Aula 2

Busca Não Informada

Rafael Geraldeli Rossi

Introdução

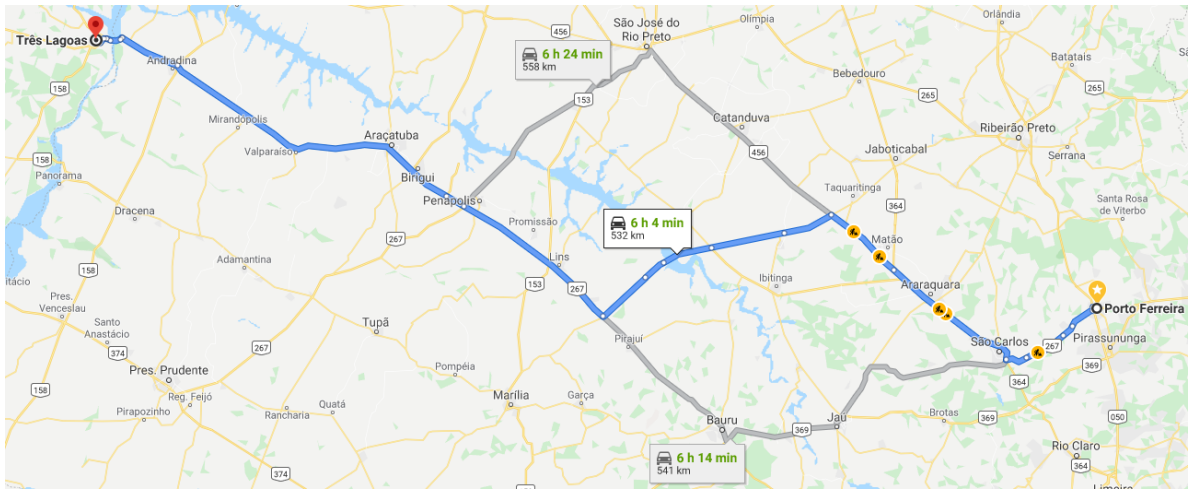
- Várias **soluções** para problemas do mundo real podem ser dadas por escolher uma solução dentre um espaço de soluções
- Porém, dependendo da quantidade de opções envolvidas, **o processo de escolha via tentativa e erro pode ser inviável devido à grande possibilidade de combinações**
- Isso tanto para um humano quanto para o computador

Introdução

- Dentro da Inteligência Artificial existem uma área denominada **Resolução de Problemas por Meio de Busca**
- Essa área consiste em métodos para achar uma solução dentro de um espaço de soluções
- Porém, ao invés do uso de força bruta (tentativa e erro de todas as possibilidades), existem técnicas para a obtenção da solução, **com um menor custo**, e **levando à solução ótima ou próximo à solução ótima**

Introdução

- Várias **soluções** para problemas do mundo real podem ser dadas por uma **sequência de ações**
- **Exemplos:**
  - Definir uma rota de viagem
    - Definir a cidade de início da rota
    - A partir da cidade de início, escolher a próxima cidade, depois a próxima, e assim por diante até chegar a cidade de destino



Introdução

- **Exemplos:**
  - Jogos de tabuleiros
    - 8-puzzle: dada a configuração inicial, mover sequencialmente as pedras em direção ao espaço em branco até obter um sequencia de pedras ordeandas
    - Jogo da velha: marcar símbolo que represente o seu jogado em um tabuleiro, um após o outro, na tentativa de obter uma sequência de símbolos

Introdução

PERGUNTA: Como automatizar a escolha da sequência de passos para que estes possam ser executados por meio de máquinas?

RESPOSTA: Realizando um busca em um espaço de estados, em que cada estado caracteriza uma solução parcial ou final de um problema

Grandes parte dos problemas de IA posdem ser resolvidos por meio de busca

Introdução

- Para os problemas e respectivos algoritmos para suas soluções que iremos considerar nesta aula, vamos assumir que o ambiente é:
  - **Observável**: é sempre possível saber qual é o estado atual
  - **Discreto**: dado um estado, existe um número finito de próximos estados a serem escolhidos
  - **Conhecido**: é possível saber quais são os próximos estados a serem escolhidos
  - **Determinístico**: as mesmas escolhas sempre levam aos mesmos estados

Introdução

- **Nesse contexto**, temos que formular o prooblema no qual queremos econtrar a solução baseando-se baseado em um **"estado inicial"**, como escolher o próximo estado e a **definição de um objetivo**
- O processo de **procurar por uma sequencia de ações** que **alcançam um objetivo** é chamada de **BUSCA**
- Nesta e na próxima aula, estudaremos algoritmos de busca em espaços de estados observáveis, conhecidos, discretos, e determinístico, e será retornado como **solução** uma **SEQUÊNCIA DE AÇÕES**

Formulação de um Problema

- **5 componentes são fundamentais**:
  - **Estado Inicial**: o estado em que o agente inicia
  - **Possíveis ações**: uma descrição de todas as ações disponíveis (e possíveis) dado um estado s
  - **Função sucessor**: retorna um estado que pode ser alcançado a partir de uma determinada ação em um estado s
  - **Teste de objetivo**: determina se um determinado estado é um estado objetivo
  - **Custo do caminho**: função que atribui um custo numérico para cada caminho

Formulação de um Problema

- Ao considerar o componentes apresentados anteriormente, podemos considerar que o problema a ser resolvido pode ser representador por um grafo de **espaço de estados**
  - Cada estado representa um **nó do grafo** (solução parcial ou final para um problema)
  - Estados que pode ser alcançados a partir de outros estados são conectados por **arestas**
  - Essas **arestas podem conter informações** para auxiliar na busca (ex: custo para mover-se de um estado para outro)

Formulação de um Problema

- **OBSERVAÇÃO 1**: assume-se que os custos dos caminhos são sempre **não negativos**

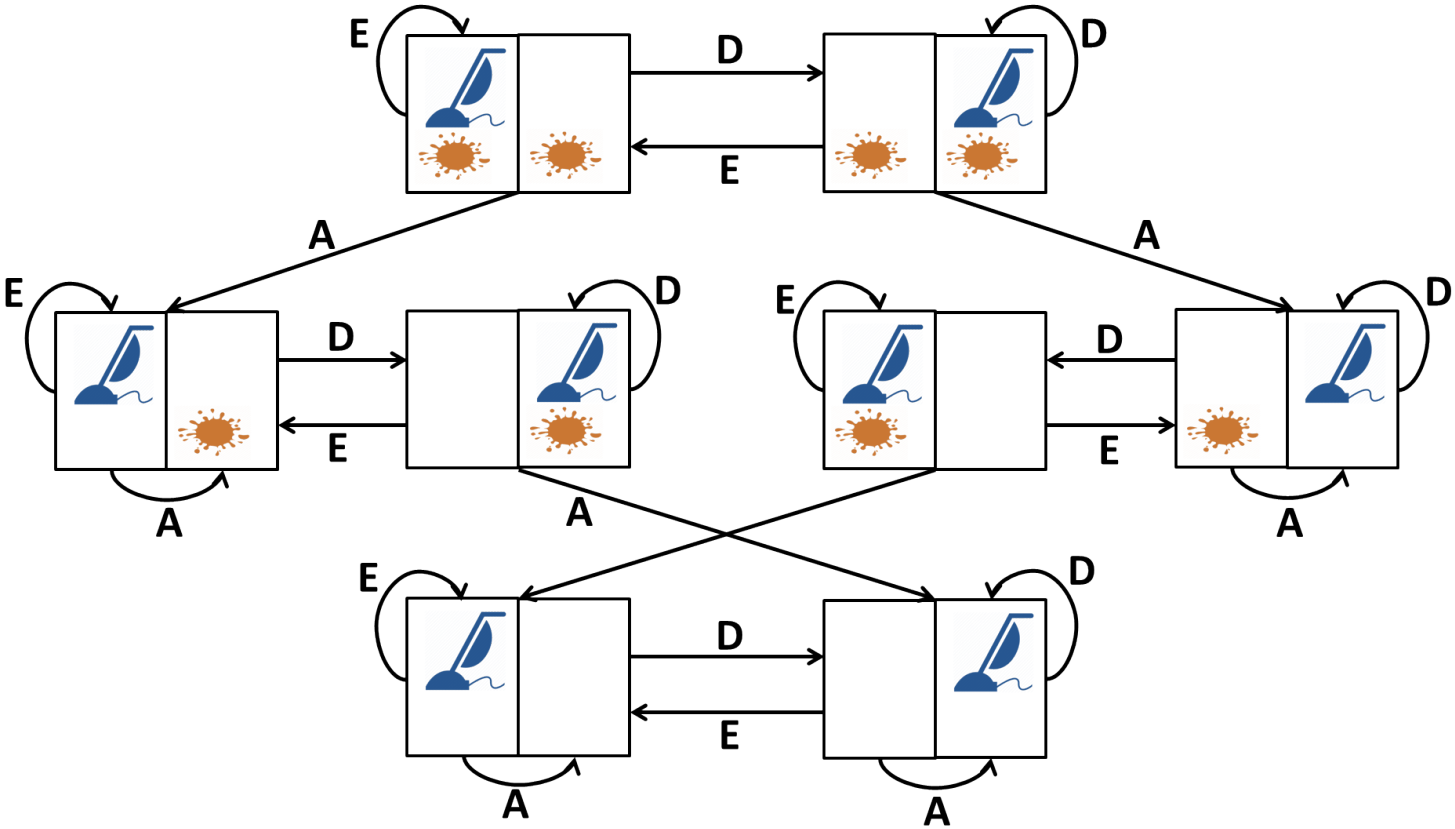
- **OBSERVAÇÃO 2**: muitas vezes objetivo é especificado por uma propriedade abstrata ao invés de um conjunto bem definido de estados
  - **Ex**: No xadrez, o objetivo é alcançar o um estado denominado "*checkmate*", no qual o rei do jogador oponente está sobe ataque e qualquer movimento movimento do rei resultará também em uma situação de ataque

Formulação de um Problema

- **OBSERVAÇÃO 3**: a qualidade da solução é medida pelo custo do caminha até a solução $\rightarrow$ **a solução ótima é aquela que apresenta um caminho de menor custo**

- **OBSERVAÇÃO 4**: dependendo de um problema, **pode haver mais de um estado objetivo**

Mundo do Aspirador de Pó



Ações

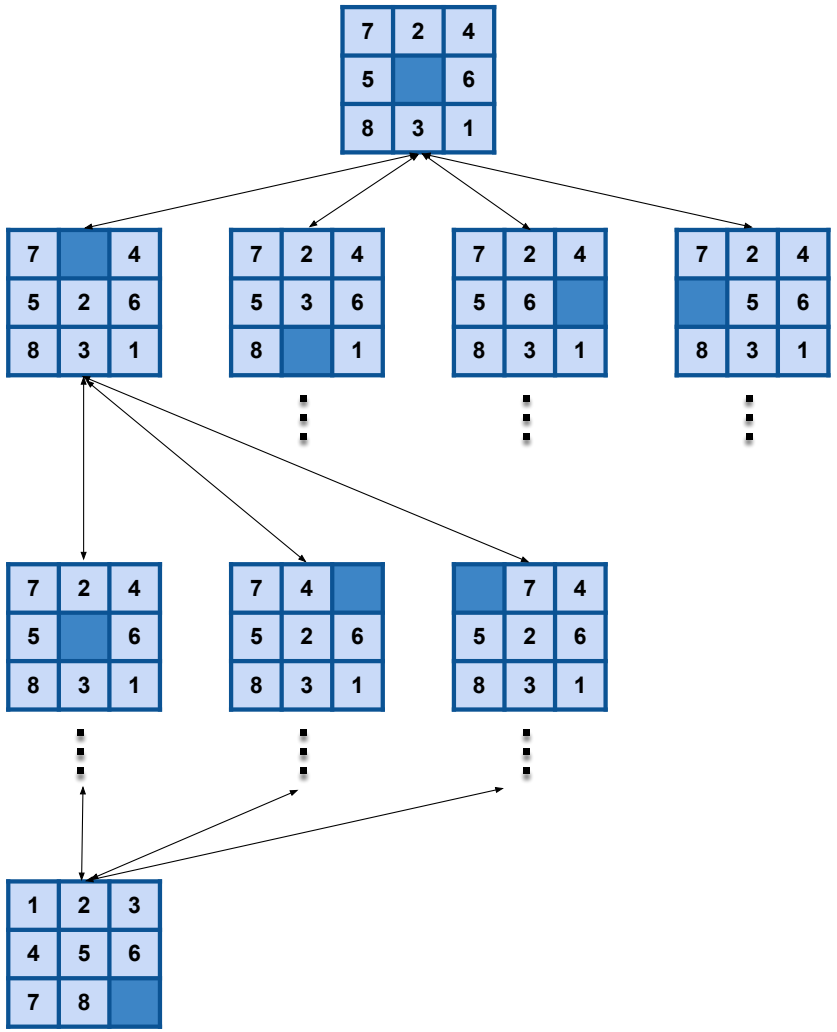
A: aspirar **D:** mover para a direita **E:** mover para a esquerda

Mundo do Aspirador de Pó

- **Estados**: o estado é determinado pela localização do aspirador e pela localização da sujeira
- **Estado Inicial**: qualquer um dos estados no topo da figura pode ser considerado como estado inicial
- **Possíveis ações**: pode-se realizar as ações de mover para a esquerda, mover para a direita e limpar
- **Função sucessor**: retorna um estado decorrente das possíveis ações

- **Teste objetivo**: verifica se todos os quadrados estão limpos
- **Custo do caminho**: cada passo tem custo 1 e portanto o custo do caminho é o número de passos realizados para se encontrar a solução

8-puzzle


7 2 4
5 6
8 3 1
7 4
5 2 6
8 3 1
7 2 4
5 3 6
8 1
7 2 4
5 6
8 3 1
7 2 4
5 6
8 3 1
7 2 4
5 6
8 3 1
7 4
5 2 6
8 3 1
7 4
5 2 6
8 3 1
1 2 3
4 5 6
7 8

8-puzzle

- **OBSERVAÇÕES**:
  - É interessante notar que o espaço de estados completo do quebra-cabeça do 8-puzzle consiste em dois subgrafos desconectados
  - Isso faz com que metade dos estados possíveis no espaço de busca seja impossível de ser alcançada a partir de qualquer estado inicial

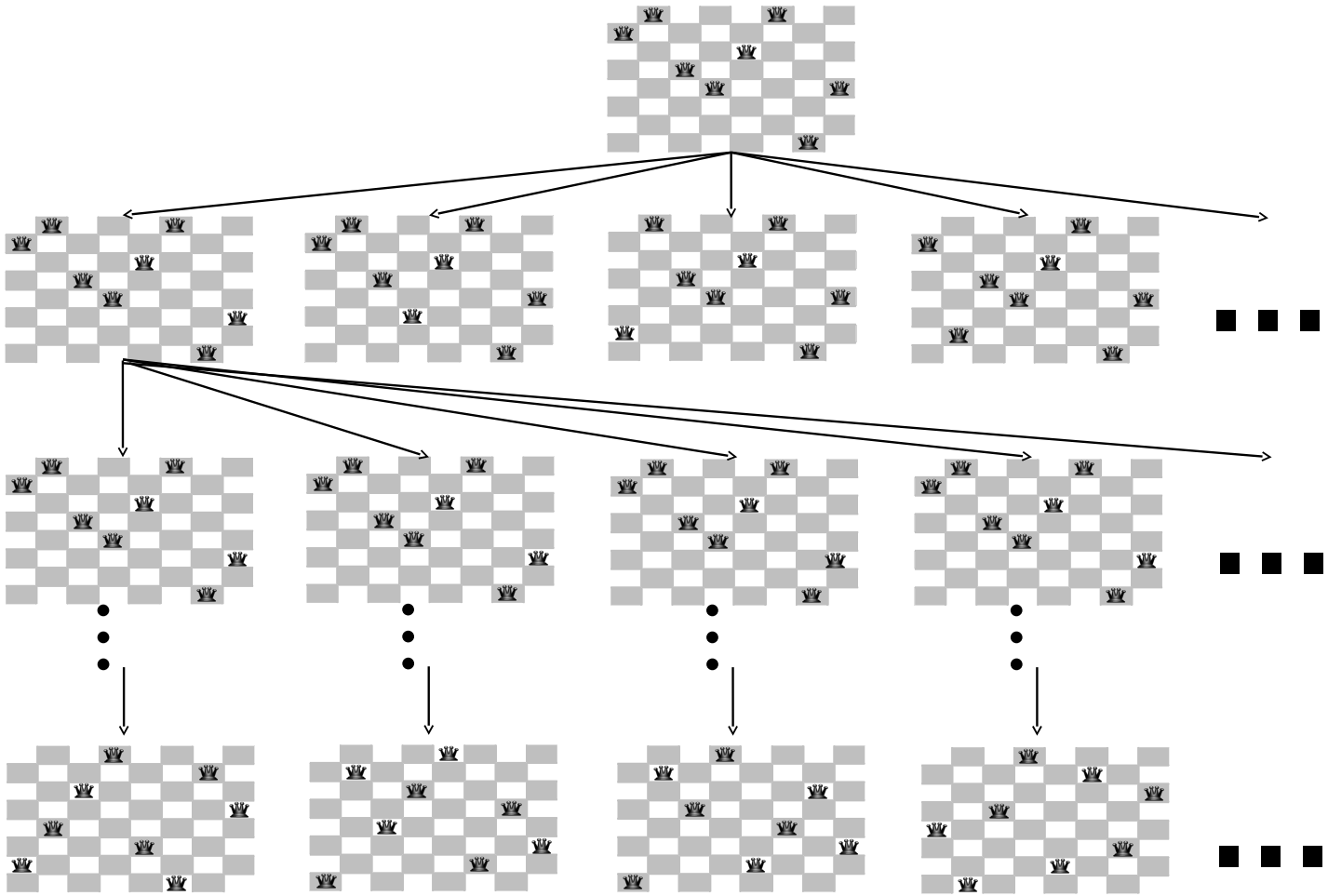
    Da forma como vimos até agora, a partir do estado inicial, vamos gerando estados sucessivos na expectativa de atingir o estado objetivo (**busca guiada por dados**)

    Porém, podemos gerar os estados a partir do estado objetivo em direção ao estado inicial (**busca guiada por objetivo**)
  - Busca guiada por dados e busca guiada por objetivos (colocar as figuras aqui).

8-puzzle

- **Estados**: determina a localização de cada um das peças numéricas e do espaço em branco
- **Estado Inicial**: qualquer configuração do tabuleiro pode ser designada como possível estado inicial
- **Possíveis Ações**: definem os movimentos do espaço em branco possíveis, isto é, `esquerda`, `direita`, `cima` e `baixo`, desde que não extrapolem os limites do tabuleiro
- **Função Sucessor**: retorna um estado decorrente de uma ação possível
- **Teste Objetivo**: verifica se o estado é equivalente ao estado final desejado, isto é, as pedras estão ordenadas crescentemente de acordo com seu valor
- **Custo do caminho**: cada movimento do espaço em branco possui custo 1, e o custo do caminho é dado pela quantidade de movimento (comprimento do caminho) até encontrar a solução desejada

8 Rainhas



8 Rainhas

- **Estados** qualquer arranjo de n rainhas, sendo uma por coluna
- **Estado inicial**: rainhas distribuídas aleatoriamente (uma em cada coluna)
- **Possíveis ações**: movimentar uma rainha de forma a diminuir o número de ataques
- **Função sucessor**: retorna uma estado decorrente da movimentação de uma rainha de forma que este estado apresente menos rainhas atacando-se
- **Teste objetivo**: 8 rainhas estão no tabuleiro de forma que nenhuma ataque a outra
- **Custo do caminho**: cada movimento de rainha possui custo 1

8 Rainhas - Formulação Alternativa

- **Estados** qualquer arranjo de i rainhas ($0 \leq i \leq n$), uma por coluna
- **Estado Inicial**: sem rainhas no tabuleiro
- **Possíveis ações**: adicionar uma nova rainha em uma coluna vazia mais a esquerda de forma que a rainha não ataque nenhuma outra rainha
- **Função sucessor**: retorna o tabuleiro com uma rainha adicionada na posição especificada
- **Teste objetivo**: 8 rainhas estão no tabuleiro de forma que nenhuma ataque a outra
- **Custo do caminho**: cada movimento de rainha possui custo 1

8 Rainhas - Formulação Alternativa

- **OBSERVAÇÕES**:
  - No problema das 8 rainhas há 1.8×10^{14} possíveis estados a investigar
  - Uma boa formulação do problema proíbe colocar uma rainha em uma posição que já está atacada $\rightarrow$ diminui o espaço de busca

Encontrar Rotas

- **Encontrar rotas** (pacotes entre roteadores de redes, planejamento de operações militares, planos de voo, caminho entre duas cidades - GPS)

- Muito utilizado na área de **logística**

Encontrar Rotas

- **Planejamento de viagem**
  - **Estados**: cada estado incluir o local (aeroporto) e o tempo atual
  - **Estado inicial**: cidade de origem (especificada pelo consulta do usuário)
  - **Possíveis ações**: qualquer cidade vizinha deslocar-se para uma cidade vizinha
  - **Função sucessor**: escolher uma das cidades vizinhas para deslocar-se
  - **Teste objetivo**: verificar se o local atual é o mesmo especificado pelo usuário como cidade destino
  - **Custo do caminho**: custo monetário, tempo de espera, tempo de voo, custo dos procedimentos de imigração, qualidade do assento, tipo de aeronave, ...
  - **Solução**: sequência de cidades a serem visitadas desde o estado inicial até o estado final (geralmente a sequência é aquela que provê o caminho de menor custo)

Outros

- **Problema do caixeiro viajante**
  - Um turista quer conhecer todas as cidades de uma região passando por elas uma única vez
  - Além disso deve-se encontrar o caminho mais curto
  - Mesmo problema pode ser aplicado em diversas outras situações

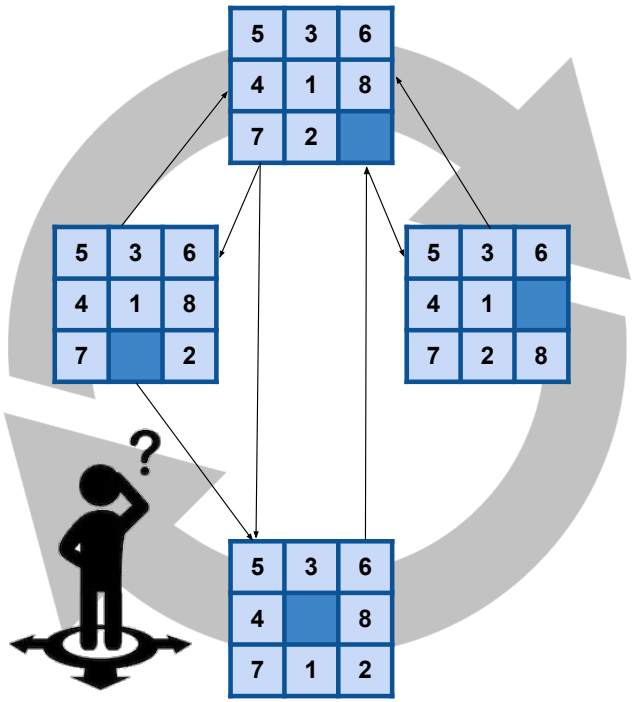
- **Very-large-scale integration (VLSI)**: posicionar milhões de componentes e conexões em um *chip* para minimizar a área, tempo de transferência de dados e maximizar a manufatura

Outros

- **Sequenciamento de montagem automática**
  - Encontrar automaticamente sequências de inclusão de peças de forma a montar um objeto corretamente
  - Sequências corretas impossibilitam a montagem correta sem refazer o trabalho
  - Qualquer algoritmo prático deve evitar explorar todas as possíveis sequências mas apena um pequena porcentagem do espaço de estados
  - Também pode ser aplicado ao design de proteínas, na qual o objetivo é encontrar sequências de aminoácidos que formam uma estrutura de proteína tridimensional com propriedades para curar algumas doenças

Buscando por Soluções

- Estados as vezes podem ser alcançados por diferentes caminhos
- Vários caminhos a um estado podem levar a laços ou ciclos
- Podem impedem que o algoritmo alcance um objetivo



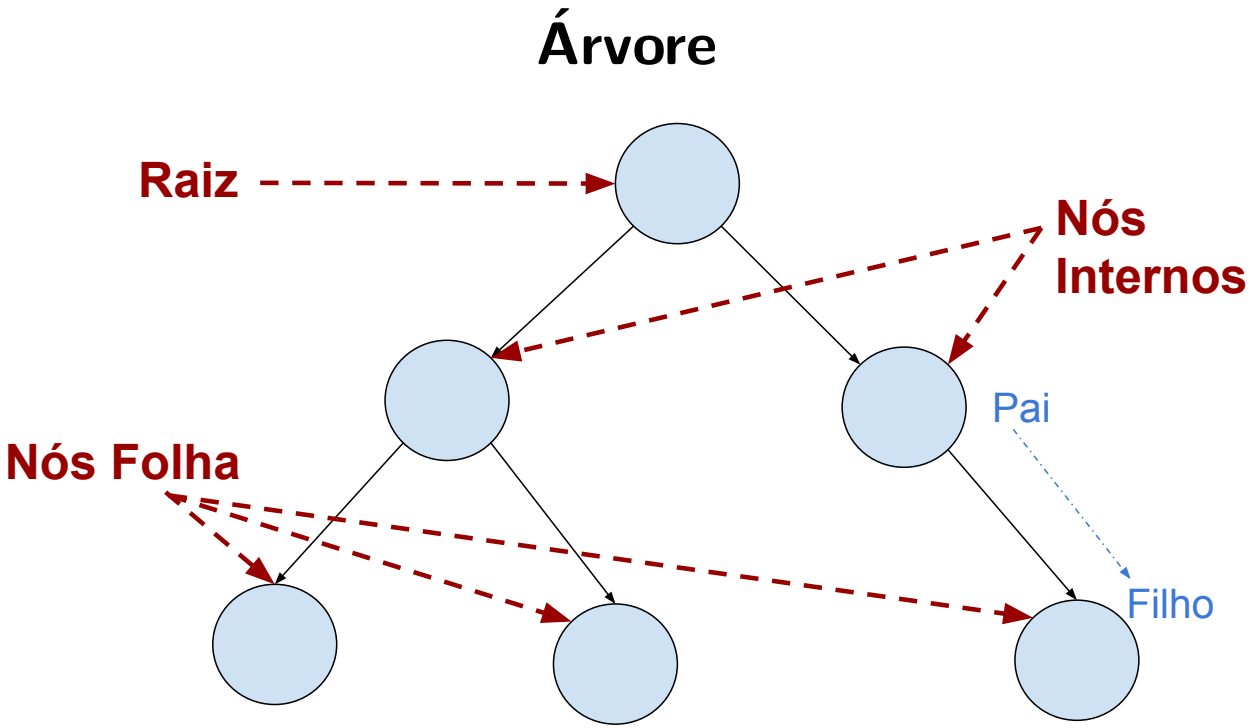
Buscando por Soluções

- Felizmente não há necessidade de considerar ciclos
- Uma vez que custos de caminho são aditivos (e não negativo), passar por um ciclo nunca levará a um caminho melhor em comparaão a um caminho sem considerar um ciclo
- Portanto, ao considerar grafos de estados, é necessário evitar os ciclos
- **Ou** ...

Buscando por Soluções

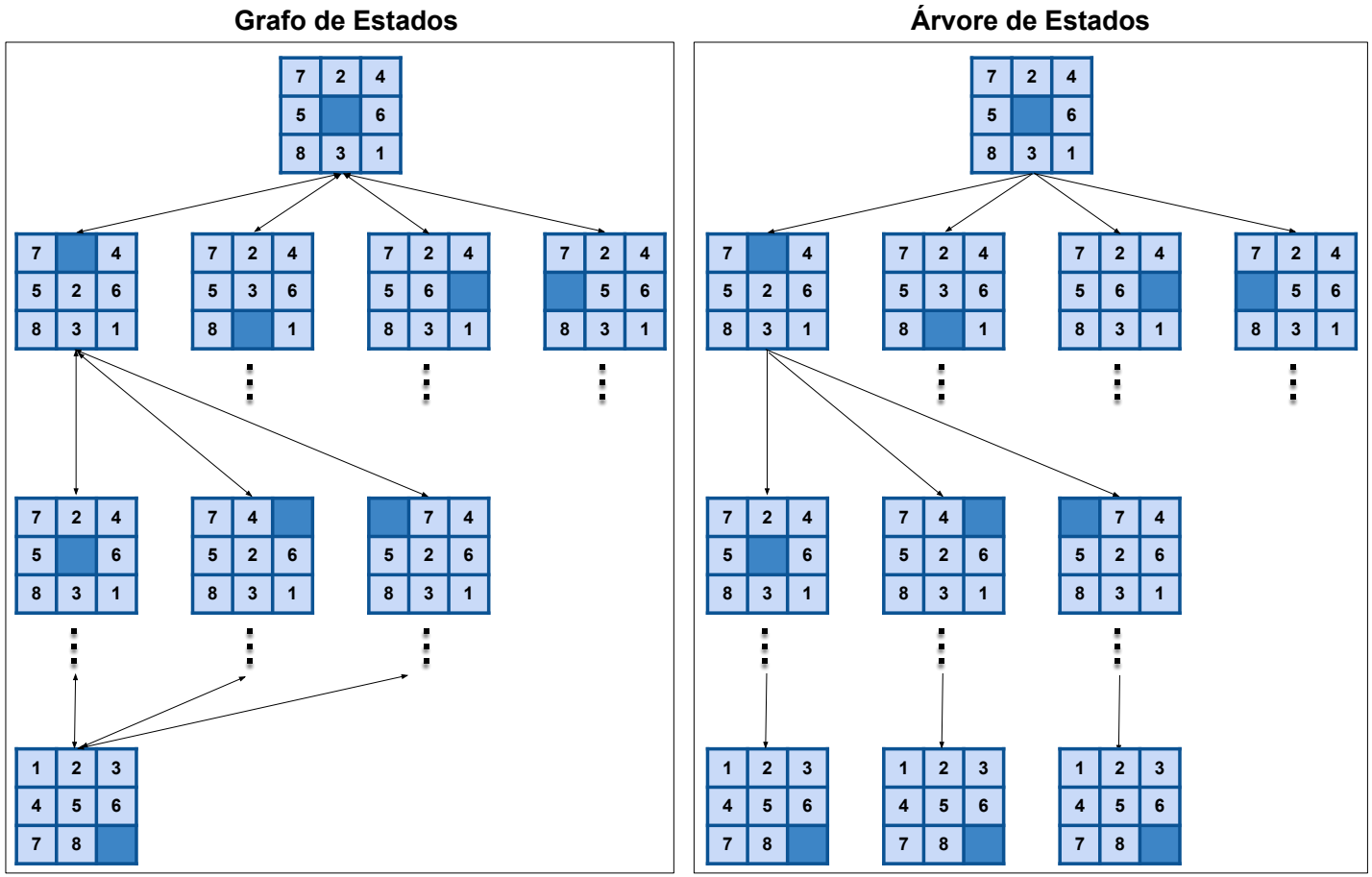
- Considerar a modelagem de um problema como uma árvore de estados
- Uma ávore é um caso especial de grafo direcionado onde não há ciclos
- Estrutura da árvore:
  - **Raiz**: elemento que não recebe nenhum tipo de ligação (estado inicial)
  - **Nós internos**: elementos que recem ligações e que estão conectados a outros elementos (soluções intermediárias)
  - **Nós folha**: nó que não se liga a nehum outro nó (solução do problema ou alguma solução parcial a qual não é capaz de gerar outros estados)
  - **Ligações**: transições entre os estados

Buscando por Soluções



Na árvore, há no máximo um caminho entre dois nós

Buscando por Soluções

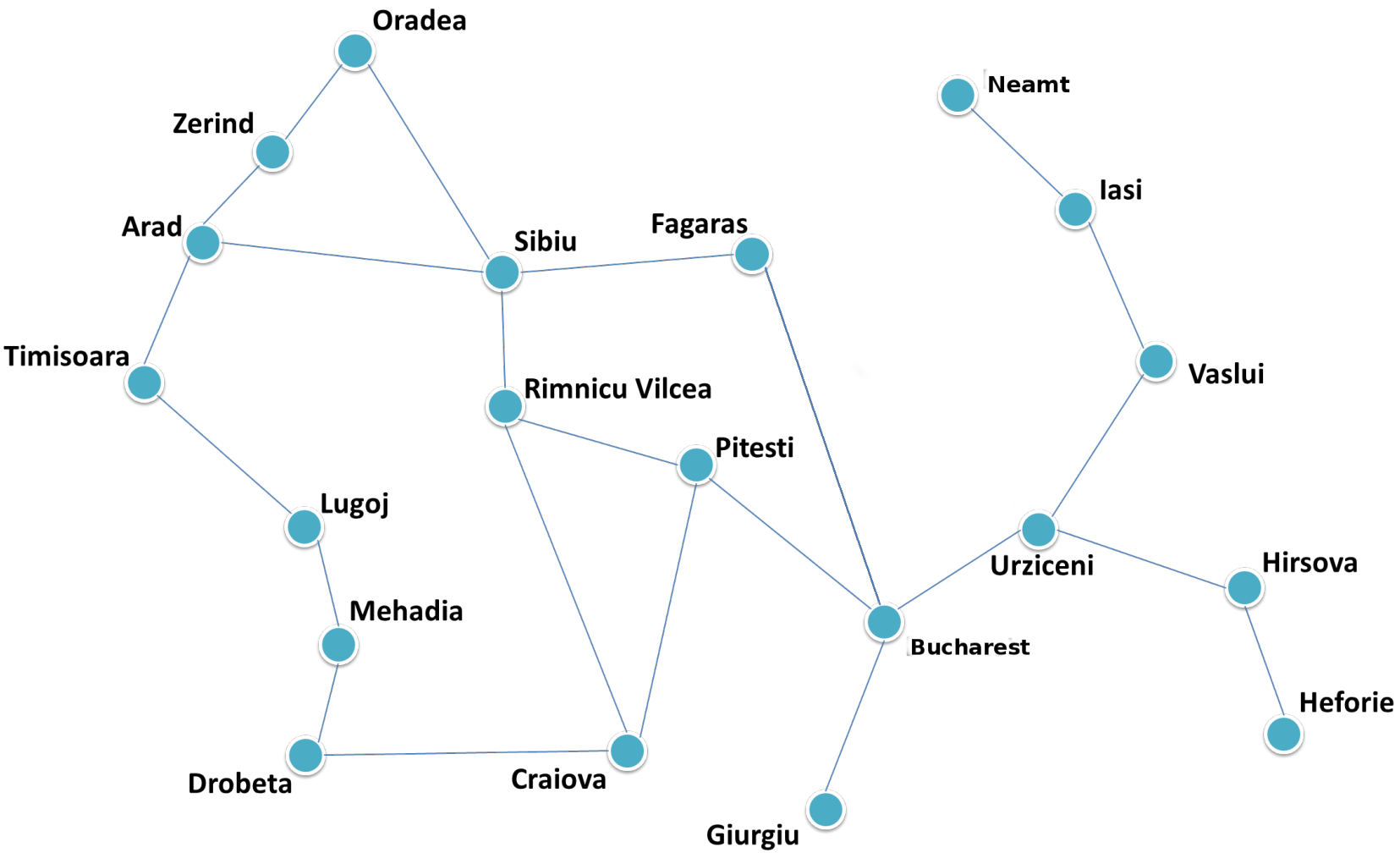


Buscando por Soluções

- A **construção da árvore se dá expandindo os possíveis estados** a partir do nó inicial, depois os estados a partir dos filhos do nó inicial e assim por diante, **até atingirem um nó em que não é mais possível expandir, ou até encontrar o estado objetivo**

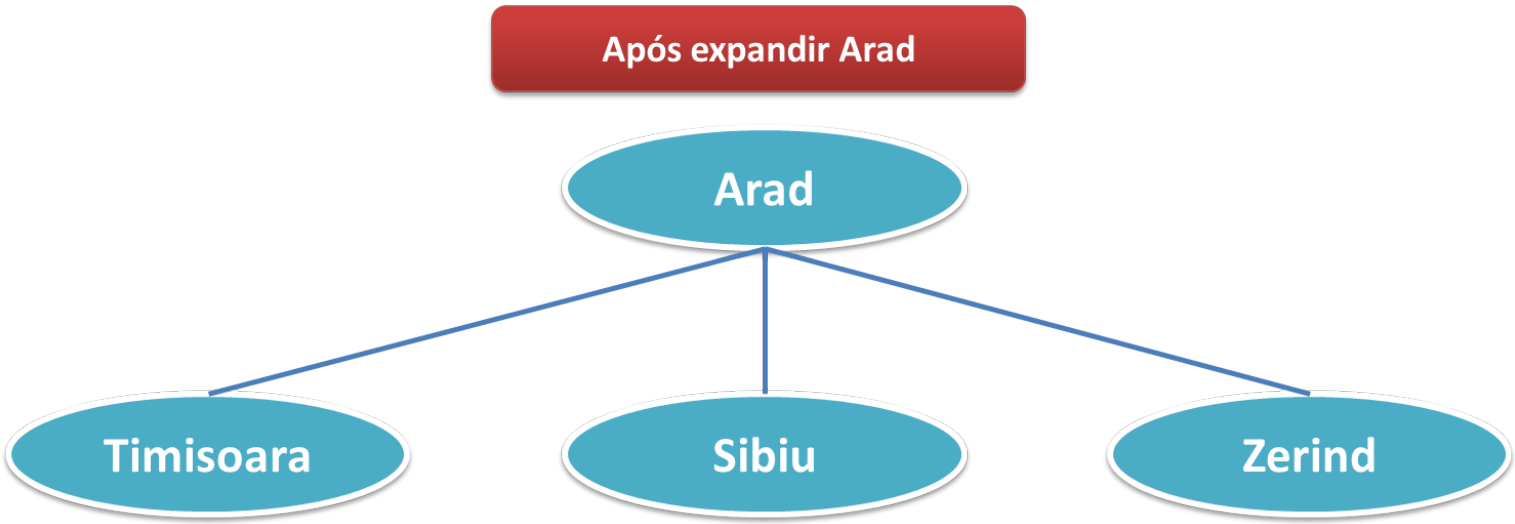
- A ordem com que os nós da árvore são considerados para a expansão é denominada **ESTRATÉGIA DE BUSCA**

Ex: Mapa da Romênia

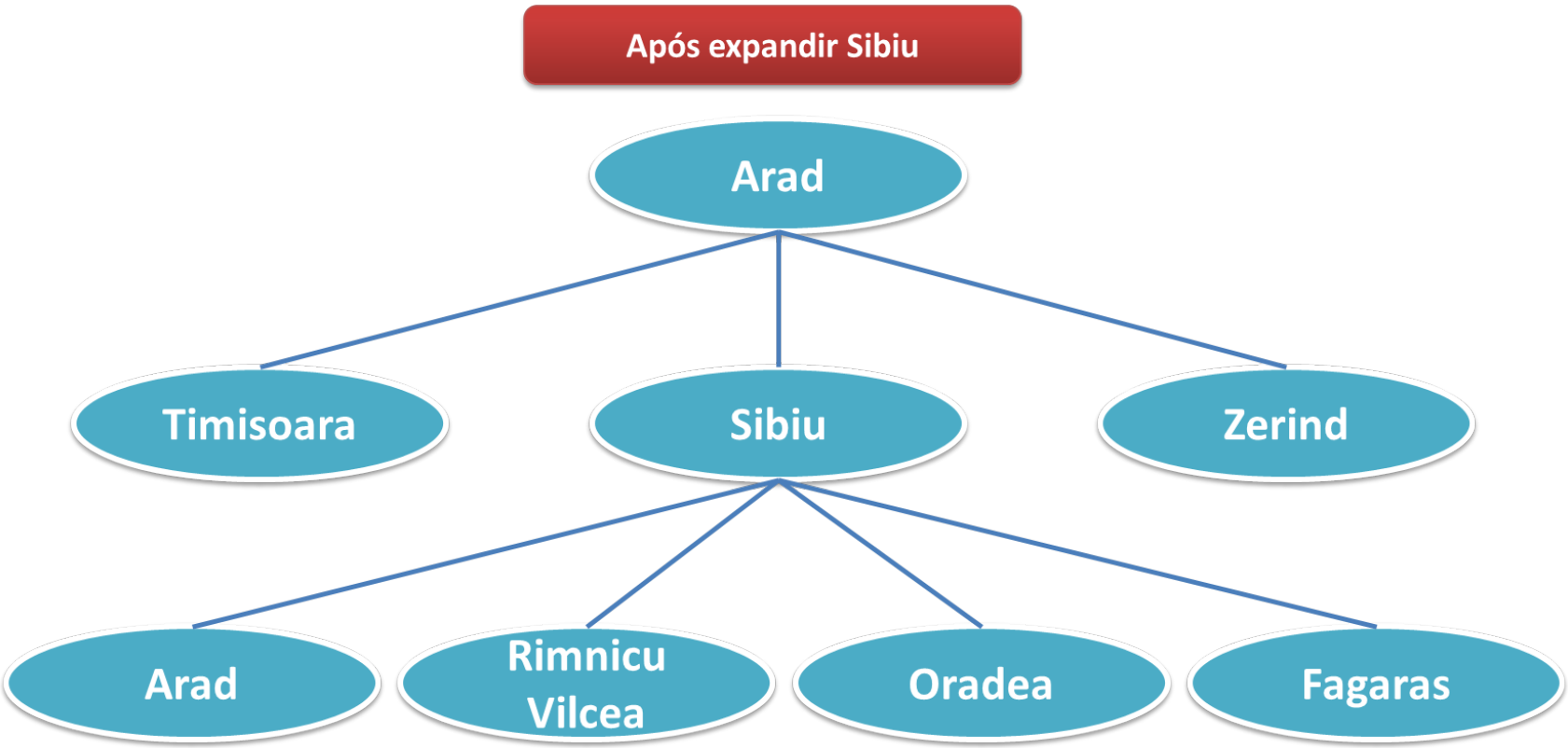


Ex: Mapa da Romênia

Ex: Mapa da Romênia



Ex: Mapa da Romênia



Descrição Informal do Algoritmo de Busca

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

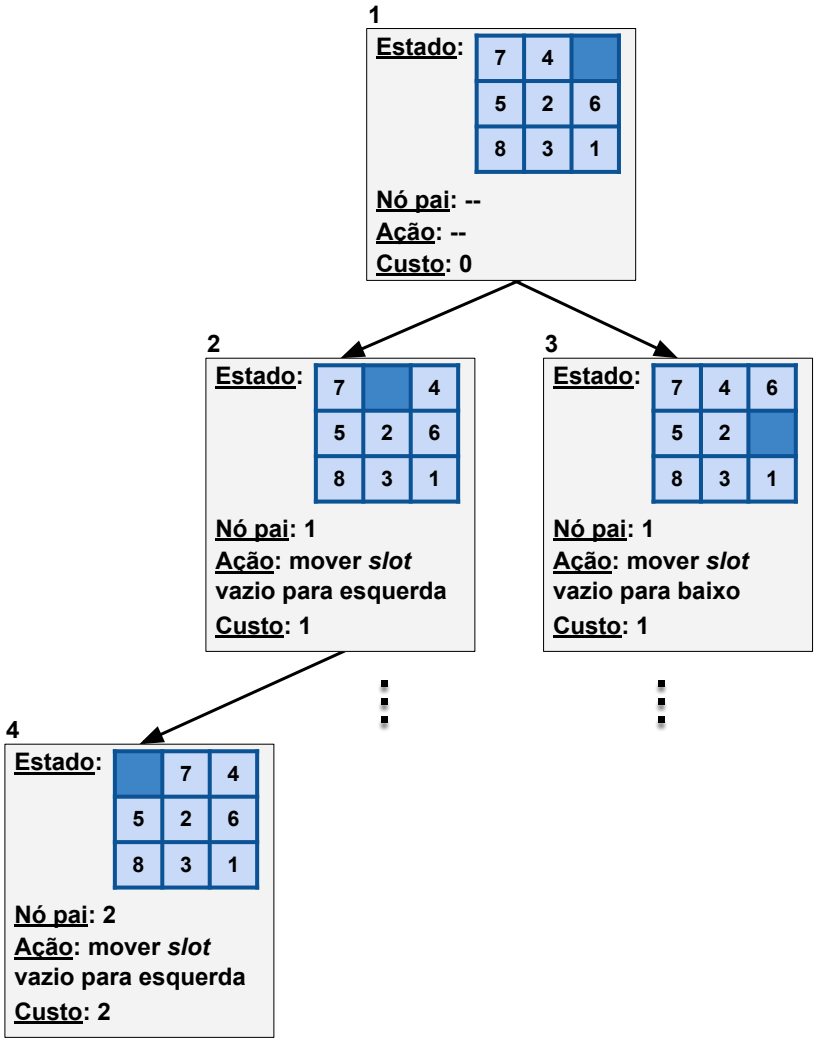
Buscando por Soluções

- **OBSERVAÇÃO 1:** o conjunto de nós disponíveis para expansão em qualquer ponto da busca é denominado de **FRONTEIRA**
- **OBSERVAÇÃO 2:** em muitos problema a expansão de estados de um determinado caminho da árvore também pode ser infinita
  - Devido à caraterística do problema
  - Devido à considerar estados repetidos
- **OBSERVAÇÃO 3:** deve-se utilizar um histórico de estados visitados para evidar as repetições denominados de conjunto de nós **EXPLORADOS**

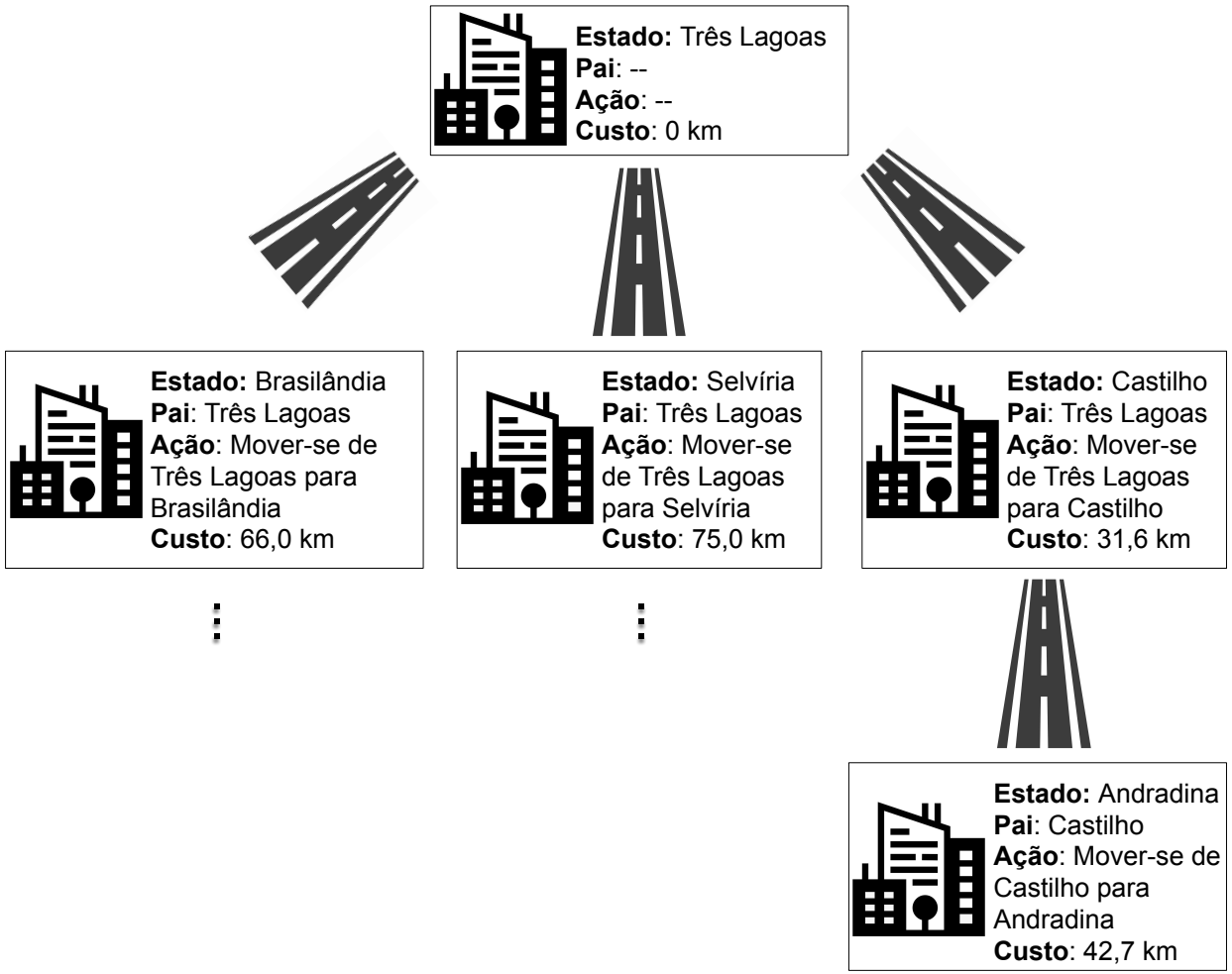
Infraestrutura dos Algoritmos de Busca

- Para cada nó n da árvore de busca, têm-se a seguinte estrutura:
  - $STATE(n)$: estado correspondente ao nó n no problema de busca
  - $PARENT(n)$: nó da árvore cuja expansão gerou o nó n
  - $ACTION(n)$: ação aplicada ao pai para gerar o no n
  - $COST(n)$ ou $g(n)$: custo do caminho desde o estado inicial até o nó n

Infraestrutura dos Algoritmos de Busca



Infraestrutura dos Algoritmos de Busca



Estruturas para Gerenciar os Nós da Fronteira

- Para se evitar a inserção de estados repetidos (ciclos) e a possível geração de caminhos infinitos na árvore de busca, normalmente utiliza-se um **conjunto de nós já explorados** (geralmente armazenados em um **conjunto hash**)

- A estrutura mais utilizada para se gerenciar os nós da **fronteira** de um problema de busca é a **LISTA**

Algoritmo de busca geral

```
function GENERAL-SEARCH( problem, QUEUING-FN) returns a solution, or failure

  nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
  loop do
    if nodes is empty then return failure
    node ← REMOVE-FRONT(nodes)
    if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
    nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))
  end
```

Estruturas para Gerenciar os Nós da Fronteira

- Três variantes comuns de listas para gerenciar os nós de fronteira são
  - **First-in, First-out (FIFO)**: elemento mais antigo na lista é o elemento a ser retornado e retirado
  - **Last-in, First-out (LIFO)**: elemento mais novo na lista é o elemento a ser retornado e retirado
  - **Lista de prioridade**: retorna o elemento da lista com maior prioridade de acordo com algum critério pré-estabelecido

- Cada uma dessas **variantes**, além das diferentes possibilidades de prioridades, irá definir um **algoritmo de busca diferente**

Medidas de Desempenho para Resolução de Problemas

- Antes de estudar algoritmos de busca, é necessário considerar alguns **critérios** que podem ser utilizados para escolhê-los em situações práticas
- Pode-se avaliar o **desempenho** de algoritmos de busca de **4 maneiras**:
  - **COMPLETUDE**: o algoritmo garante encontrar uma solução quando existir uma solução
  - **OTIMALIDADE**: encontra a solução ótima (aquela com menor custo)
  - **COMPLEXIDADE DE TEMPO**: quanto tempo o algoritmo demora pra encontrar a solução (número nós visitados)
  - **COMPLEXIDADE DE ESPAÇO**: quanto de memória é necessário para executar a busca (número máximo de nós inseridos na fronteira)

Medidas de Desempenho para Resolução de Problemas

- **Para se estimar o tempo e espaço serão considerados**:
  - **Branching factor (ou fator de ramificação)**: número máximo de sucessores de qualquer nó (será denotado pela letra **b**)
  - **Depth (ou profundidade)**: tamanho do caminho da raiz até o nó mais profundo (será denotado pela letra **d**)
  - **Tamanho máximo de caminho**: tamanho máximo de qualquer caminho (será denotado pela letra **m**)

Estratégias de Busca Não Informada

- Nesta aula nós veremos algoritmos de **BUSCA NÃO INFORMADA** ou **BUSCA CEGA**
- Busca não informada significa que a estratégia de busca **não possui ou não considera informações adicionais** sobre os estados além do estado atual, ou não considera o custo dos caminhos até o estado atual, ou ainda considera que **todas as ações possuem o mesmo custo**
- Tudo o que esses algoritmos fazem é **expandir os estados sucessores** e **verificar se um estado é ou não um estado objetivo**
- Os algoritmos distinguem-se pela **ordem com que os estados sucessores são inseridos na fronteira**

Busca em Largura

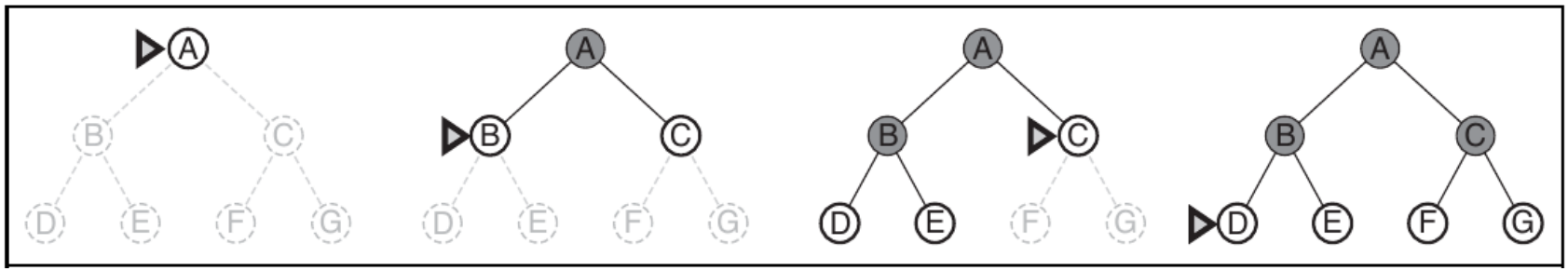
- **A busca em largura utiliza a seguinte estratégia:**
  - Primeiro a raiz é expandida
  - Todos os sucessores da raiz são expandidos em seguida
  - Os sucessores dos sucessores são expandindo
  - ...

- A característica geral da busca em largura é que todos os nós de uma profundidade i são expandidos antes da profundidade $i + 1$

Busca em Largura

- Portanto, com a busca em largura, os **nós mais rasos são expandidos primeiro**
- Isso é alcançado utilizando a **FIFO** para construir a lista dos nós de fronteira
- **OBSERVAÇÃO:** para a busca em largura funcionar da forma como foi especificada, deve-se descartar nós com estados repetidos

Busca em Largura

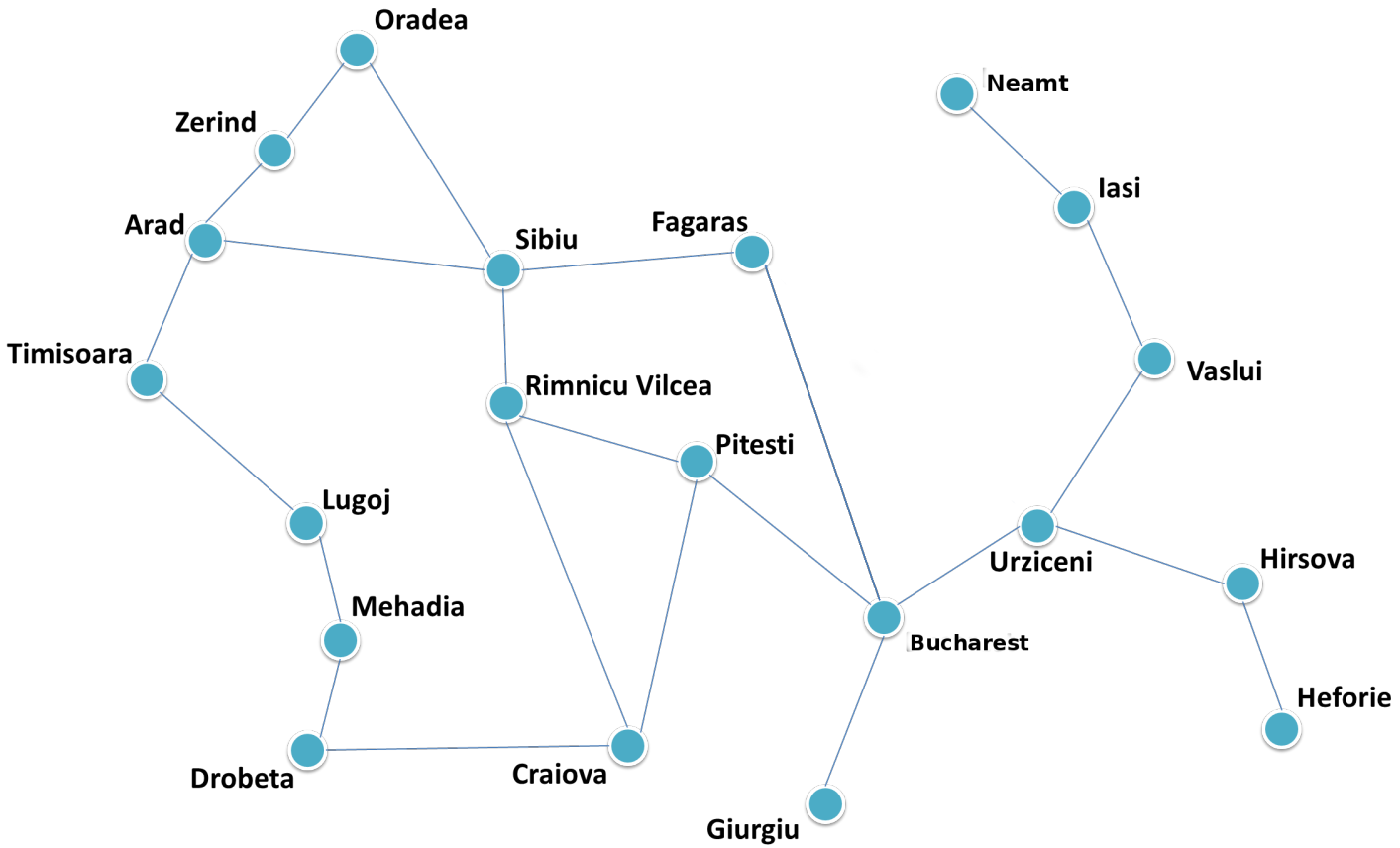


Busca em Largura

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)   /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
              frontier ← INSERT(child, frontier)
```

Busca em Largura

EXERCÍCIO: execute a busca em largura para achar um caminho entre Timisoara e Rimnicu Vilcea



Busca em Largura

- **Características da busca em largura**:
  - **A busca é COMPLETA**: se o nó objetivo menos profundo está presente em alguma profundidade finita d, a busca em largura eventualmente o encontrará
  - **A busca em largura é ÓTIMA**: é ótima se o custo do caminho é uma função não decrescente da profundidade do nó
  - Considerando que cada nó sucessor possui b sucessores e que a solução se encontra em uma profundidade d
    - **Complexidade de tempo:** $O(b^{d-1})$ nós explorados
    - **Complexidade de espaço:** $O(b^d)$ nós na fronteira

Busca em Largura

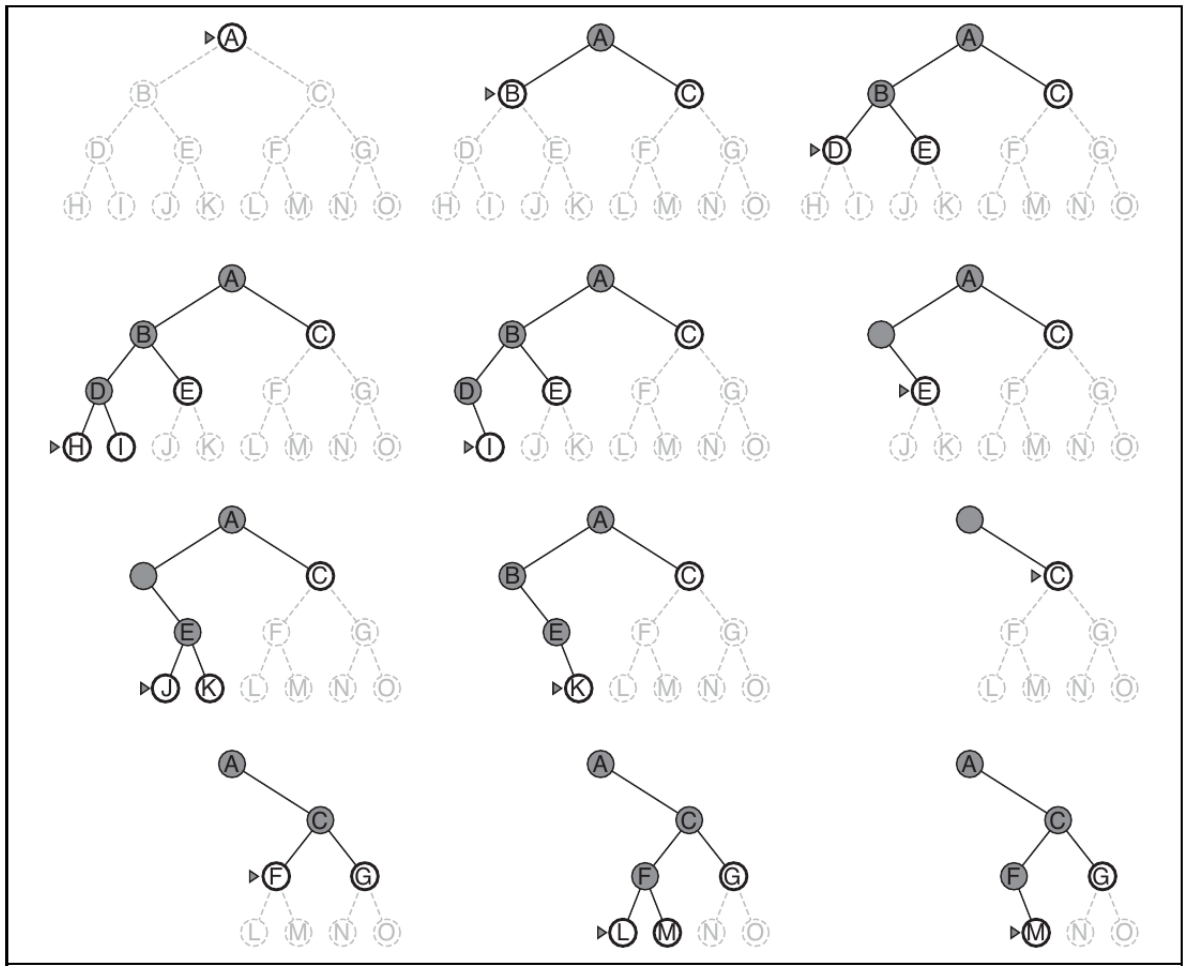
- **OBSERVAÇÃO:** deve-se tomar cuidado com a quantidade de nós inseridos na fronteira pela busca em largura
- Considerando fator de ramificação $b = 10$, um processamento de 1 milhão de nós por segundo e 1000 bytes por nó tem-se:

Produndidade	Nº Nós	Tempo	Memória
2	110	0.11 milisegundos	107 kilobytes
4	11110	11 milisegundos	10,6 megabytes
6	10^6	1,1 secundos	1 gigabyte
8	10^8	2 minutos	103 gibagytes
10	10^{10}	3 horas	10 terabytes
12	10^{12}	13 dias	1 petabyte
14	10^{14}	3,5 anos	99 petabytes
16	10^{16}	350 anos	10 exabytes

Busca em Profundidade

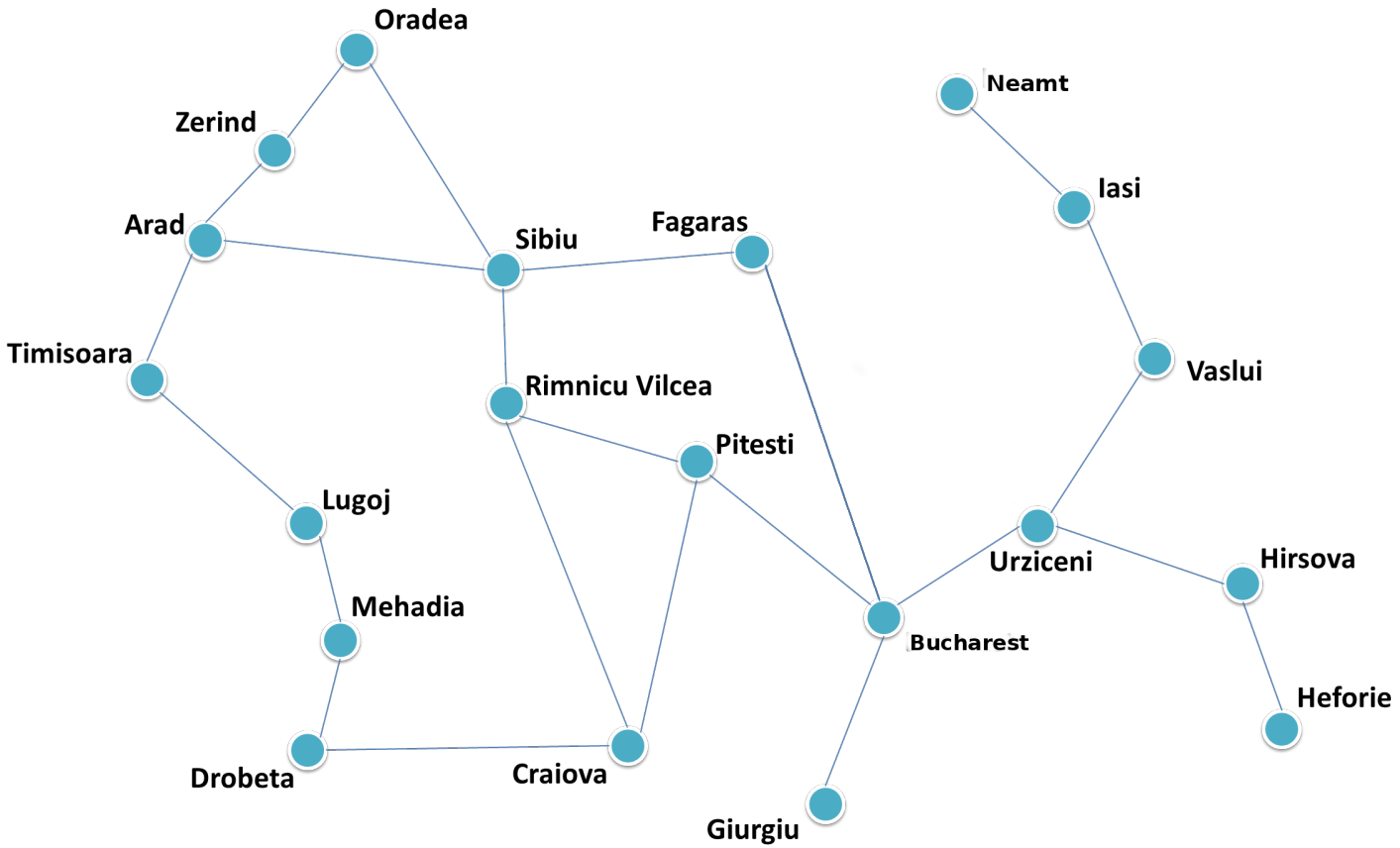
- **A busca em profundidade sempre expande o nó mais profundo na fronteira da árvore de busca**
- Quando a busca encontra um estado que não possibilita mais expansões e este estado não é um estado objetivo, a busca "retorna" e expande o nó mais profundo ainda não expandido
- Ao invés de usar uma fila, como na busca em largura, **a lista de fronteira possui um formato de pilha** (LIFO)

Busca em Profundidade



Busca em Profundidade

EXERCÍCIO: execute a busca em profundidade para achar um caminho entre Timisoara e Rimnicu Vilcea



Busca em Profundidade

- **A busca em profundidade é completa**: uma versão de construção da árvore de busca que evite estados repetitivos e caminhos redundantes produz uma busca completa uma vez de cada nó do espaço de estados será eventualmente expandido

- **A busca em profundidade não é ótima**: pode-se encontrar o estado objetivo em uma profundidade d em um determinado ramo da árvore sendo que existe um nó objetivo em outro ramo com profundidade $d' < d$

Busca em Profundidade

- **Complexidade de tempo**: a busca em profundidade pode pesquisar até $O(b^m)$ nós para encontrar a solução (se m for próximo de d, o tempo para encontrar a solução é igual a busca em largura

- **Complexidade de espaço**: o armazenamento da busca em profundidade requer $O(bm)$ nós (menor que da busca em largura)

Busca em Profundidade

- **OBSERVAÇÃO:** uma variante da busca em profundidade denominada busca em profundidade com *backtracking* utiliza ainda menos memória
  - Apenas um sucessor é gerado por vez ao invés de todos os sucessores
  - Cada sucessor deve "lembrar" qual nó já gerou
  - Complexidade de espaço: $O(m)$

Busca em Profundidade Limitada

- **OBSERVAÇÃO:** dependendo da caraterística do problema, sua modelagem por gerar uma profundidade infinita, gerando assim **FALHA** na busca em profundidade
- Tais problemas da busca em profundidade podem ser solucionados delimitando o limite da busca em *l* níveis
- Esta abordagem é denominada **Busca em Profundidade Limitada**

Busca em Profundidade Limitada

- **Problema**: se escolher uma profundidade l tal que a solução esteja em um nível s e $l < s$, a busca será **incompleta**
- **A busca em profundidade limitada é não ótima** pelos mesmos motivos da busca em profundidade original
- **Complexidade de tempo**: $O(b^l)$
- **Complexidade de espaço**: $O(bl)$

Busca em Profundidade Limitada Iterativa

- A busca em profundidade limitada iterativa gradualmente **aumenta o limite de profundidade da busca até encontrar uma solução**
- Primeiro é realizada uma busca com a profundidade limitada em um nível, depois em dois níveis, três níveis, e assim por diante
- É uma combinação dos benefícios da busca em largura com a busca em profundidade
- **Complexidade de tempo**: $O(b^d)$
- **Complexidade de espaço**: $O(bd)$

Busca em Profundidade Limitada Iterativa

- **OBSERVAÇÃO:** a busca iterativa pode parecer dispendiosa pois os mesmos estados são gerados múltiplas vezes
  - NA VERDADE NÃO É BEM ASSIM
  - A maioria dos nós da árvore se encontram nos níveis mais profundos $\rightarrow$ não impacta muito se os nós dos níveis mais baixos são gerados repetidamente
  - Dado um determinado nível d, os nós neste nível são gerados uma vez, os nós no nível $d-1$ são gerados duas vezes, os nós no nível $d-2$ são gerados 3 vezes e assim por diante até o nível 1, que é gerado 1 vez

Busca em Profundidade Limitada Iterativa

- **OBSERVAÇÃO:** a busca iterativa pode parecer dispendiosa pois os mesmos estados são gerados múltipas vezes
  - Portanto, o número de nós possíveis de serem analisados na busca em profundidade limitada iterativa é dada por

$$N(BPLI) = (d)b + (d-1)b^2 + \cdots + (1)b^d$$

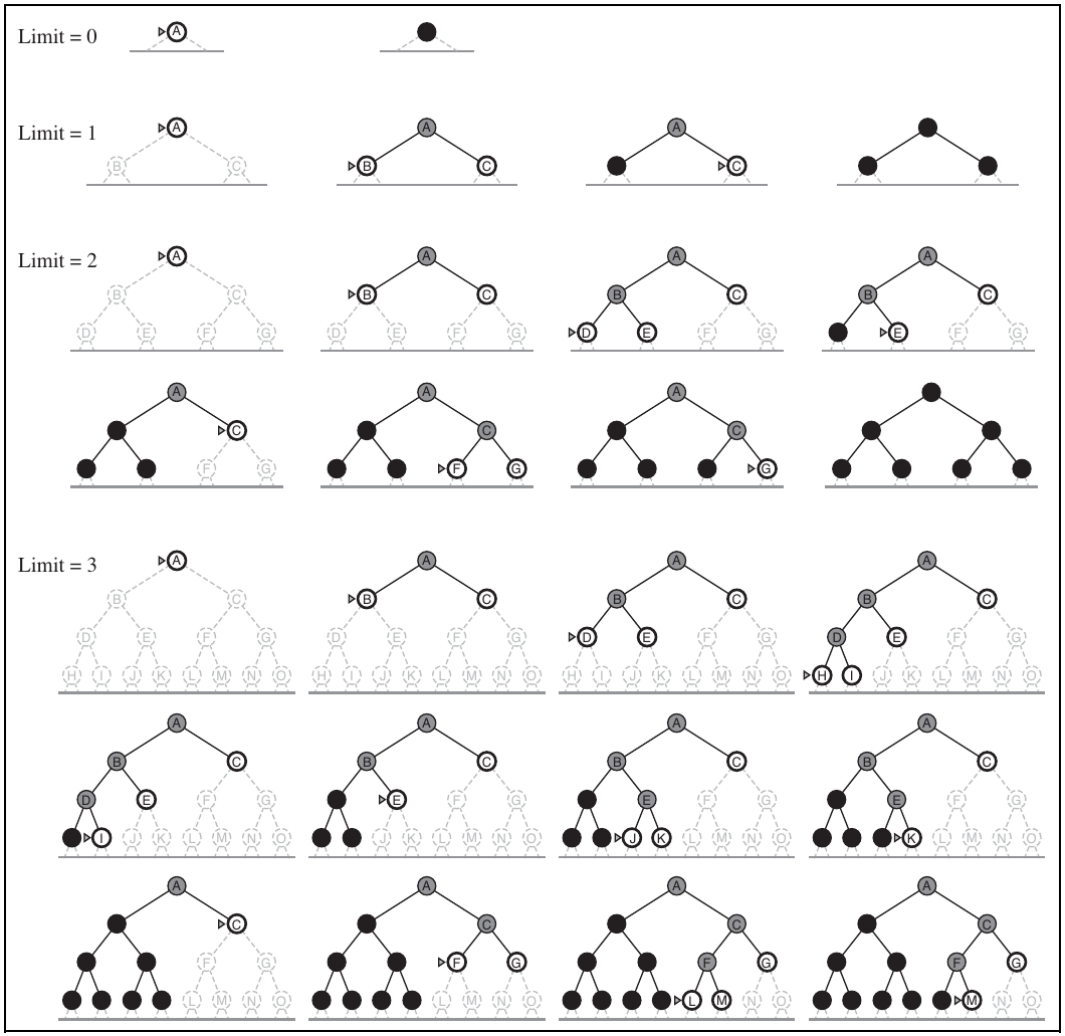
  - Sendo assim, a complexidade assintótica é $O(b^d)$, a mesma da busca em largura

Busca em Profundidade Limitada Iterativa

- **OBSERVAÇÃO:** é análoga a busca em largura uma vez que todos os nós de um determinado nível são analisados antes de se analisar os nós do próximo nível

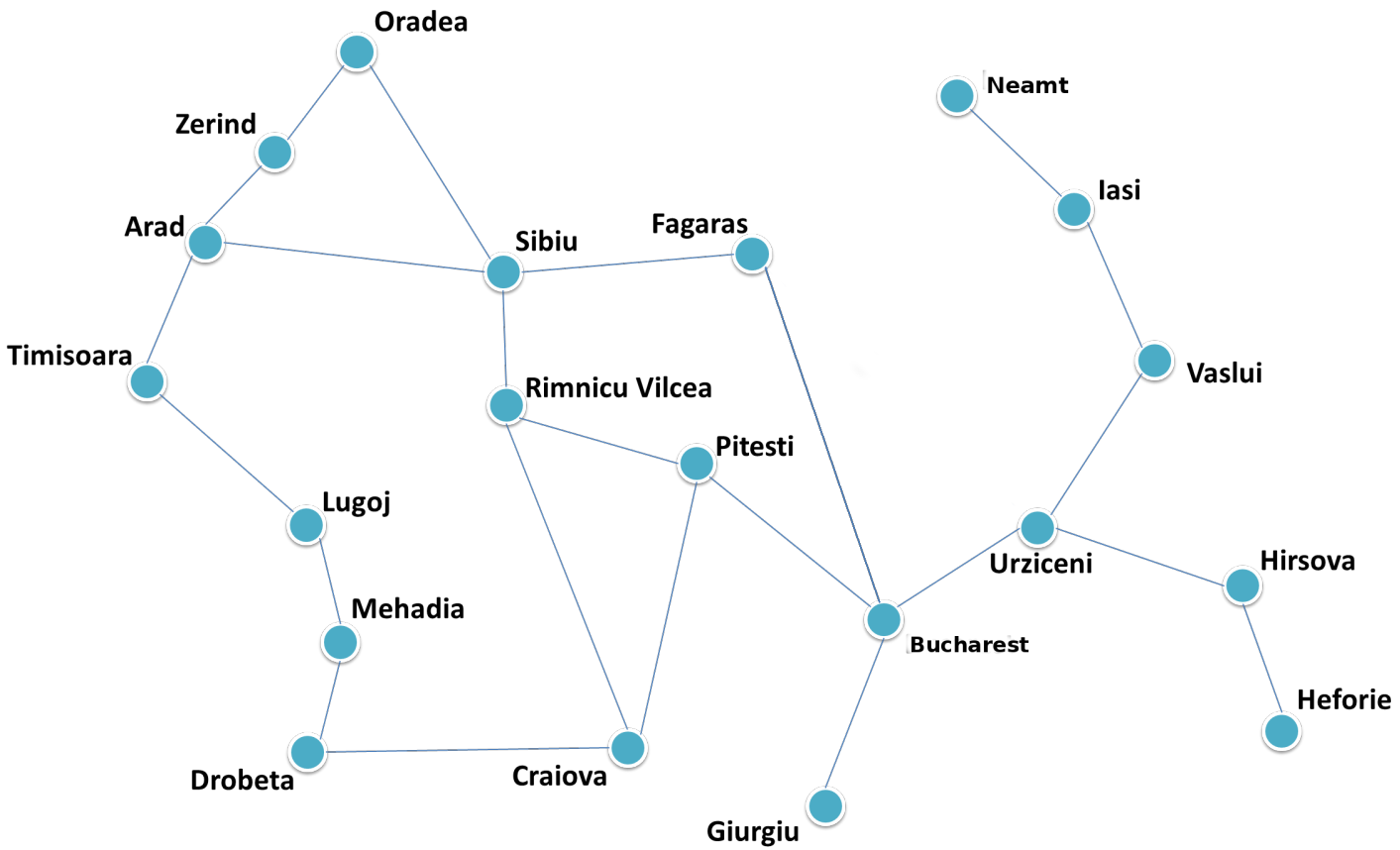
- **CURIOSIDADE:** em geral, a busca em profundidade limitada iterativa é um dos métodos de busca não informada mais utilizados quando o espaço de busca é grande e a profundidade da solução é desconhecida

Busca em Profundidade Limitada Iterativa



Busca em Profundidade Limitada Iterativa

EXERCÍCIO: execute a busca em profundidade limitada iterativa para achar um caminho entre Timisoara e Rimnicu Vilcea



Comparativo Geral dos Algoritmos de Busca Não Informada

Critério	**Largura**	**Profundidade**	**Prof. Limitada**	**Prof. Iterativa**
Completude	Sim	Não	Sim se $l > d$	Sim
Otimalidade	Sim	Não	Não	Sim
Complexidade de Tempo	b^{d-1}	b^m	b^l	b^d
Complexidade de Espaço	b^d	bm	bl	bd

Material Complementar

- 4. Search: Depth-First, Hill Climbing, Beam

  `https://youtu.be/j1H3jAAGlEA?list=PLUl4u3cNGP63gFHB6xb-kVBiQHYe_4hSi`

- Lecture 2: Uninformed Search

  `https://youtu.be/ST--VJJJqoc`

- Artificial Intelligence - Uninformed Search

  `http://cs.gettysburg.edu/~tneller/resources/ai-search/uninformed-java/`

  Sliding Puzzle

  `https://n-puzzle-solver.appspot.com/`

Material Complementar

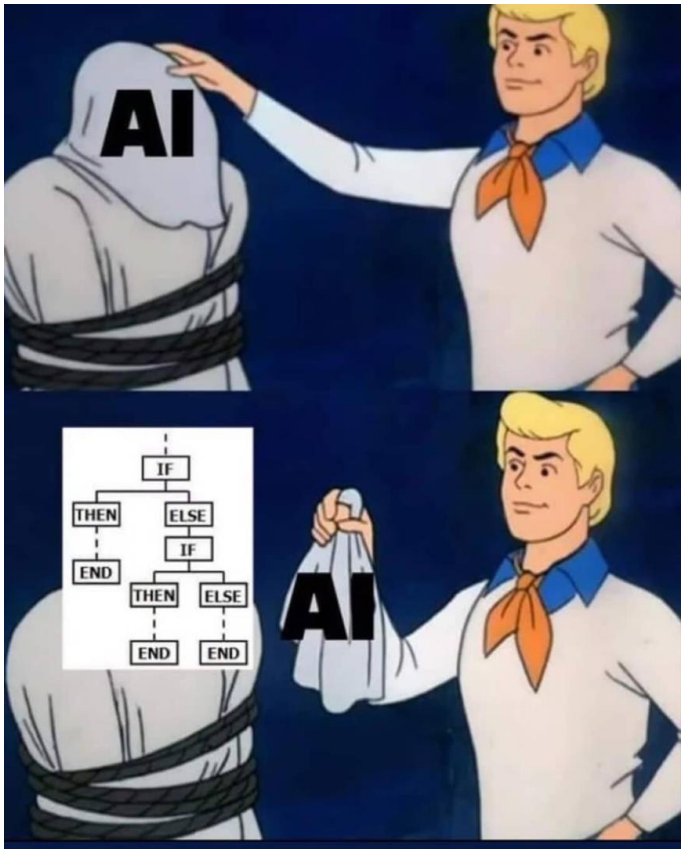
- Busca por Aprofundamento Iterativo, Largura e Profundidade

  `https://www.youtube.com/watch?v=-qj-9bvwh2Y&t=29s`

- Busca em Espaço de Estados em Inteligência Artificial

  `https://www.youtube.com/watch?v=nwvGg94ivac&t=1s`

Imagem do Dia



Inteligência Artificial
http://lives.ufms.br/moodle/

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br

Slides baseados em [Russell and Norvig, 2010] e [Luger, 2013]

Referências Bibliográficas I


Luger, G. F. (2013).
Inteligência Artificial.
6th edition.


Russell, S. and Norvig, P. (2010).
Artificial Intelligence: A Modern Approach, Global Edition.
3rd edition.