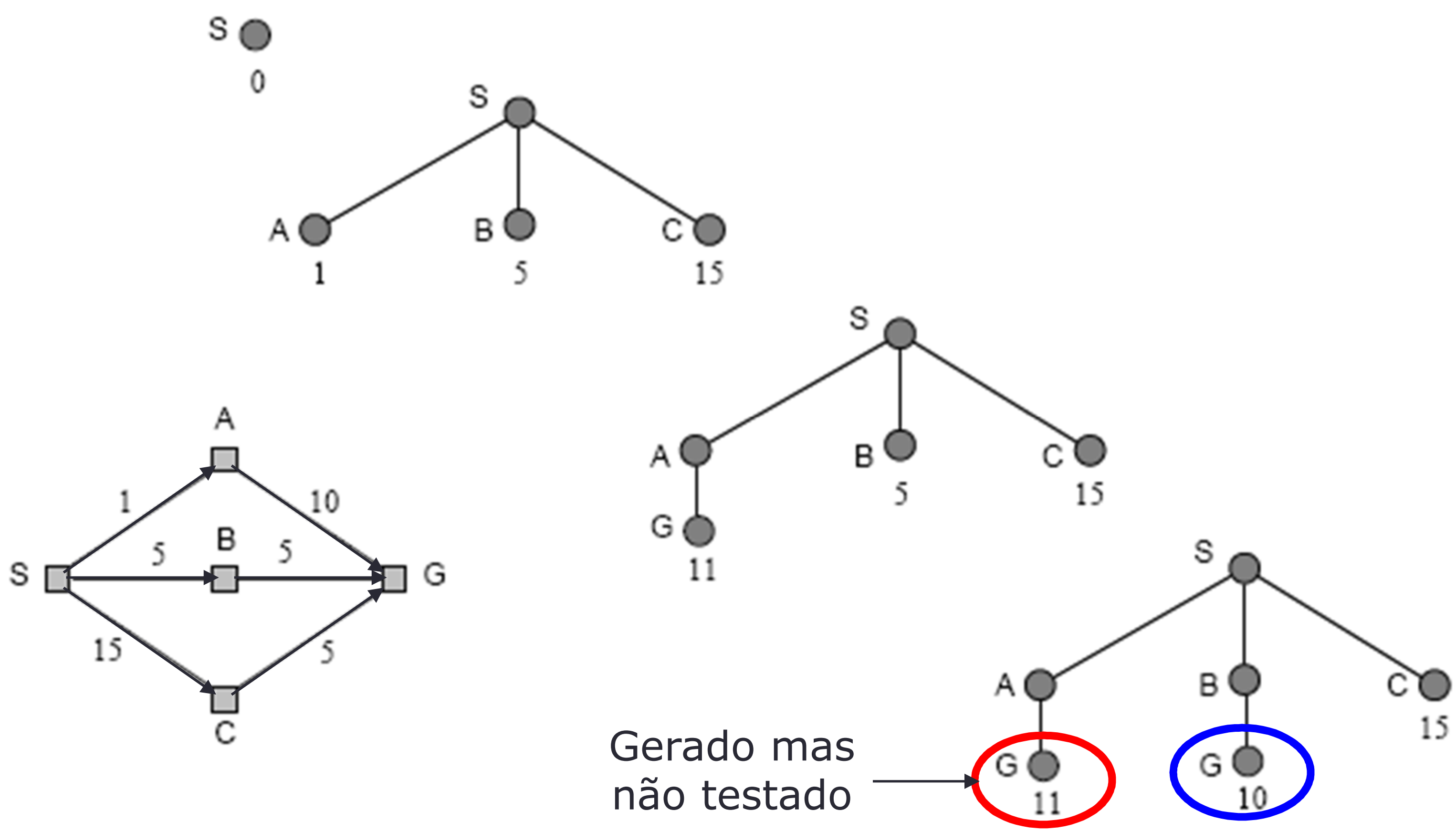
Será igual à busca em largura se $g(n)$ = PROFUNDIDADE(n) ➔ custos de passo iguais

Algoritmo:

função BUSCA-DE-CUSTO-UNIFORME(*problema*)
retorna uma solução ou falha
BUSCA-EM-ÁRVORE(*problema*, INSERE_ORDEM_CRESCENTE_CUSTO);

Busca de Custo Uniforme

Exemplo



Busca de Custo Uniforme

Fronteira do exemplo

- $F = \{S\}$
  - Testa se S é o estado-objetivo, expande-o e guarda seus filhos A, B e C ordenadamente na fronteira
- $F = \{A, B, C\}$
  - Testa A, expande-o e guarda seu filho G_A na ordem
    - Lembrete: o algoritmo guarda na fronteira todos os nós gerados e não expandidos, testando se um nó é o objetivo apenas quando ele é retirado da lista!
- $F = \{B, G_A, C\}$
  - Testa B, expande-o e guarda seu filho G_B na ordem
- $F = \{G_B, G_A, C\}$
  - Testa G_B e pára

Busca de Custo Uniforme

- Esta estratégia é completa! ↑
  - Desde que o custo de cada passo seja $\geq \varepsilon$ (cons-tante positiva pequena), pois, se existir um nó que tenha uma ação de custo nulo indo p/ o mesmo estado, o algoritmo pode ficar em um laço infinito
- É ótima se
  - $g(\text{sucessor}(n)) > g(n)$, ou seja, se o custo de caminho sempre aumenta ao se percorrer o caminho
- Complexidade temporal e espacial: ↓↓
  - Quando todos os custos de passo forem iguais, tais complexidades serão de $O(b^{d+1})$, pois, no pior caso, serão expandidos todos os nós com $g(\cdot) \leq C^*$ (custo da solução ótima); caso contrário, podem ser piores

Busca em Profundidade

Estratégia: o nó-folha de **maior** profundidade e mais à **esquerda** da árvore é escolhido para ser testado e expandido primeiro

Obs: Qdo um nó-folha final não é solução e não tem sucessores, ele é retirado da memória e o algoritmo retorna p/ o mesmo nível ou p/ o nível anterior a fim de expandir os nós que ainda estão na fronteira

Algoritmo:

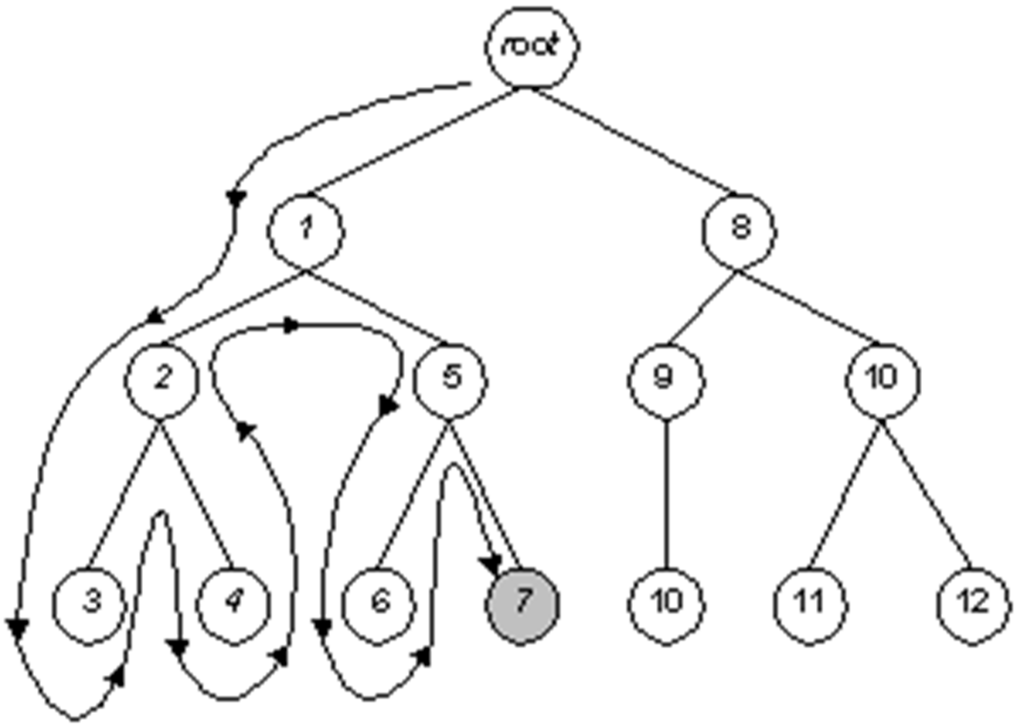
função BUSCA-EM-PROFUNDIDADE(*problema*)
retorna uma solução ou falha
BUSCA-EM-ÁRVORE(*problema*,
INSERE_NO_COMEÇO);

Busca em Profundidade

- **Algoritmo tem como base uma pilha LIFO (Last In First Out)**

1. Seja *L* uma lista de nós não examinados.
2. n = 1º nó de *L.* Se *L* é vazia, retorna.
3. Se *n* é o objetivo, fim e retorne o caminho percorrido
4. Caso contrário, some ao começo de *L* todos os filhos de *n,* retorne ao passo 2

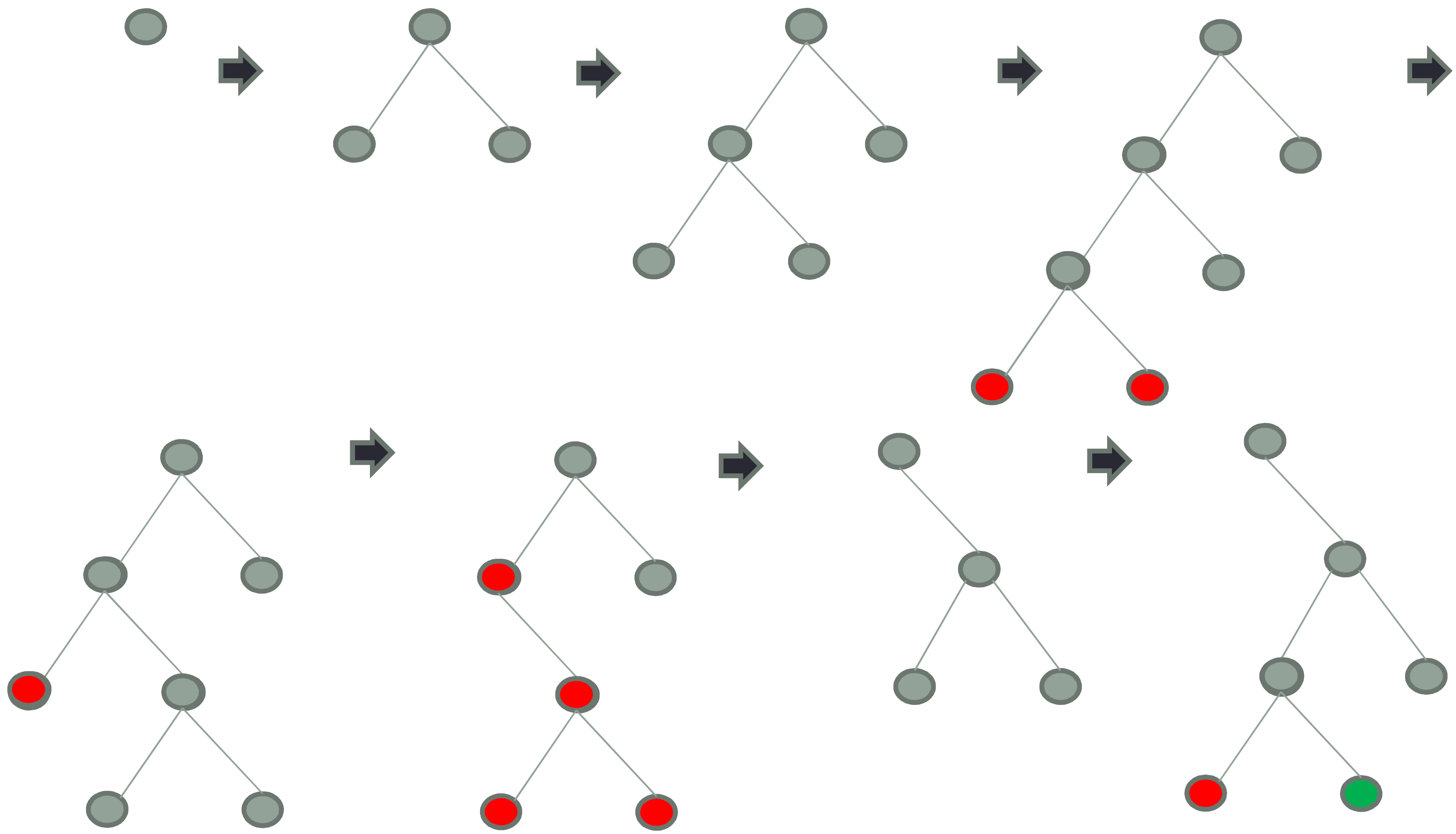
Busca em Profundidade



Busca em Profundidade

- **Análise de Complexidade Temporal e Espacial:**
  - Tem requisitos de memória modestos em relação a busca em largura: só precisa armazenar um caminho da raiz até um nó folha, junto com os nós-irmãos não-expandidos de cada nó do caminho:
    - Número de nós para fator de ramificação ***b*** e profundidade máxima ***m*** = *bm+1*
  - Usando as mesmas suposições (b=10, 10.000 nós/segundo, 1000 bytes/nó), e supondo que nós na profundidade do nó estado-objetivo não tem sucessores:
    - Para d=12 necessitariamos de 118kbytes de memória (10 bilhões de vezes menor que na busca em largura)
    - Complexidade Temporal é exponencial = $O(b^m)$
    - Complexidade Espacial (memória) <<< Complexidade Temporal

Busca em Profundidade



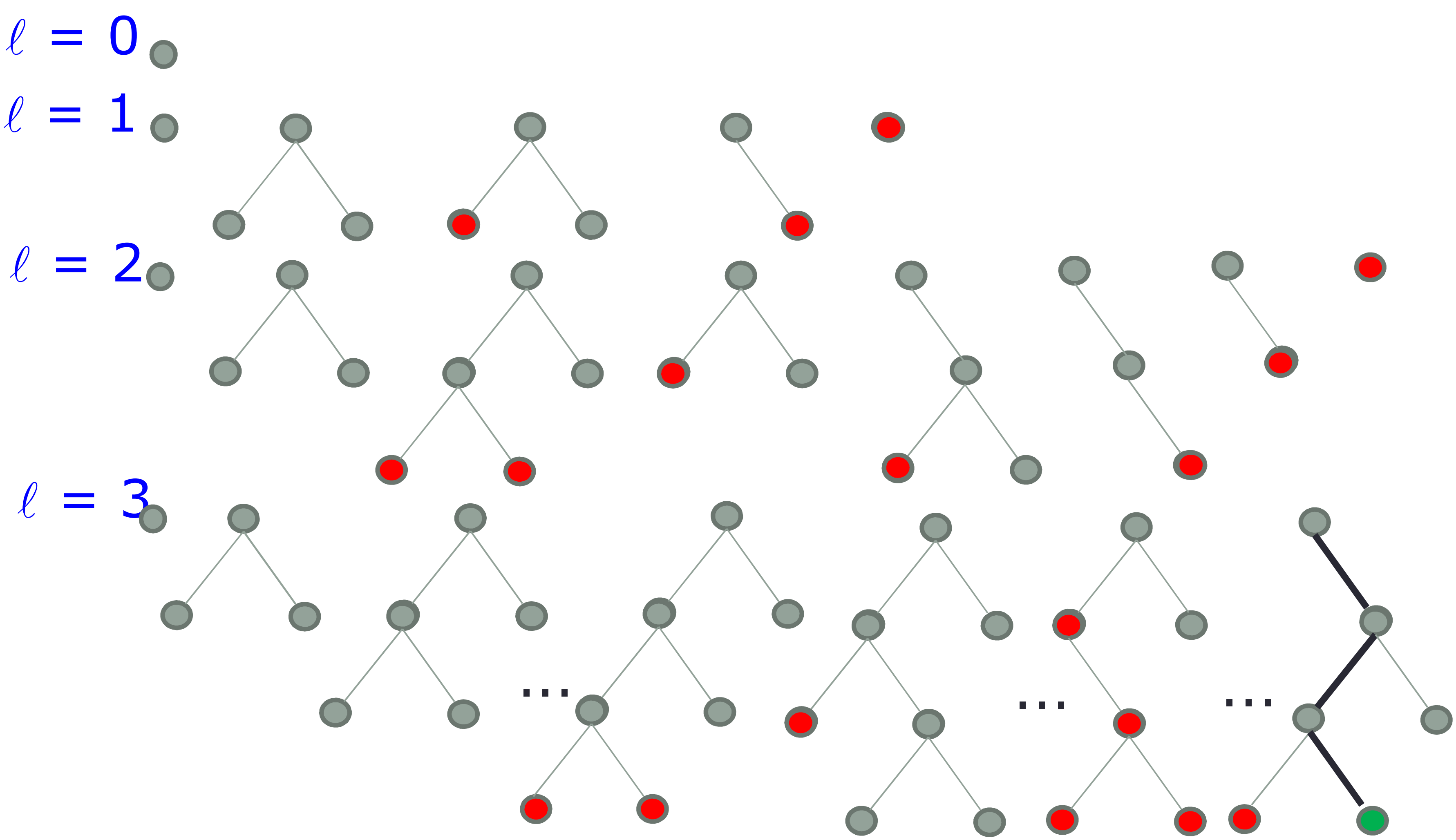
Busca em Profundidade

- Esta estratégia não é completa nem ótima! ↓↓
  - Basta a subárvore da esquerda ter profundidade infinita
  - Dá preferência a estados-objetivos mais "profundos", os quais não são necessariamente os ótimos
- Complexidade espacial: ↑
  - Mantém na memória somente o caminho que está sendo expandido no momento e os nós-irmãos (de mesmo pai) não expandidos dos nós que integram o caminho (p/ possibilitar o retorno de nível)
  - Assim: pouca exigência de memória, $b.m + 1$ ($=O(b.m)$), o que é bom p/ problemas c/ muitas soluções → pode ser bem mais rápida que busca em largura
- Complexidade temporal: ↓
  - $O(b^m)$, no pior caso → ruim se $m >> d$ ou se $m \rightarrow \infty$
  - Deve ser evitada quando as árvores geradas são muito profundas ou geram caminhos infinitos

Busca Cegas – Demais estratégias

- Busca em profundidade limitada
  - Estratégia: evita o problema de caminhos muito longos ou infinitos impondo um limite máximo (ℓ) de profundidade p/ os caminhos
    - Só é completa se $\ell \geq d$ → mas geralmente não se sabe *a priori* o valor de *d*. Não é ótima ↓
    - Complexidade temporal: $O(b^{\ell})$; ↓ e espacial: $O(b\ell)$ ↑
- Busca com aprofundamento iterativo
  - Estratégia: executa a busca em profundidade limitada, iterativamente, aumentando sempre o valor do limite da profundidade ℓ
    - Fixa $\ell(0) = i$ e executa a BPL;
    - Se insucesso, recomeça a BPL c/ $\ell(t) = \ell(t\text{-}1) + n$ (p/ qquer $n > 0$ fixo);

Busca com aprofundamento iterativo

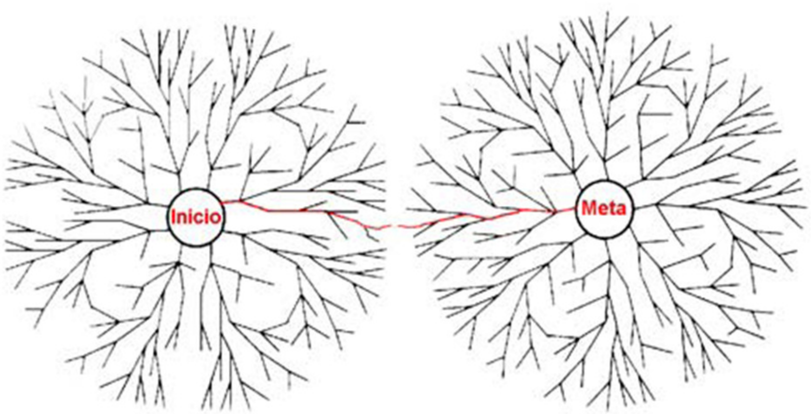


Busca com aprofundamento iterativo

- Esta estratégia combina as vantagens da busca em largura com as da busca em profundidade
- É ótima e completa ↑↑
  - Quando $n = 1$ e operações com custos de passo iguais
- Complexidade espacial: ↑
  - Pouca exigência de memória, necessitando armazenar no máximo na ordem de $O(b.d)$ nós
- Complexidade temporal: ↓
  - No. total de nós gerados: $d(b) + (d\text{-}1)b^2 + \ldots (1)b^d = O(b^d)$
- ➔ Em geral, é a melhor estratégia cega quando o espaço de estados é grande e a profundidade da solução é desconhecida

Busca Bidirecional

- Idéia geral: executar duas buscas simultâneas, uma para frente, a partir do estado inicial, e outra para trás, desde o estado-objetivo
  - A busca pára qdo. os 2 processos geram nós representando um mesmo estado intermediário → pelo menos uma das árvores precisa ser mantida inteiramente na memória
  - Aspecto interessante: é possível se utilizar de estratégias de busca diferentes em cada direção da busca!



Busca Bidirecional

- É completa e ótima ↑↑
  - Ao se adotar busca em largura em ambas as direções
- Complex. temporal/espacial: $O(2b^{d/2}) = O(b^{d/2})$
- Apresenta alguns problemas: ↓
  - Como gerar os sucessores na busca p/ trás?
    - Se todas as operações forem reversíveis, então o conjunto de sucessores de um nó qualquer será igual ao conjunto de seus antecessores
    - Mas se as operações não forem reversíveis, então a geração dos antecessores fica comprometida!
      - Como determinar exatamente todos os estados que precedem um estado de xeque-mate?
  - E no caso de se ter vários estados-objetivos?
    - Cria-se um estado-objetivo fictício cujos predecessores imediatos são os estados-objetivos reais e inicia c/ ele

Busca de soluções - Estratégias

Resumo da ópera

Critério	BL	BCU	BP	BPL	BPI	BB(*)
TEMPO	$O(b^{d+1})$	$O(b^{d+1})$	$O(b^{m})$	$O(b^{\ell})$	$O(b^{d})$	$O(b^{d/2})$
ESPAÇO	$O(b^{d+1})$	$O(b^{d+1})$	$O(b.m)$	$O(b.\ell)$	$O(b.d)$	$O(b^{d/2})$
ÓTIMA?	SIM	SIM	NÃO	NÃO	SIM	SIM
COMPLETA?	SIM	SIM	NÃO	NÃO (se $\ell < d$)	SIM	SIM

BL = Busca em largura
BCU = Busca de custo uniforme
BP = Busca em profundidade

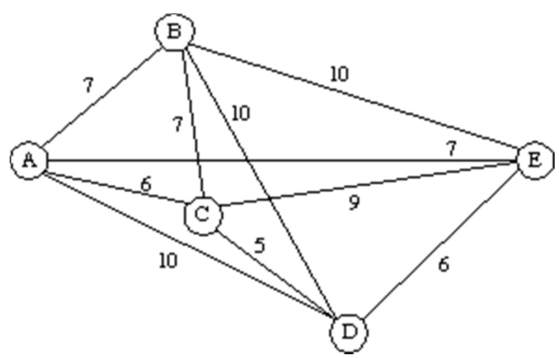
BPL = BP limitada
BPI = BP iterativa
BB = Busca bidirecional

(*) se aplicável

b: fator de ramificação
d: profundidade da solução
m: máxima profundidade da árvore

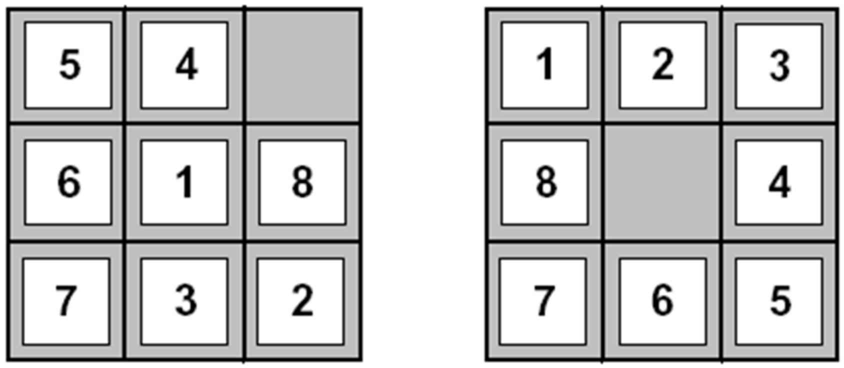
Exemplos de problemas clássicos

- **Problema das jarras de água**
  - Você tem duas jarras, uma de 4 litros e uma de 3 litros. Nenhuma delas tem qualquer marcação de medidas. Há uma bomba que pode ser usada para encher as jarras com água. Além disso, pode-se jogar a água de uma jarra para a outra ou jogar fora. Como é que se consegue colocar exatamente 2 litros de água na jarra de 3 litros?
- **Problema do caixeiro-viajante**
  - Um vendedor tem uma lista de cidades que precisa visitar exatamente uma vez. Há estradas diretas entre cada par de cidades da lista. Encontre a rota que o vendedor deverá seguir para que a viagem seja a menor possível, e que comece e termine em uma mesma cidade, que poderá ser qualquer uma da lista.



Exemplos de problemas clássicos

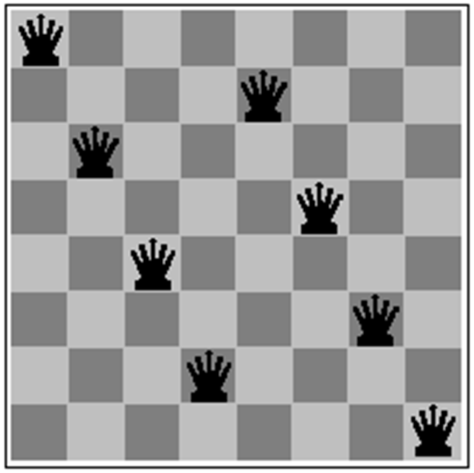
- **Puzzle-8**
  - Pretende-se encontrar a menor seqüência de movimen-tos para passar de uma configuração inicial do quebra-cabeça para a situação final, tal como na figura abaixo.



- **Problema dos missionários e canibais**
  - 3 missionários e 3 canibais estão numa das margens do rio com um barco que só leva 2 pessoas. Encontrar uma série de travessias de forma a levar os 6 tripulantes para a outra margem do rio sem nunca deixar mais canibais do que missionários em qualquer uma das duas margens durante o processo!

Exemplos de problemas clássicos

- **8-Rainhas em um tabuleiro de xadrez**
  - Colocar 8 rainhas num tabuleiro de xadrez sem se colocarem mutuamente em xeque. Lembre-se que a região de xeque de uma rainha é toda a extensão da vertical, horizontal e diagonais em relação à posição onde se encontra. Enunciado de uma outra forma: como colocar as 8 rainhas no tabuleiro de xadrez de tal forma que cada linha, cada coluna e cada diagonal contenha uma e apenas uma rainha?



Exemplos de problemas clássicos

- **Criptogramas**
  - Encontrar dígitos (todos diferentes), um para cada letra, de forma a que a soma seja correta.

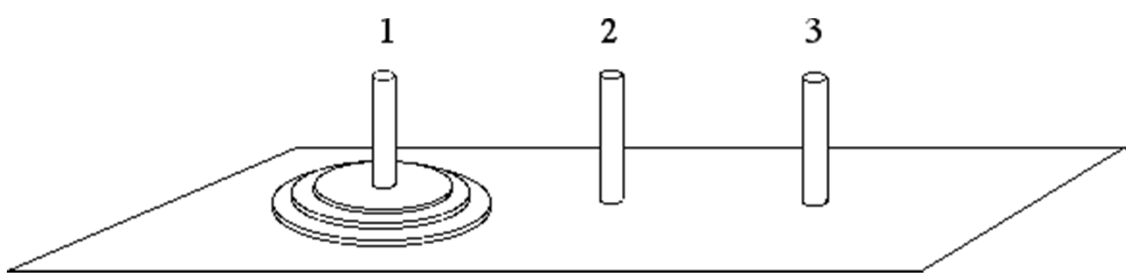
```
   FORTY
     TEN
+    TEN
---------
   SIXTY
```

```
C1 C2 C3 C4
    S E N D
  + M O R E
-----------
  M O N E Y
```

- **Coloração de mapas**
  - Dado um mapa (de países ou outras regiões) e um conjunto de cores pretende-se colorir as regiões de maneira que nunca duas regiões adjacentes tenham a mesma cor. Uma outra formulação pode ser: qual o número mínimo de cores necessárias para colorir um dado mapa de maneira que nunca duas regiões adjacentes tenham a mesma cor?

Exemplos de problemas clássicos

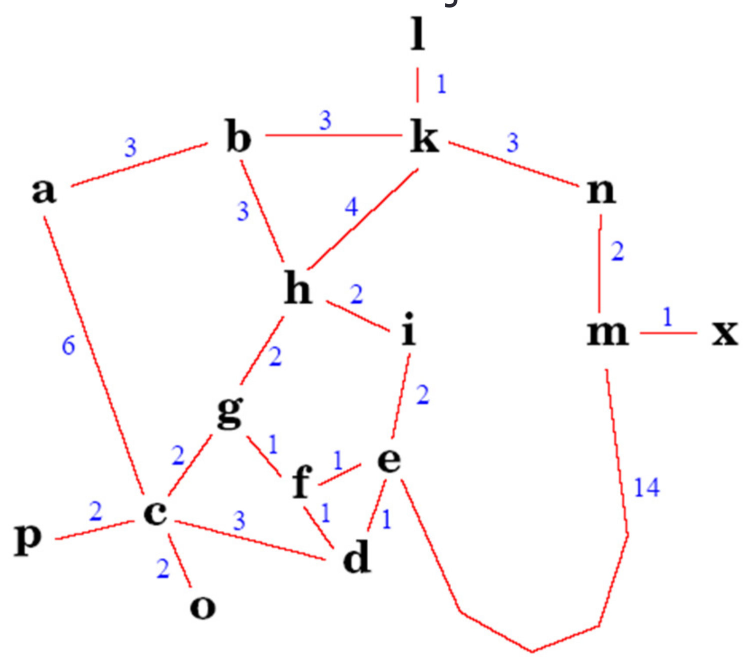
- **Torres de Hanói**
  - Em algum lugar perto de Hanói há um mosteiro onde os monges dedicam suas vidas a uma tarefa muito importante. No pátio há três postes bem altos. Em cima deles há 64 discos, cada um com um buraco no centro e cada um com um raio diferente. Quando o mosteiro foi criado, todos os discos estavam em um só poste, e cada disco estava em cima daquele com tamanho imediatamente maior que o seu. A tarefa dos monges é mover todos estes discos para um dos outros postes. Apenas um disco pode ser deslocado de cada vez, e todos os outros discos precisam estar em um dos postes. Além isso, em nenhum momento durante o processo um disco pode ser colocado sobre um disco menor. É claro que o terceiro poste pode ser usado como local temporário para os discos. Qual a maneira mais rápida para os monges concluírem sua missão?



Atividade extra-sala

Exercício 1)

Indicar o caminho de solução gerado pelas estratégias de busca cega a seguir, considerando o seguinte espaço de estados e o objetivo de ir do estado inicial "*i*" ao estado-destino "*x*": BL, BCU, BP e BPL (ℓ = 3). Assumir que um nó-sucessor só é gerado se o estado correspondente ainda não fizer parte da árvore. Apresentar a árvore de busca e o caminho de solução.



Atividade extra-sala

Exercício 2)

Indicar o caminho de solução para alguns dos problemas clássicos, gerado pelas estratégias de busca cega: BL, BCU, BP e BPL ($\ell = 3$). Apresentar a árvore de busca e o caminho de solução.