

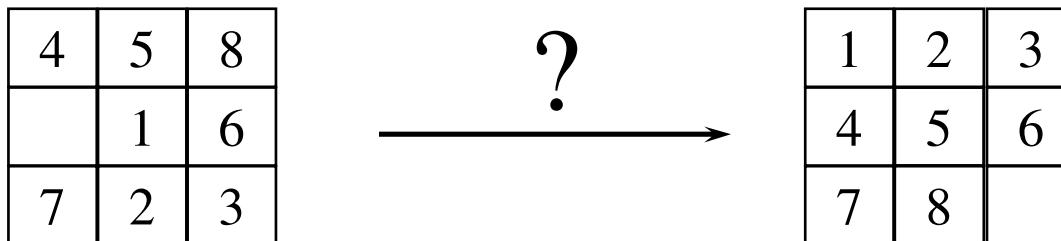


Resolução de Problemas

Agente de Resolução de Problemas (1/2)

■ O agente reativo

- Escolhe suas ações com base apenas nas percepções atuais
 - *não pode pensar no futuro, não sabe “aonde vai”*



■ Já o agente baseado em objetivo...

- sabe, pois segue um **objetivo** explícito



Agente de Resolução de Problemas (2/2)

- **Dentre as maneiras de implementar um agente baseado em objetivo existe o chamado Agente de Resolução de Problemas**
 - serve para alguns tipos de problemas
 - requer pouco conhecimento explícito
 - basicamente busca uma seqüência de ações que leve a estados desejáveis (objetivos)
- **Questões**
 - O que é um problema e como formulá-lo?
 - Como buscar a solução do problema?


Problemas e Soluções bem Definidos (1/2)

Um problema em IA é definido em termos de...

1) um espaço de estados possíveis, incluindo um estado inicial e um estado final (objetivo)

- exemplo 1: dirigir do Rio de Janeiro a Juiz de Fora
- exemplo 2: jogo de 8-números

4	5	8
	1	6
7	2	3



1	2	3
4	5	6
7	8	

2) um conjunto de ações (ou operadores) que permitem passar de um estado a outro

- ex1. dirigir de uma cidade a outra
- ex2. mover uma peça do jogo de n-números (*n-puzzle*)

Problemas e Soluções bem Definidos (2/2)

■ Espaço de Estados:


- conjunto de todos os estados alcançáveis a partir do estado inicial por qualquer seqüência de ações.

■ Definição do objetivo:

- propriedade abstrata
 - ex., condição de xeque-mate no Xadrez
- conjunto de estados finais do mundo
 - ex., estar em na cidade-destino

■ Solução:

- caminho (seqüência de *ações* ou *operadores*) que leva do estado inicial a um estado final (objetivo).



Solucionando o Problema: formulação, busca e execução

■ **Formulação do problema e do objetivo:**

- quais são os **estados** e as **ações** a considerar?
- qual é (e como representar) o **objetivo**?

■ **Busca (solução do problema):**

- processo que gera/analisa seqüências de ações para alcançar um objetivo
- *solução* = caminho entre estado inicial e estado final.

■ **Execução:**

- Executar (passo a passo) a solução **completa** encontrada



Agentes Solucionadores de Problemas

formulação, busca e execução

função Agente-Simples-SP(p) **retorna** uma *ação*
entrada: p , um dado perceptivo

$estado \leftarrow$ Atualiza-Estado ($estado, p$)

se s (seqüência de ações) está vazia

então

o (objetivo) \leftarrow Formula-Objetivo ($estado$)

$problema \leftarrow$ Formula-Problema ($estado, o$)

$s \leftarrow$ **Busca** ($problema$)

$ação \leftarrow$ Primeira ($s, estado$)

$s \leftarrow$ Resto ($s, estado$)

retorna $ação$

Medida de Desempenho na Busca

■ Desempenho de um algoritmo de busca:

- 1. O algoritmo encontrou alguma solução?
- 2. É uma boa solução?
 - **custo de caminho** (*qualidade da solução*)
- 3. É uma solução computacionalmente barata?
 - **custo da busca** (*tempo e memória*)

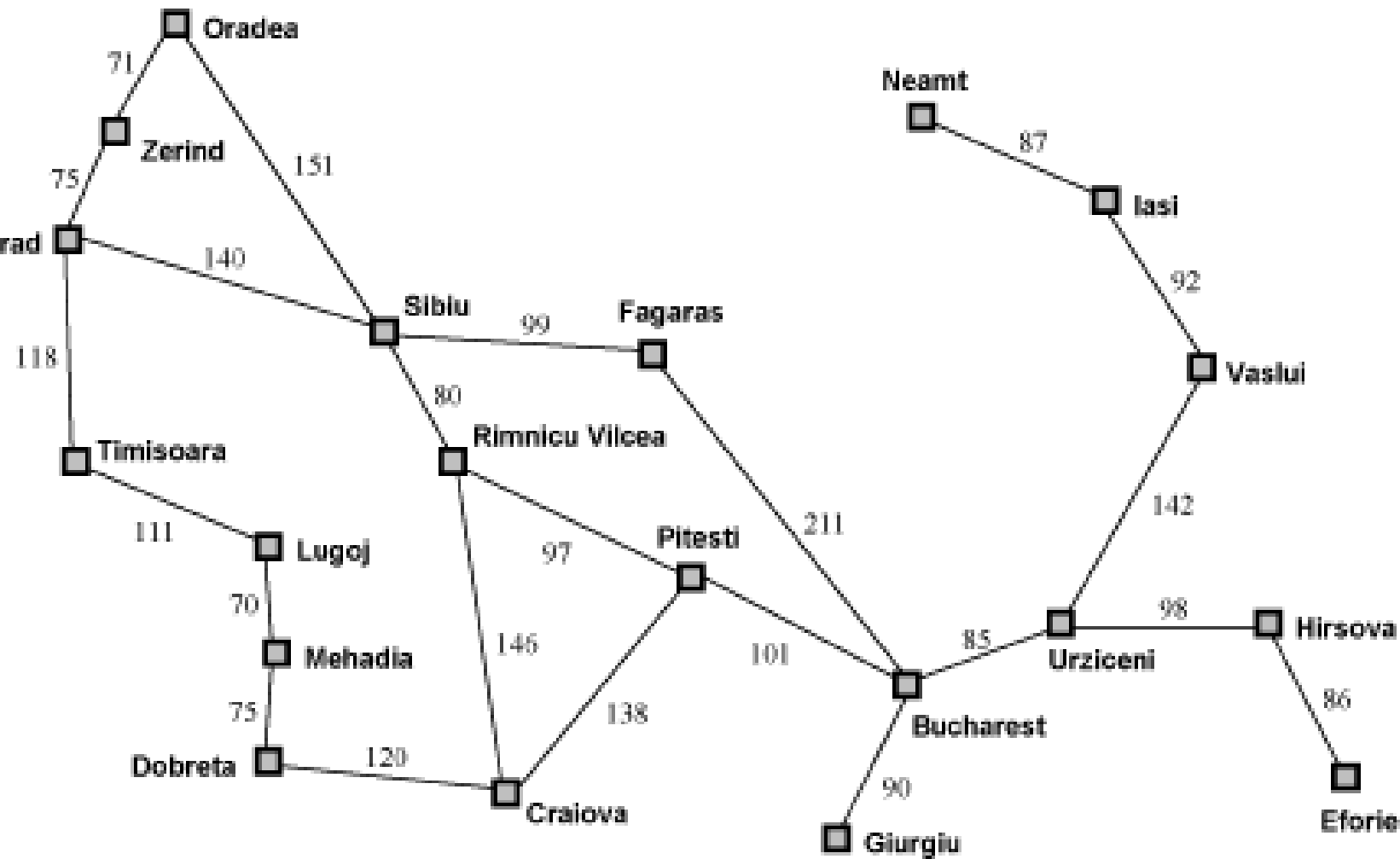
■ Custo total

- custo do caminho + custo de busca

■ Espaço de estados grande:

- compromisso (conflito) entre a melhor solução e a solução mais barata

Outro Exemplo: Ir de Arad a Bucharest



Straight line distance
to Bucharest

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

Exemplo Romênia

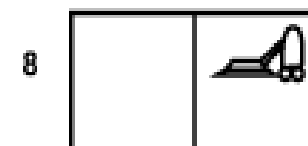
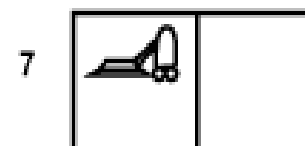
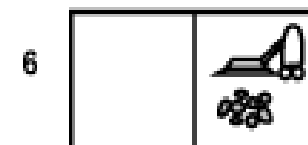
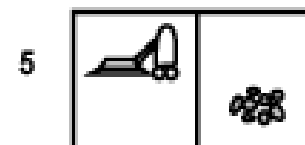
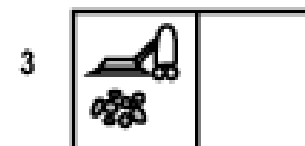
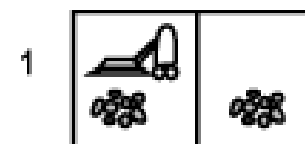
■ Ida para Bucharest:

- **estados** = cada possível cidade do mapa
- **estado inicial** = Arad
- **teste de término** = estar em Bucarest
- **operadores** = dirigir de uma cidade para outra
- **custo do caminho** = número de cidades visitadas, distância percorrida, tempo de viagem, grau de divertimento, etc

Mais um Exemplo...

■ Aspirador de pó

- estados =
- estado inicial =
- teste de término =
- operadores =
- custo da solução =



Custo Diferente => Solução Diferente

■ **Função de *custo de caminho***

- (1) número de cidades visitadas,
- (2) distância entre as cidades,
- (3) tempo de viagem, etc.

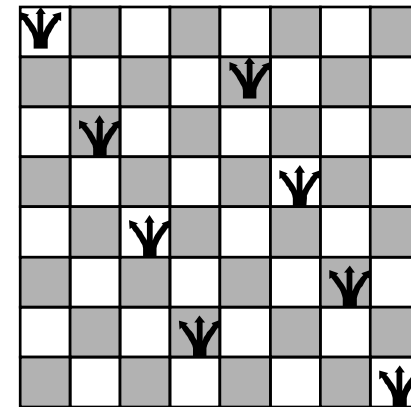
■ **Solução mais barata:**

- (1) Rio, Niterói, São Gonçalo, ...
- (2) Rio, Niterói, Mage, ...
- (3) Rio, Mage, Araruama, Cabo Frio, São Gonçalo

Importância da formulação: 8 rainhas

■ Jogo das 8 Rainhas

- dispor 8 rainhas no tabuleiro de forma que não possam se “atacar”
 - *não pode haver mais de uma rainha em uma mesma linha, coluna ou diagonal*
- somente o custo da busca conta
 - *não existe custo de caminho*



■ Existem diferentes estados e operadores possíveis

- essa escolha pode ter consequências boas ou nefastas na complexidade da busca ou no tamanho do espaço de estados

Importância da formulação: 8 rainhas

■ **Formulação A**

- estados: qualquer disposição com n ($n \leq 8$) rainhas
- operadores: adicionar uma rainha a qualquer quadrado
- 64^8 possibilidades: vai até o fim para testar se dá certo

■ **Formulação B**

- estados: disposição com n ($n \leq 8$) rainhas sem ataque mútuo (teste gradual)
- operadores: adicionar uma rainha na coluna vazia mais à esquerda em que não possa ser atacada
- melhor (2057 possibilidades), mas pode não haver ação possível

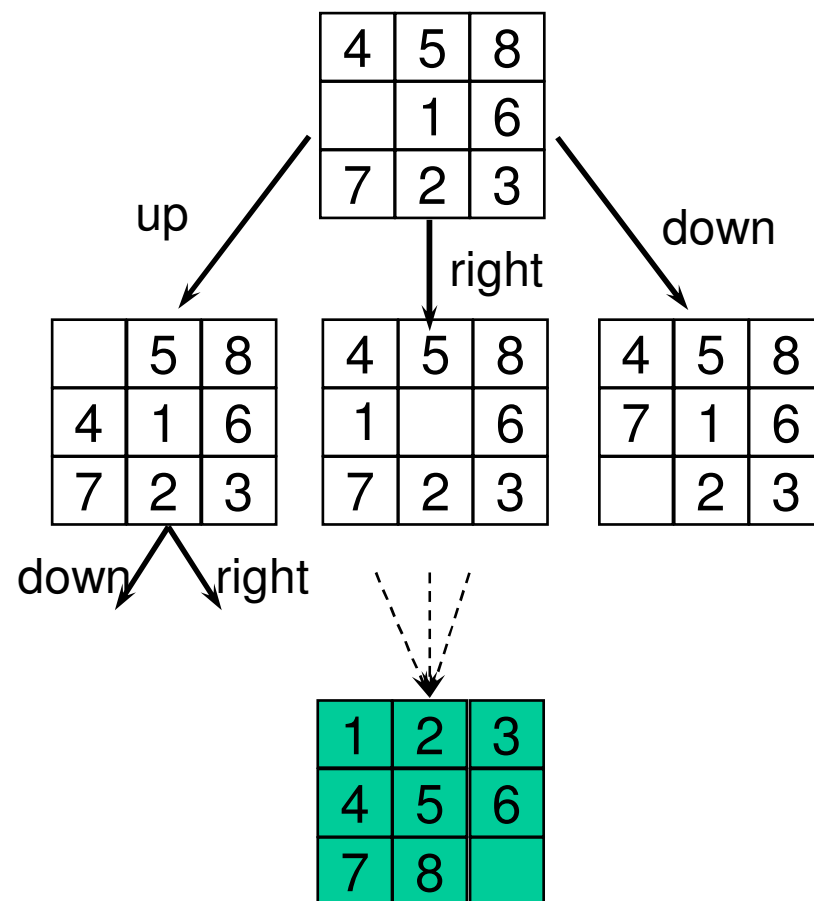
■ **Formulação C**

- estados: disposição com 8 rainhas, uma em cada coluna
- operadores: mover uma rainha atacada para outra casa na mesma coluna

Importância da formulação: 8-números

■ Jogo de 8 números:


- **estados** = cada possível configuração do tabuleiro
- **estado inicial** = qualquer um dos estados possíveis
- **teste de término** = ordenado, com branco na posição [3,3]
- **operadores** = mover branco (esquerda, direita, para cima e para baixo)
- **custo da solução** = número de passos da solução





Algumas Aplicações

Aplicações de Busca: “Toy Problems”

- 
- Jogo das n rainhas
 - Jogo dos n números (*n -puzzle*)
 - Torre de Hanoi
 - Palavras cruzadas
 - Canibais e missionários

Aplicações: Problemas Reais

■ **Cálculo de rotas (pathfinding)**

- rotas em redes de computadores
- sistemas de planejamento de viagens
- planejamento de rotas de aviões
- Caixeiro viajante
- Jogos de computadores (rotas dos personagens)

■ **Alocação (Scheduling)**

- Salas de aula
- Máquinas industriais (job shop)

■ **Projeto de VLSI**

- Cell layout
- Channel routing

Aplicações: Problemas Reais

■ **Navegação de robôs:**

- generalização do problema da navegação
- robôs movem-se em espaços contínuos, com um conjunto (infinito) de possíveis ações e estados
 - *controlar os movimentos do robô no chão, e de seus braços e pernas requer espaço multi-dimensional*

■ **Montagem de objetos complexos por robôs:**

- ordenar a montagem das diversas partes do objeto

■ **etc...**



Problemas com Informação Parcial

Problemas com informação Parcial

■ Até agora só vimos problemas de estado único

- o agente sabe em que estado está e pode determinar o efeito de cada uma de suas ações
 - *sabe seu estado depois de uma seqüência qualquer de ações*
- Solução: seqüência de ações

■ Porém existem 3 outros tipos de problemas...

Problemas com Informação Parcial

■ Sensorless or conformant problem

- Agente não sabe seu estado inicial (percepção deficiente)
- Deve raciocinar sobre os conjuntos de estados
- Solução: seqüência de ações (via busca)

■ Problema de contingência

- Efeito das ações não-determinístico e/ou mundo parcialmente observável => novas percepções depois de ação
 - *ex. aspirador que suja ao sugar e/ou só percebe sujeira localmente*
- Solução: árvore de decisão (via planejamento)

■ Problema exploratório (on-line)

- Espaço de estados desconhecido
 - *ex. dirigir sem mapa*
- Solução.... via aprendizagem por reforço

Problemas com Informação Parcial

■ Estado simples

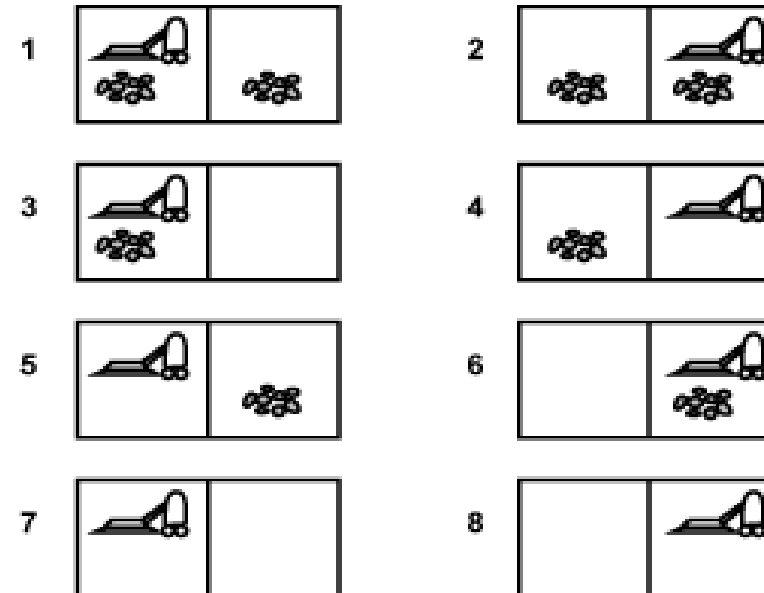
- Início: 5, Solução: [dir, suga]

■ Conformant problem

- Percepção deficiente
- Início: {1,2,3,4,5,6,7,8}
- Direita => {2,4,6,8}, Sugar => {4,8},...
- Solução: [dir, suga, esq, suga]

■ Problema de contingência

- Efeito das ações não-determinístico
- Início: [lado esq, sujo] = {1,3}
- Solução? Sugar => {5,7}, Dir => {6,8}, Sugar no 6 => 8 mas sugar no 8 => 6
- Solução: [sugar, dir, se sujo sugar]
- Solução geral: [dir, se sujo suga, esq, se sujo suga]





Buscando Soluções

Busca Cega

Busca em Espaço de Estados

- Uma vez o problema bem formulado... o estado final deve ser “buscado”
- Em outras palavras, deve-se usar um **método de busca** para saber a **ordem correta de aplicação dos operadores** que levará do estado inicial ao final
- Isto é feito por um processo de **geração** (de estados possíveis) **e teste** (para ver se o objetivo está entre eles)
- Uma vez a busca terminada com sucesso, é só **executar a solução** (= conjunto ordenado de operadores a aplicar)

Busca em Espaço de Estados: Geração e Teste

■ Fronteira do espaço de estados

- nós (estados) a serem expandidos no momento.

■ Algoritmo:

Obs: começa com a fronteira contendo o estado inicial do problema.

1. Selecionar o primeiro nó (estado) da *fronteira* do espaço de estados;
- se a fronteira está vazia, o algoritmo termina com falha.
2. Testar se o nó é um estado final (objetivo):
- se “sim, então retornar nó - a busca termina com sucesso.
3. Gerar um novo conjunto de estados pela aplicação dos operadores ao nó selecionado;
4. Inserir os nós gerados na *fronteira*, de acordo com a estratégia de busca usada, e voltar para o passo (1).

Métodos de Busca

■ Busca exaustiva ou cega

- Não sabe qual o **melhor** nó da fronteira a ser expandido = menor custo de caminho desse nó até um ***nó final*** (*objetivo*).

■ Busca heurística - informada

- Estima qual o melhor nó da fronteira a ser expandido com base em ***funções heurísticas*** => conhecimento



Busca Cega

■ Estratégias para determinar a ordem de ramificação dos nós:

1. Busca em largura
2. Busca de custo uniforme
3. Busca em profundidade
4. Busca com aprofundamento iterativo

■ Direção da ramificação:

1. Do estado inicial para um estado final
2. De um estado final para o estado inicial
3. Busca bi-direcional

Busca Cega (Blind Search ou Uninformed Search)

- Uma estratégia de busca é dita cega se ela não leva em conta informações específicas sobre o problema a ser resolvido.
- Tipos de Busca Cega
 - Busca em largura
 - Busca pelo custo uniforme
 - Busca em profundidade
 - Busca em profundidade limitada
 - Busca por aprofundamento iterativo
 - Busca bidirecional



Cr terios de Avalia  o das Estrat gicas de Busca

■ Completa?

- a estrat gia **sempre** encontra uma solu  o quando existe alguma?

■  tima?

- a estrat gia encontra **a melhor solu  o** quando existem solu   es diferentes?
 - *menor custo de caminho*

■ Custo de tempo?

- quanto **tempo** gasta para encontrar uma solu  o?

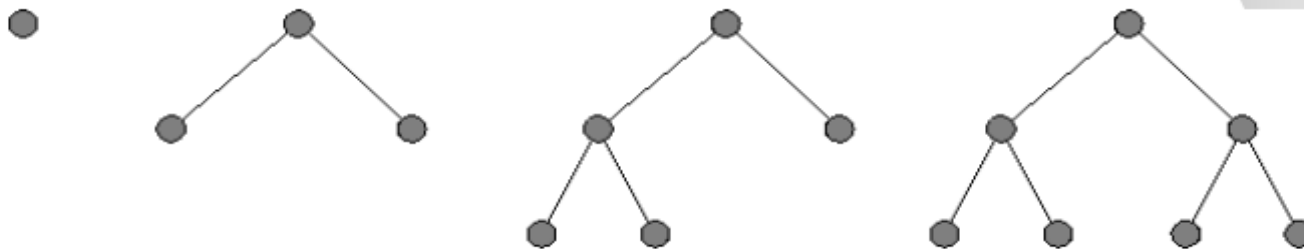
■ Custo de mem ria?

- quanta **mem ria**   necess ria para realizar a busca?

Busca Cega

Busca em Largura (Amplitude)

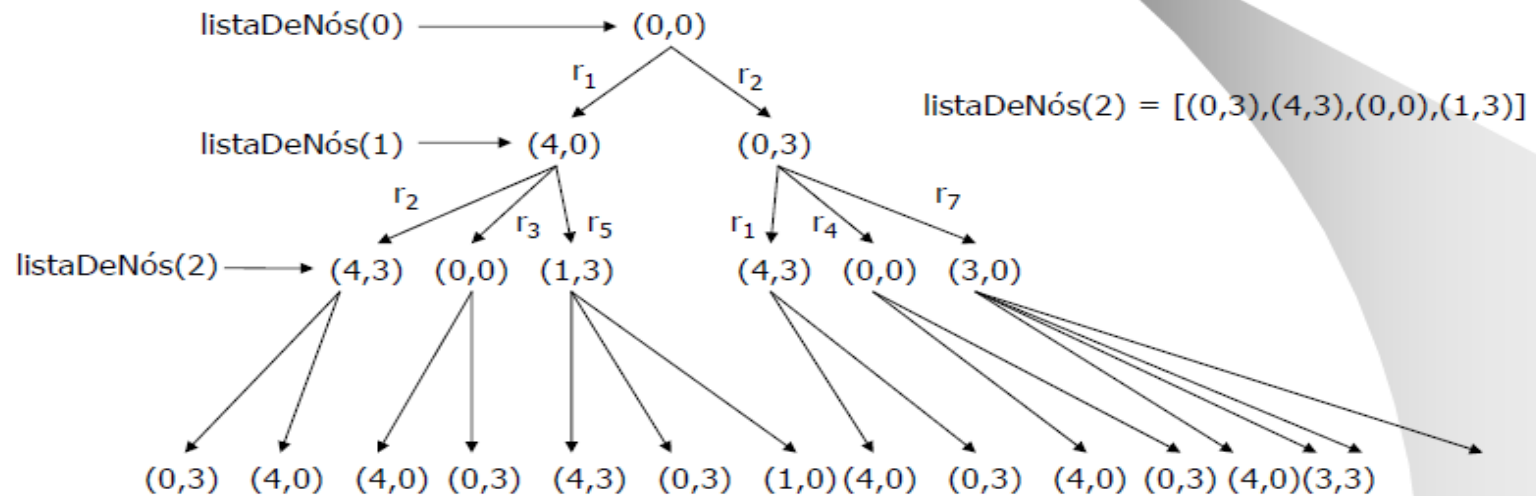
- Consiste em construir uma árvore de estados a partir do estado inicial, aplicando a cada momento, todas as regras possíveis aos estados do nível mais baixo, gerando todos os estados sucessores de cada um destes estados. Assim, cada nível da árvore é completamente construído antes de qualquer nodo do próximo nível seja adicionado à árvore



Busca Cega

Busca em Largura (Amplitude)

- Exemplo: Um balde de 4 litros e um balde de 3 litros.
Inicialmente vazios.
 - Estado Final: um dos baldes com 2 litros de água.



Busca Cega

Busca em Largura (Amplitude)

- Características: Completa e Ótima
 - Se existe solução, esta será encontrada;
 - A solução encontrada primeiro será a de menor profundidade.
- Análise de Complexidade - Tempo e Memória
 - Seja um fator de ramificação b .
 - Nível 0: 1 nó
 - Nível 1: b nós
 - Nível 2: b^2 nós
 - Nível 3: b^3 nós
 - Nível d (solução) b^d nós

Busca Cega

Busca em Largura (Amplitude)

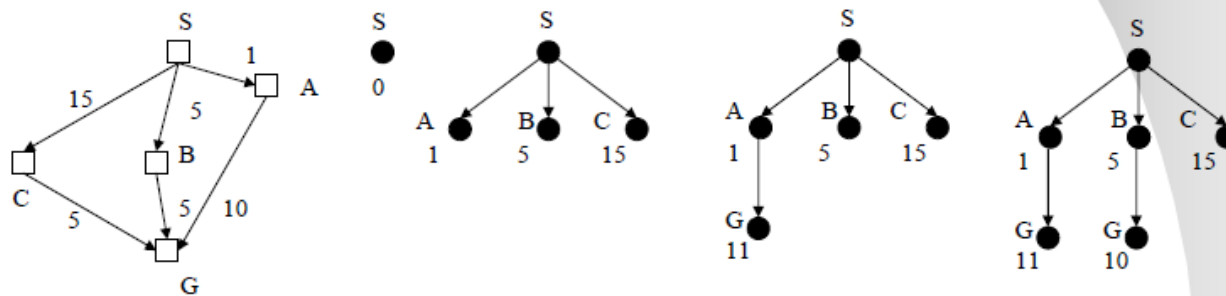
- Análise de Complexidade - Tempo e Memória

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Busca Cega

Método do Custo Uniforme

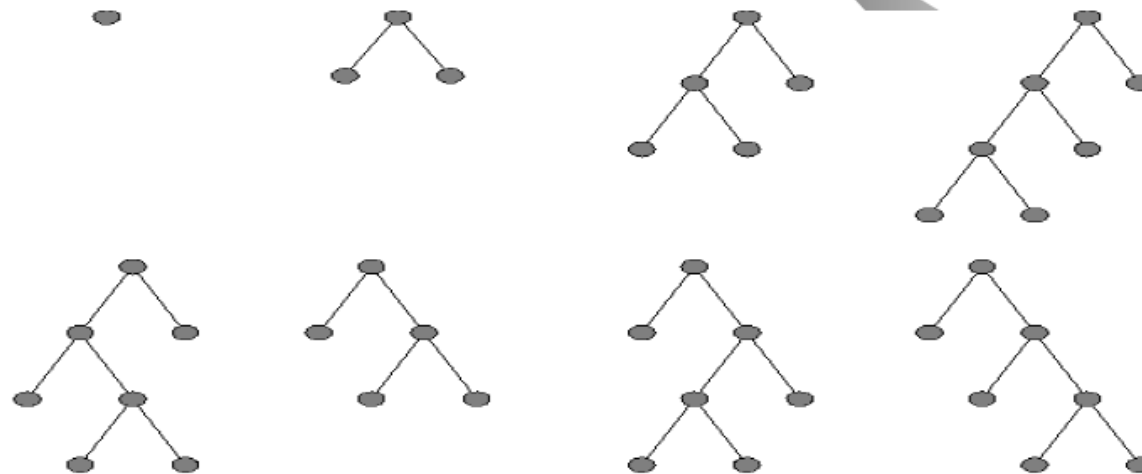
- Supondo que exista um "custo do caminho" associado a cada nó percorrido e que se deseje achar o caminho de custo mínimo.
- Neste caso, o algoritmo anterior é modificado para expandir primeiro o nó de menor custo.
- Exemplo: Problema de Rota entre S e G



Busca Cega

Busca em Profundidade

- Procurar explorar completamente cada ramo da árvore antes de tentar o ramo vizinho.



Busca Cega

Busca em Profundidade

- Exemplo: Um balde de 4 litros e um balde de 3 litros. Inicialmente vazios. Estado Final: um dos baldes com 2 litros de água.
- O que acontece quando nenhuma regra pode ser aplicada, ou a árvore atinge uma profundidade muito grande sem que tenha encontrado uma solução?
 - Neste caso ocorre o BACKTRACKING, ou seja, o algoritmo volta atrás e tenta outro caminho.
 - Considere o seguinte sistema de produção:
 $E = \{0, 1, 2, 3, 4, 5\}$
 $e_0 = 0$
 $F = \{3\}$
 $R = \{ r_1 = (x | x \geq 1 \text{ e } x \leq 2) \rightarrow (2 * x)$
 $r_2 = (x | \text{é Par}(x)) \rightarrow (x + 1) \}$

Busca Cega

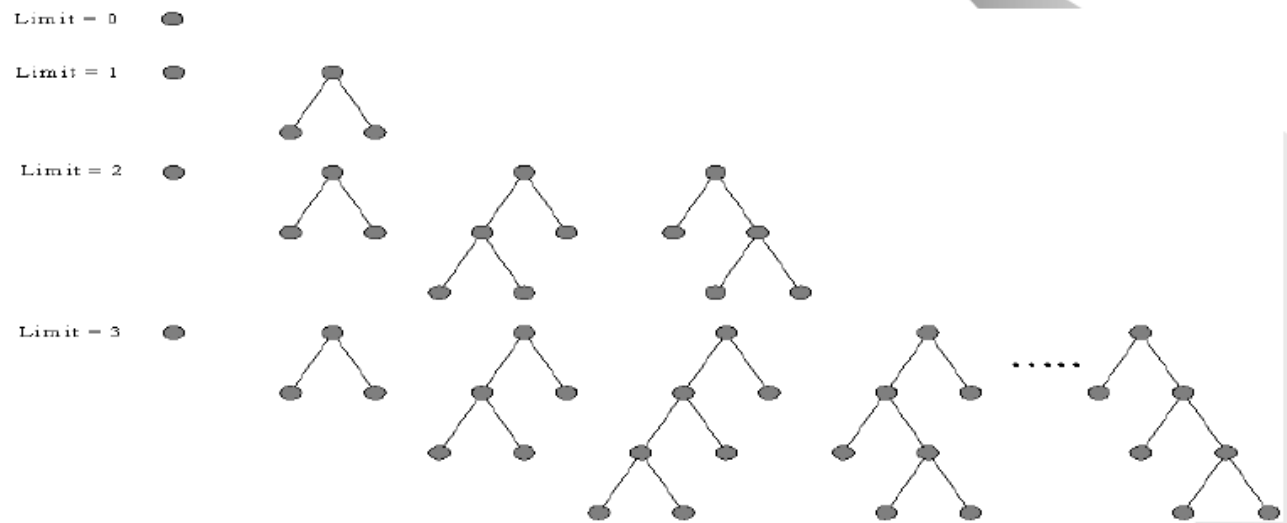
Busca em Profundidade

- ♦ Características: Não é Completa e Não é Ótima
 - Se admitir estados repetidos ou um nível máximo de profundidade, pode nunca encontrar a solução.
 - A solução encontrada primeiro poderá não ser a de menor profundidade.
 - O algoritmo não encontra necessariamente a solução mais próxima, mas pode ser **MAIS EFICIENTE** se o problema possui um grande número de soluções ou se a maioria dos caminhos pode levar a uma solução.
- ♦ Análise de Complexidade - Tempo e Memória
 - Seja m a profundidade máxima e um fator de ramificação b .
 - Tempo: b^m
 - Memória: $b.m$

Busca Cega

Busca por Aprofundamento Iterativo

- Teste de todos os possíveis limites com busca por profundidade limitada.
- Em geral é o melhor método quando o espaço de busca é grande e a profundidade é desconhecida.



Busca Cega

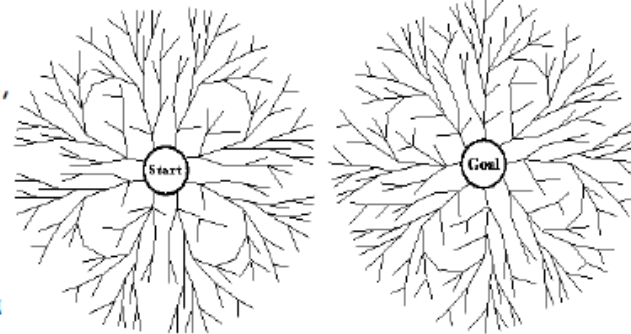
Busca Bidirecional

- A idéia deste método de busca é procurar simultaneamente "para a frente" a partir do estado inicial e "para trás" a partir do estado final, e parar quando as duas buscas se encontrarem no meio.
- Nem sempre isto é possível, para alguns problemas os operadores não são reversíveis, isto é, não existe a função predecessora e portanto não é possível fazer a busca "para trás".

- **Análise de Complexidade**

- Comparando com a busca em largura, o tempo e o espaço para a busca é proporcional a $2b^{d/2}$, onde d é o nível onde está a solução e b é o fator de ramificação da árvore.

- Exemplo: Para $b=10$ e $d=6$, na busca em largura seriam gerados 1.111.111 nós, enquanto que na busca bidirecional seriam gerados 2.222 nós.





Comparando Estratégias de Busca Exaustiva

Critério	Largura	Custo Uniforme	Profundidade	Aprofundamento Iterativo
Tempo	b^d	b^d	b^m	b^d
Espaço	b^d	b^d	bm	bd
Otima?	Sim	Sim*	Não	Sim
Completa?	Sim	Sim	Não	Sim

Comparação entre Métodos de Busca

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

b: fator de ramificação

d: profundidade da solução mais rasa

m: profundidade máxima da árvore de busca

l: limite de profundidade

Evitar Geração de Estados Repetidos

■ Problema geral em busca

- expandir estados presentes em caminhos já explorados

■ É inevitável quando existe operadores reversíveis

- ex. encontrar rotas, canibais e missionários, 8-números, etc.
- a árvore de busca é potencialmente infinita

■ 3 soluções com diferentes níveis de eficácia e custo de implementação...



Evitar Estados Repetidos: soluções

- 1. Não retornar ao estado “pai”**
- 2. Não retorna a um ancestral**
- 3. Não gerar qualquer estado que já tenha sido criado antes (em qualquer ramo)**
 - requer que todos os estados gerados permaneçam na memória: custo $O(b^d)$
 - pode ser implementado mais eficientemente com *hash tables*
 - quando encontra nó igual tem de escolher o melhor (menor custo de caminho até então)

Fim

- Obrigado pela presença!

