Busca não informada

Capítulo 3

Parte II

Leliane Nunes de Barros

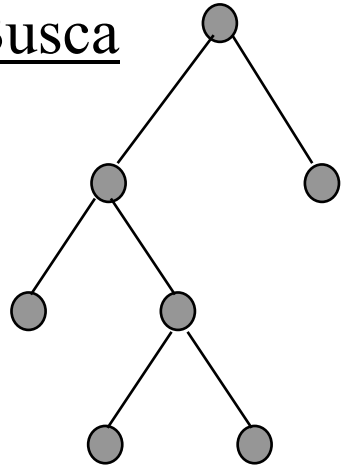
leliane@ime.usp.br

Agente baseado em metas: ingredientes

- Formulação de problema
  - *estado inicial*:
  - *ações*:
  - *função custo*:
  - *teste de meta:*

- Nós da árvore de busca -
  - estado
  - o nó *pai*
  - ação
  - profundidade
  - o custo do caminho da raiz até o nó

- Árvore de Busca



Algoritmo de busca geral

função Busca-Geral(*problema*, *estratégia*) **devolve** uma solução, ou falha

inicializa a árvore de busca com o estado inicial do *problema*

laço faça

se não há mais candidatos para expandir **então devolve** falha

escolha um nó folha para expandir

se o nó contém um estado meta **então devolve** a solução

senão expanda o nó de acordo com a *estratégia*

fim

Obs: *estratégia* = QUEUING-FN

Estratégias de busca

- A estratégia de busca é definida pela ordem em que os nós são expandidos
- Estratégias são avaliadas através das seguintes dimensões:
  - completeza (encontra a solução se existir uma)
  - complexidade de tempo (número de nós gerados e expandidos)
  - complexidade de espaço (número de nós na memória)
  - solução ótima (encontra a solução de menor custo)

Estratégias de busca

- Complexidade é medida em termos de:
  - ***b*** - fator de ramificação da árvore de busca
  - ***d*** - profundidade da solução de menor custo
  - ***m*** - profundidade máxima do espaço de busca

Estratégias de busca não-informada

Variam quanto a ordem em que nós são expandidos:

- Busca em largura - *Breadth first*
- Busca de custo uniforme
- Busca em profundidade - *Depth first*
- Busca em profundidade limitada
- Busca em profundidade iterativa
- Busca bidirecional

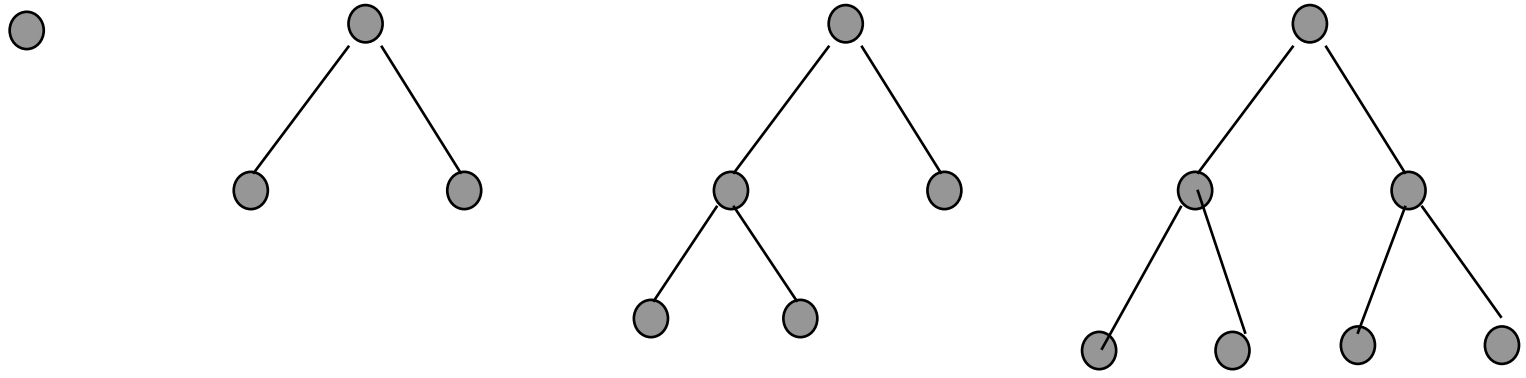
Não possuem informação sobre a distância do nó à meta.

Estratégias de busca não-informada (outros nomes)

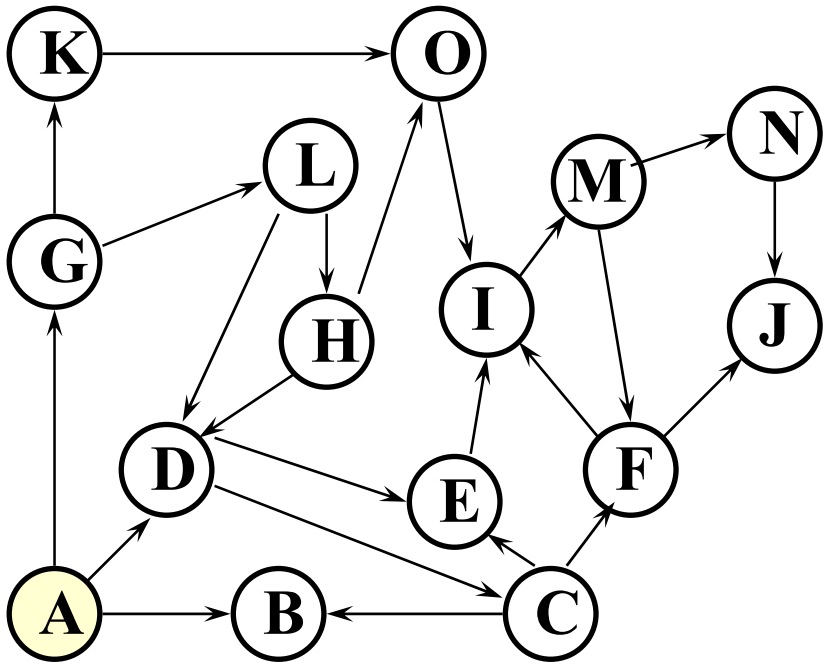
- *Busca cega*
- *Busca exaustiva*
- *Busca completa*
- *Busca de força bruta*
- *Busca sistemática*

Busca em Largura

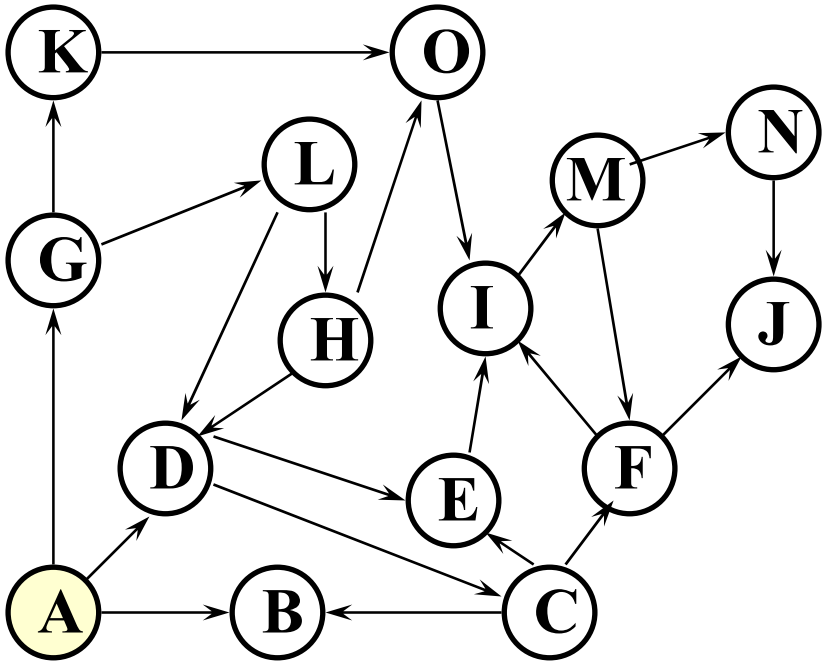
QUEUING-FN at-end

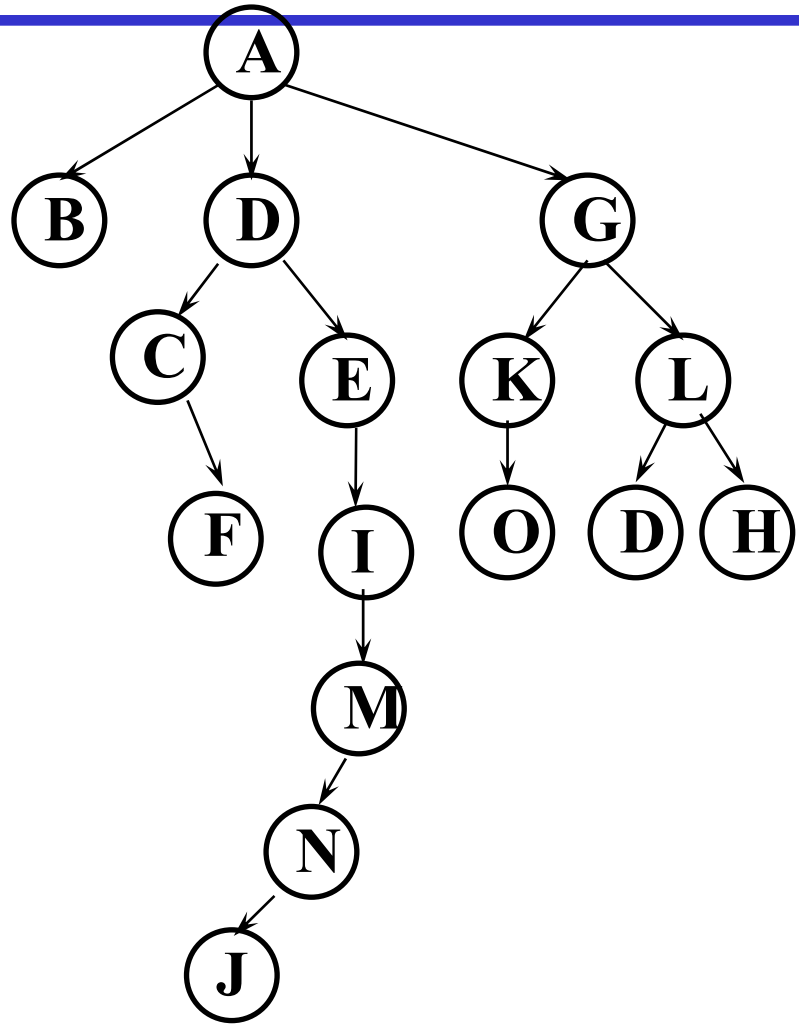


Busca em Largura



Busca em Largura





Busca em Largura

A BDG CEKL FIODH M N J ...

Algoritmo de busca em largura

função Busca-Geral(*problema*, *estratégia*) **devolve** uma solução, ou falha

inicializa a árvore de busca com o estado inicial do *problema*

laço faça

se não há mais candidatos para expandir **então devolve** falha

escolha um nó folha para expandir

se o nó contém um estado meta **então devolve** a solução

senão expanda o nó de acordo com a *estratégia*

fim

Obs: *estratégia* = fila (FIFO)

Propriedades da busca em largura

- **Completa?**
- **Solução ótima?**
- **Tempo?**
- **Espaço?**

Propriedades da busca em largura

- **Completa?** Sim. Garante encontrar a solução (se existir uma)
- **Solução ótima?** Sempre encontra primeiro a solução mais rasa. Devolve a solução ótima se a função custo for igual a 1.
- **Tempo?** $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$
- **Espaço?** $O(b^d)$ (guarda todos os nós na memória)

Busca em largura

Prof.	Nós	Tempo	Memória
0	1	1 ms	100 bytes
2	111	.1 s	11 Kbytes
4	11.111	11 s	1 Mbytes
6	10^6	18 min	111 Mbytes
8	10^8	31 hs	11 Gbytes
10	10^{10}	128 dias	1 Tbytes
12	10^{12}	35 anos	111 Tbytes
14	10^{14}	3500 anos	11.111 Tbytes

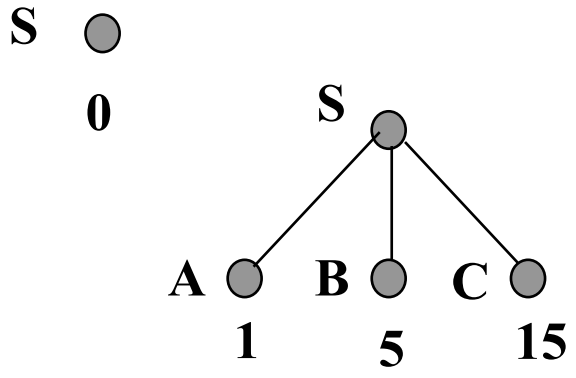
B=10 1000 nós/seg 100 bytes/nó

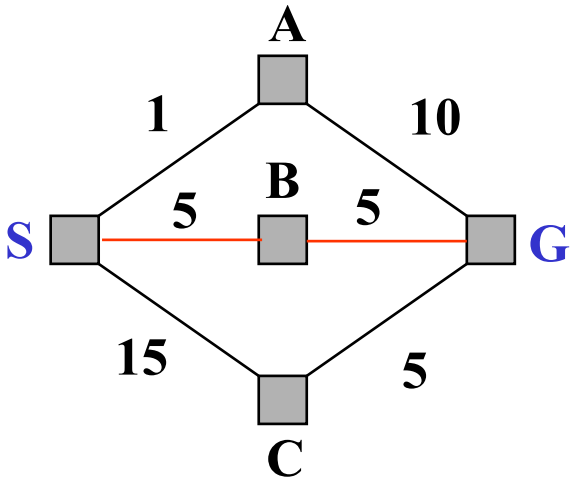
Busca em largura (BFS)

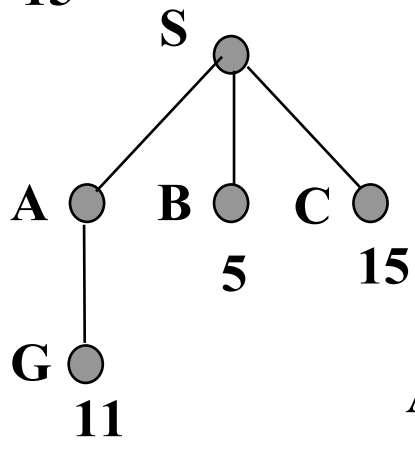
- Desvantagens:
  - complexidade exponencial: $O(b^d)$
  - problema de memória maior que o de tempo
- Vantagens:
  - garante encontrar a solução
  - encontra a solução mais próxima da raiz

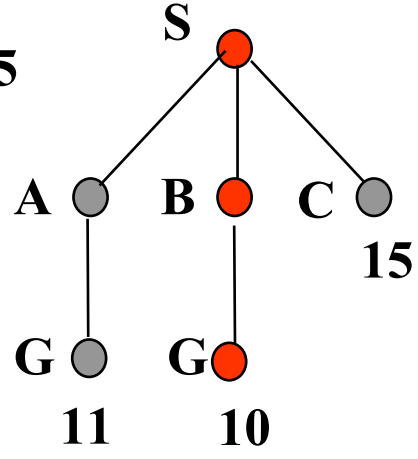
    $g(n) = Profundidade(n)$

Busca de custo uniforme



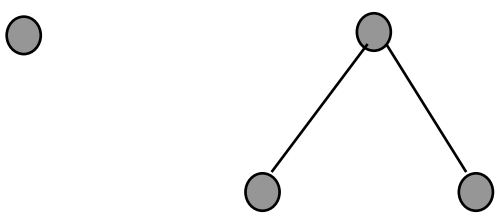


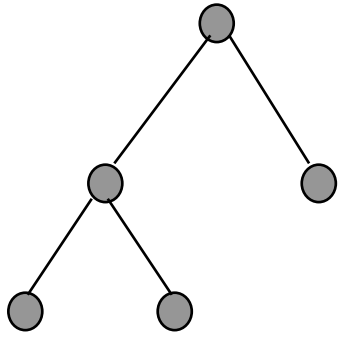


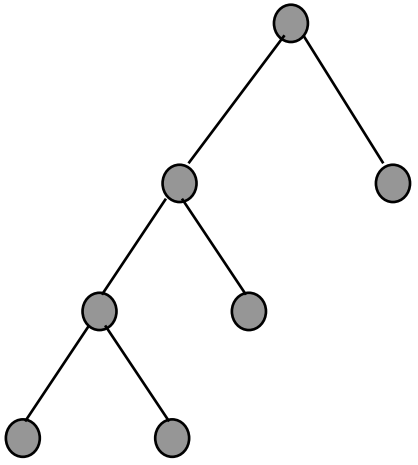


Problema de roteamento

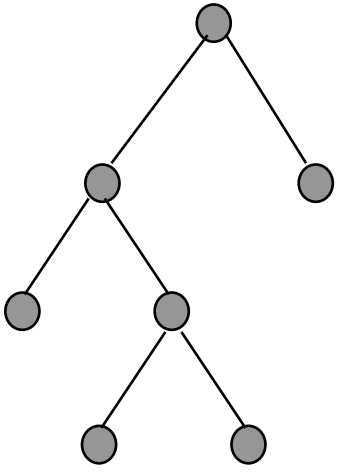
Busca em profundidade (DFS)

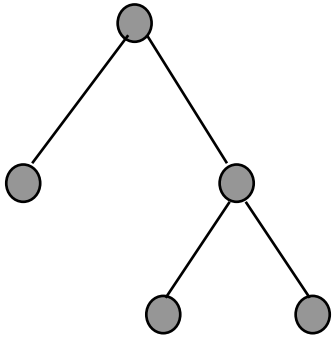


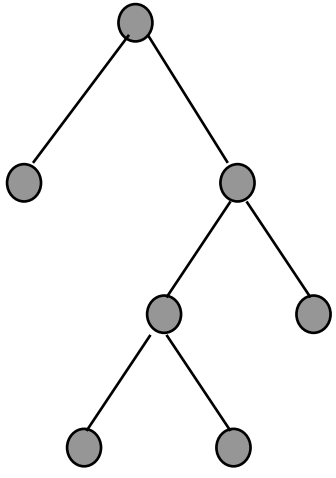


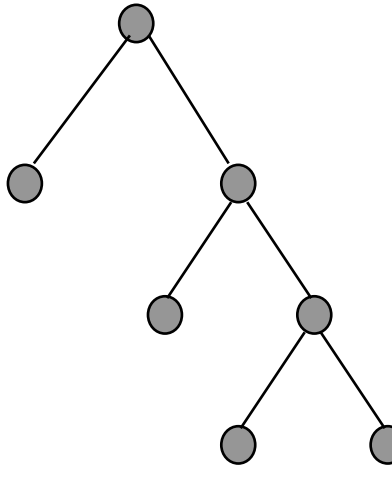


QUEUING-FN at-front



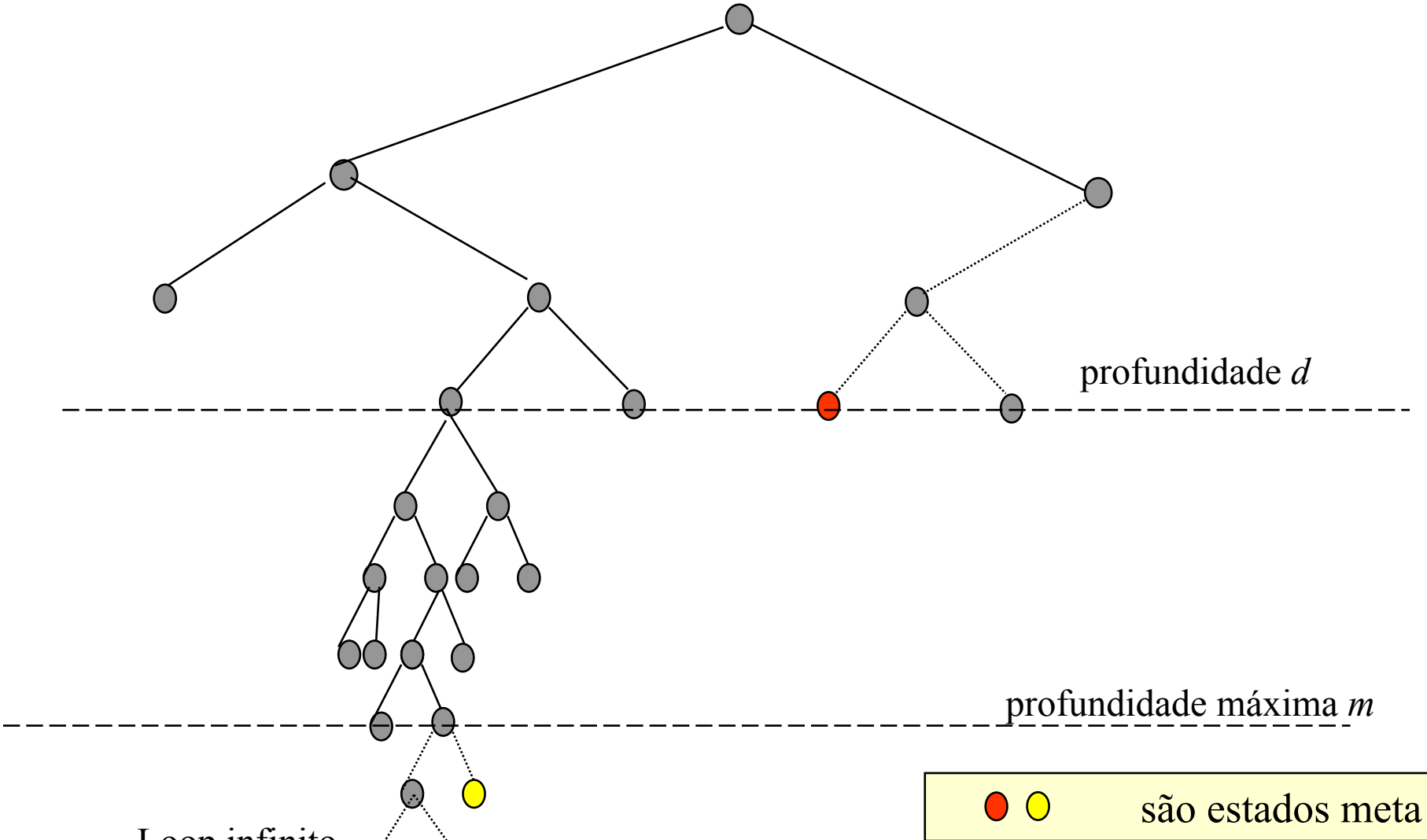




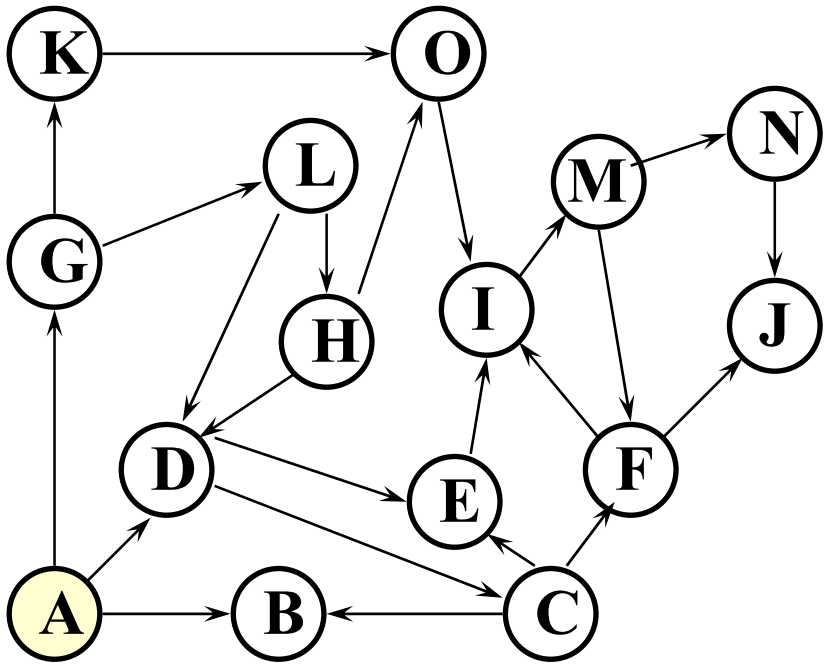


Expande os nós de profundidade maior / b*m nós armazenados

Busca em profundidade



Busca em Largura



Algoritmo de busca em profundidade

função Busca-Geral(*problema*, *estratégia*) **devolve** uma solução, ou falha

inicializa a árvore de busca com o estado inicial do *problema*

laço faça

se não há mais candidatos para expandir **então devolve** falha

escolha um nó folha para expandir

se o nó contém um estado meta **então devolve** a solução

senão expanda o nó de acordo com a *estratégia*

fim

Obs: *estratégia* = pilha (LILO)

Propriedades da busca em profundidade

- **Completa?**
- **Tempo?**
- **Espaço?**
- **Solução ótima?**

Propriedades da busca em profundidade

- **Completa?** Não. Falha em espaços com profundidade infinita, ou com loops. Se modificado para evitar estados repetidos *é completo em espaço finitos*
- **Tempo?** $O(b^m)$*:* ruim se m é muito maior que d
- **Espaço?** $O(bm)$ linear com m
- **Solução ótima?** Não. Devolve a primeira solução encontrada

DFS pode realizar ciclos infinitos. Requer um espaço de busca finito e não cíclico (verificação de estados repetidos)

Busca em profundidade (DFS)

- Expande os nós de profundidade maior
  - requer pouca memória (*bm versus* b^m)

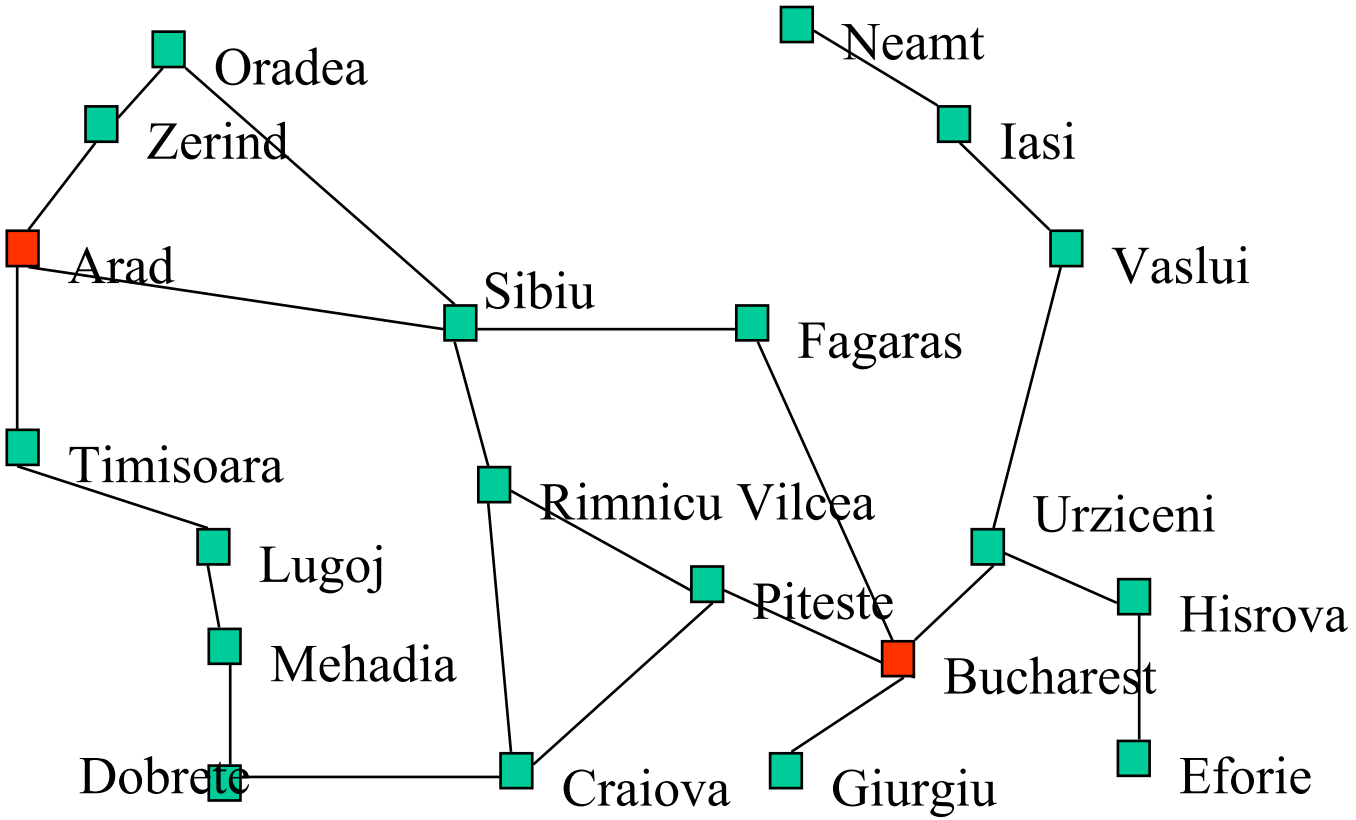
Busca	**Prof.**	**Nós**	**Memória**
BFS	**12**	**10^{12}**	**111 Tbytes**
DFS	**12**	**120**	**12 Kbytes**

  - não garante encontrar a solução ótima.
  - não recomendado para grandes árvores de busca ou quando a profundidade máxima é desconhecida.

Busca em profundidade limitada

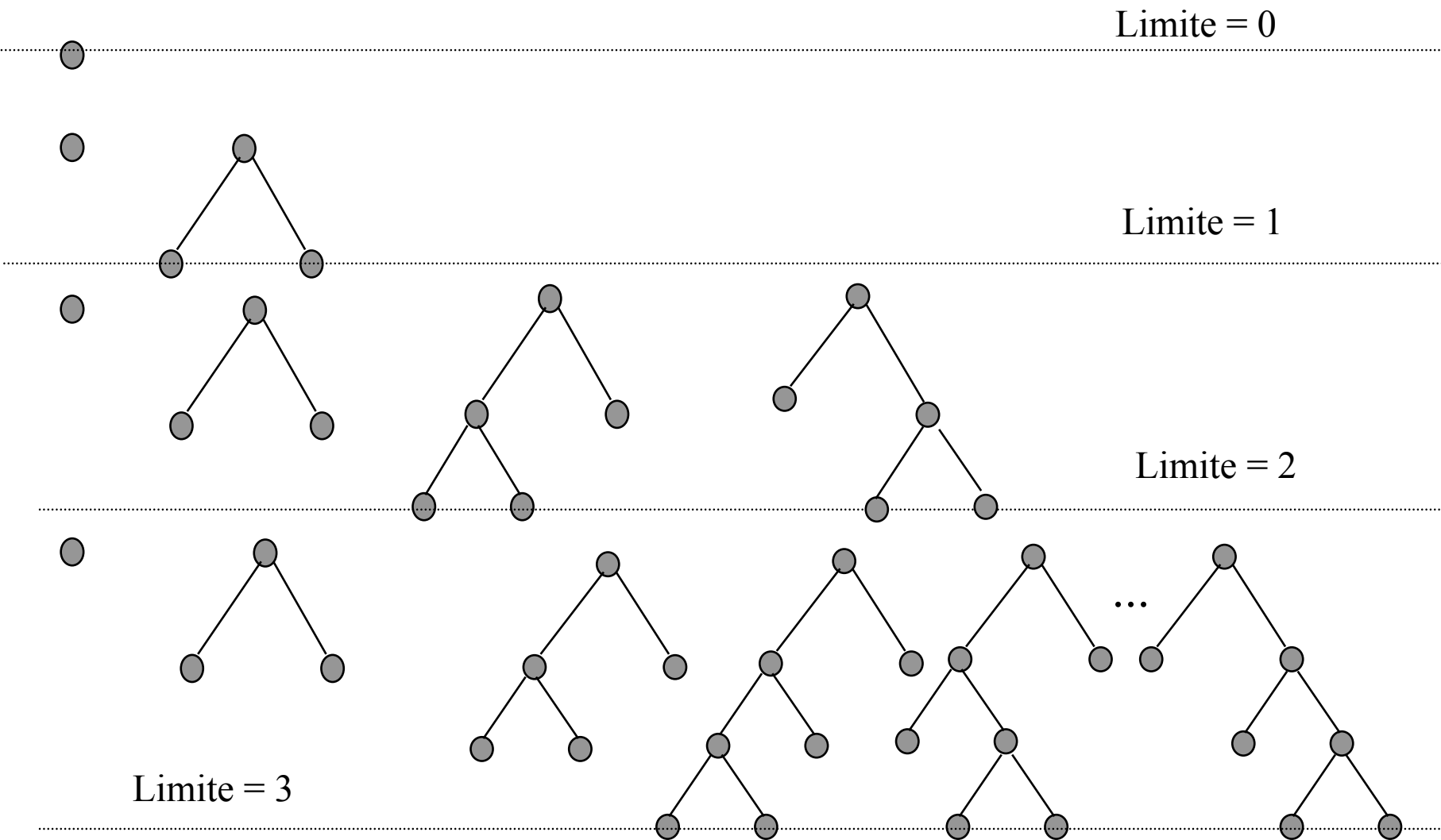
- Exemplo de roteamento no mapa da Romenia: caminho máximo: passar pelas 20 cidades
- Mesma complexidade da DFS
- **Tempo?** $O(b^l)$
- **Espaço?** $O(bl)$

Busca em profundidade limitada



Profundidade máxima para 20 cidades: 19

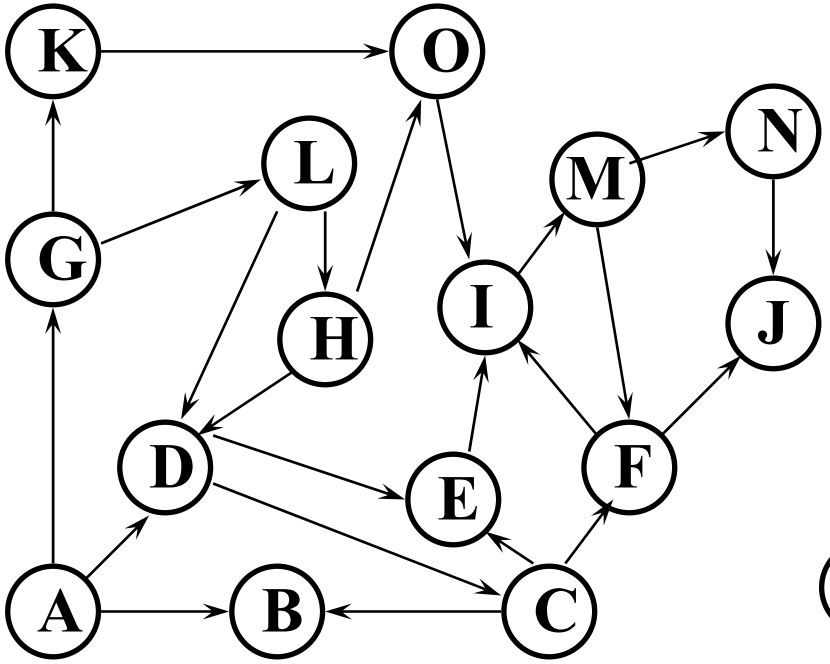
Busca em profundidade iterativa

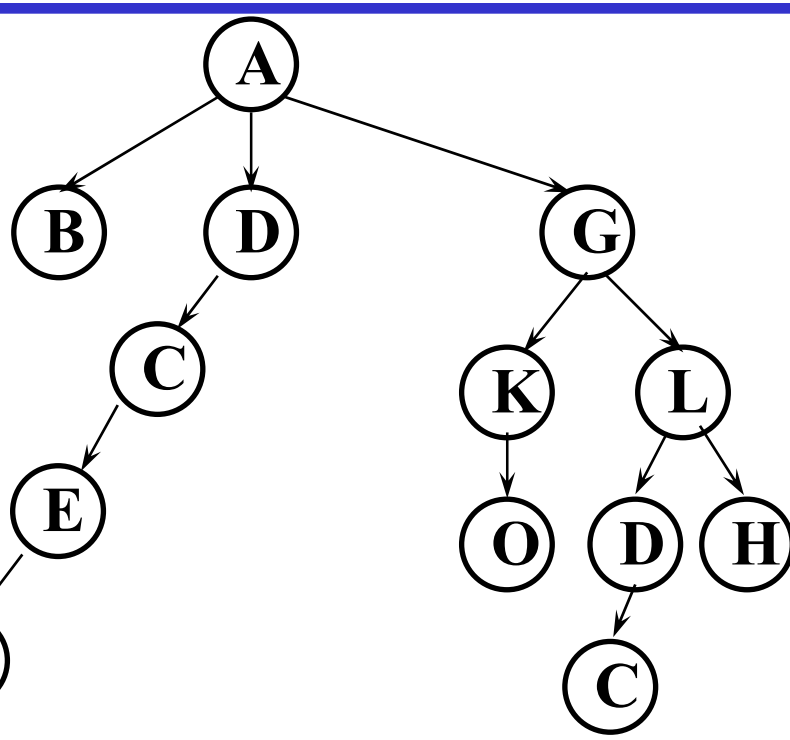


Propriedades da busca em profundidade iterativa

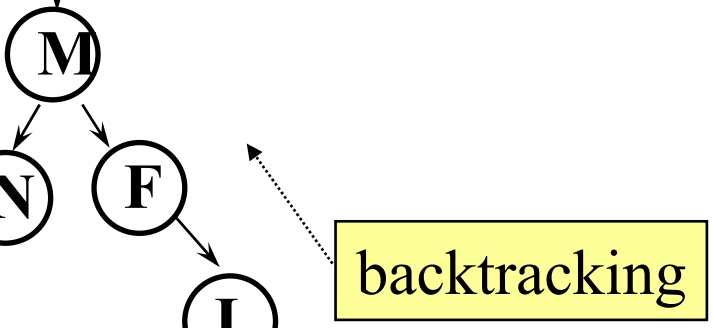
- **Completa?** Sim
- **Tempo?** $O(b^d)$
- **Espaço?** $O(bd)$
- **Solução ótima?** Sim, se custo = 1 (por passo).

Busca em profundidade





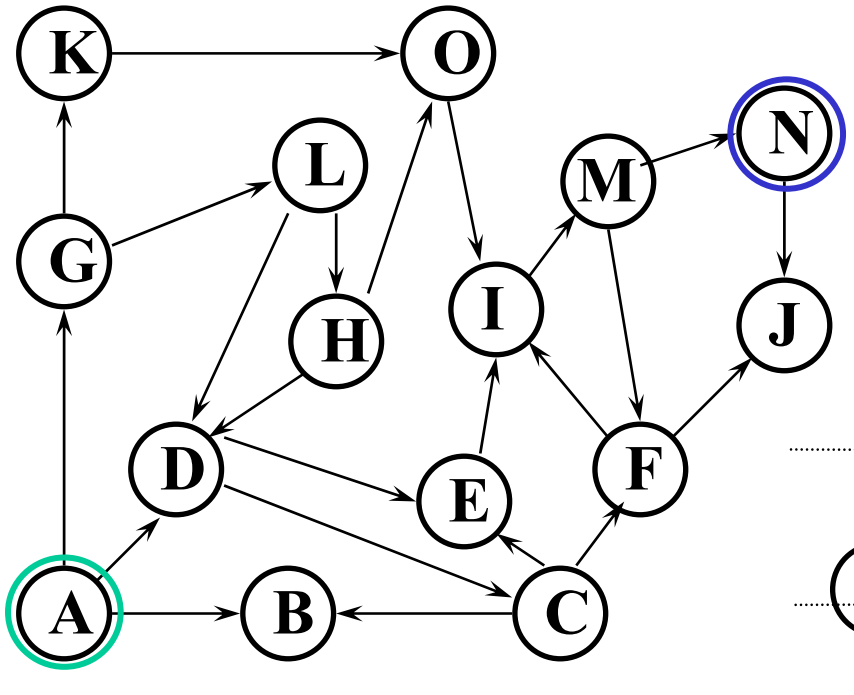
AB DCEIMNJ FI F MNJ FI ...

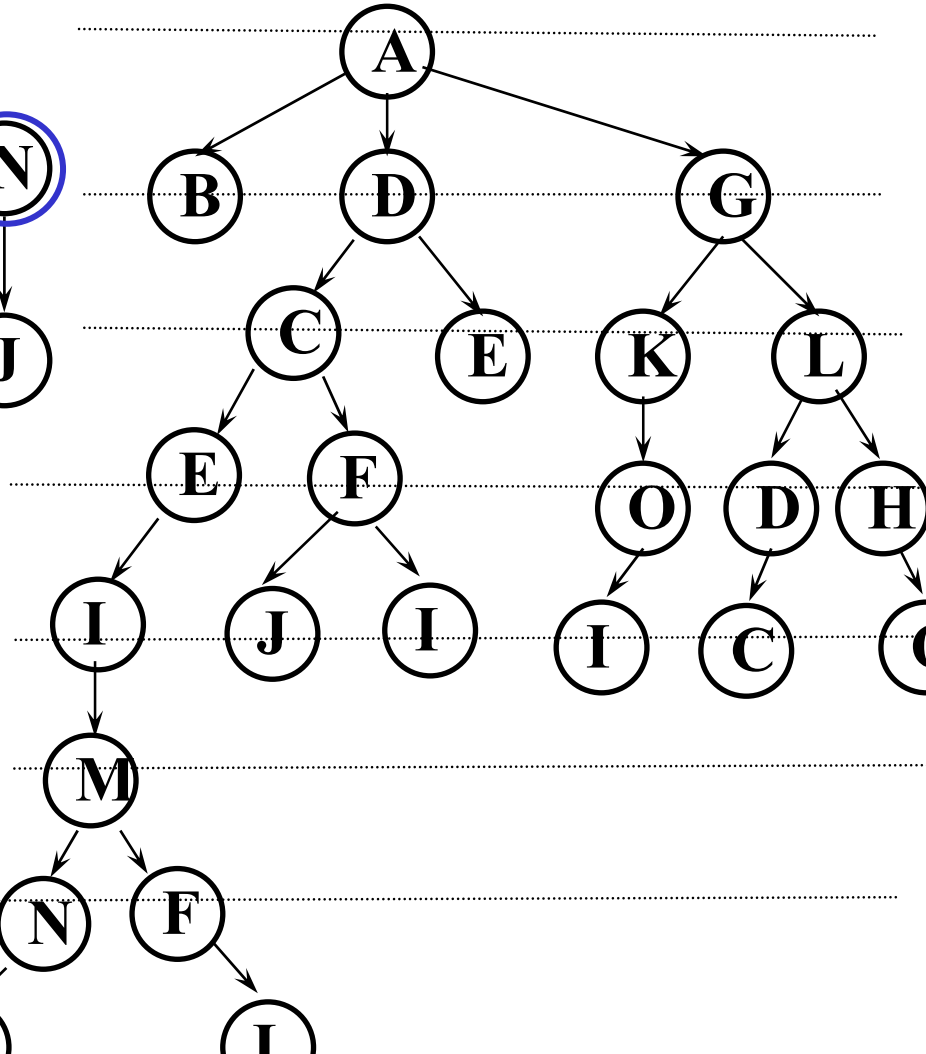


Evitando nós repetidos:

A BDG C E I M NF JI KL O DH C...

Busca em profundidade iterativa





A

A B D G

A B D C E G K L

A B D C E F E I G K O L D H

Direções de busca

- **Encadeamento para frente** *(Forward chaining)*:
  - começa do estado inicial até encontrar o estado meta.
  - gera sucessores
- **Encadeamento para trás** *(Backward chaining)*:
  - começa do estado meta até encontrar o estado inicial do mundo (só é possível se o estado meta for completamente conhecido).
  - gera precessores (operadores reversíveis ?)
- **Biderecional ...**

A escolha depende da estrutura do espaço de problema

Exemplos de direções:

- 8-puzzle: estado inicial e estado meta são conhecidos
- 8-rainhas, xadrez: o estado meta não é completamente conhecido

Busca bidirecional

- útil se o estado inicial e o estado meta são conhecidos
- Explora duas árvores busca simultaneamente: uma com o estado inicial como raiz e outra com o estado meta, expandindo nós em ambas as direções com a esperança de se encontrarem no meio
- Alternativas:
  - Expande alternativamente a partir dos dois lados
  - Expande o lado com menor número de nós na lista

Propriedades da busca biderecional

- **Completa?** Sim
- **Tempo?** $O(b^{d/2})$
- **Espaço?** $O(b^{d/2})$ (guarda todos os nós na memória)
- **Solução ótima?** Sim