Inteligência Artificial
Tópico 03 - Parte 02
Resolução de Problemas por Buscas
(Buscas sem Informação)

Profa. Dra. Priscila Tiemi Maeda Saito
priscilasaito@ufscar.br

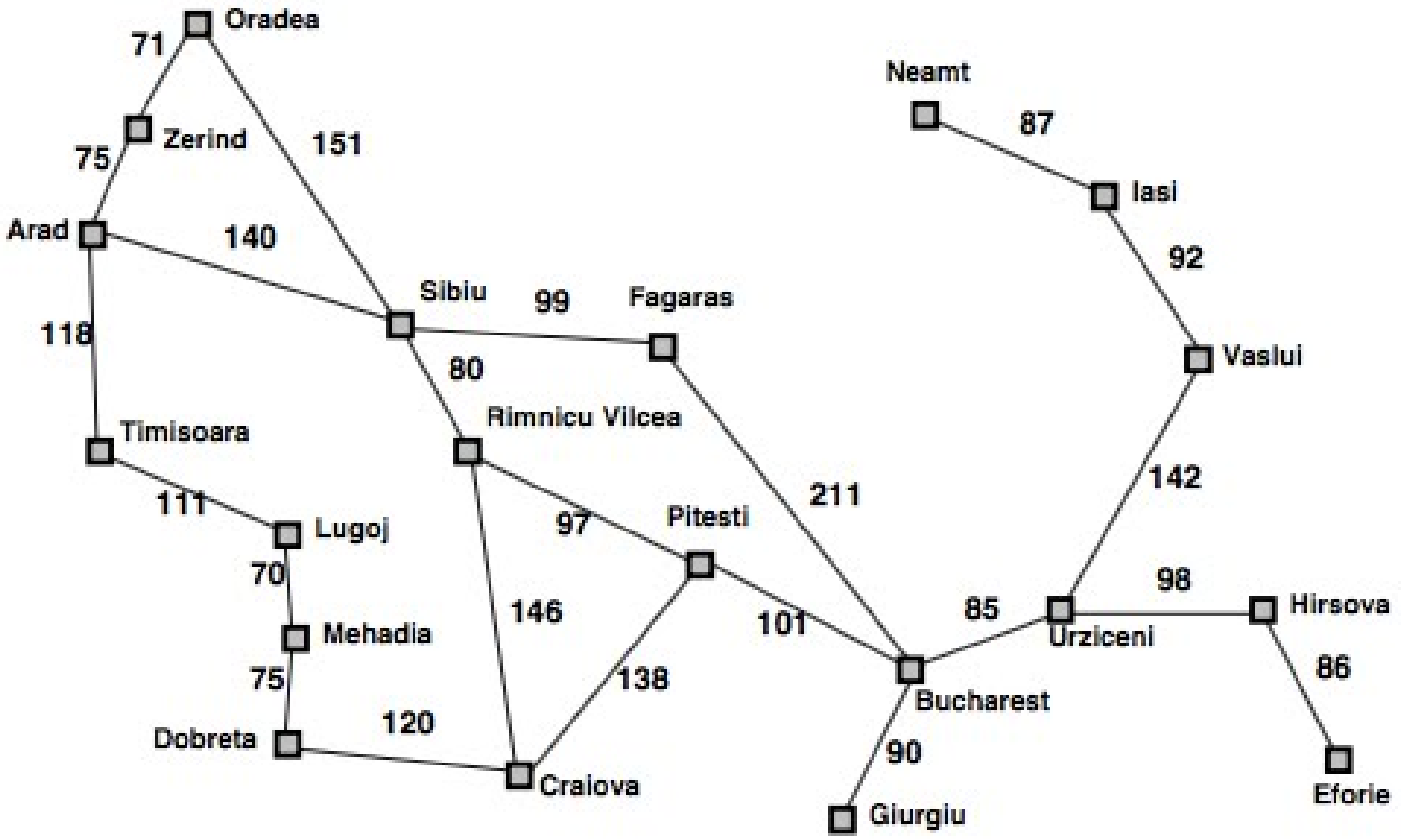
Roteiro

1. Resolução de Problemas por Buscas

2. Estratégias de Busca
   - Busca sem Informação

Exemplo: Problema Romênia

Romênia com custos por passo em km

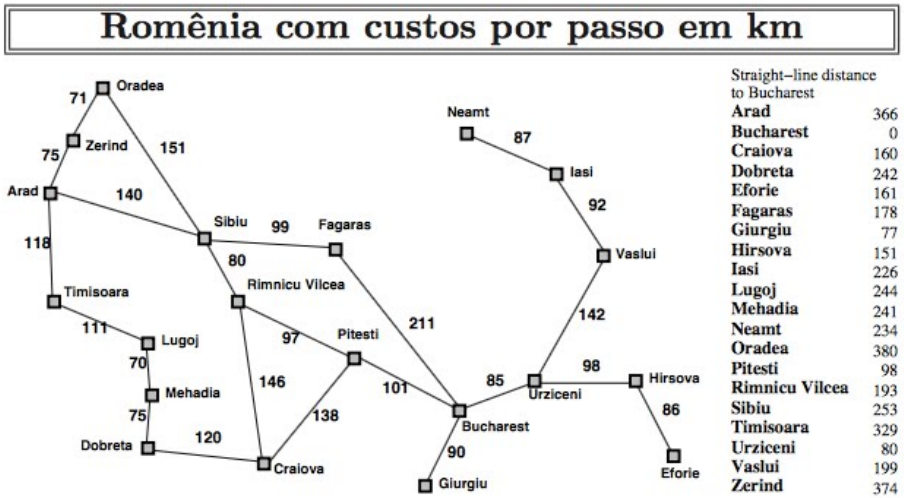


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Exemplo: Problema Romênia

- Contexto:
  - de férias na Romênia, estamos em Arad (e queremos ver mais do país). Temos voo marcado para amanhã, saindo de Bucareste
- Formular objetivo:
  - estar em Bucareste
- Formular problema (definir abstrações relevantes)
  - estados: {Arad, Timisoara, Zerind}
  - ações: Ir para Sibiu, Ir para Fagaras, ...
- Buscar solução:
  - sequência de cidades, e.g., Arad, Sibiu, Fagaras, Bucareste

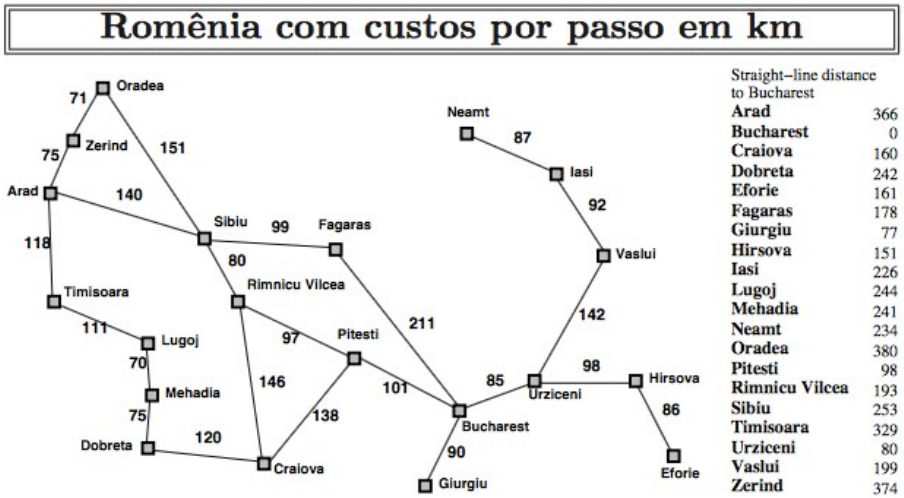
Exemplo: Problema Romênia



Formulação do Problema:

- **estados?**
- **estado inicial?**
- **ações?**
- **modelo de transição?**
- **teste objetivo?**
- **custo do caminho?**

Exemplo: Problema Romênia



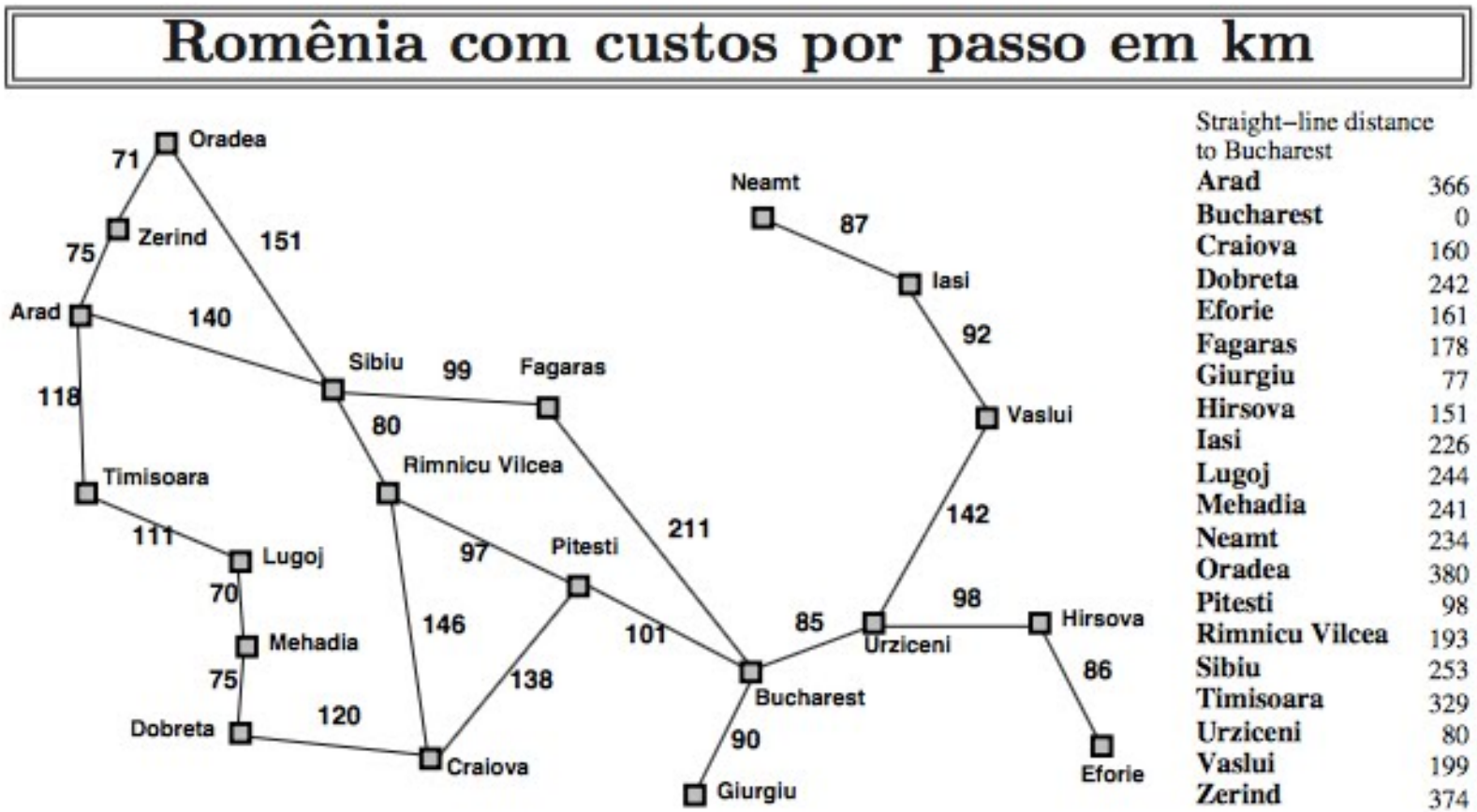
Formulação do Problema:

- **estados?** {todos as cidades alcançáveis a partir do estado inicial por meio de qualquer sequência de ações}
- **estado inicial?** Em(Arad)
- **ações?** Ações(Arad) = {IrPara(Zerind), IrPara(Timisoara), ...}
- **modelo de transição?** Suc(Arad, IrPara(Zerind)) = Zerind)
- **teste objetivo?** Em(Bucharest)?
- **custo do caminho?** soma das ações executadas - custo do caminho c(s, a, s')

Solução? Solução Ótima?

Exercício

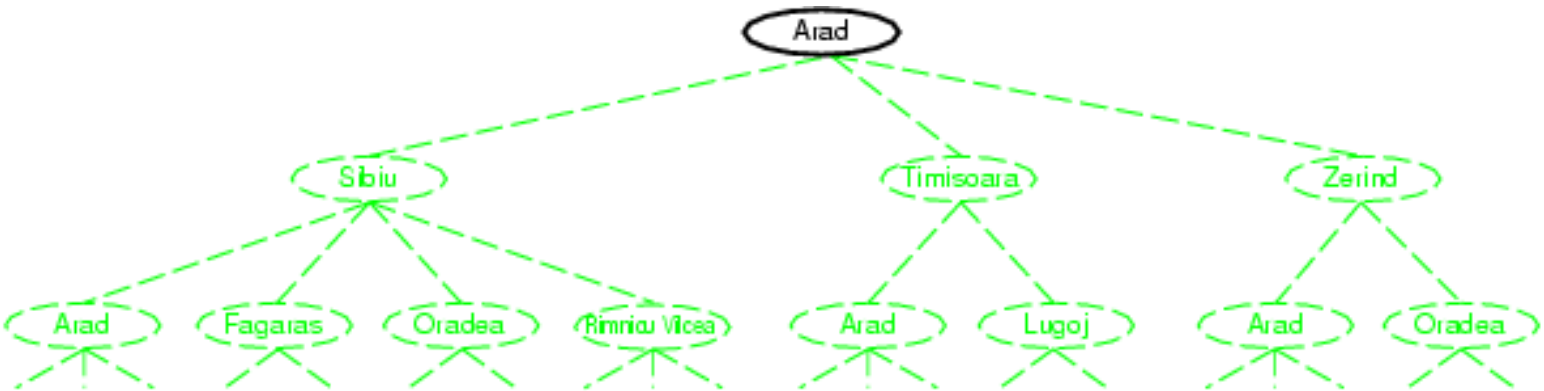
- Considere o problema de encontrar uma rota de Arad a Bucareste. Responda:
  - que rota você pegaria e por quê?
  - descreva o raciocínio que você usou para encontrar essa rota



Busca

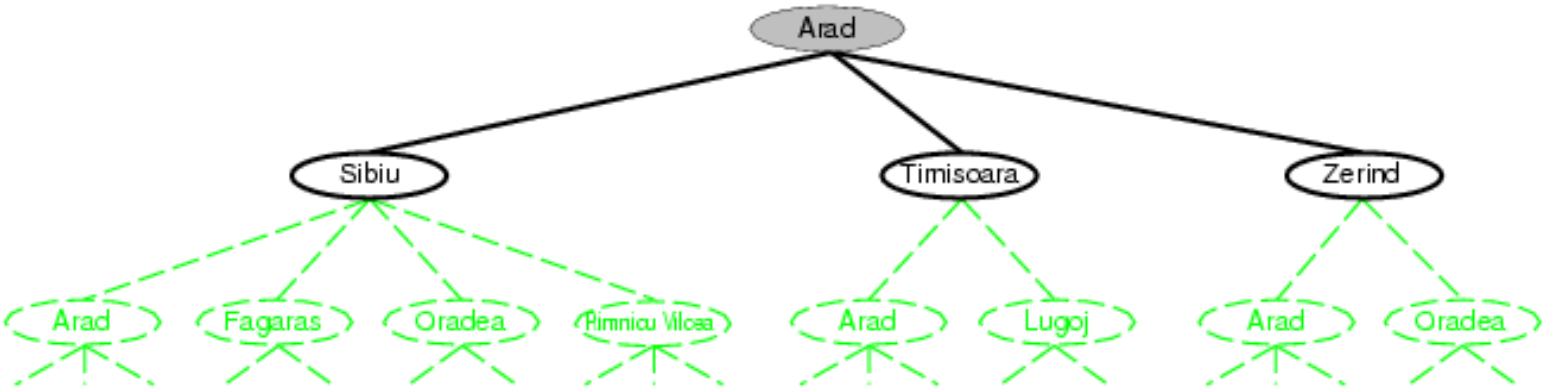
- Passos básicos da busca:
  1. escolhe estado,
  2. testa se é objetivo
  3. se não for, expande (gera sucessores) e retorna ao 1 até
     - ★ encontrar solução ou
     - ★ não existirem mais estados a serem visitados

Busca em Árvore



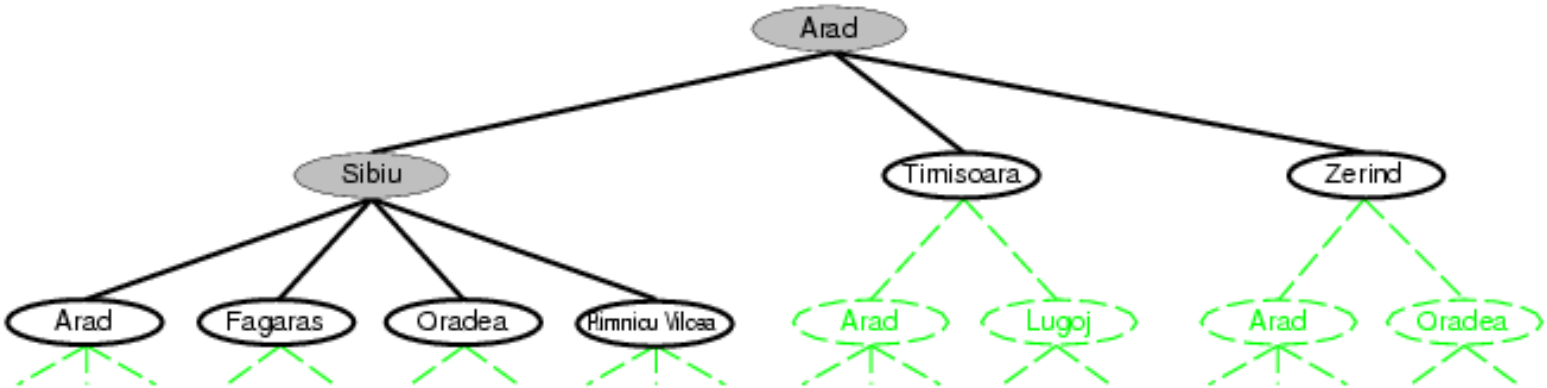
- Escolhe. Teste. Se não for objetivo, expande o nó corrente

Busca em Árvore



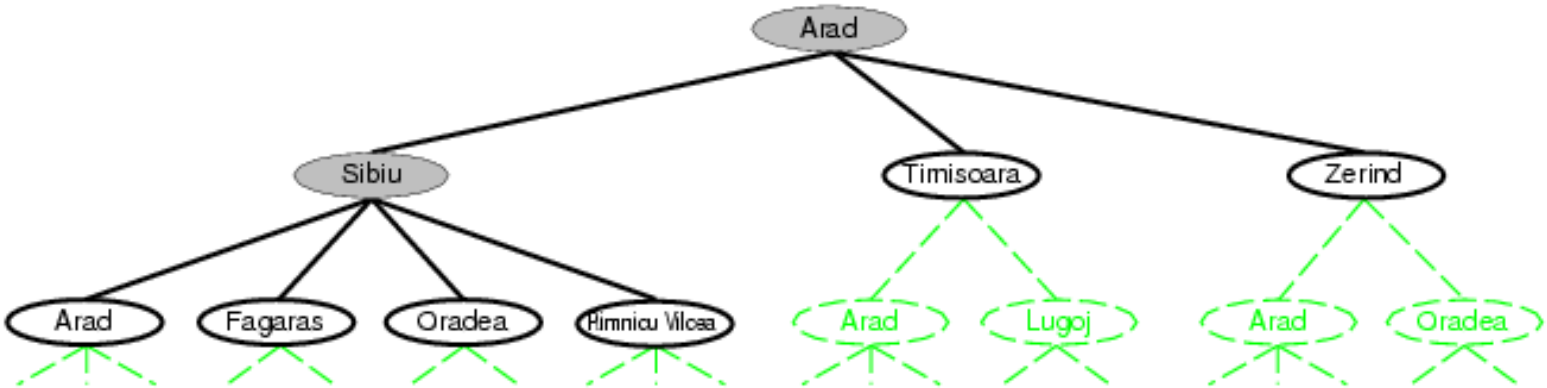
- Expandir o nó corrente resulta na árvore em negrito

Busca em Árvore



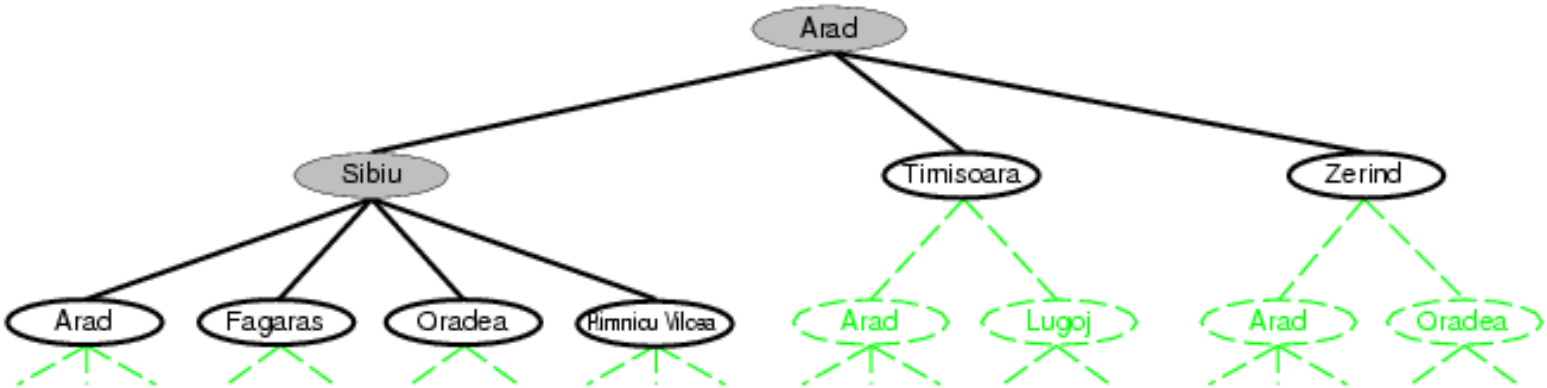
- Expandir o nó corrente resulta na árvore em negrito

Busca em Árvore



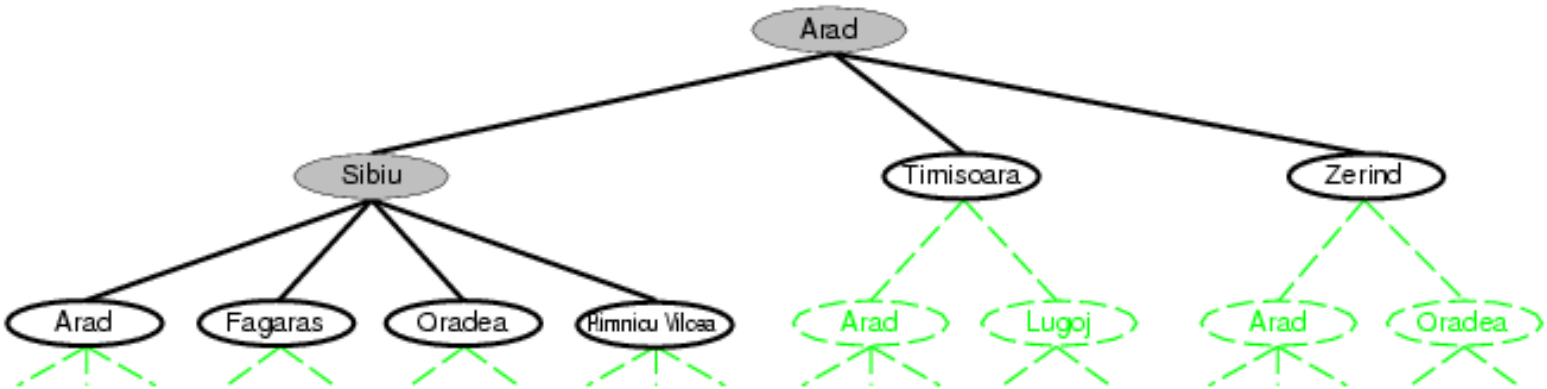
- Fronteira da árvore de busca: ?

Busca em Árvore



- Fronteira da árvore de busca: todos os nós folhas em um dado momento

Busca em Árvore



- Fronteira da árvore de busca: todos os nós folhas em um dado momento
  - Arad, Fagaras, Oradea, Rimnicu Vilcea, Timisoara, Zerind

Algoritmos de Busca em Árvore

- Ideia básica:
  - explorar o espaço de estados de forma simulada (off-line), gerando-se estados sucessores (expandindo) até atingir o estado objetivo

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

inicialize a *fronteira* da árvore usando o estado inicial do *problema*

repita

se a *fronteira* está vazia **então retornar** falha

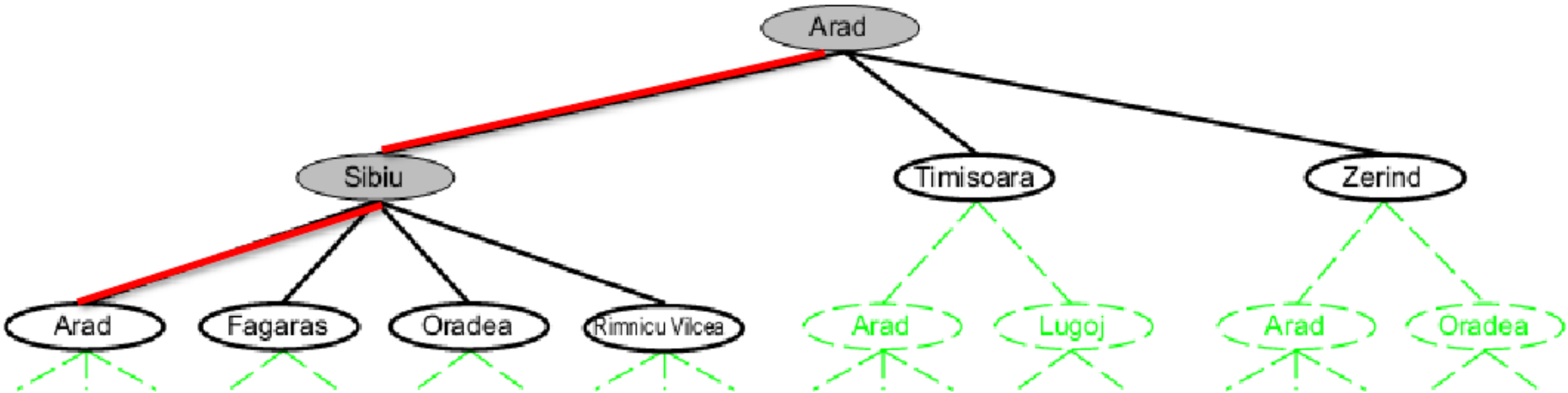
escolha um nó folha para expansão e remova-o da *fronteira*

se o nó contém o estado objetivo **então retornar** a solução correspondente

expandir o nó e adicionar os nós resultantes à *fronteira* da árvore de busca

Problemas da Busca em Árvore

- Ciclos causados por caminhos que são bidirecionais ou por ciclos no próprio mapa (caminhos redundantes)



Algoritmos de Busca em Grafo

- Ideia básica:
  - solucionar o problema de redundância da busca em árvore por meio de uma lista de nós já visitados

função BUSCA-EM-GRAFO(*problema*) **retorna** uma solução, ou falha

inicialize a *fronteira* da árvore usando o estado inicial do *problema*

inicialize *explorados* com vazio

repita

se a *fronteira* está vazia **então retornar** falha

escolha um nó folha para expansão e remova-o da *fronteira*

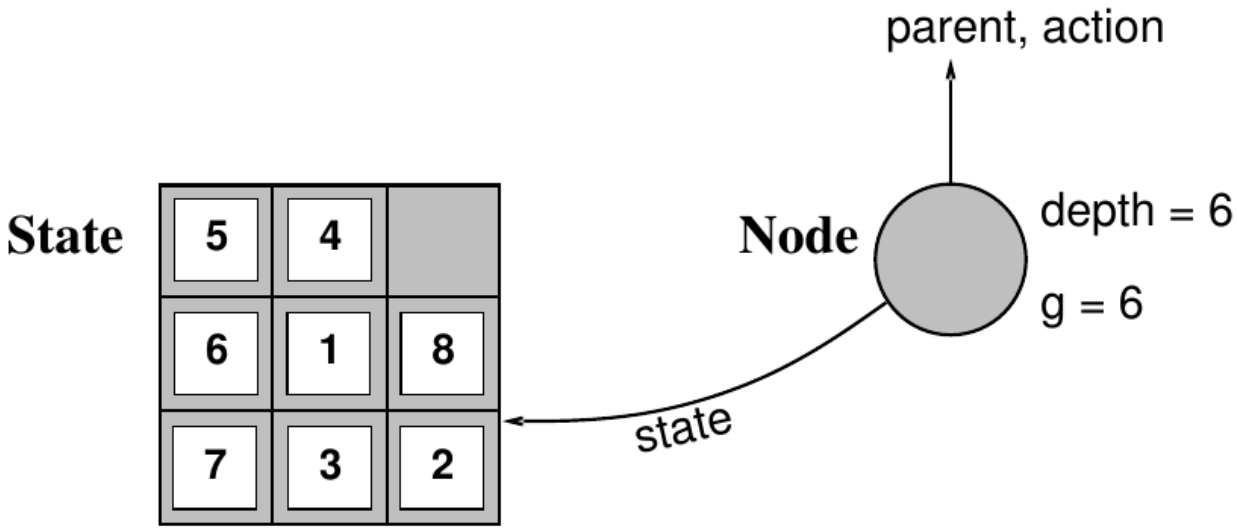
se o nó contém o estado objetivo **então retornar** a solução correspondente

adicione o nó ao conjunto dos *explorados*

expandir o nó e adicionar os nós resultantes à *fronteira* da árvore de busca somente se não estiverem nos já *explorados*

Implementação: Estados x Nós

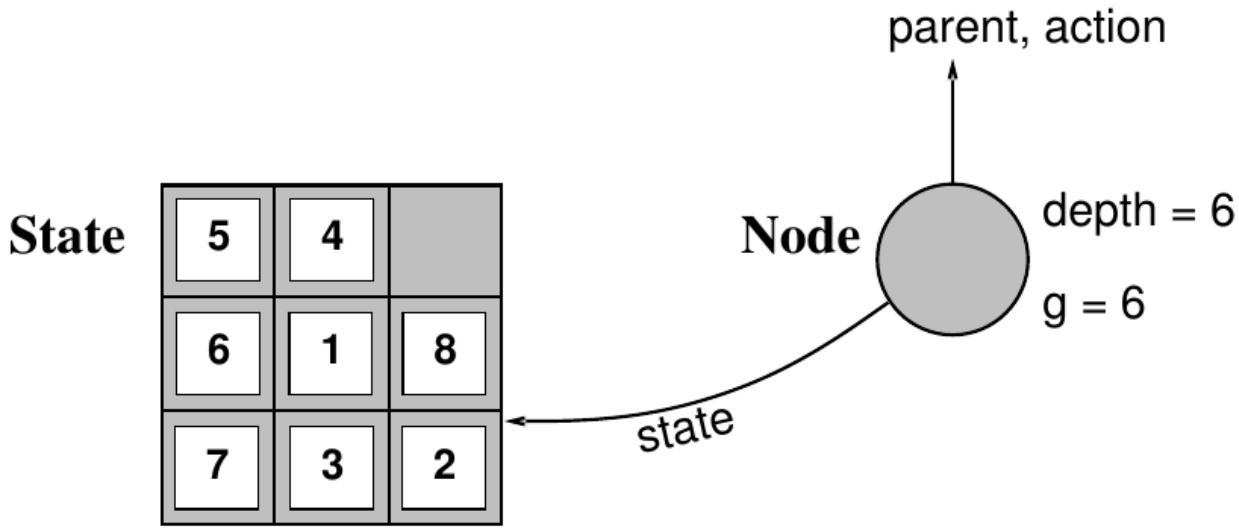
- Estado: (representação de) uma configuração física
- Nó: estrutura de dados que constitui parte da árvore de busca
  - inclui pai, filhos, profundidade, custo do caminho
- Estados não tem pais, filhos, profundidades ou custo de caminho!



A função "Expandir" cria novos nós, preenchendo os diversos campos do nó e usando a função "Sucessor" do problema para criar os estados correspondentes

Implementação: Estados x Nós

- Dois nós podem conter (ou apontar) para o mesmo estado - quando há redundância de caminhos
- Número de nós da árvore de busca **pode ser maior** do que o número de estados do mundo



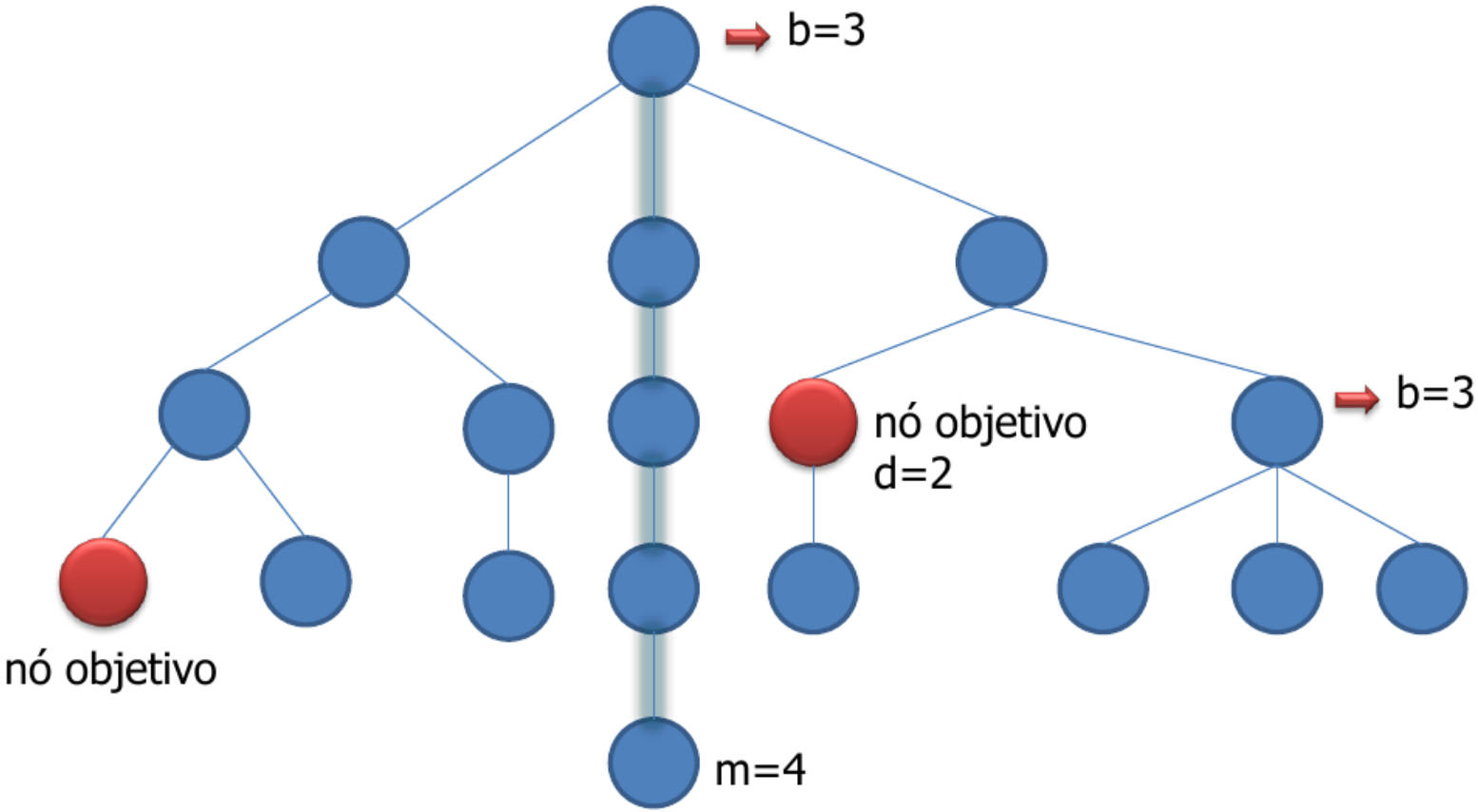
Avaliação das Estratégias de Busca

- Uma estratégia de busca é definida pela escolha da ordem de expansão dos nós
- Estratégias são avaliadas nas seguintes dimensões:
  - **completa:** sempre encontra uma solução (se ela existir)?
  - **ótima:** sempre encontra a solução de menor custo?
  - **complexidade de tempo:** número de nós gerados (tempo que leva)
  - **complexidade espacial:** número máximo de nós armazenados na memória (espaço)

Avaliação das Estratégias de Busca

- Complexidades de tempo e de espaço são medidas em função da dificuldade do problema:
  - **b:** número máximo de sucessores de um nó qualquer
  - **d:** profundidade do nó objetivo mais raso (tamanho do caminho, não custo)
  - **m:** tamanho máximo dos caminhos no espaço de estados (também não leva em conta o custo do caminho)

Exemplo das Medidas



Estratégias de Busca

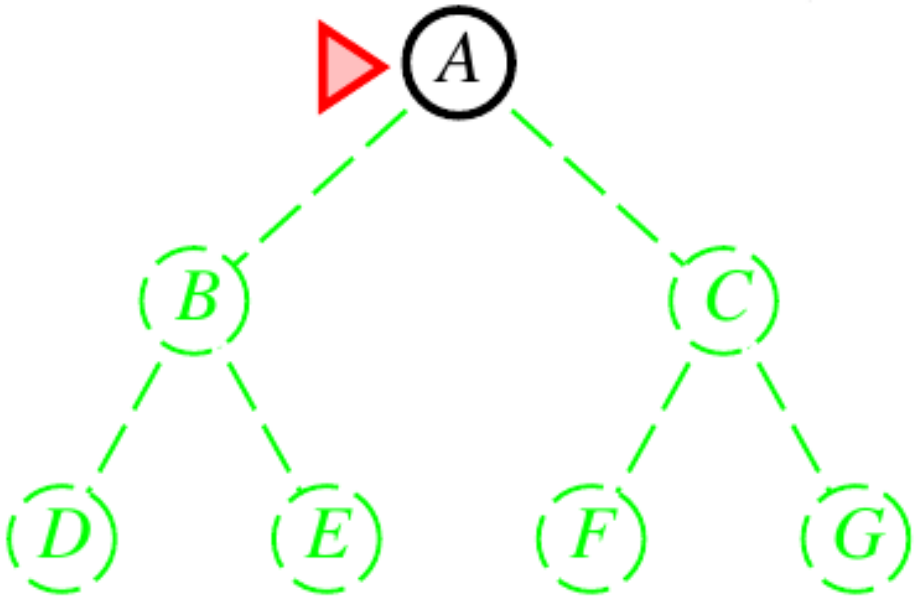
- Busca sem Informação ou cega
  - busca em largura
    - busca de custo uniforme (menor custo)
  - busca em profundidade
    - busca em profundidade limitada
    - busca de aprofundamento iterativo em profundidade

Busca sem Informação ou Cega

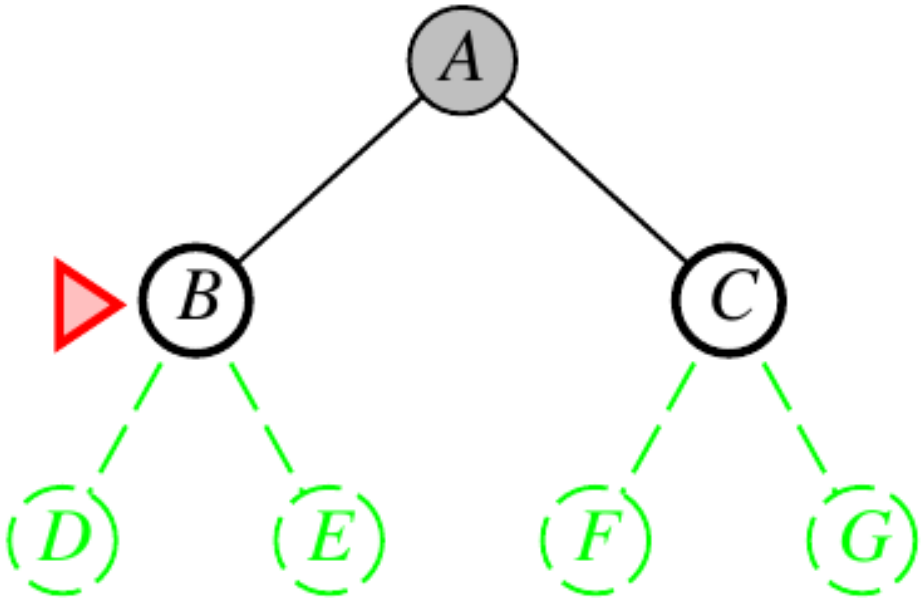
- Utilizam somente a informação disponível quando da definição do problema
- Estratégias de busca diferenciam-se apenas pela ordem em que expandem os nós da árvore de busca
- Estratégias que exploram primeiramente nós mais promissores com uso de informação extra-problema serão vistas mais tarde
  - busca informada ou heurística

Busca em Largura

- Busca em Extensão ou Breadth-First Search (BFS)
- Dentre os nós da fronteira, expande o mais raso
  - sempre explora o caminho mais curto antes
- A fronteira é uma FILA (FIFO) - nós sucessores vão para o final da fila de fronteira - porém sempre pega o mais raso

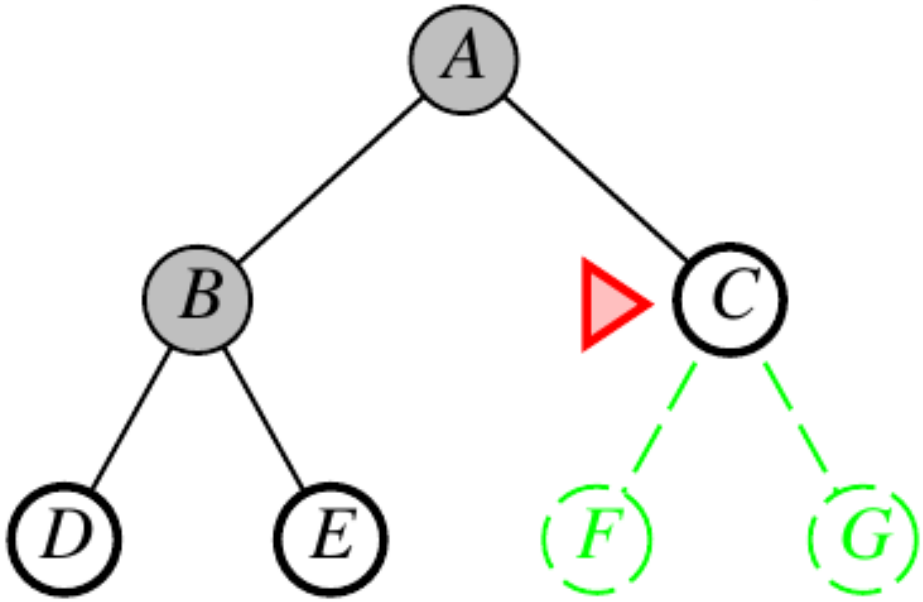


Busca em Largura



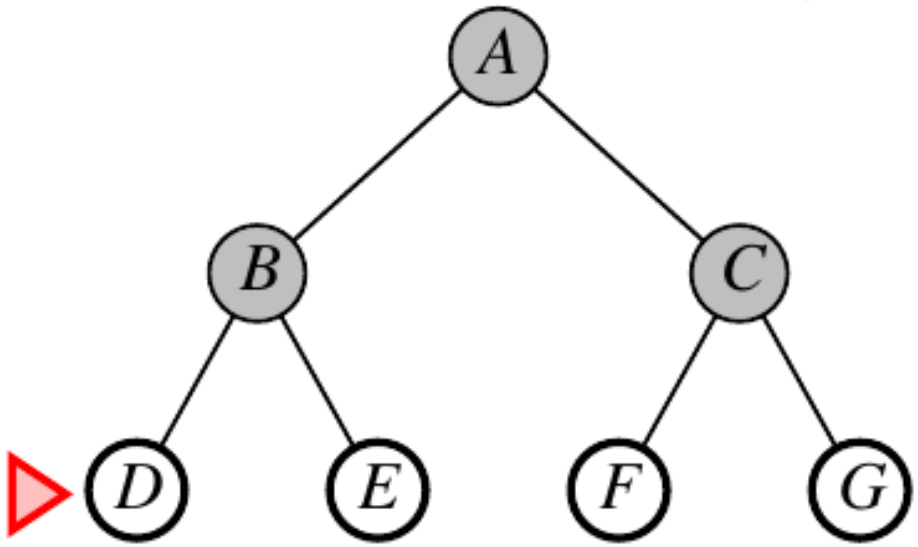
- Explorados = {A}
- Fronteira = B < C

Busca em Largura



- Explorados = {A, B}
- Fronteira = C < D < E

Busca em Largura

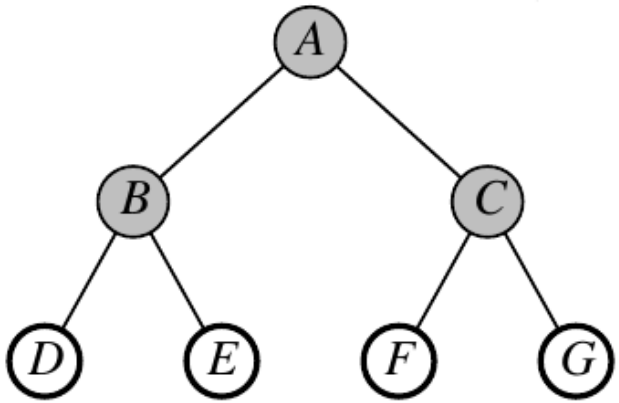


- Explorados = {A, B, C}
- Fronteira = D < E < F < G

Busca em Largura - Análise

- Completa? Sim (se b é finita)
- Ótimo? Sim, se custo for igual para todos os passos
- Tempo? $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- Espaço? $O(b^d)$ (mantém todos os nós na memória)
- Espaço: guarda todos os nós na memória
  - nós que já foram explorados
  - nós que estão na fronteira

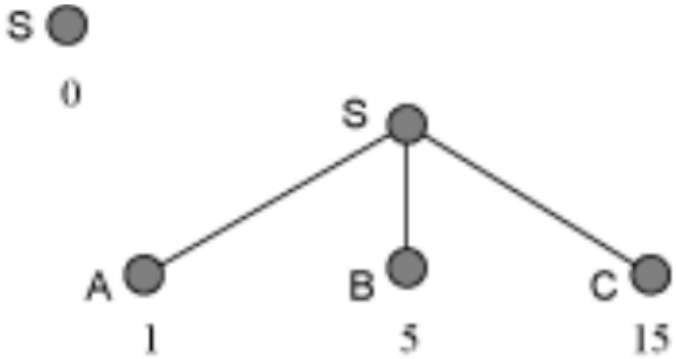
b: número máximo de sucessores de um nó qualquer (neste caso, 2)
d: profundidade do nó objetivo menos profundo (vamos supor G, então d = 2)

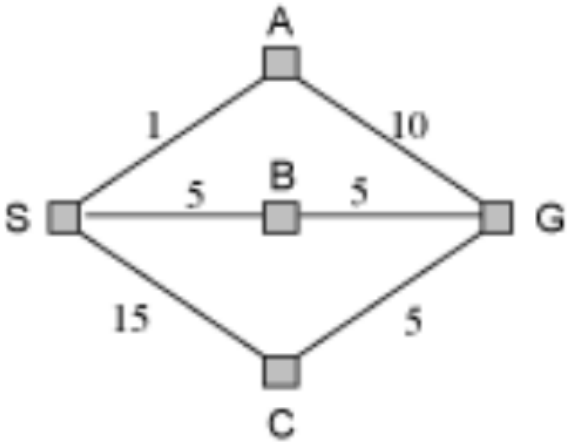


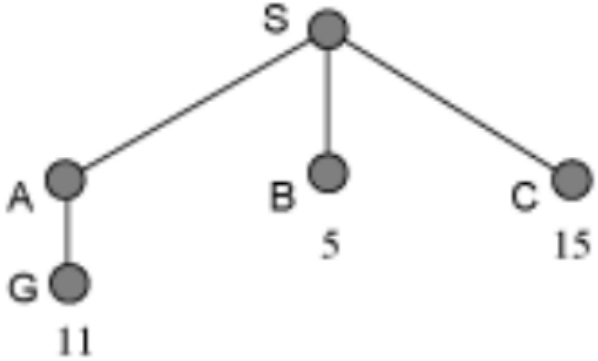
Busca de Custo Uniforme

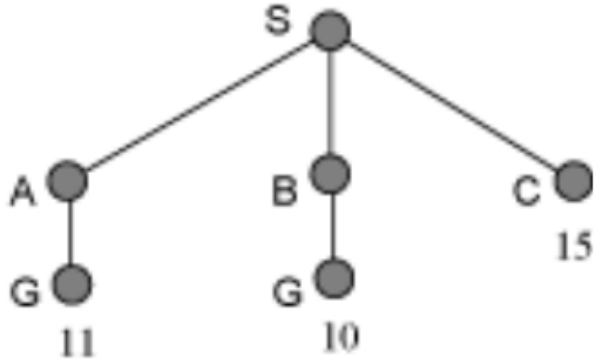
- **Busca de Custo Uniforme** ou **Uniform-Cost**
- Estende a Busca em Largura de modo a encontrar a solução ótima para qualquer valor de passo
- Expande o nó *n* que apresentar **menor custo de caminho** = g(n)
- Fronteira = fila ordenada pelo g(n)
- Sofre ligeira alteração em relação à busca em largura:
  - teste de objetivo é feito quando um nó é selecionado para expansão (e não quando é gerado)
  - se o nó já está na fronteira, mesmo assim é necessário verificar se o caminho encontrado é mais barato que aquele guardado

Busca de Custo Uniforme



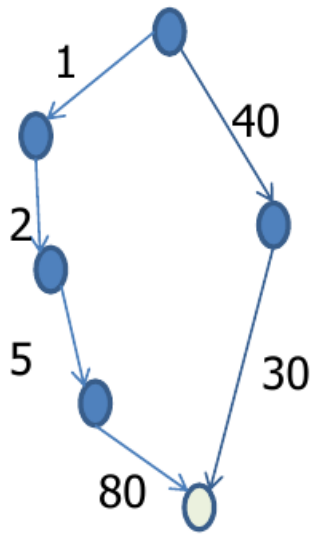






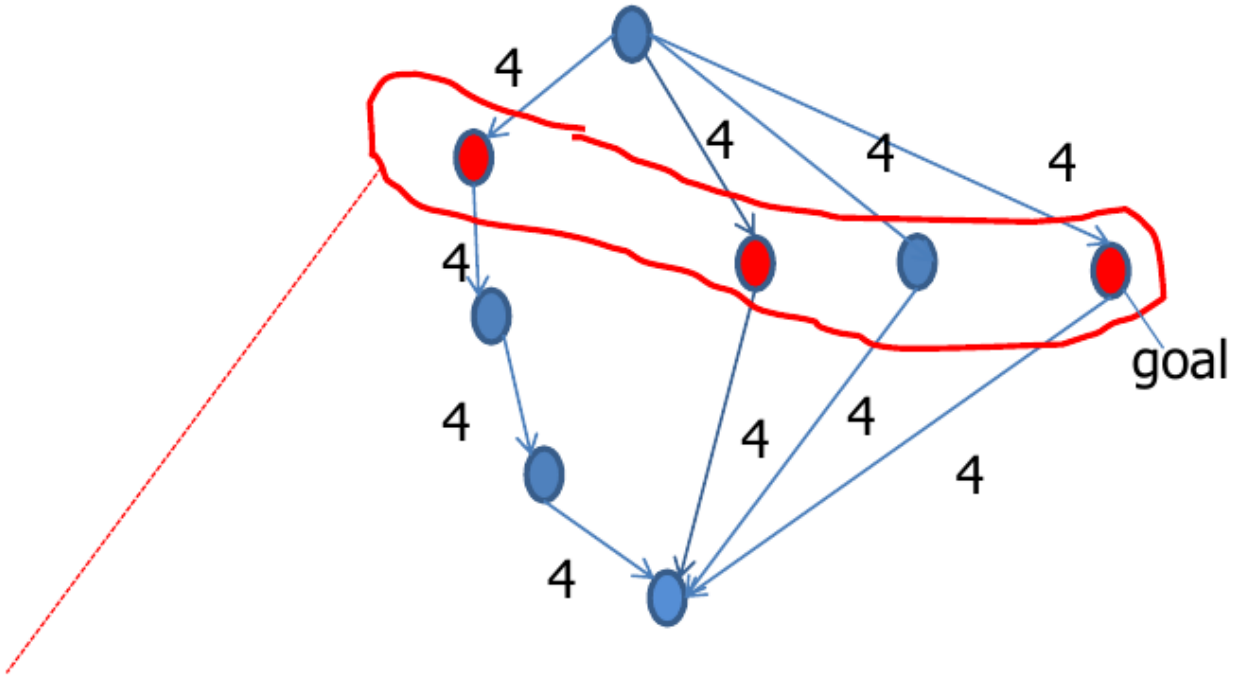
Avaliação Busca Custo Uniforme

- **Completo e Ótimo:** sim, garantida se cada passo for positivo
  - nós são expandidos em ordem crescente de custo de caminho
- **Complexidade de tempo e de espaço:** pode ser maior do que a da busca em largura, pois pode examinar grandes caminhos com pequenos passos antes de examinar caminhos com grandes passos que podem levar mais rapidamente a solução ótima



Avaliação Busca Custo Uniforme

- Se os custos das ações forem iguais, a busca por custo uniforme gastará mais tempo (e memória) que o breadth-first
  - examina todos os nós que estão na profundidade do objetivo para verificar se há algum de menor custo (expande todos os nós da profundidade *d* desnecessariamente)

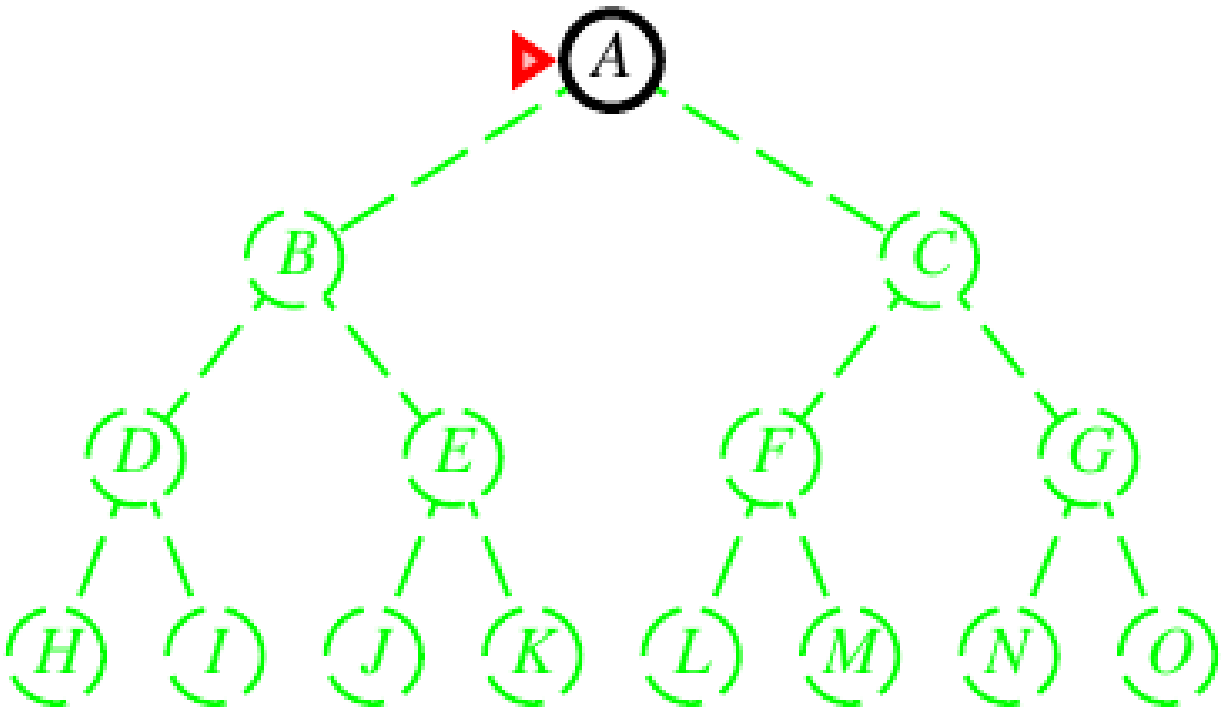


- Qualquer um dos nós na fronteira representa um estado objetivo:
  - custo uniforme deve examinar todos antes de retornar a solução
  - Breadth-first retorna o primeiro encontrado

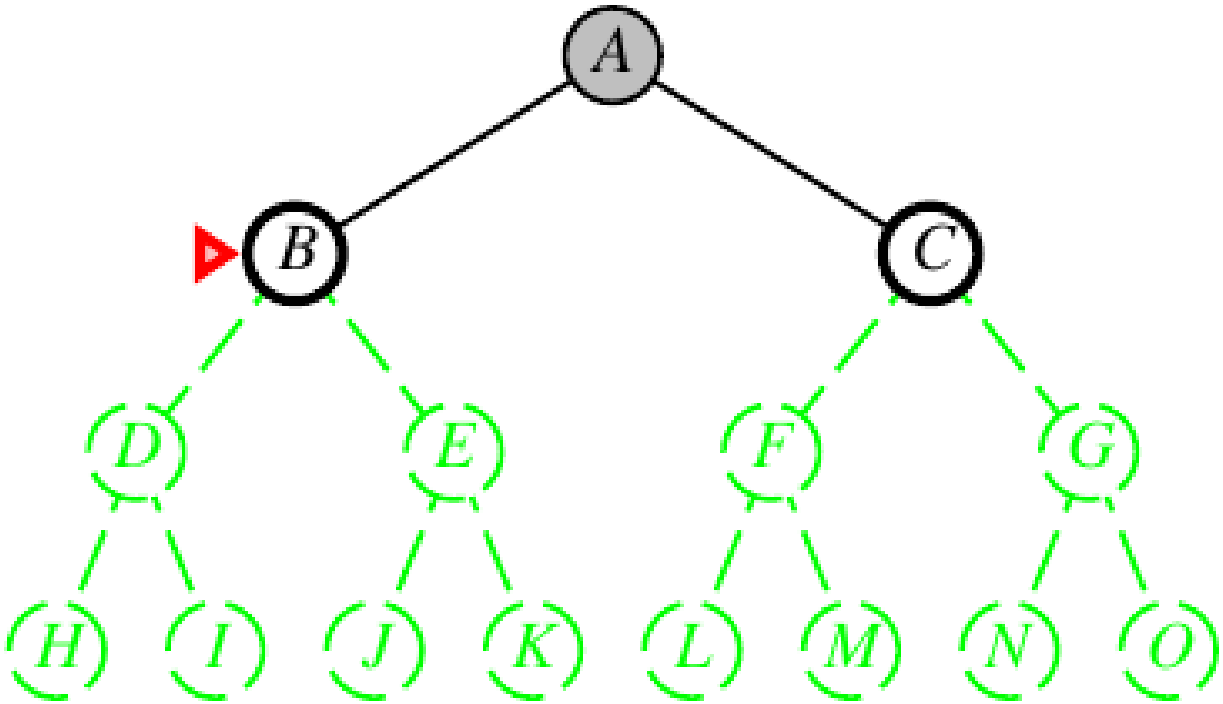
Busca em Profundidade

- **Busca em Profundidade** ou **Depth-First Search (DFS)**
- Expande o nó de maior profundidade que esteja na fronteira da árvore de busca
- Fronteira = Pilha (LIFO)
  - novos sucessores são colocados no final
  - último ou topo da pilha é selecionado a cada passo

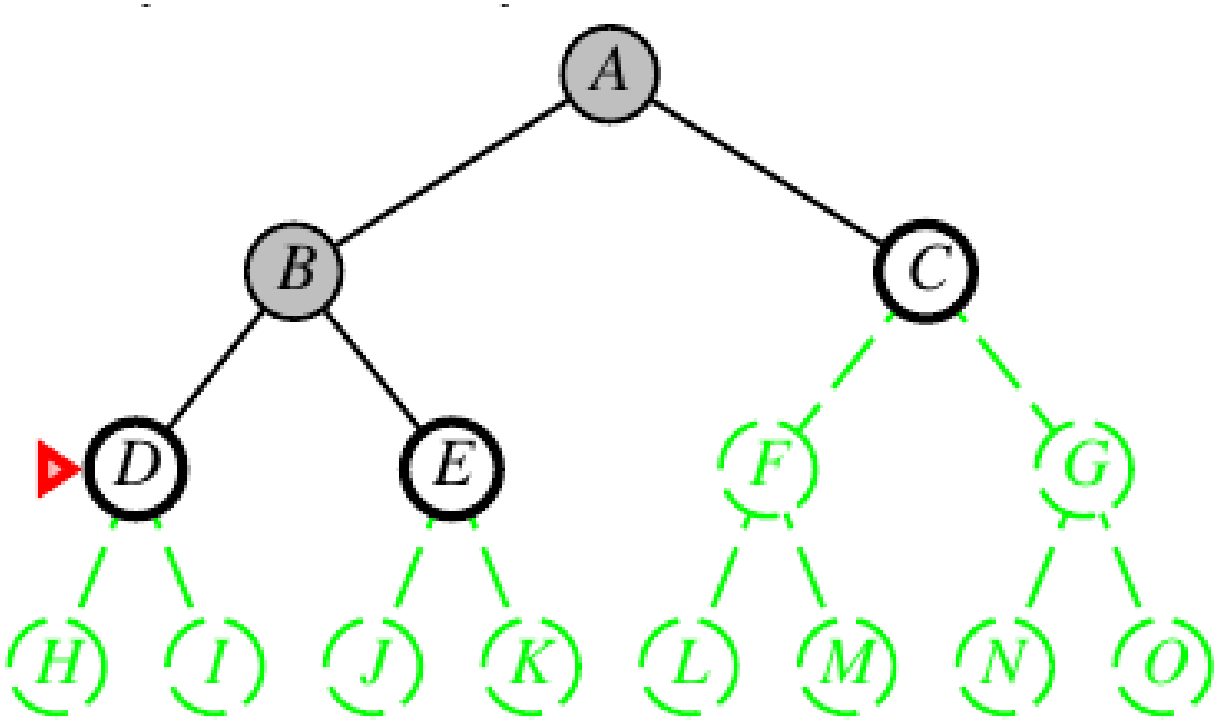
Busca em Profundidade



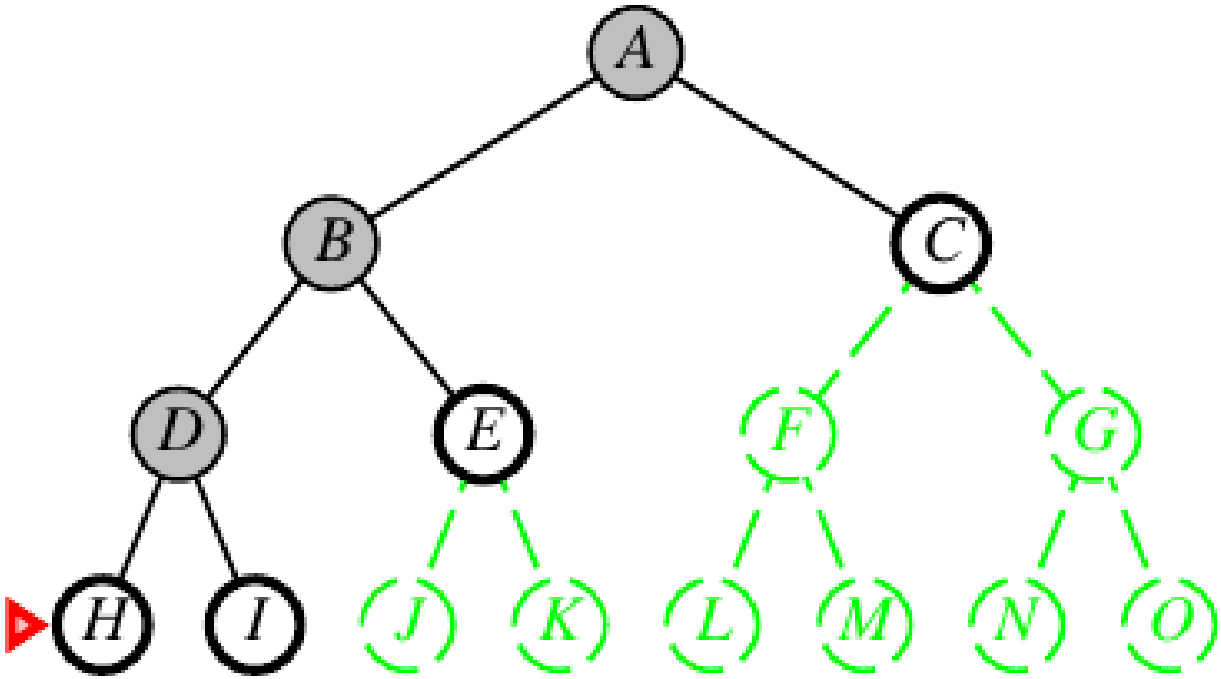
Busca em Profundidade



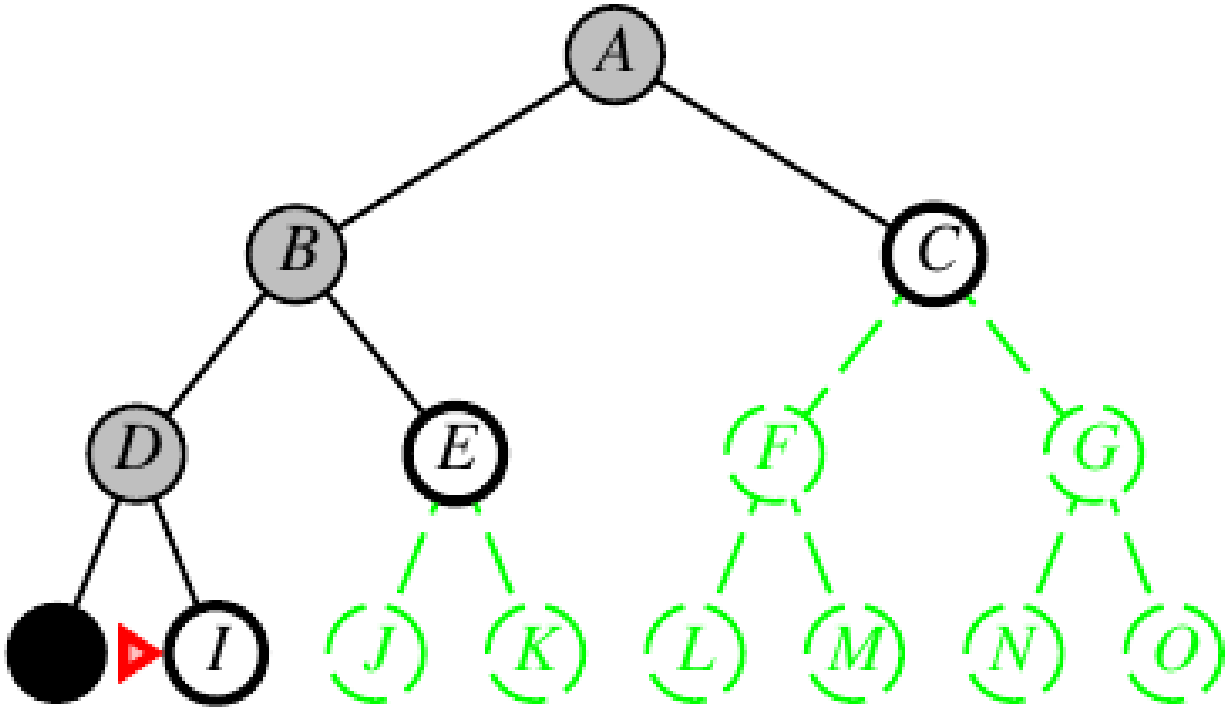
Busca em Profundidade



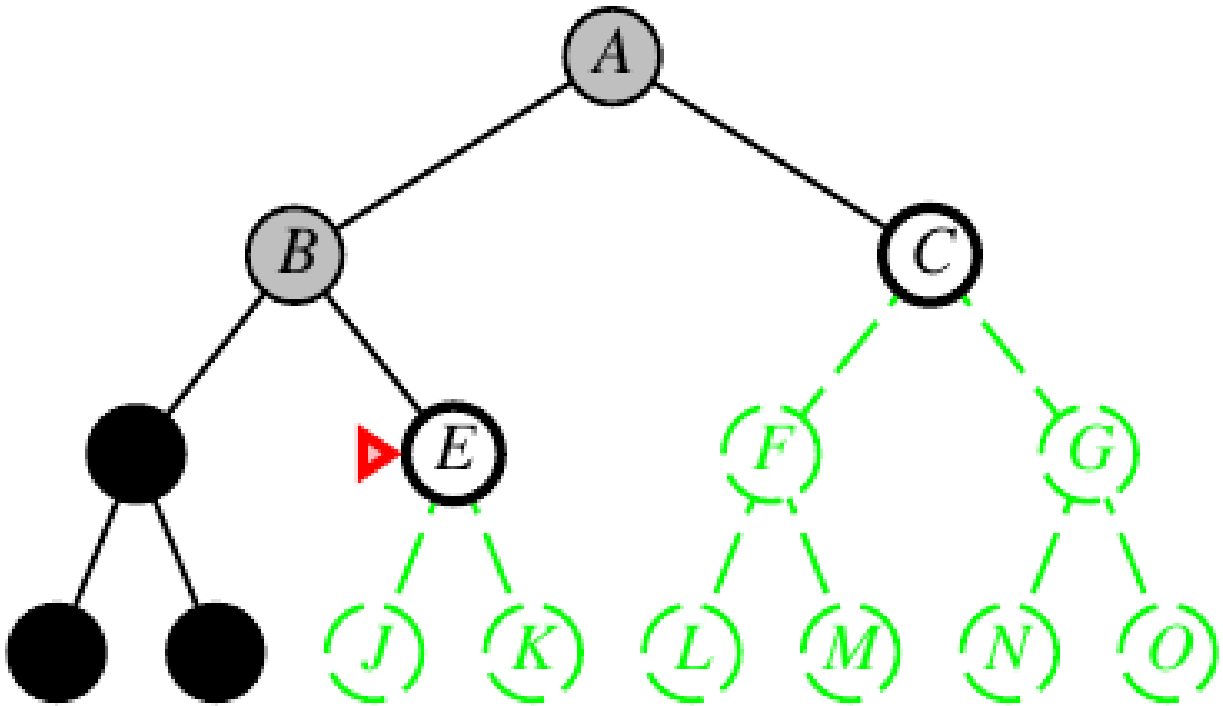
Busca em Profundidade



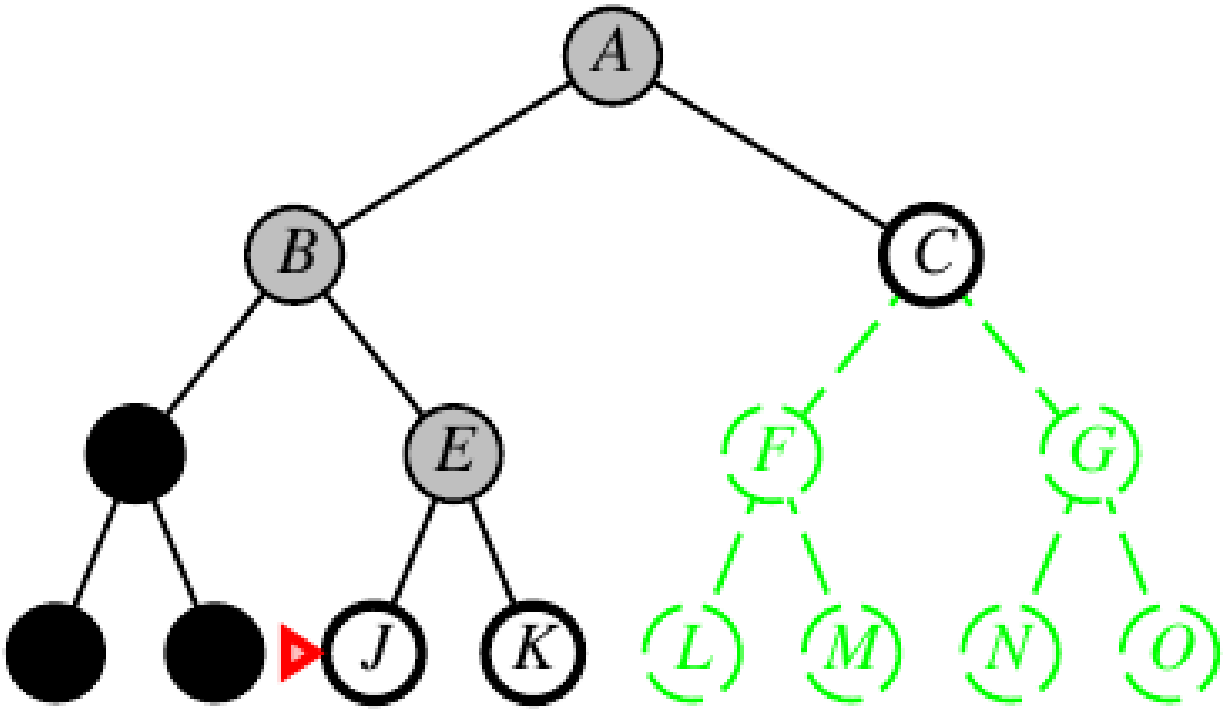
Busca em Profundidade



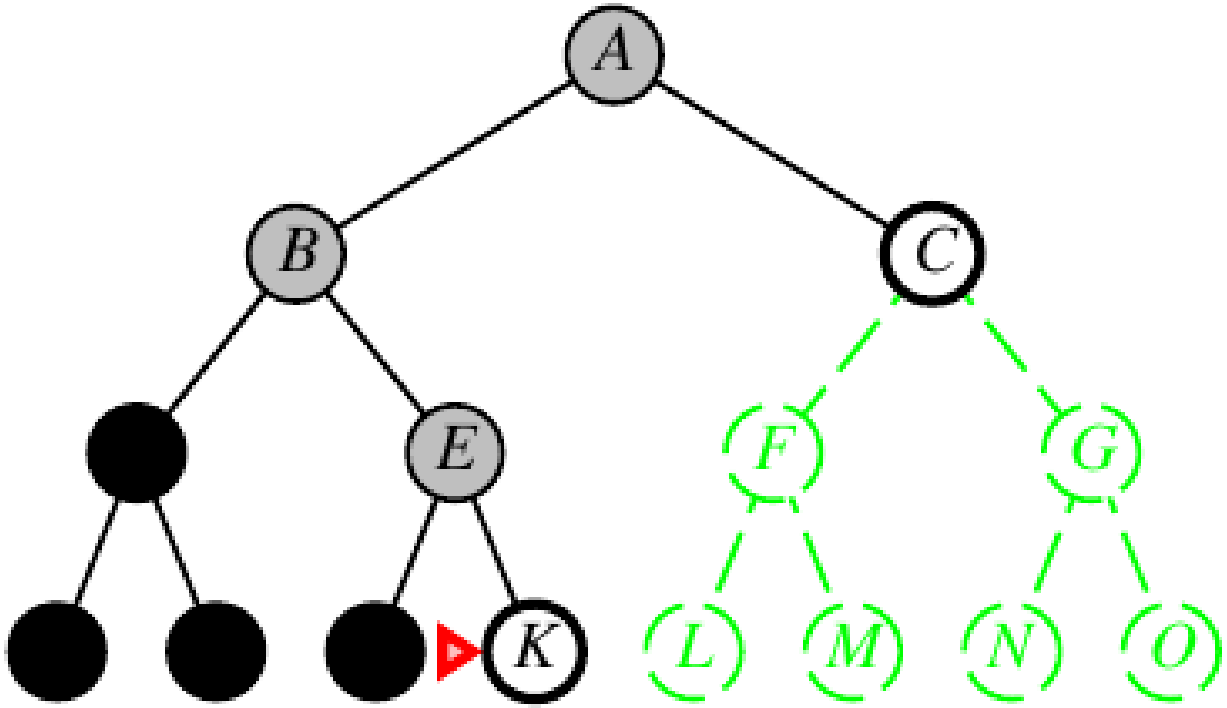
Busca em Profundidade



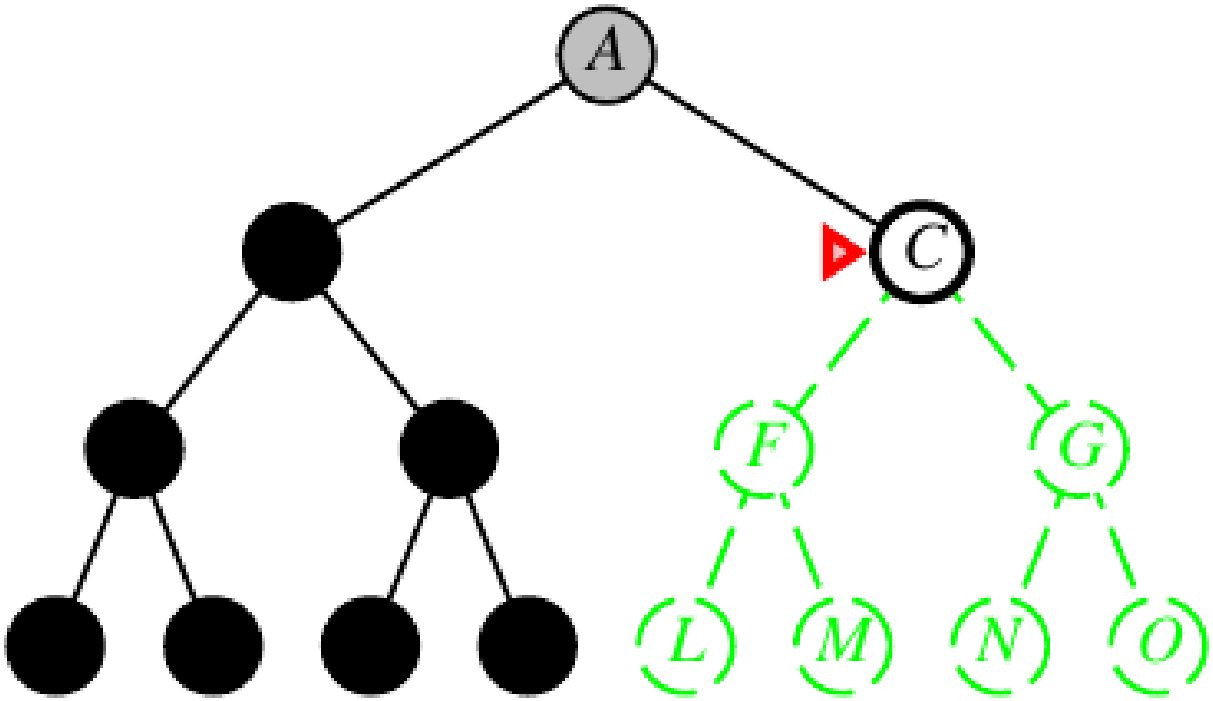
Busca em Profundidade



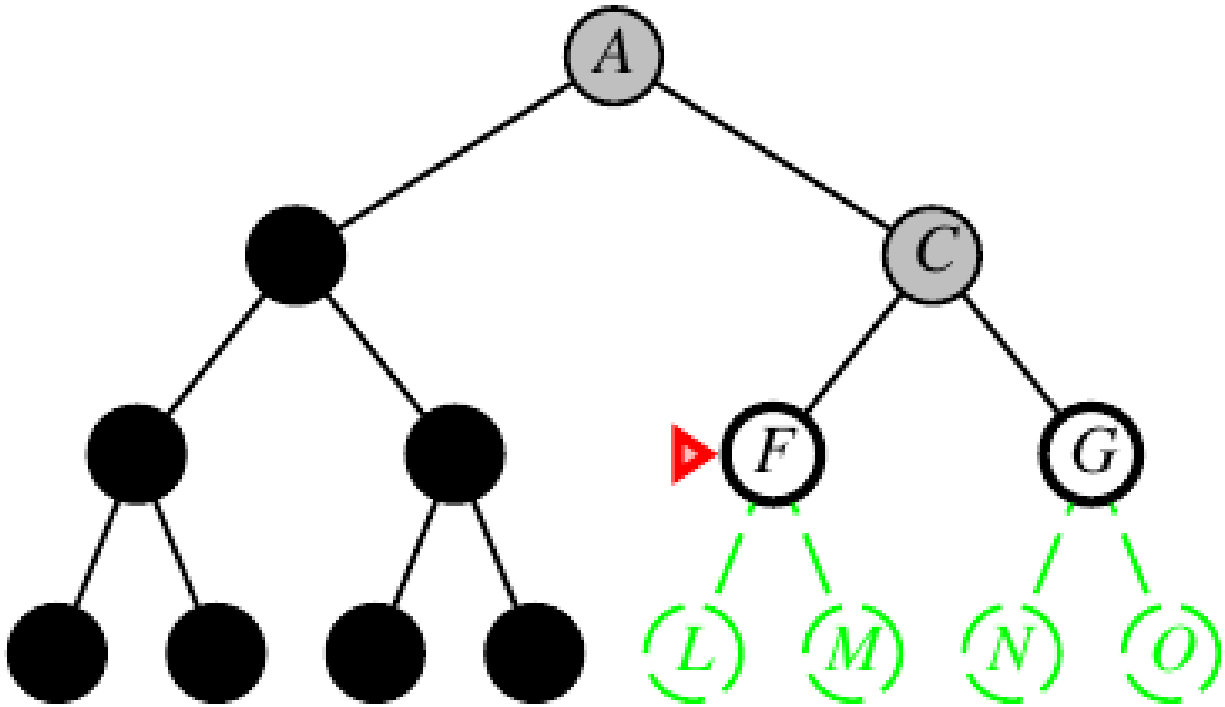
Busca em Profundidade



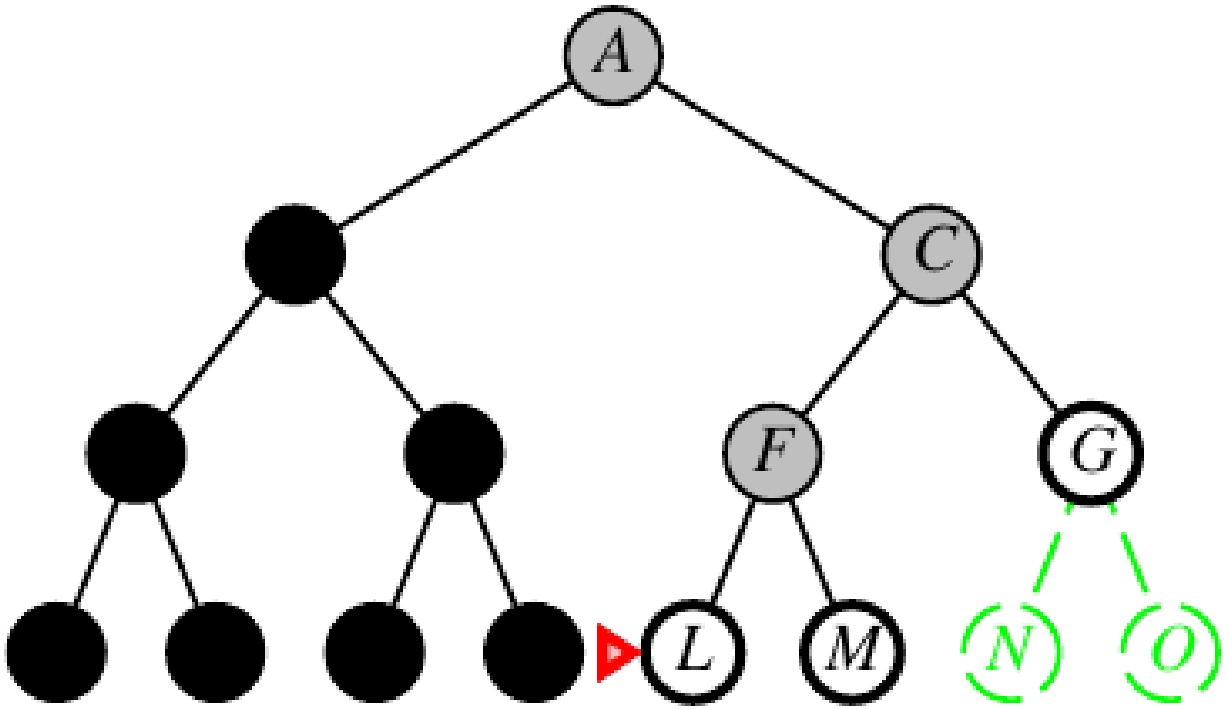
Busca em Profundidade



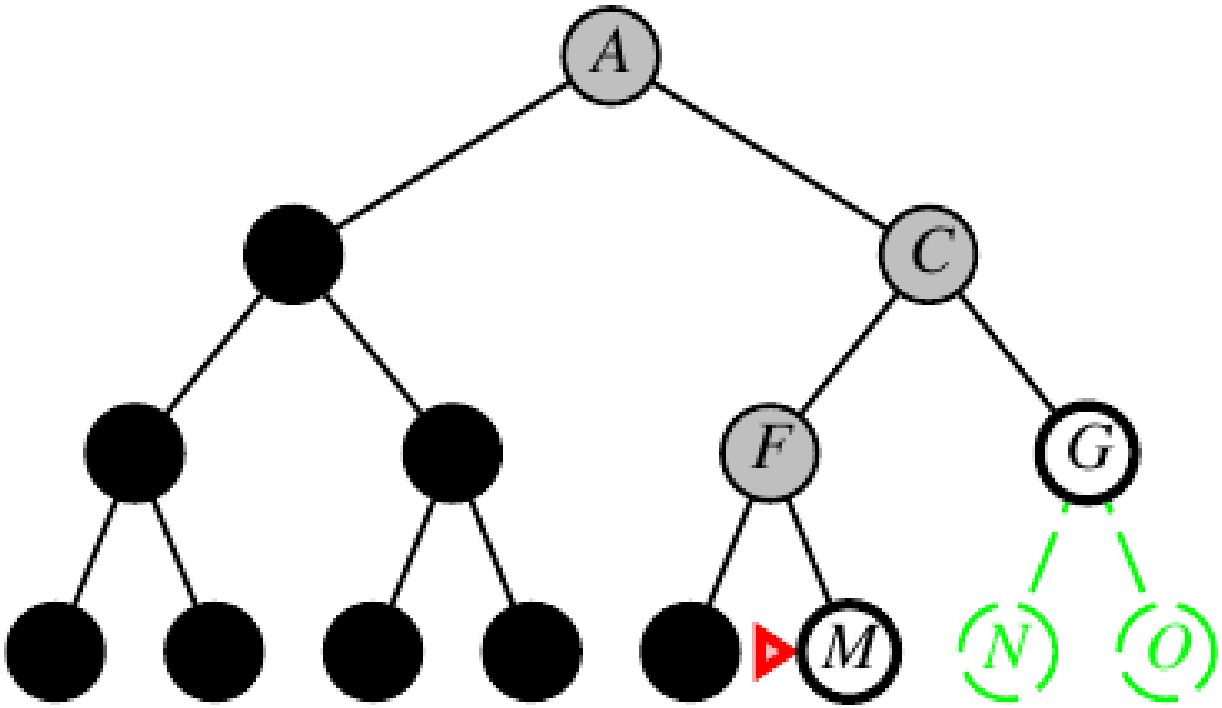
Busca em Profundidade



Busca em Profundidade



Busca em Profundidade



Avaliação da Busca em Profundidade

- **Completa?** Não. Falha em espaços com profundidade infinita (loops)
  - se modificada para evitar estados repetidos é completa para espaços finitos
- **Ótima?** Não. Pode retornar solução em profundidade maior na árvore de busca do que a mais rasa
- **Espaço?** $O(b \cdot m) \rightarrow$ espaço linear!
  - precisa armazenar: único caminho de raiz a uma folha e nós irmãos não expandidos no caminho. Nós expandidos podem ser removidos da memória quando seus descendentes forem todos explorados
- **Tempo?** $O(b^m)$. Péssimo quando m é muito maior que d

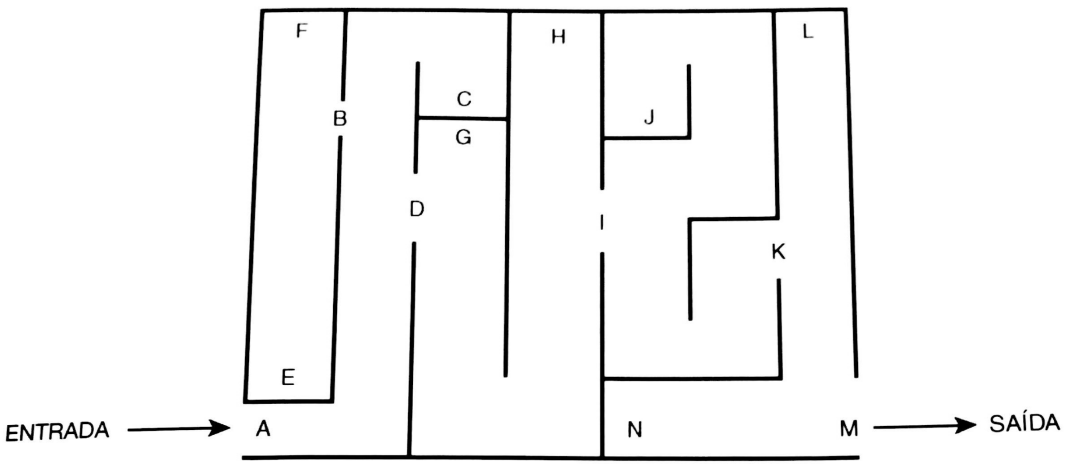
Estratégia deve ser evitada quando as árvores geradas são muito profundas ou geram caminhos infinitos

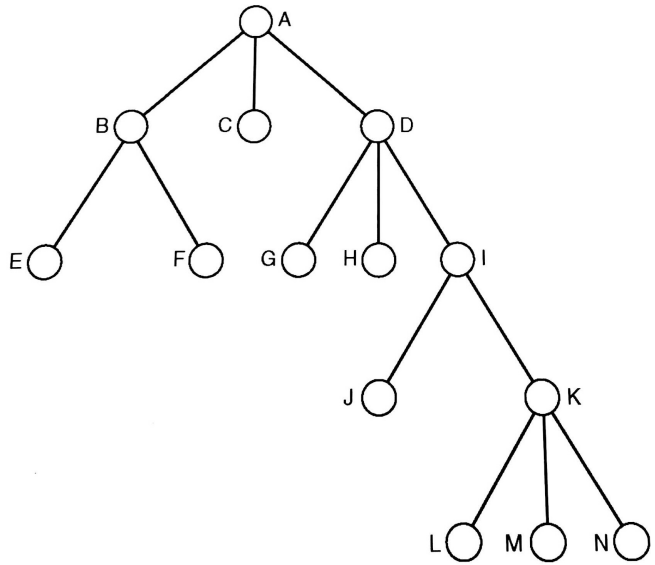
Busca em Profundidade (BP) x Busca em Largura (BL)

- Tempo
  - $m = d$: BP tipicamente ganha
  - $m > d$: BL pode ganhar
  - m é infinito: BL provavelmente será melhor
- BP geralmente é melhor que BL

Busca em Profundidade (BP) x Busca em Largura (BL)

- BP e BL fáceis de implementar
- BP mais fácil $\rightarrow$ modo natural de humanos realizarem buscas
  - explorar totalmente cada caminho antes de mudar para outro caminho
  - ao invés de visitar e expandir determinados caminhos várias vezes
  - ex.: percorrendo um labirinto, comprando um presente
  - perambular a esmo? busca racional? "andar em círculos"?





Busca em Profundidade Limitada

- **Busca em Profundidade Limitada** ou **Depth-Limited**
- Tenta amenizar o problema da busca em profundidade em espaços de estado de tamanho infinito
- Impõe um limite l para a máxima profundidade a ser expandida
- Nós na profundidade l são tratados como se não tivessem sucessores

Busca em Profundidade Limitada

- Problemas deste artifício:
  - se $l < d$ a solução não será encontrada
  - e, portanto, é fonte de **incompletude**
  - se $l > d$ **não encontra o ótimo**
- Complexidade de **tempo**: $O(b^l)$
- Complexidade de **espaço**: $O(b.l)$

Busca em Profundidade Limitada

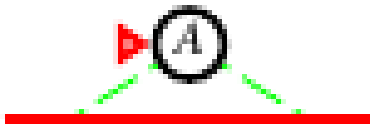
- O problema é determinar o **valor de** /!
- Às vezes, o conhecimento sobre o problema pode dar limite de profundidade
  - um modo é pegar o **diâmetro do espaço de estados do grafo**
  - ex.: tamanho máximo do caminho que liga uma cidade a outra
  - analisando o mapa da Romênia, qualquer cidade pode ser alcançada a partir de outra com no máximo 9 passos
    - ★ neste caso, é 9 - Neamt-Lugoj ou Neamt-Timisoara
- Diâmetro do espaço de estados
  - infelizmente, não é conhecido para a maioria dos problemas

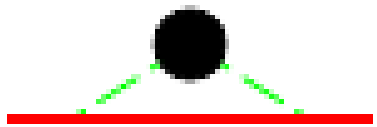
Busca de Aprofundamento Iterativo

- **Busca em Profundidade por Aprofundamento Iterativo**
- Estratégia utilizada com frequência em conjunto com busca em profundidade
- Encontra o melhor limite *l* de profundidade
  - aumenta gradualmente *l*
    - 0, 1, 2, ...
    - até encontrar um estado objetivo
- Isto ocorre quando a profundidade alcançar **d**
  - profundidade do objetivo mais raso

Busca de Aprofundamento Iterativo

- Limite = 0



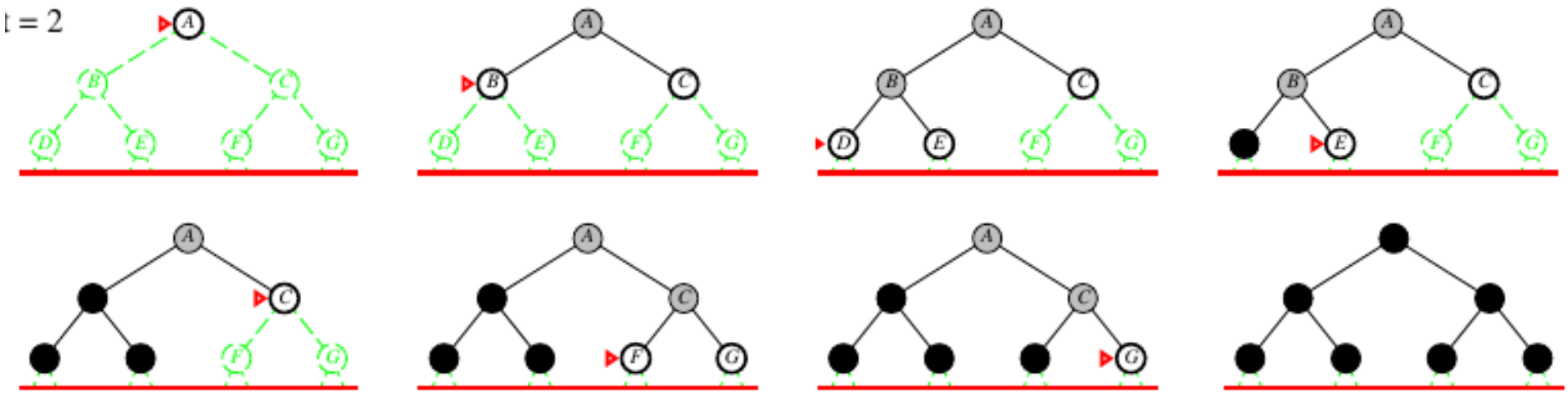


Busca de Aprofundamento Iterativo

- Limite = 1

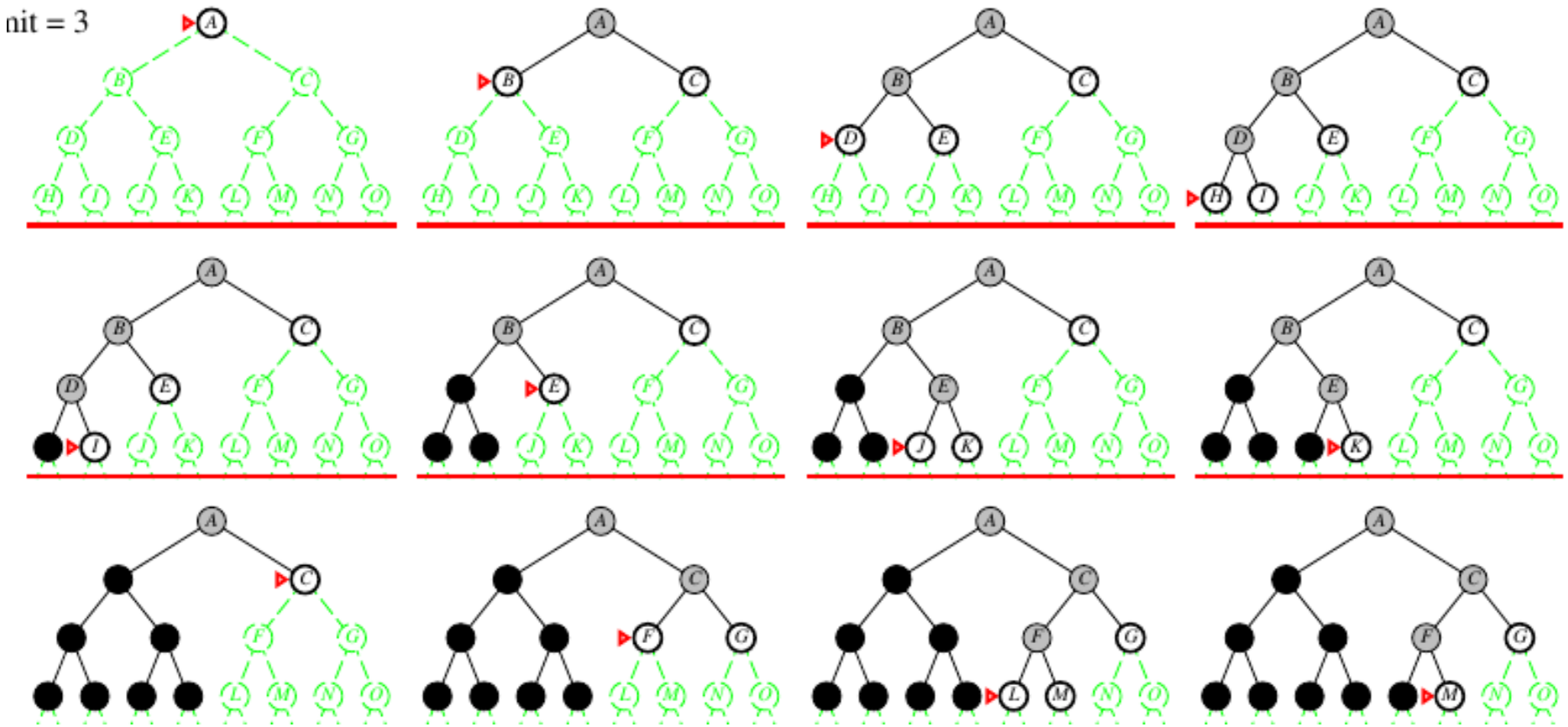
Busca de Aprofundamento Iterativo

- Limite = 2



Busca de Aprofundamento Iterativo

- Limite = 3



Avaliação da Busca de Aprofundamento Iterativo

- Pode parecer desperdício
  - estados são gerados várias vezes
  - custo não é muito alto, a maior parte dos nós estará em níveis inferiores
    - ★ nós do nível inferior (profundidade d) são gerados uma vez
    - ★ nós do penúltimo são gerados duas vezes
    - ★ e assim por diante, ...
    - ★ até filhos da raiz, gerados d vezes

Avaliação da Busca de Aprofundamento Iterativo

- **Completa?** Sim, quando b for finito = busca em largura
- **Ótima?** Sim, se custos dos passos forem iguais / crescem com a profundidade
- **Tempo?** $O(b^d)$ = busca em largura
- **Espaço?** $O(b.d)$ = busca em profundidade

Indicado quando o espaço de estados é grande e a profundidade da solução não é conhecida

Resumo

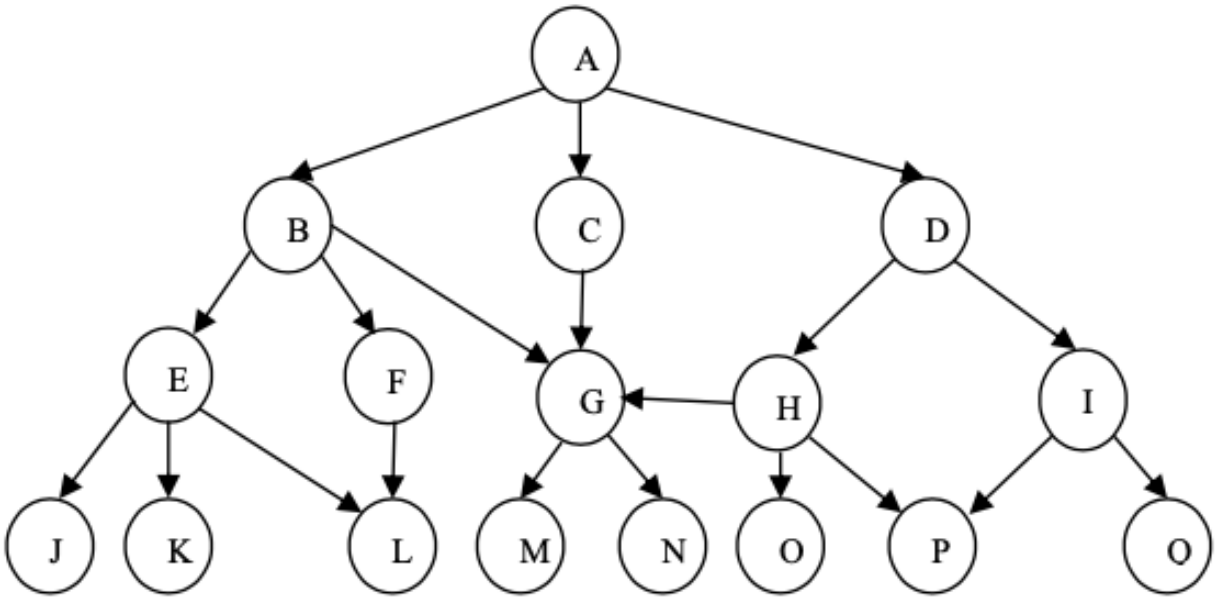
- Estratégias de busca apresentadas realizam busca exaustiva no espaço de estados, não fazem uso de conhecimento para encontrar sua solução
- Para contornar este problema, métodos heurísticos podem ser utilizados

Referências e Leituras Complementares

- Cap. 03 $\rightarrow$ livro Russel e Norvig
- Cap. 03, cap. 04 $\rightarrow$ livro Ben Coppin

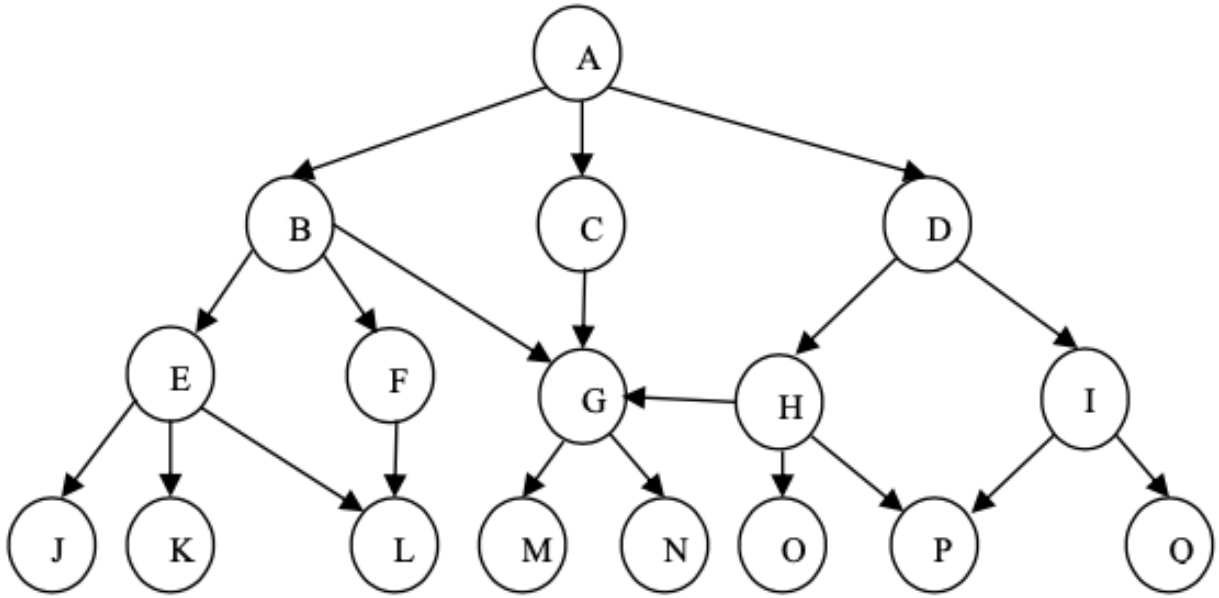
Exercícios

- Considere o grafo abaixo, representando o espaço de estados para um problema. Utilize o algoritmo de **busca em largura** para encontrar a solução, sendo o nó inicial A e o nó objetivo N.
- Apresente o conteúdo das listas (fronteira e explorados) a cada passo e construa árvore de busca
- Indique as ações realizadas quando um nó gerado já estiver em uma das duas listas e diga qual o caminho da solução



Exercícios

- Considere o mesmo grafo do exercício anterior, representando o espaço de estados para um problema. Utilize o algoritmo de **busca em profundidade** para encontrar a solução, sendo o nó inicial A e o nó objetivo N.
- Apresente o conteúdo das listas (fronteira e explorados) a cada passo e construa árvore de busca
- Indique as ações realizadas quando um nó gerado já estiver em uma das duas listas e diga qual o caminho da solução



Exercícios

Considere o **problema do mundo do aspirador de pó**:

- um cenário é representado por uma grade de 2x2, sendo que cada quadrado pode ter ou não sujeira
- um aspirador de pó pode se mover nesse cenário com os seguintes movimentos (aspirar a sujeira, andar para a esquerda, andar para a direita, andar para cima e andar para baixo)
- os operadores devem ser aplicados na ordem:
  1. aspirar
  2. mover à direita
  3. mover para baixo
  4. mover à esquerda
  5. mover para cima
- A Figura ilustra o estado inicial em que o aspirador de pó está no quadrado superior esquerdo e que os dois quadrados da direita estão sujos

A	S
	S

Exercícios

- Encontre a sequência de movimentos para chegar a um estado onde todos os quadrados estão limpos
- Para tanto, construa a árvore de busca o algoritmo de busca em largura e busca em profundidade limitada com lim=3
- Enumere os nós da árvore de busca de acordo com a sequência em que foram expandidos
- Indique as ações realizadas quando um nó gerado já estiver em uma das listas (fronteira e explorados)
- Mostra a sequência de movimentos que determina a solução
- Compare e comente as duas soluções encontradas quanto ao nível em que estava o objetivo encontrado, o tamanho da solução (número de movimentos) e o número de nós gerados
- Não é necessário descrever os conteúdos das listas (fronteira e explorados) passo a passo