

# Inteligência Artificial

## Aula 04 – Algoritmos de Busca Cega

Ms Gustavo Molina  
<msc.gustavo.unip@gmail.com>

- **Resolução de Problemas** =  
descrição formal de problemas  
+  
uso de uma estratégia de controle da busca, que  
leve de um estado inicial a um estado-objetivo.
- Busca:
  - é um **mecanismo geral**;
  - um **método fraco**;
  - em oposição a **métodos baseados em conhecimento** sobre a solução do problema.

# Estratégias de Controle da Busca

- A busca se dá construindo uma **árvore de busca**:
  - Saindo de um dos estados iniciais e
  - Aplicando uma seqüência de operadores,
  - Construa um ramo da árvore que conduza a um estado-objetivo.
- A questão:
  - **Que estratégia seguir para desenvolver a árvore de busca?**
  - A cada instante, que nó escolher para desenvolver?
- As duas principais estratégias de controle da busca que **não são guiadas por conhecimento** (i.e., cegas) são:
  - Em **largura** ou
  - Em **profundidade**. (Assunto da próxima aula)

# Busca em Largura

## • Busca em Largura (amplitude/extensão):

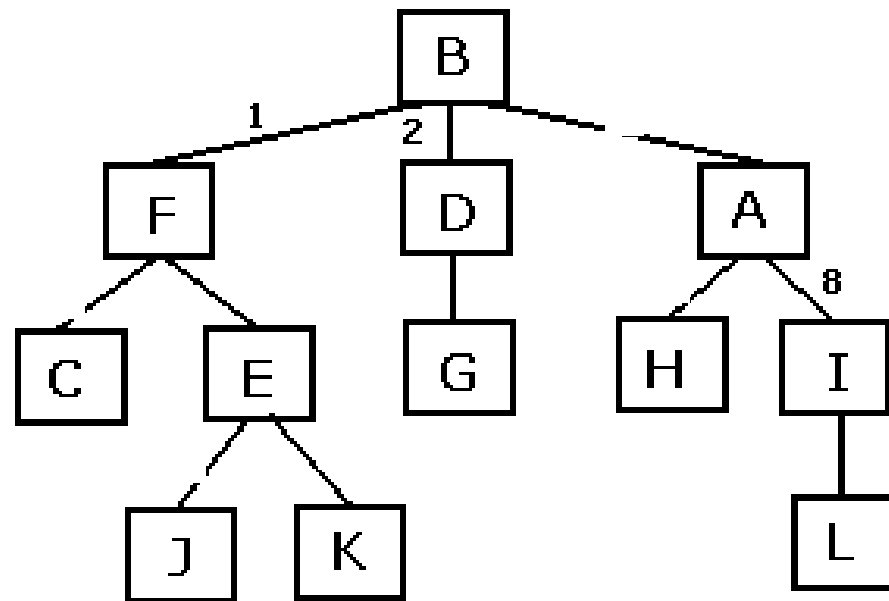
- escolha um **estado inicial** para raiz da árvore;
- para cada **nó da árvore**, recursivamente:  
    **gere todos os descendentes** deste nó,  
    aplicando todas as regras possíveis,
- até que se chegue a um **estado-objetivo** (ou se chegue a um **estado final**).

# Busca em Largura

```
lista_nós := estado_inicial;  
até que [achado_estado_objetivo = verdade ou  
        lista_nós = vazio] faça:  
  início  
    E := cabeça(lista_de_nós);  
    se [E = vazio]  
      então retorne(vazio)  
    para cada regra R que case com o estado descrito por E faça:  
      início;  
      aplique a regra R, gerando um novo estado E';  
      se estado_objetivo(E')  
        então retorne(E')  
        senão adicione_à_cauda (E', lista_nós);  
      fim;  
  fim;  
fim;
```

# Busca em Largura (*Breadth-First Search*)

- Faz uma busca sistemática examinando primeiro os nós da árvore de busca no mesmo nível de profundidade antes de explorar nós em níveis mais abaixo.
- **Exemplo:** Caminho para encontrar o nó **G**, usando a Busca em Largura: **Caminho = { B, F, D, A, C, E, G }**



# Busca em Largura (*Breadth-First Search*)

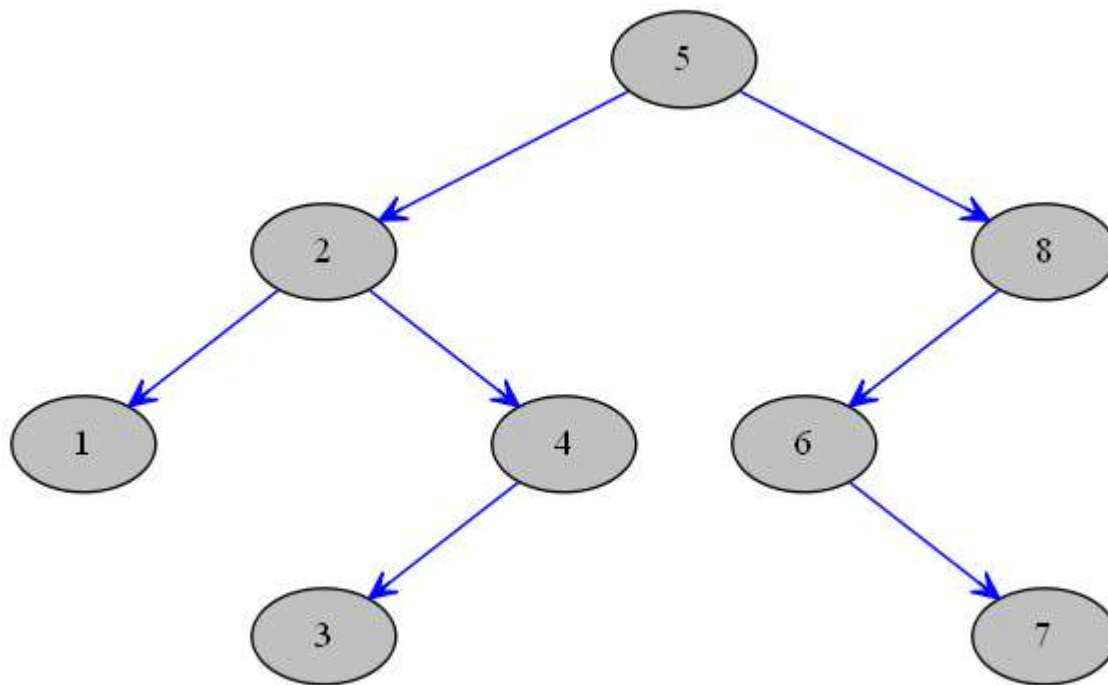
0	x	x
0		x
		0

0	x	x
0	x	x
		0

0	x	x
0		x
x		0

0	x	x
0		x
	x	0

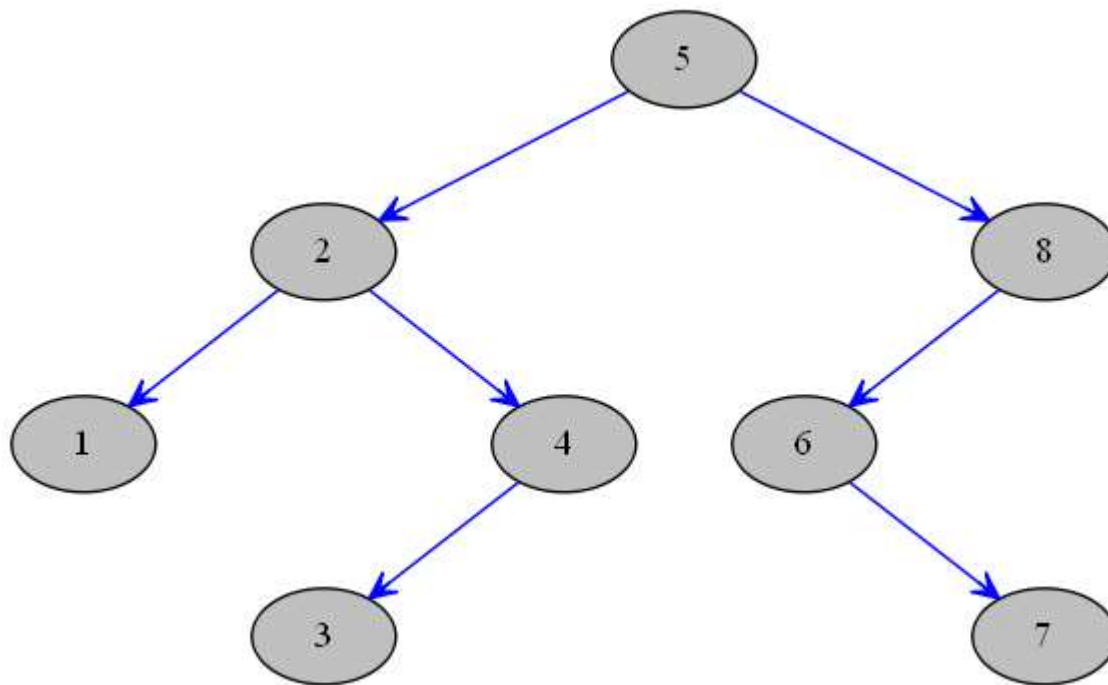
# Busca em Largura (*Breadth-First Search*)



Qual a sequência que  
será percorrida para  
chegar ao nó 7 ?



# Busca em Largura (*Breadth-First Search*)

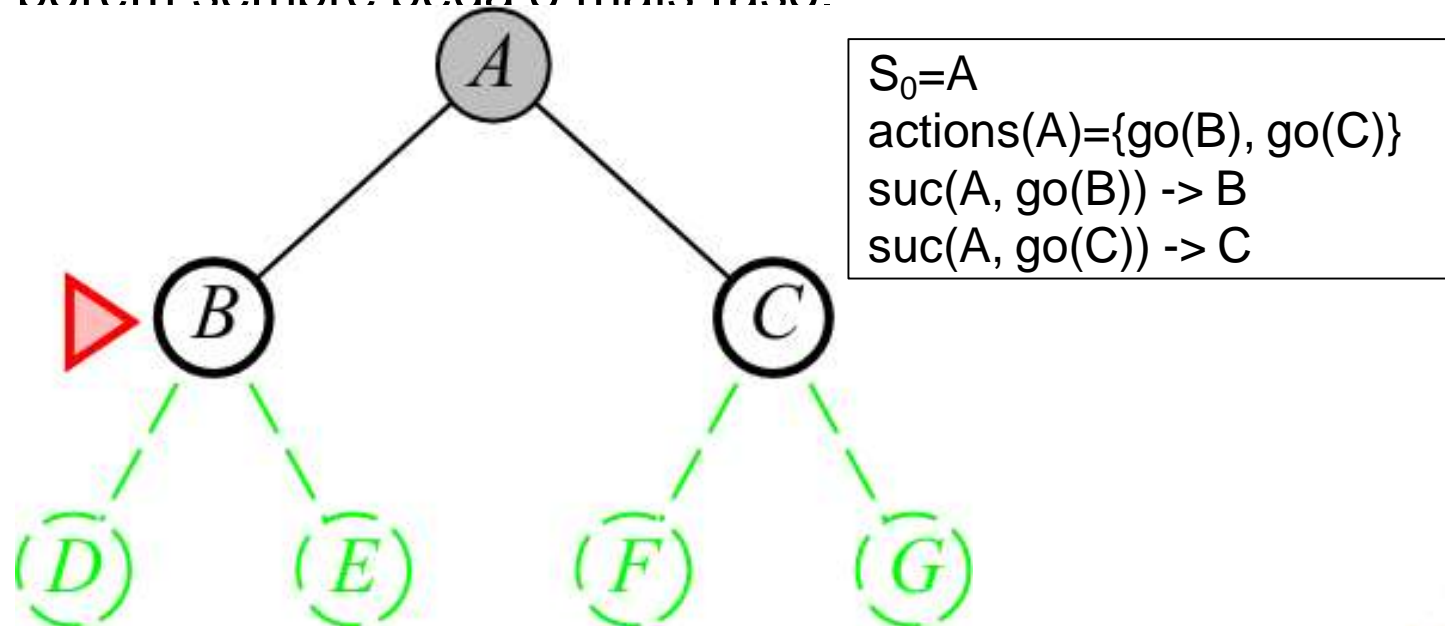


Qual a sequência que  
será percorrida para  
chegar ao nó 7 ?

**5 – 2 – 8 – 1 – 4 – 6 – 3 – 7**

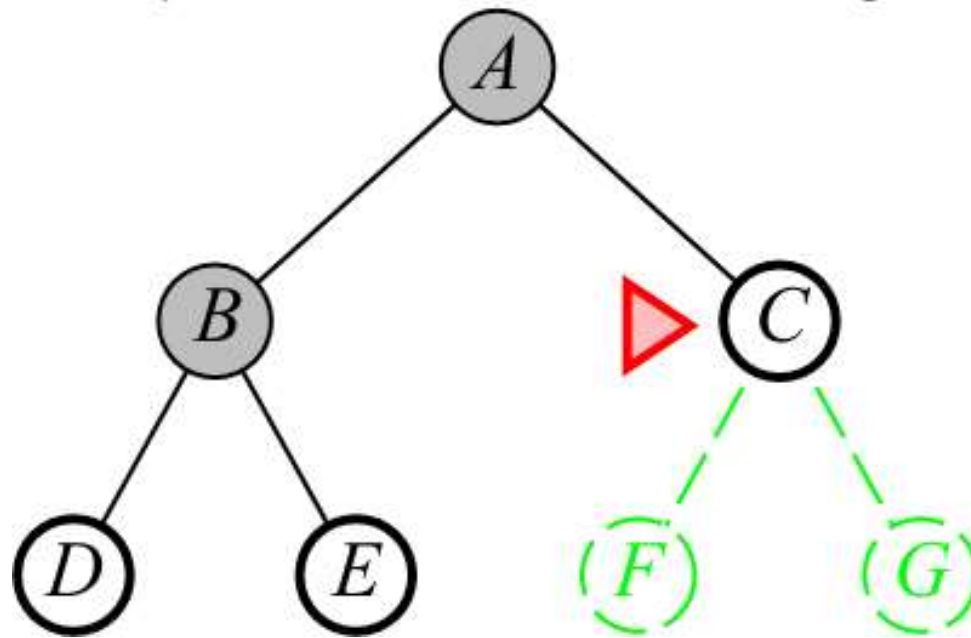
# BUSCA EM LARGURA

- Dentre os nós da fronteira, expande o **MAIS RASO** (sempre explora o caminho mais curto em qtd de ações antes)
- A fronteira é uma **FILA (FIFO)** – nós sucessores vão para o final da fila de fronteira – porém sempre pega o mais raso.



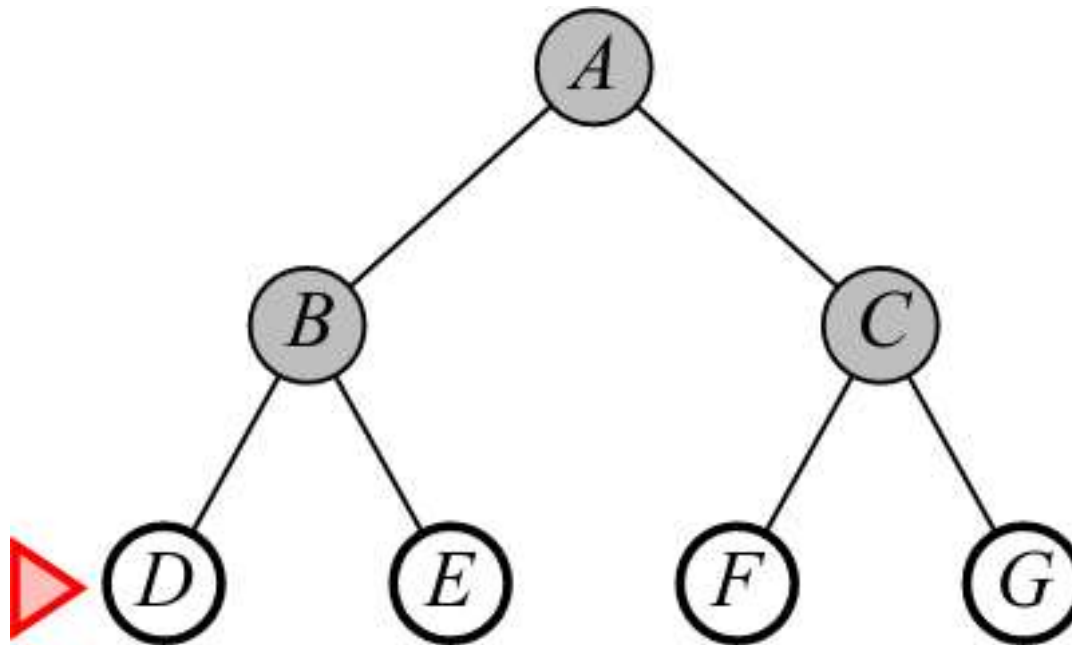
Explorados={A}  
Fronteira=B<C

# BUSCA EM LARGURA



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

# BUSCA EM LARGURA

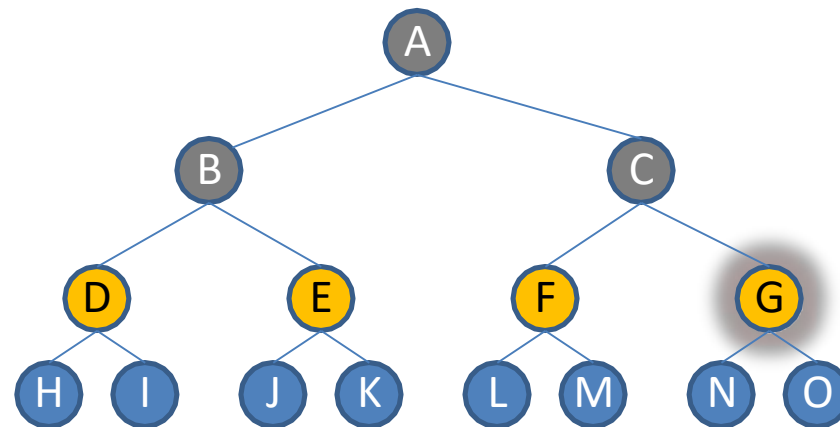


Explorados={A, B, C}

Fronteira=D<E<F<G

# Busca em largura: análise

Critério	Análise
Completo	sim, se $b$ for finito.
Ótimo	sim, se todas as ações tiverem o mesmo custo.
Complex. espaço	<b><math>O(b^{d+1})</math> – mantém todos os nos na memória</b>
Complex. tempo	<b><math>O(b^{d+1})</math></b>



# Busca em Largura (*Breadth-First Search*)

## ●●Vantagens:

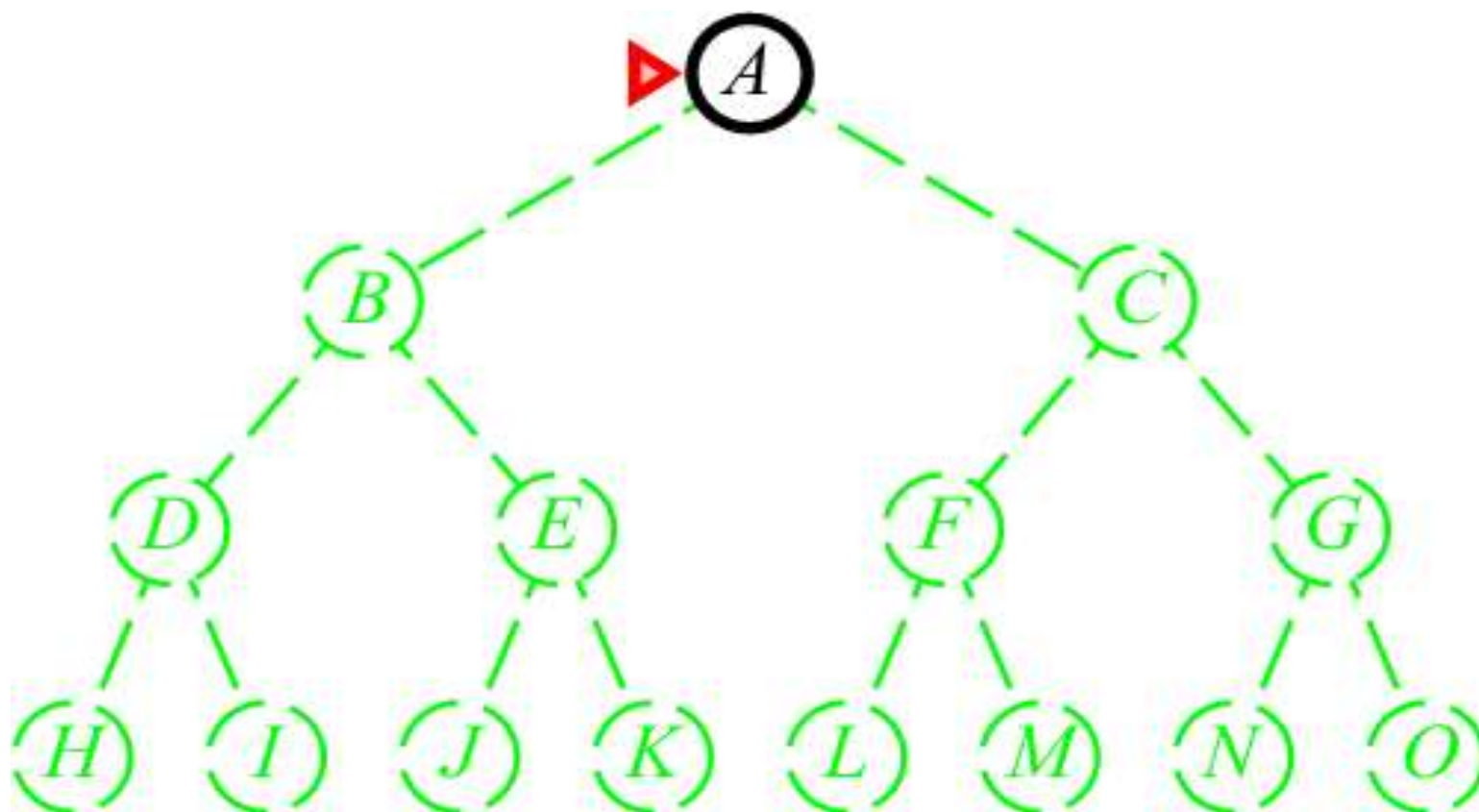
●●Se existe uma **solução** para o problema então é **garantido** que a **busca em largura** vai achar essa **solução**.

●●Além do mais, essa **solução** vai ser a **mais curta** possível.

# Busca em profundidade

- Expande o nó de **MAIOR PROFUNDIDADE** que esteja na fronteira da árvore de busca
- Fronteira = PILHA (LIFO)

# Busca em profundidade

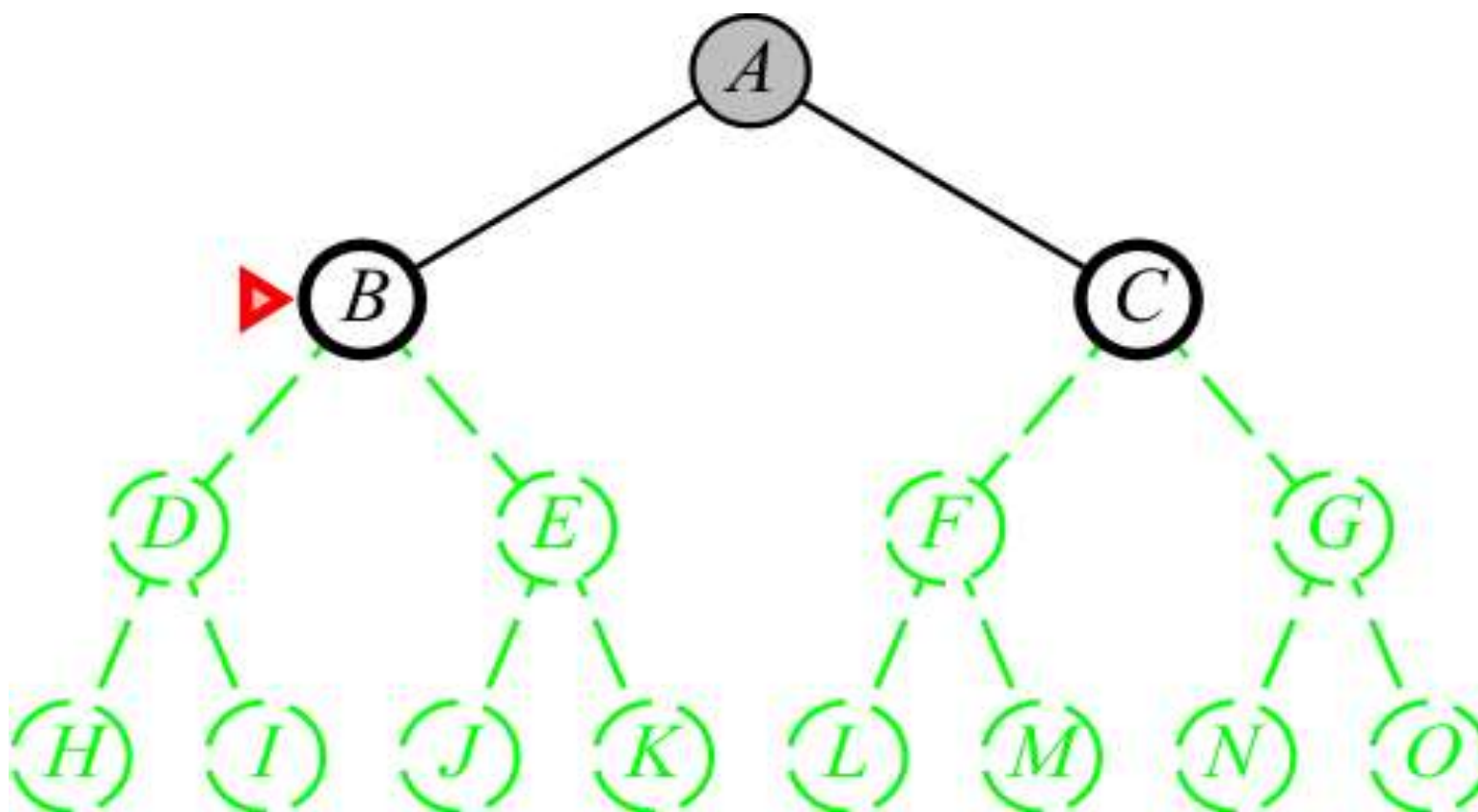


A

BASE  
FRONTEIRA



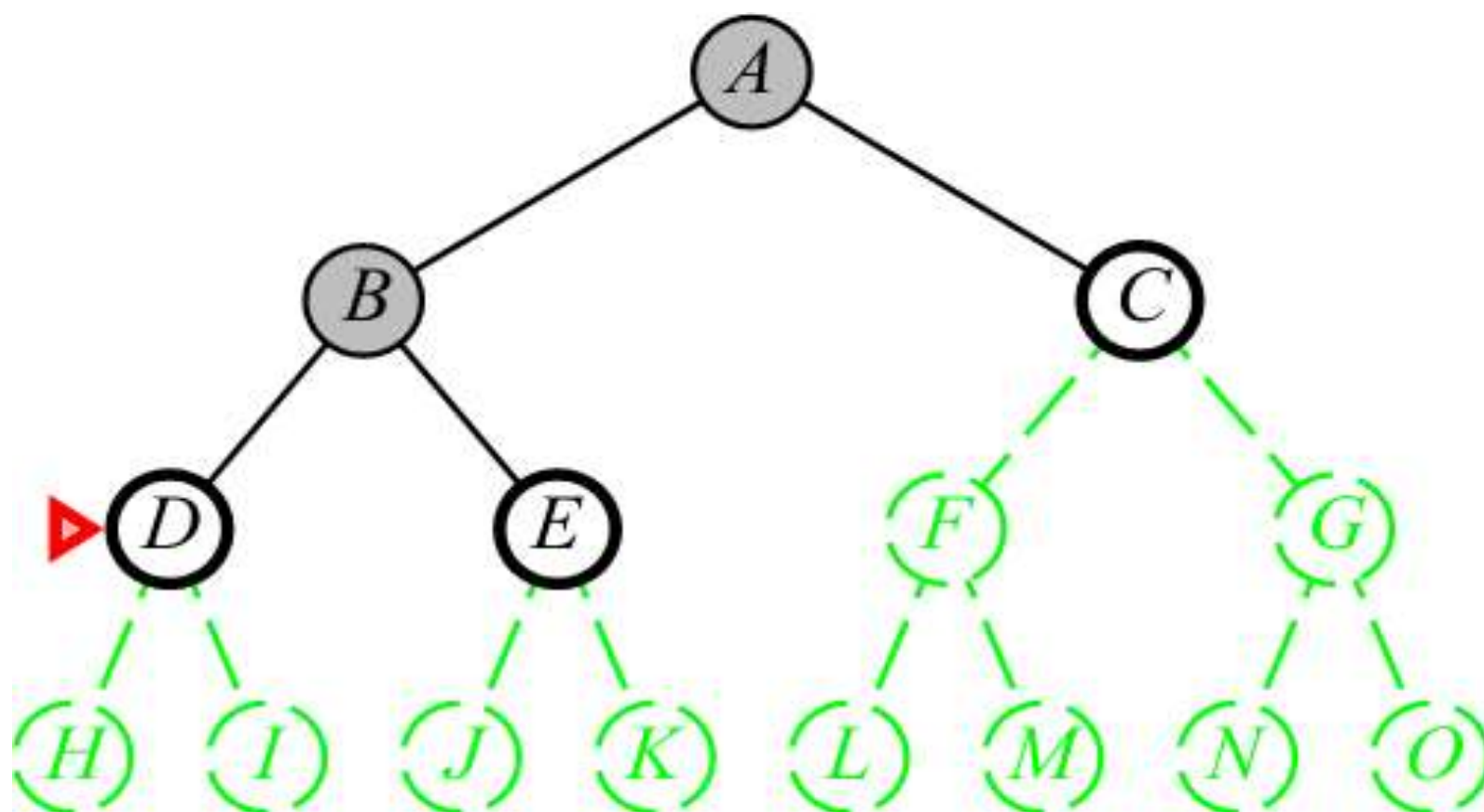
# Busca em profundidade



B
C

BASE  
FRONTEIRA

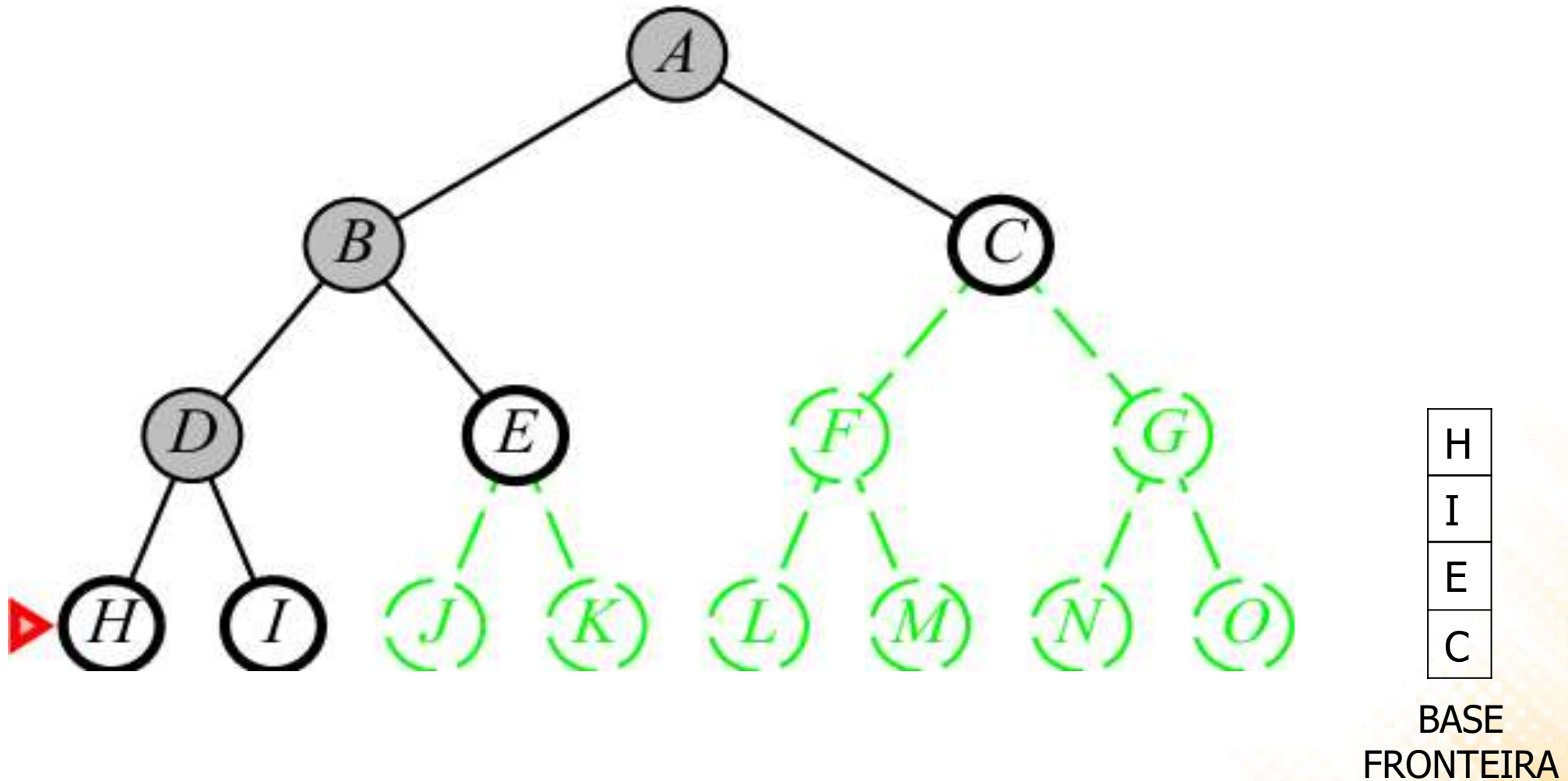
# Busca em profundidade



D
E
C

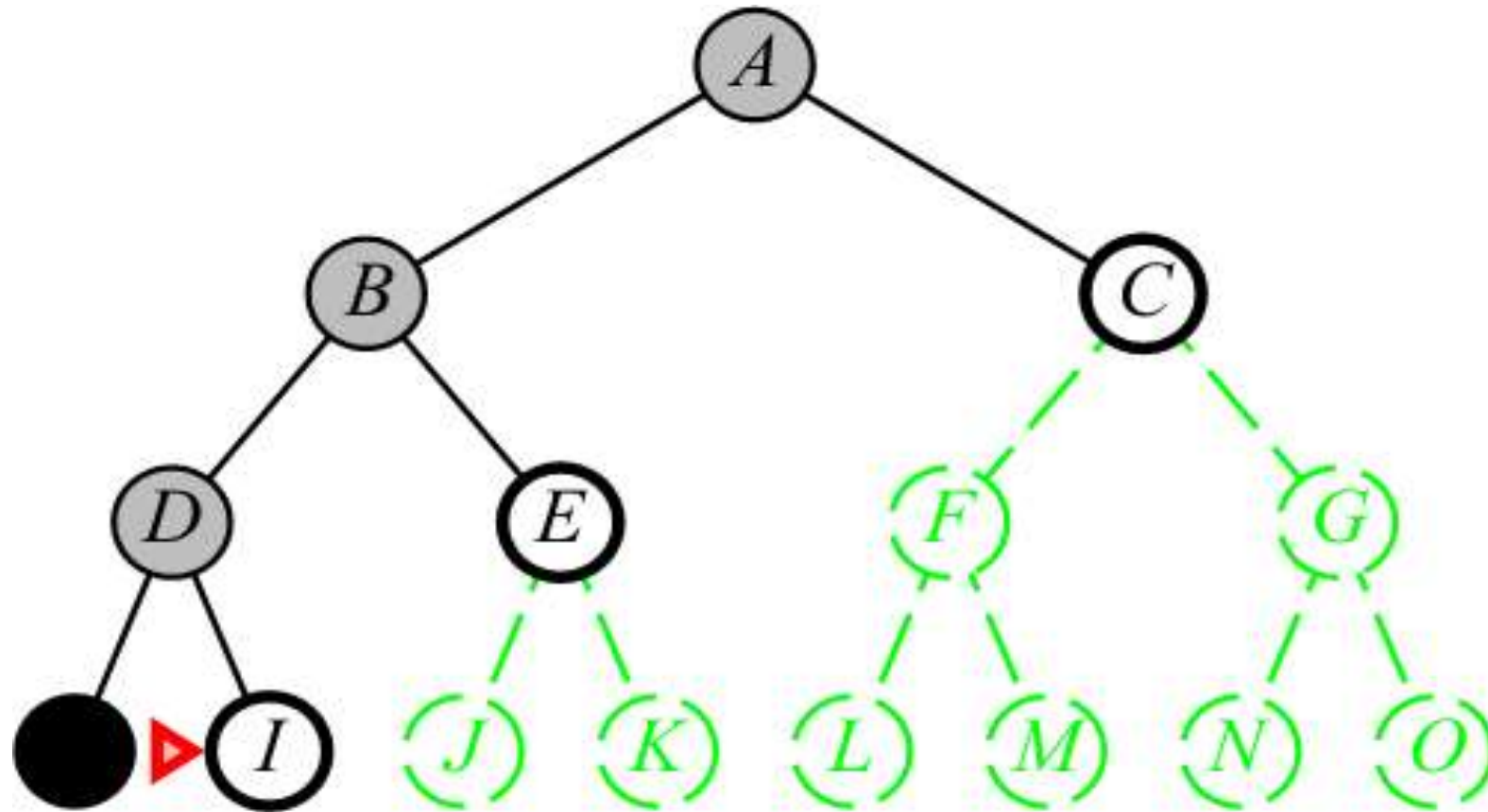
BASE  
FRONTEIRA

# Busca em profundidade



Situação de maior ocupação de memória:  
(3 nós na fronteira + 3 já explorados)  $1 + 2 + 2 + 2 = 1 + 3.2 = 1 + m.b$

# Busca em profundidade

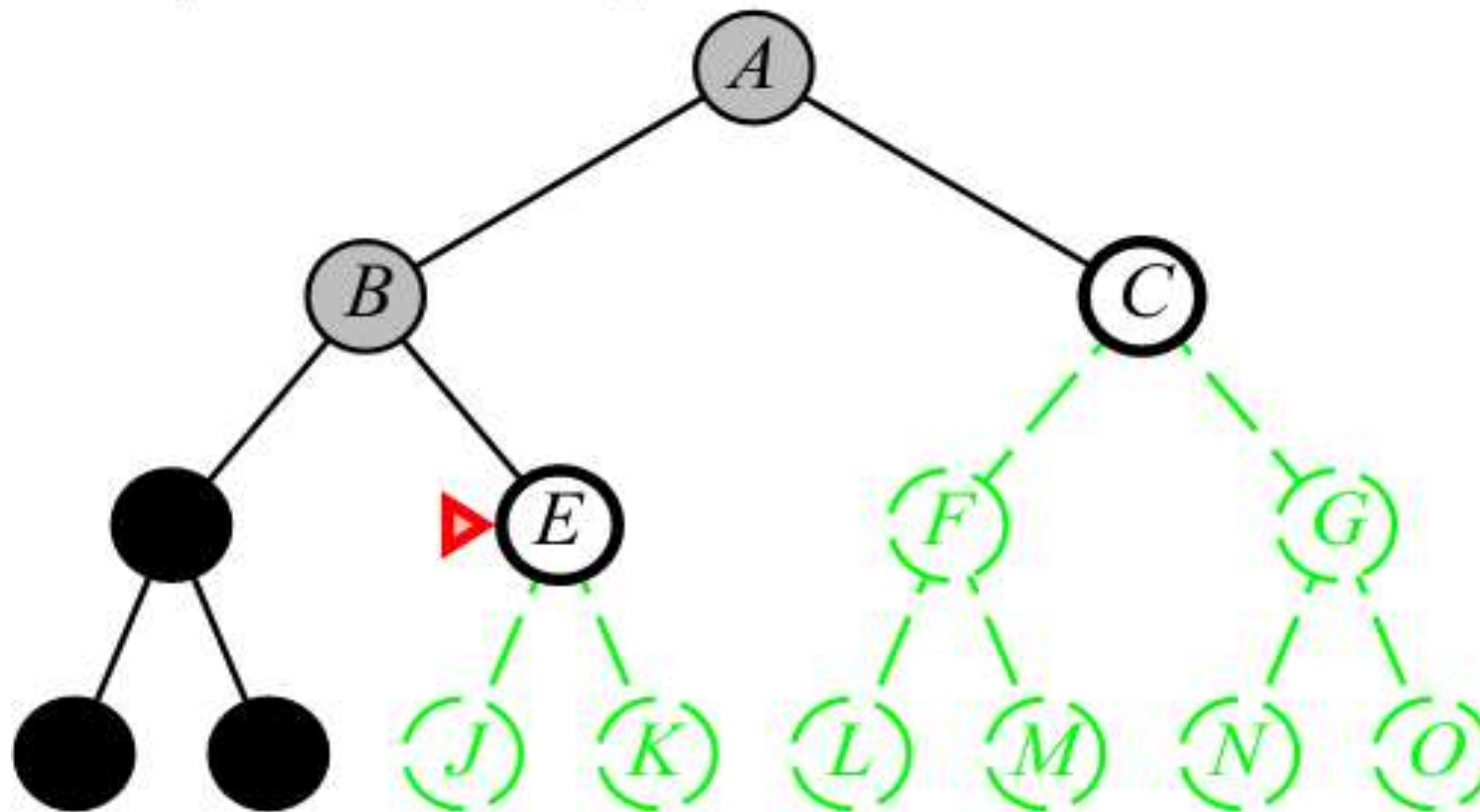


I
E
C

BASE  
FRONTEIRA

*\*nó pode ser removido da memória uma vez que todos seus descendentes tenham sido explorados (em preto na árvore)*

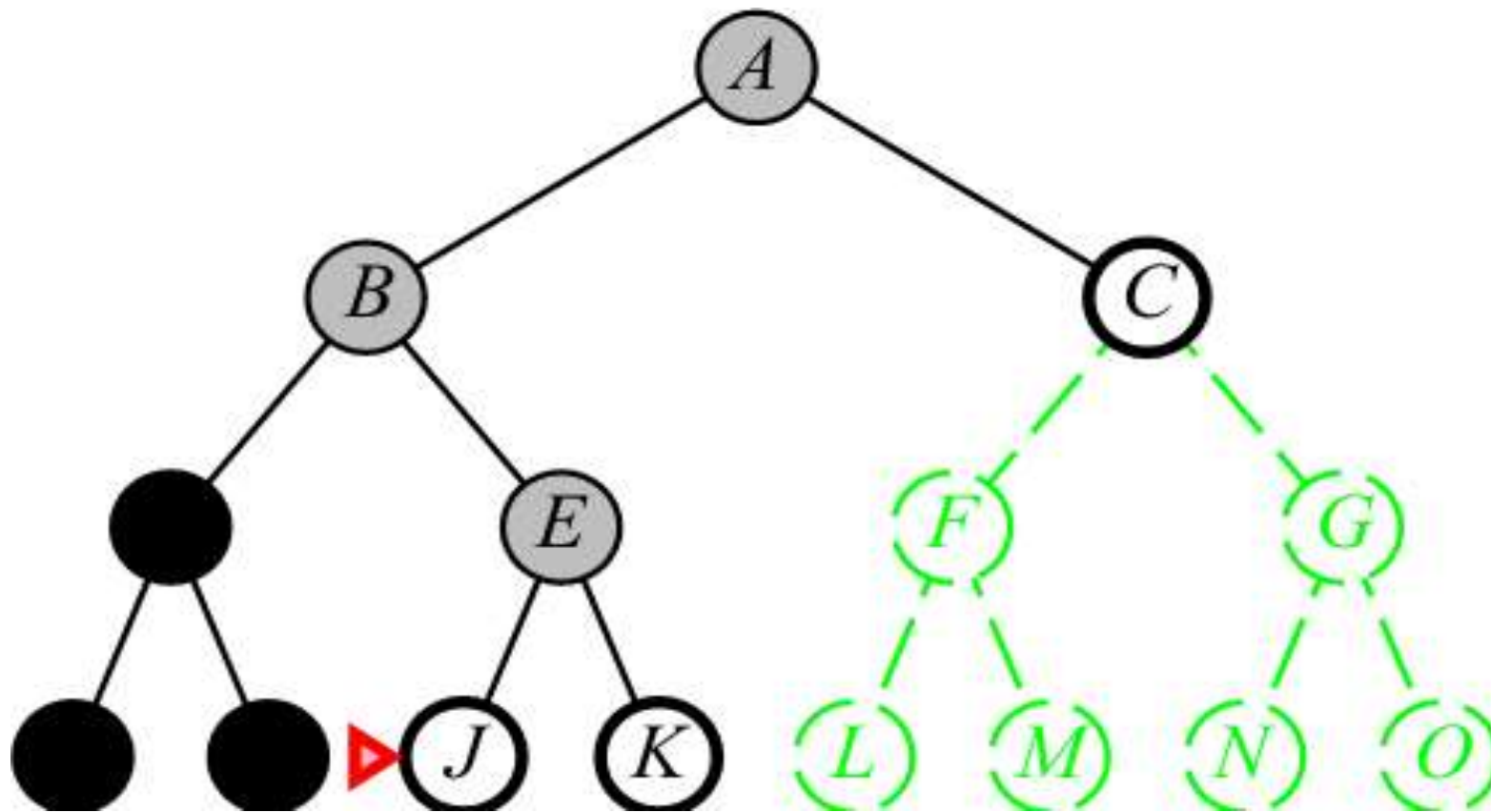
# Busca em profundidade



E
C

BASE  
FRONTEIRA

# BUSCA EM PROFUNDIDADE

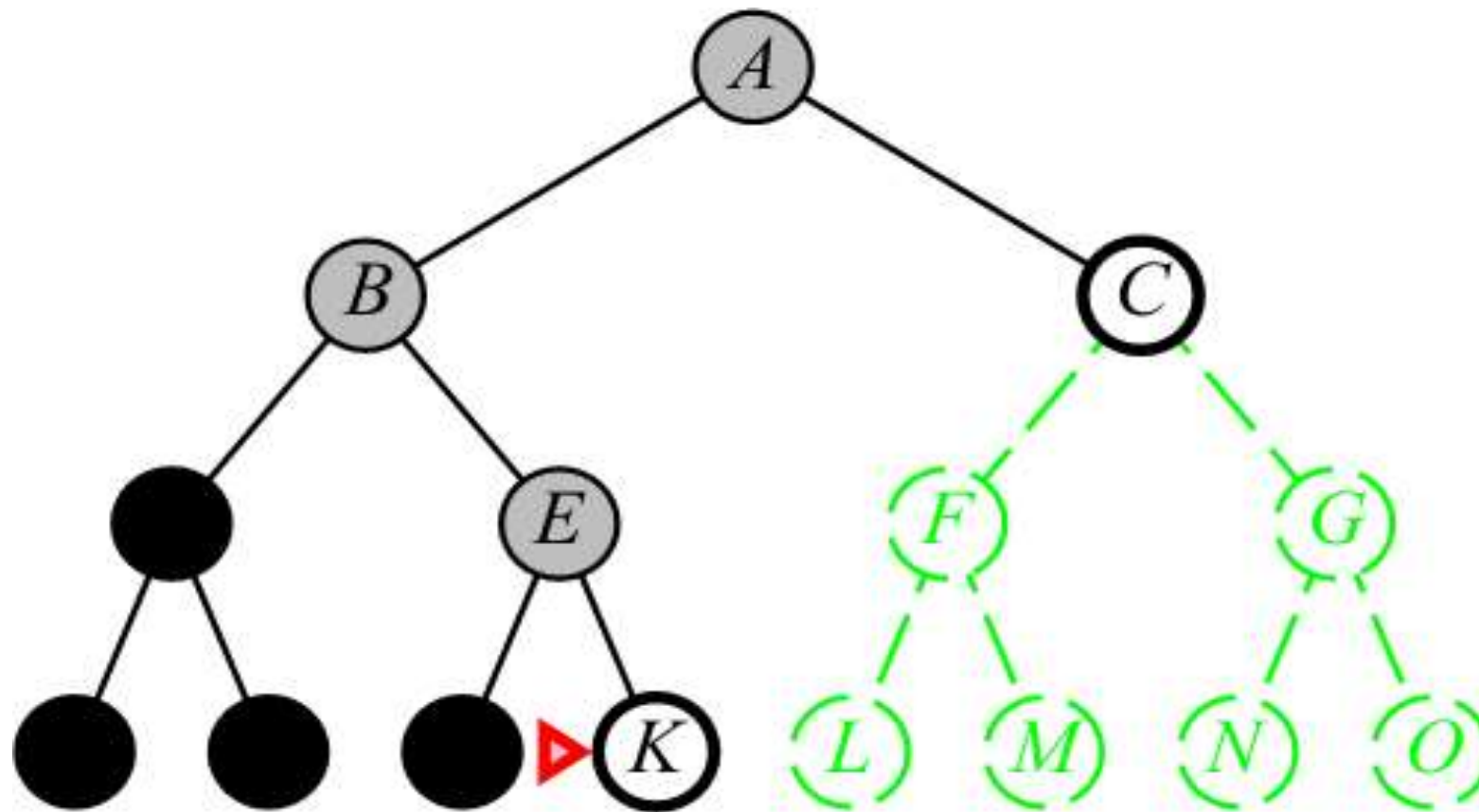


J
K
C

BASE  
FRONTEIRA



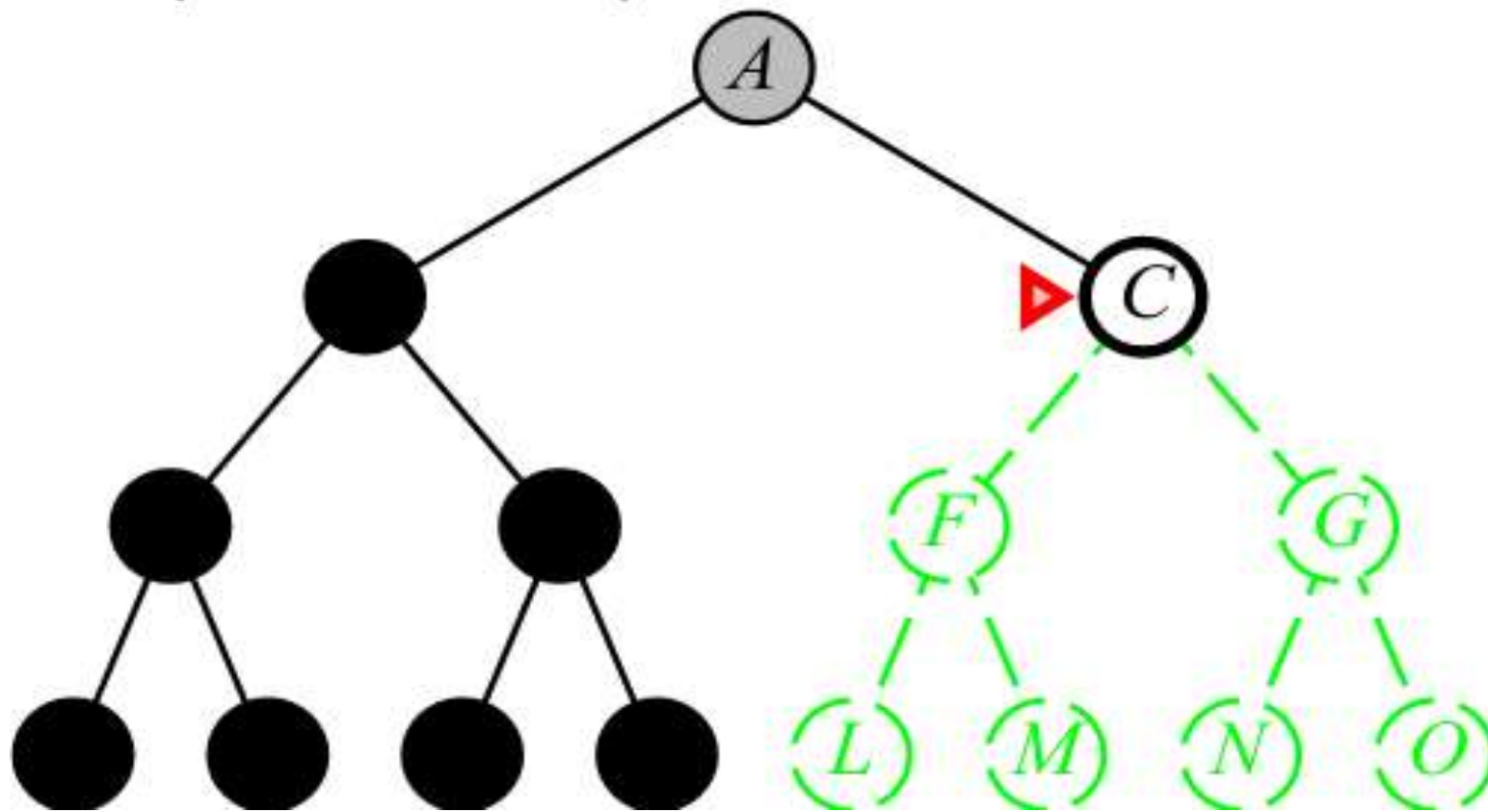
# Busca em profundidade



K
C

BASE  
FRONTEIRA

# Busca em profundidade

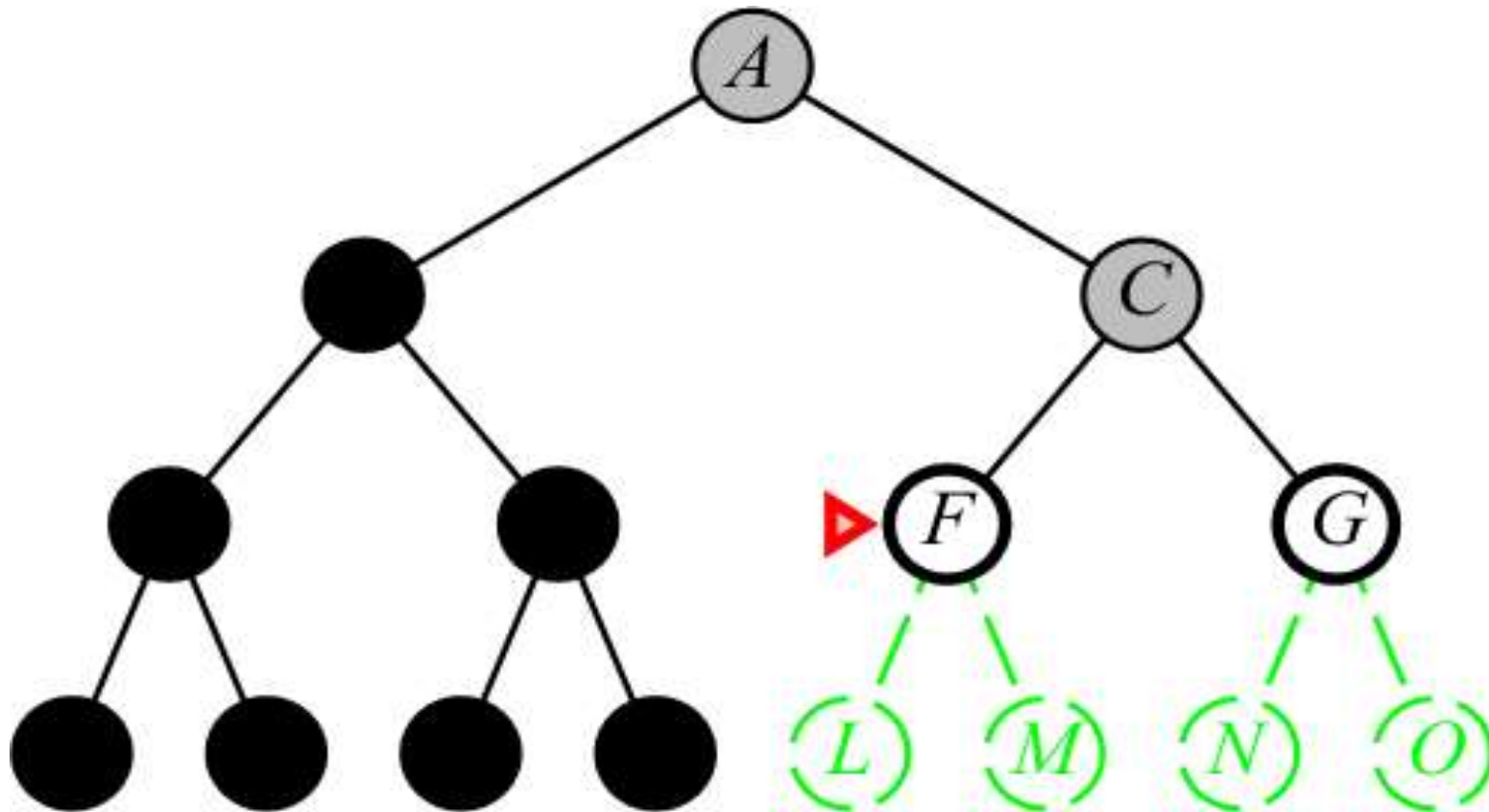


C

BASE  
FRONTEIRA



# Busca em profundidade



# Avaliação da busca em profundidade

Critério	Avaliação
Completo	Sim, se o espaço de estados for finito
Otimidade	Não, pois retorna a 1a. Solução encontrada
Espacial	$O(b.m)$
Tempo	$O(b^m)$ Se a solução estiver no primeiro ramo então é linear em relação à $m$ (pode ser infinito)


**b**: ramificação máxima entre todos os nós

**m**: tamanho máximo entre todos os caminhos no espaço de estados

# Avaliação da busca em profundidade

- A baixa complexidade espacial da busca em profundidade com busca em árvore fez com que fosse aplicado em
  - Satisfação de restrições (Constraint satisfaction)
  - Satisfabilidade em lógica Prop. (Propositional satisfiability)
  - Programação lógica (Logic programming)

# Busca em profundidade limitada

- 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
- O problema é determinar o valor de  $l$ !
  - Um modo é pegar o *tamanho máximo de caminho entre dois estados quaisquer* no espaço de estados do grafo: **16**
  - Porém, sabemos que qualquer cidade pode ser alcançada a partir de qualquer outra em no máximo 9 passos (*diâmetro do espaço de estados*): **9**
- $d$ : profundidade do nó objetivo mais raso

# Busca em profundidade limitada

- Complexidade de tempo:  $O(b^l)$
- Complexidade de espaço:  $O(b \cdot l)$

# Busca em profundidade limitada

**function** DEPTH-LIMITED-SEARCH(*problem, limit*) **returns** a solution, or failure/cutoff

**return** RECURSIVE-DLS(MAKE-NODE(*problem*.INITIAL-STATE), *problem, limit*)

**function** RECURSIVE-DLS(*node, problem, limit*) **returns** a solution, or failure/cutoff

if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)

**else if** *limit* = 0 **then return** *cutoff*

**else**

*cutoff\_occurred?*  $\leftarrow$  false

**for each** *action* in *problem*.ACTIONS(*node*.STATE) **do** *child*

$\leftarrow$  CHILD-NODE(*problem, node, action*) *result*  $\leftarrow$

RECURSIVE-DLS(*child, problem, limit-1*) **if** *result* =

*cutoff* **then** *cutoff\_occurred?*  $\leftarrow$  true **else if** *result*

$\neq$  *failure* **then return** *result*

**if** *cutoff\_occurred?* **then return** *cutoff* **else return** *failure*

# Busca de Aprofundamento Iterativo

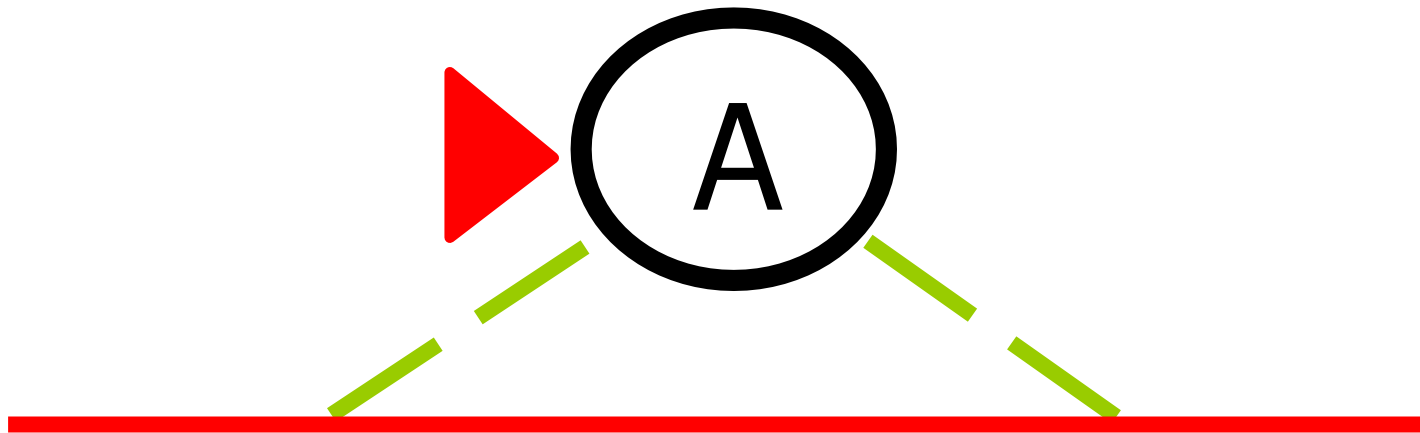
- Estratégia utilizada em conjunto com busca em profundidade para encontrar o melhor limite  $l$ 
  - aumentar gradualmente / até encontrar um estado objetivo.
- Isto ocorre quando a profundidade alcançar  $d$ 
  - (profundidade do objetivo mais raso)



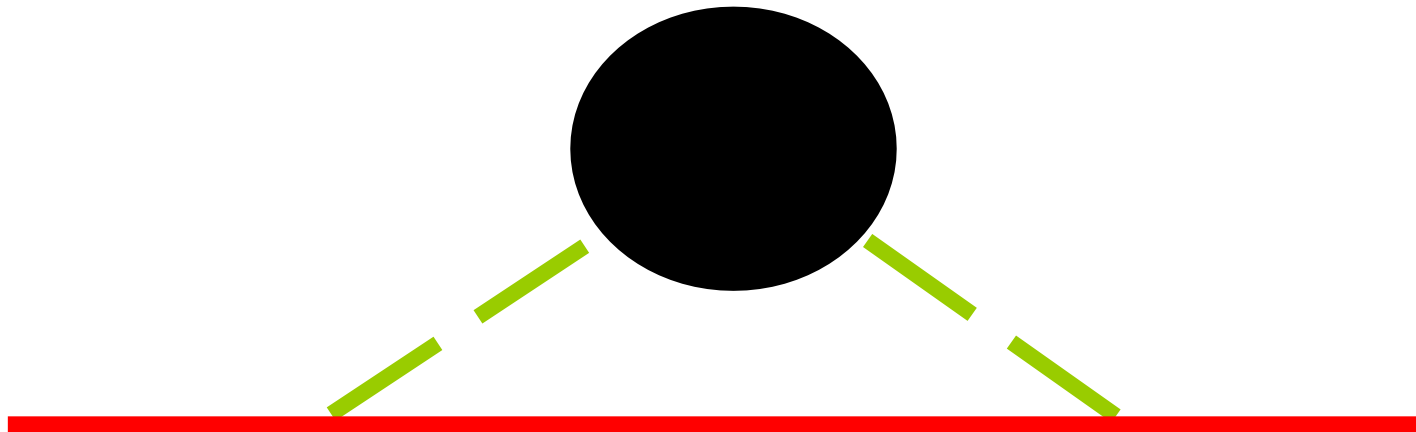
# Busca de Aprofundamento Iterativo

- Combina os benefícios da busca em largura com os benefícios da busca em profundidade.
- Evita o problema de caminhos muito longos ou infinitos.
- A repetição da expansão de estados não é tão ruim, pois a maior parte dos estados está nos níveis mais baixos.
- Cria menos estados que a busca em largura e consome menos memória.

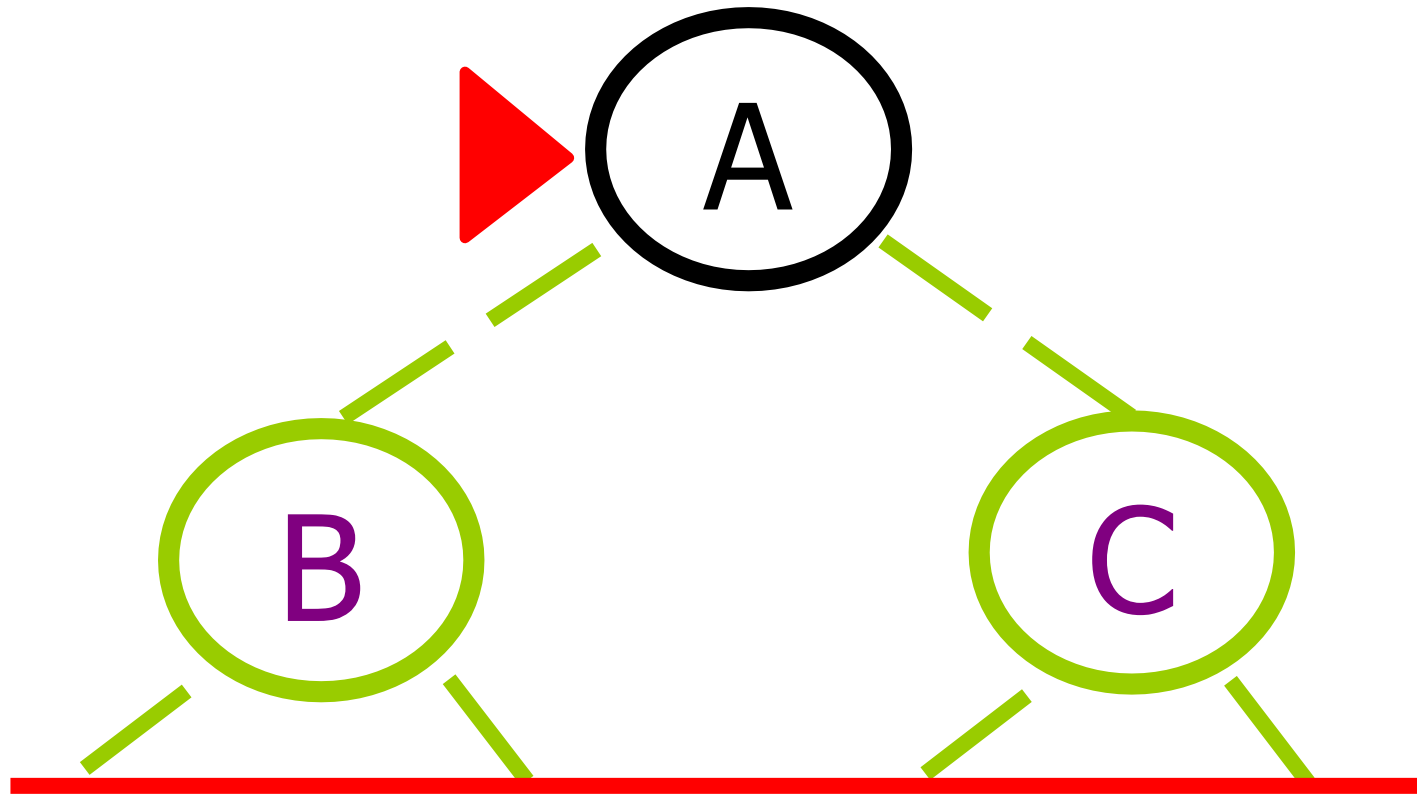
# Busca de Aprofundamento Iterativo $\ell=0$



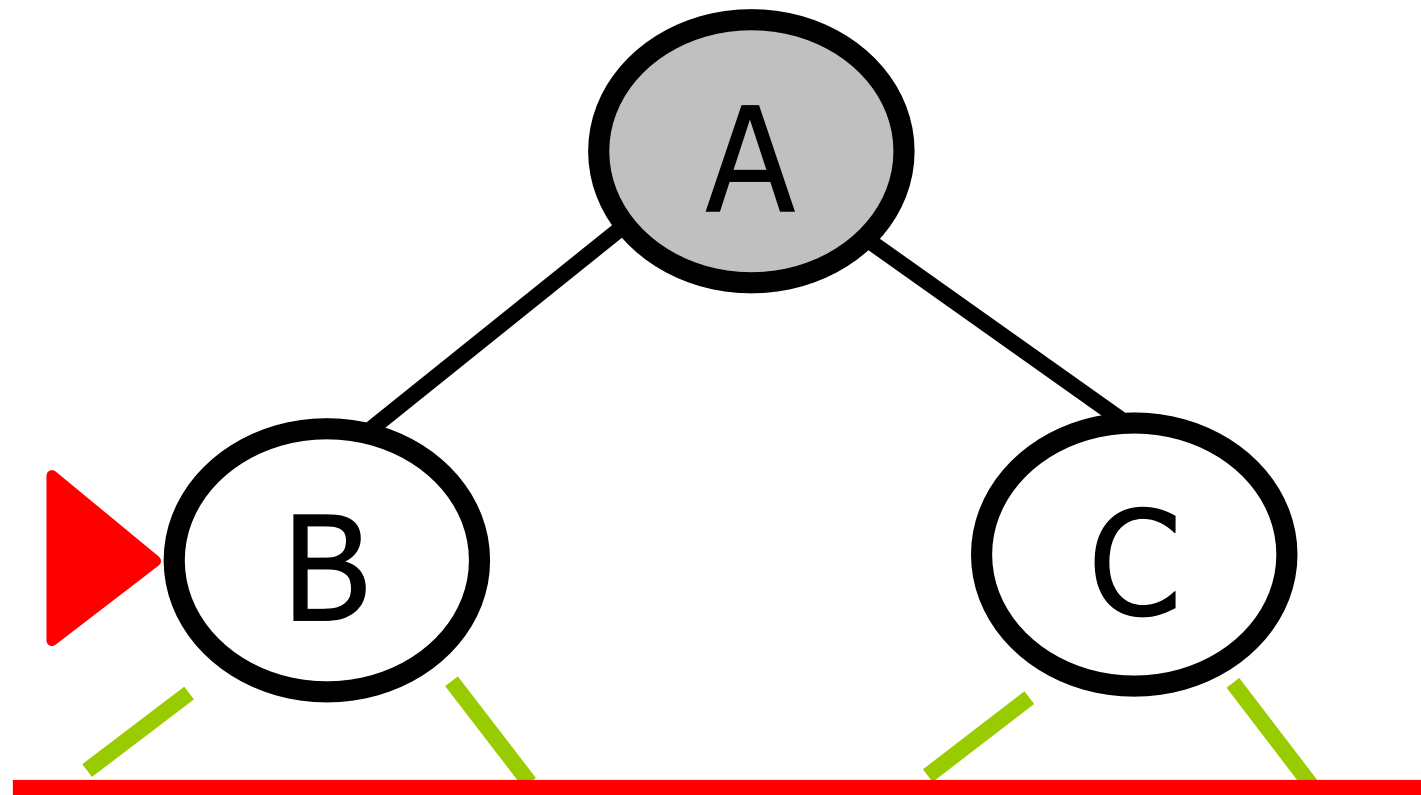
# Busca de Aprofundamento Iterativo $\ell=0$



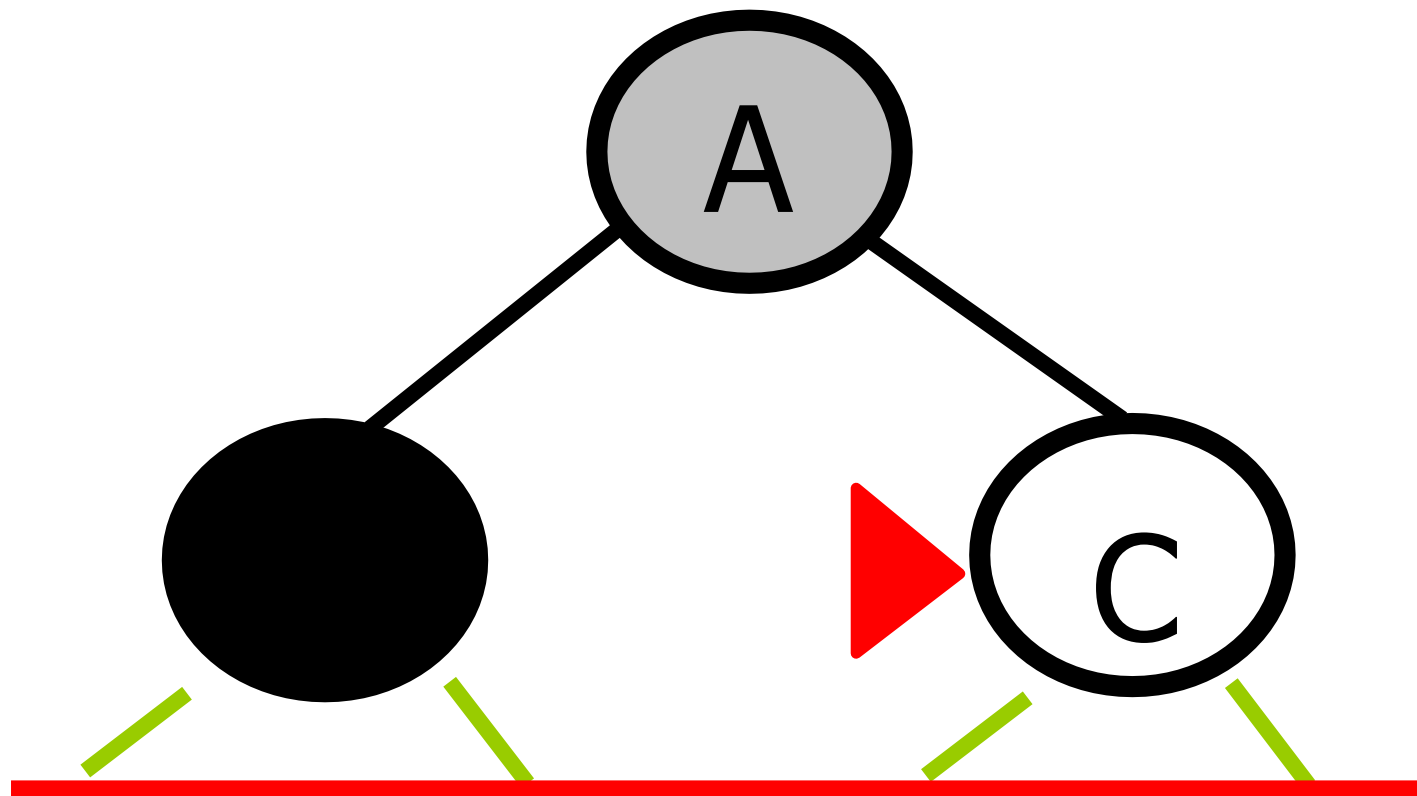
# Busca de Aprofundamento Iterativo $\ell=1$



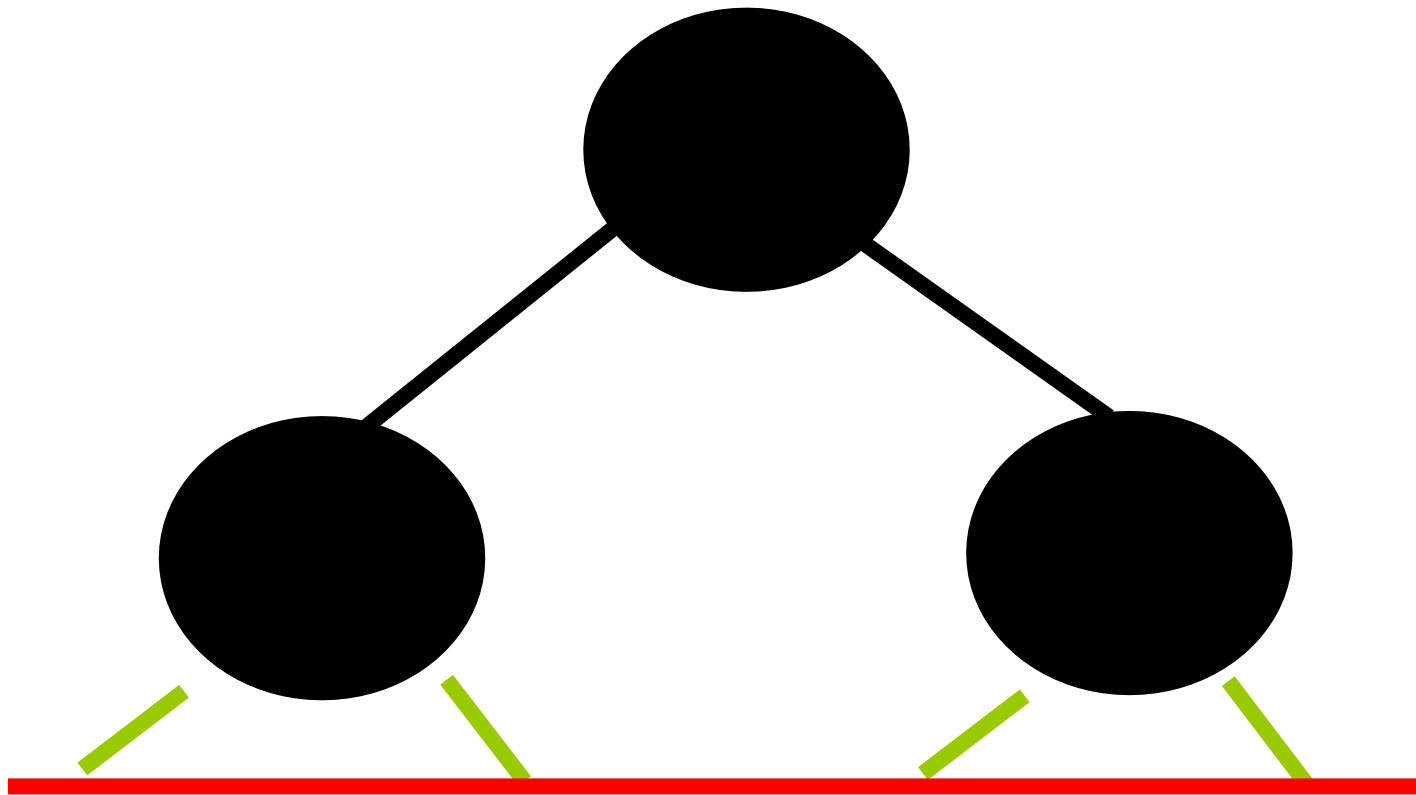
# Busca de Aprofundamento Iterativo $\ell=1$



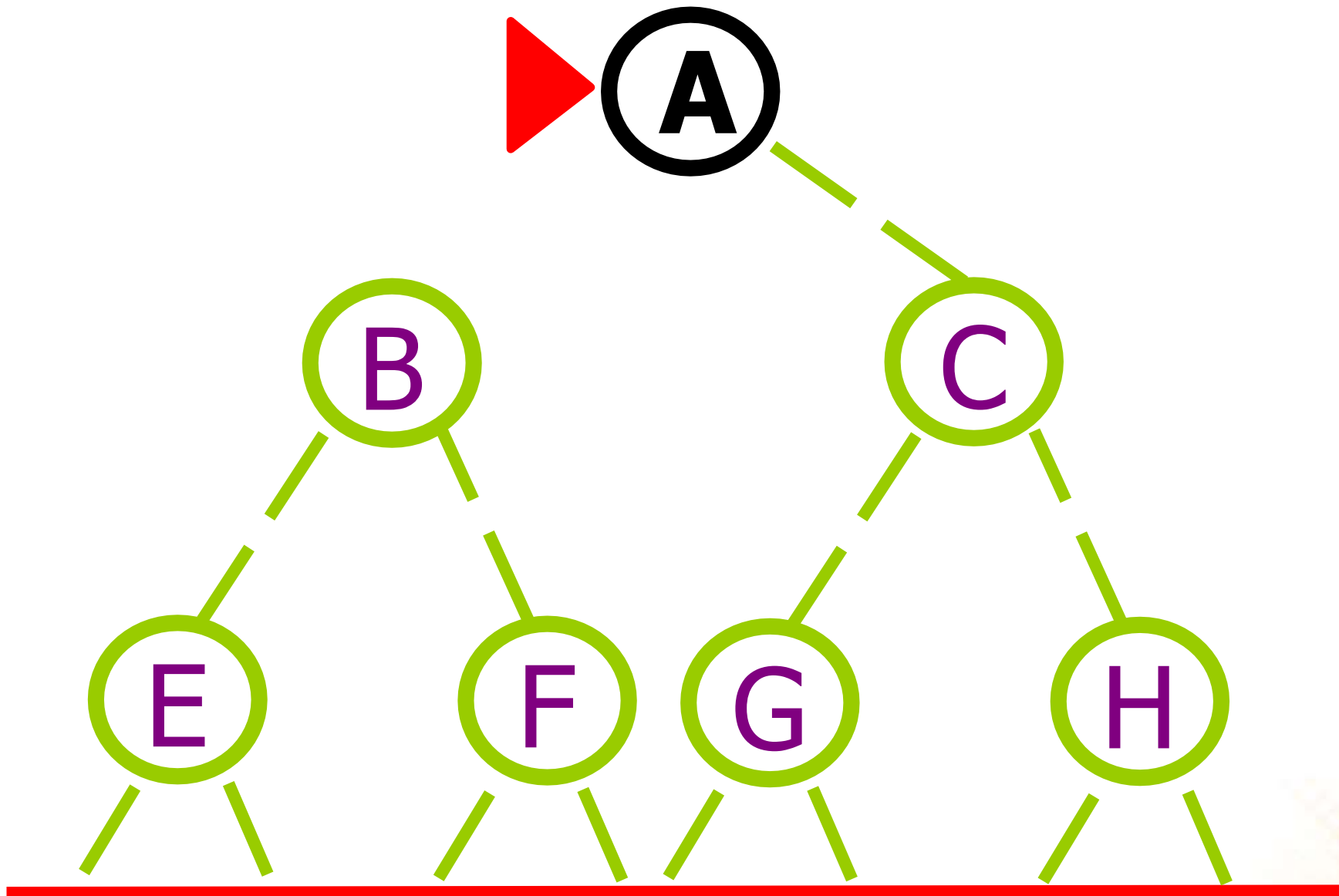
# Busca de Aprofundamento Iterativo $\ell=1$



# Busca de Aprofundamento Iterativo $\ell=1$

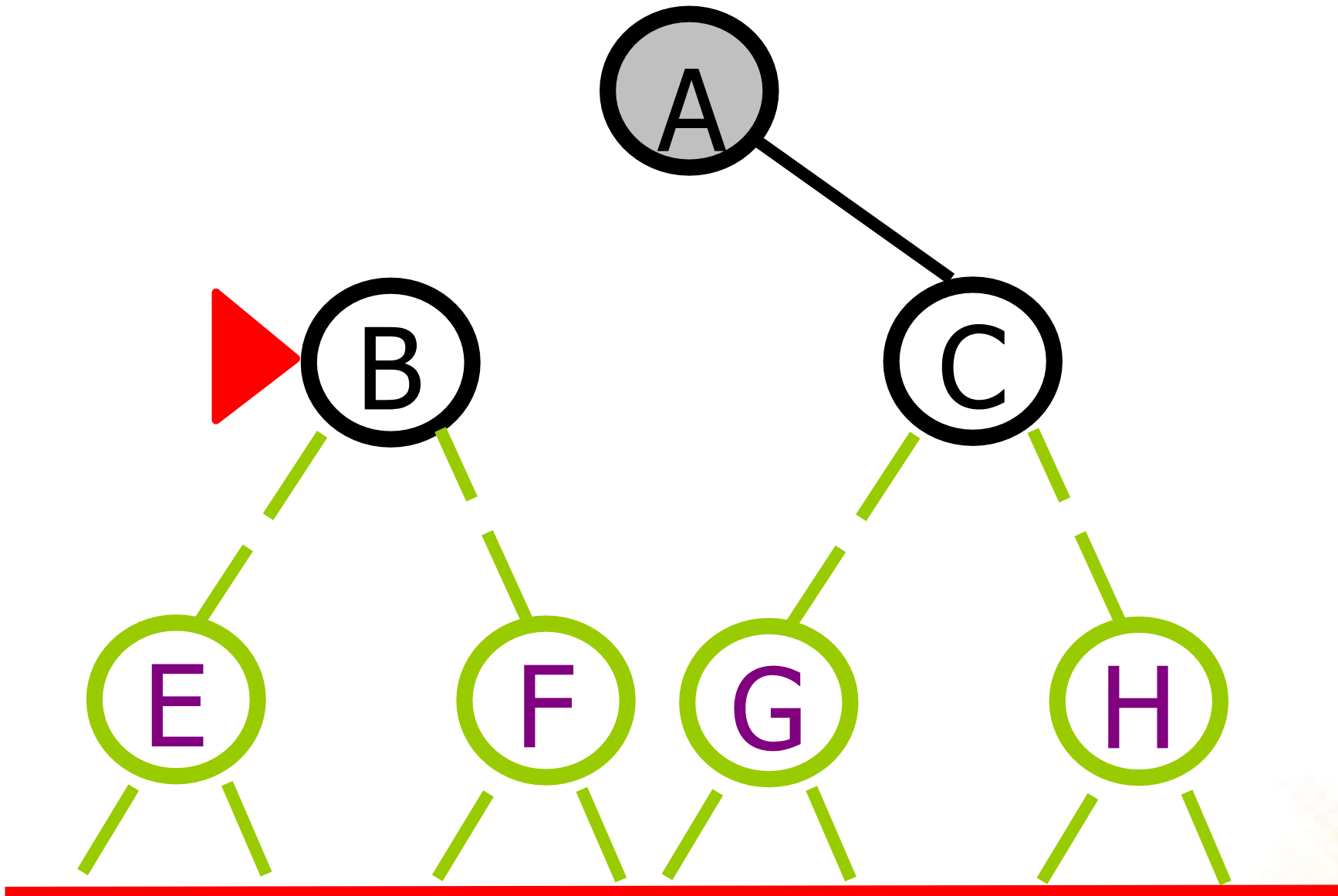


# Busca de Aprofundamento Iterativo $\ell=2$

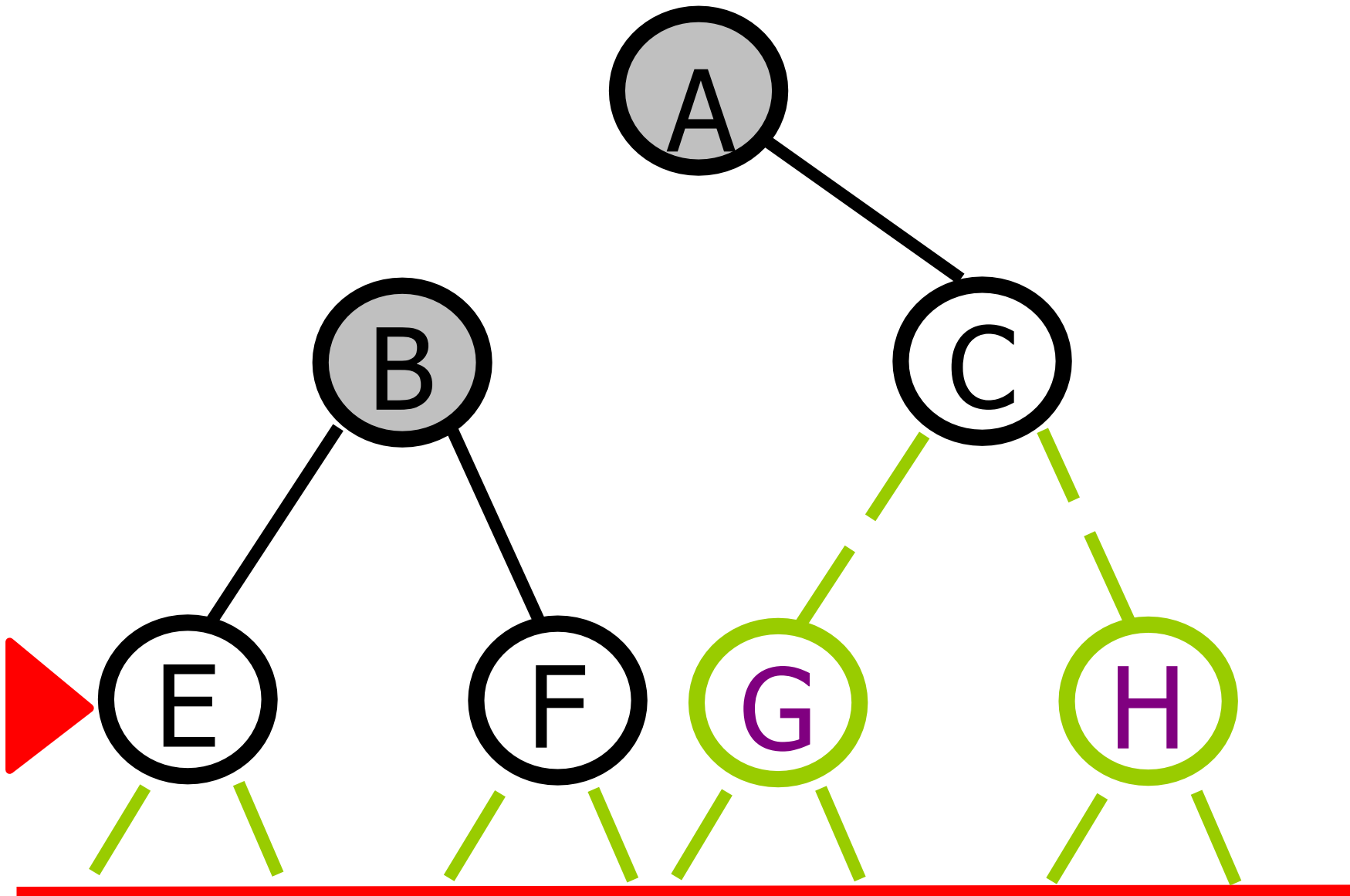




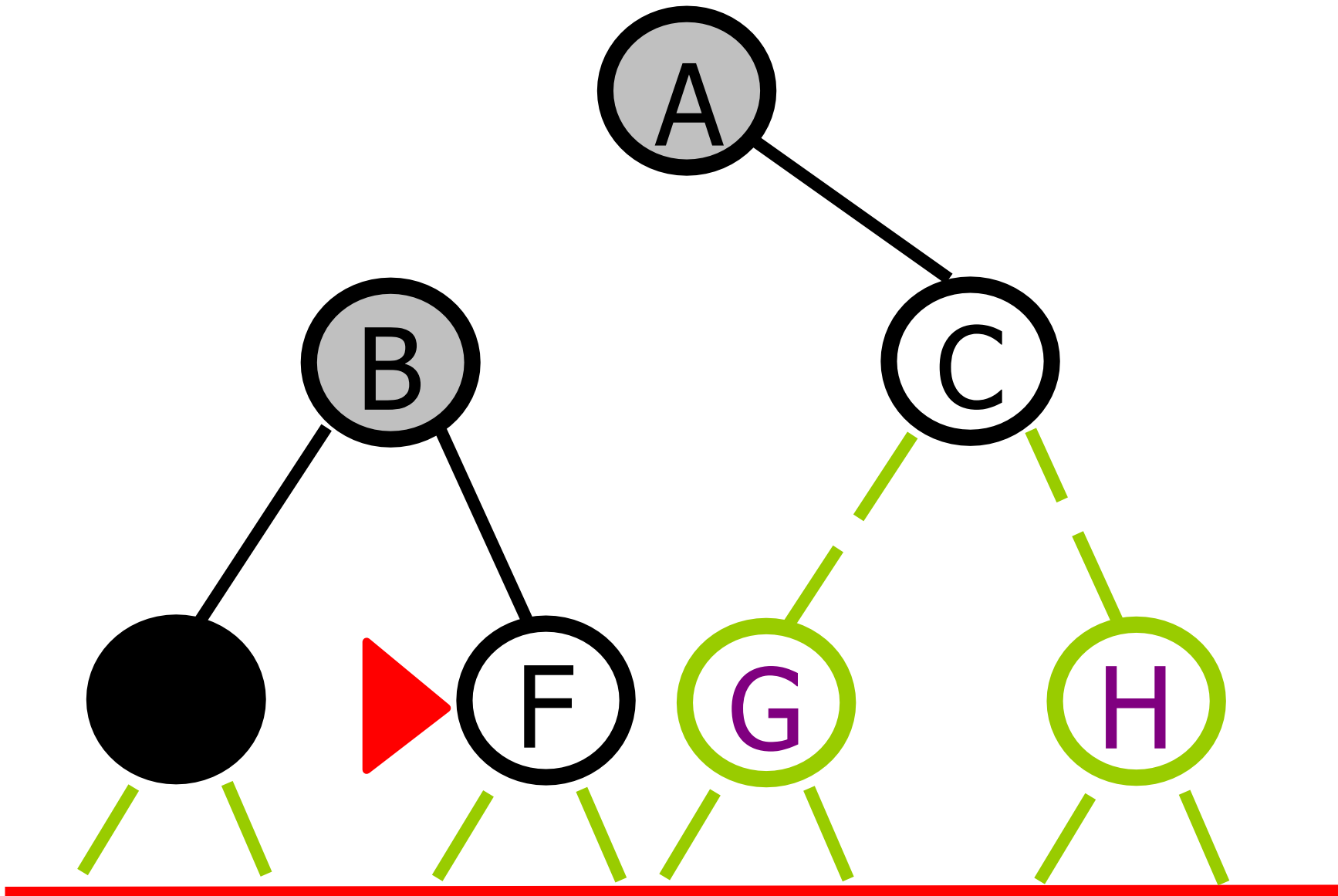
# Busca de Aprofundamento Iterativo $\ell=2$



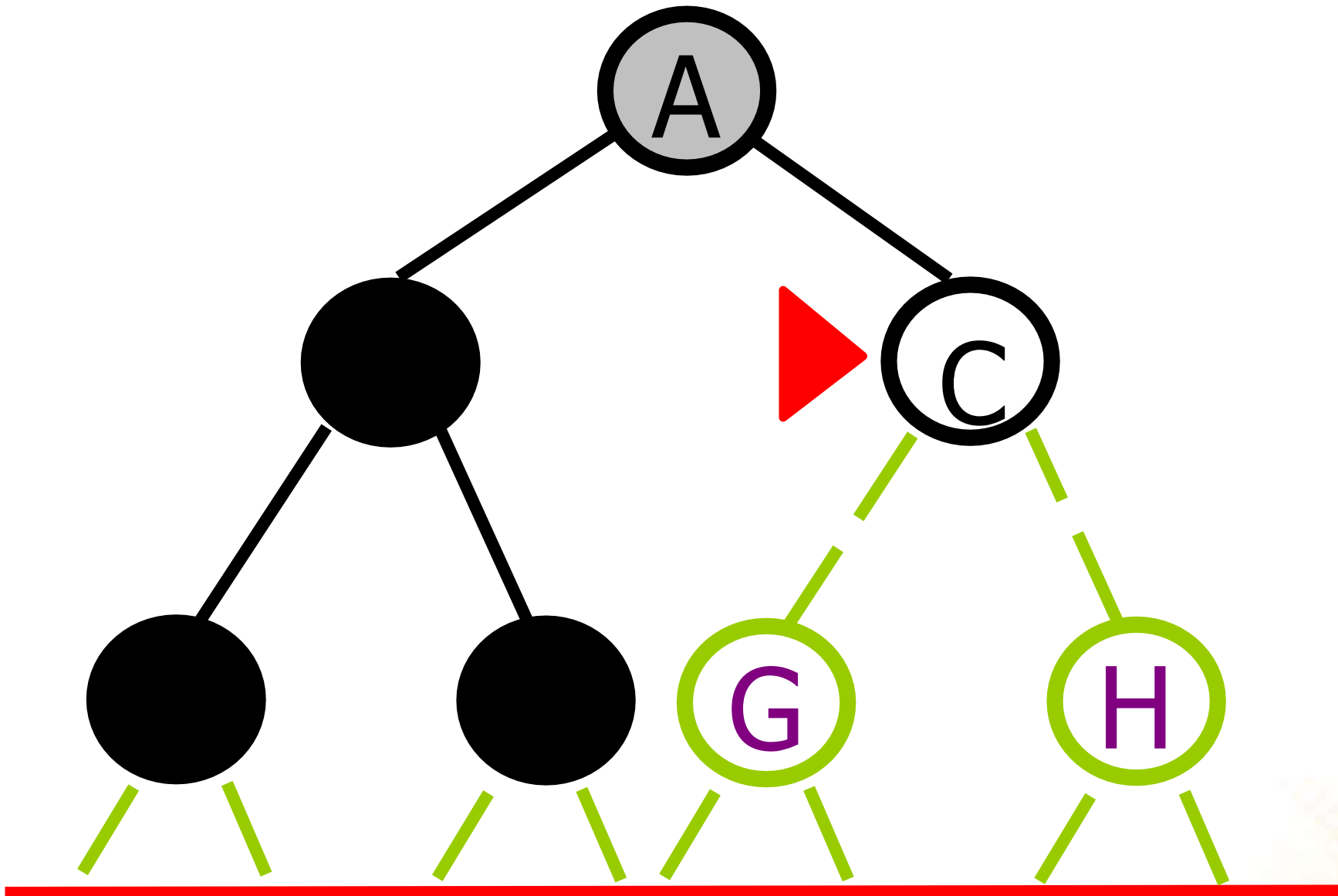
# Busca de Aprofundamento Iterativo $\ell=2$



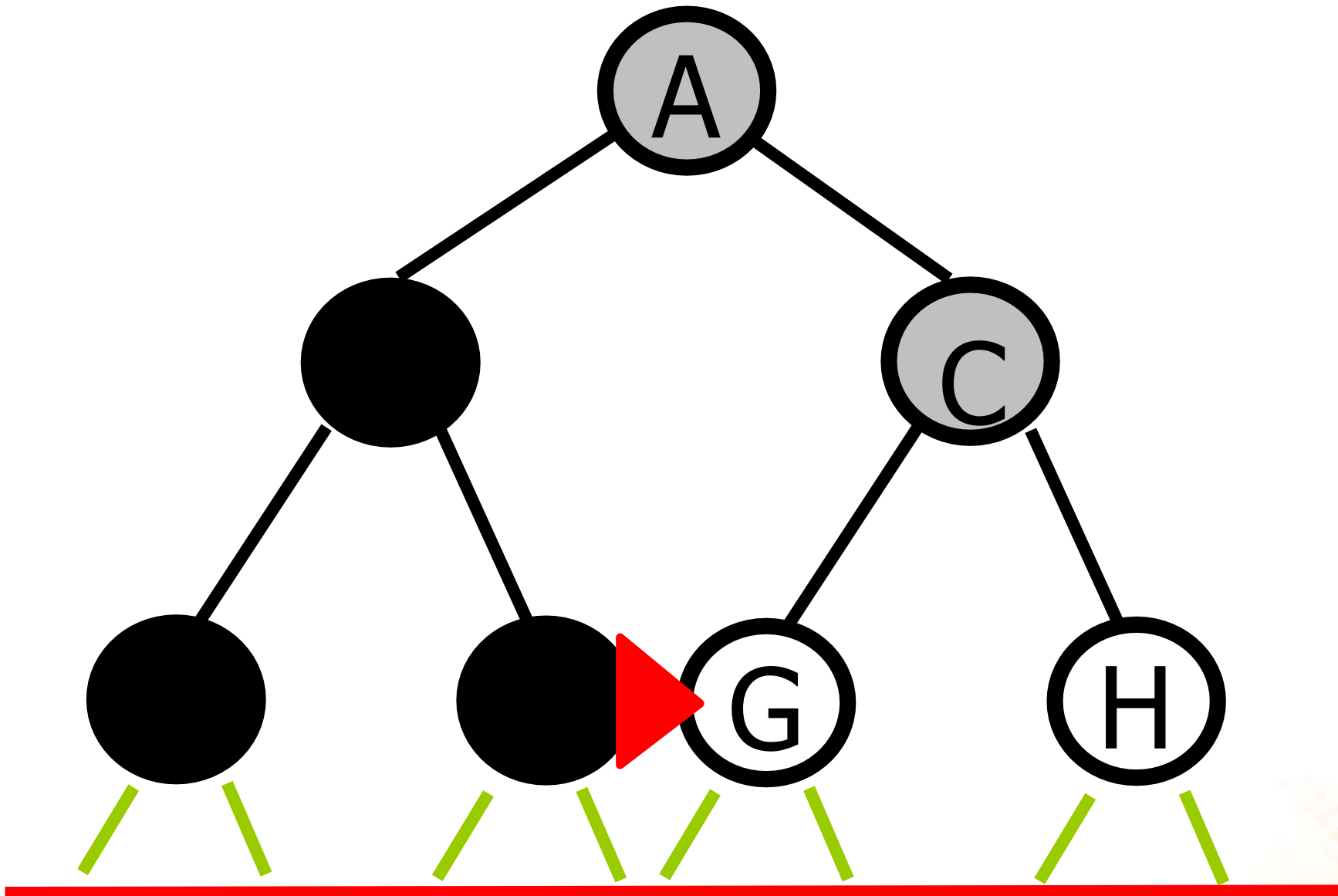
# Busca de Aprofundamento Iterativo $\ell=2$



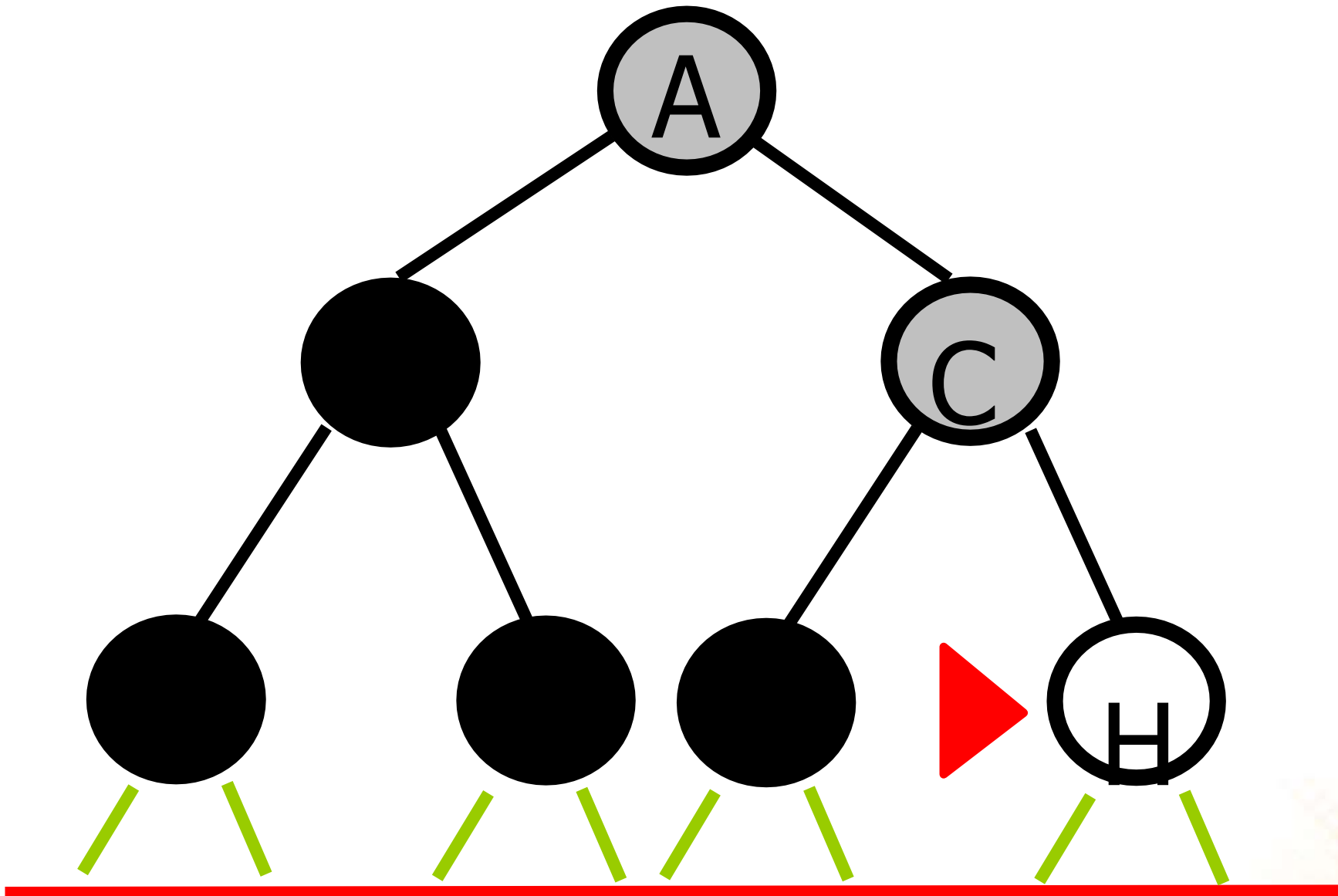
# Busca de Aprofundamento Iterativo $\ell=2$



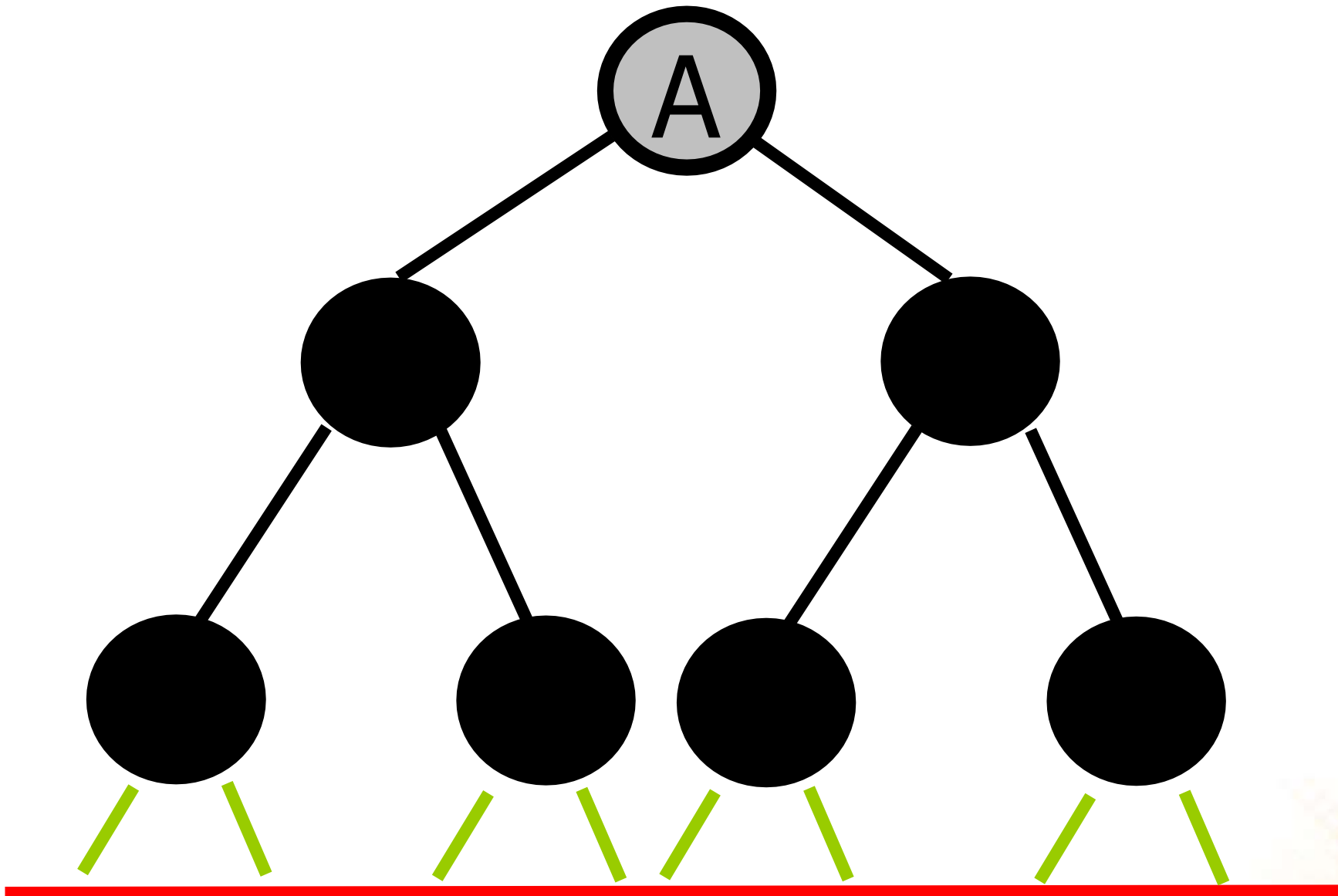
# Busca de Aprofundamento Iterativo $\ell=2$



# Busca de Aprofundamento Iterativo $\ell=2$



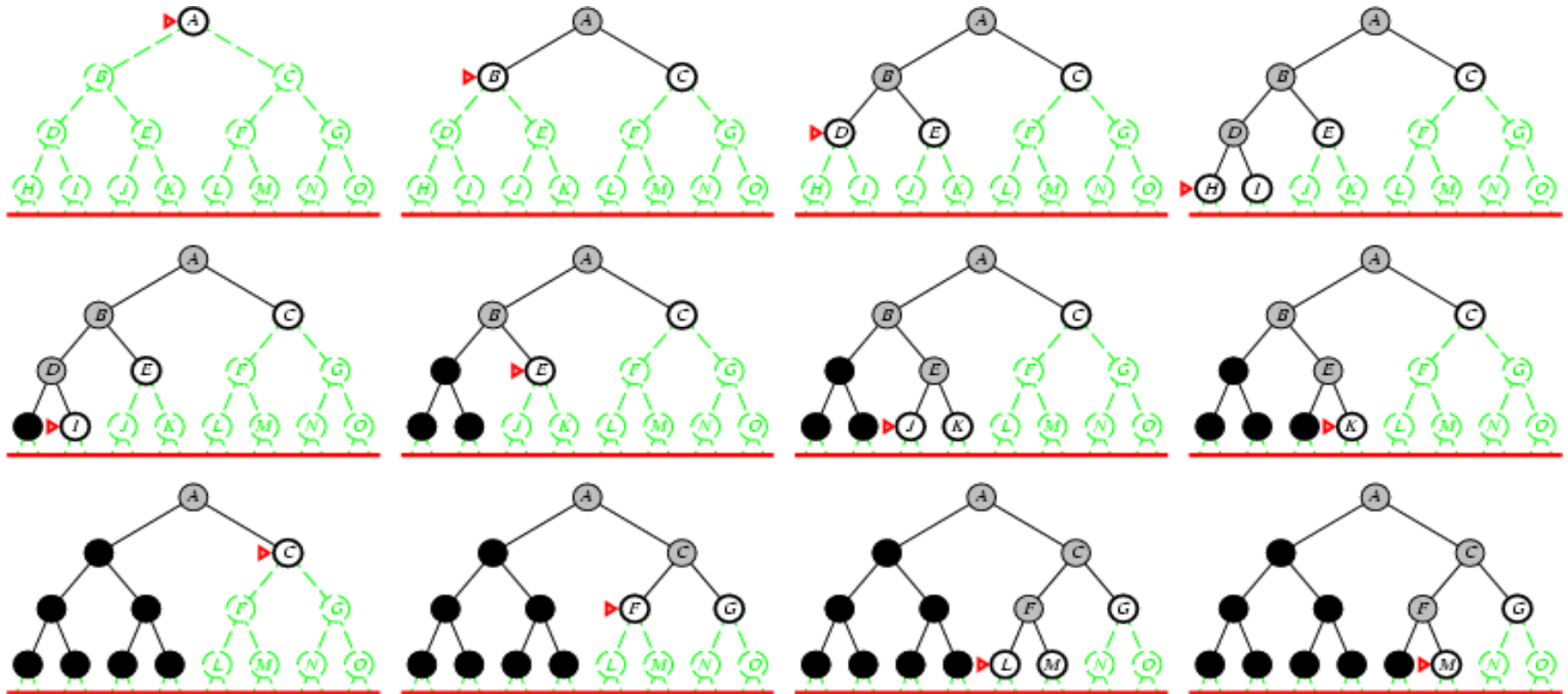
# Busca de Aprofundamento Iterativo $\ell=2$



# Busca de Aprofundamento Iterativo

$l=3$

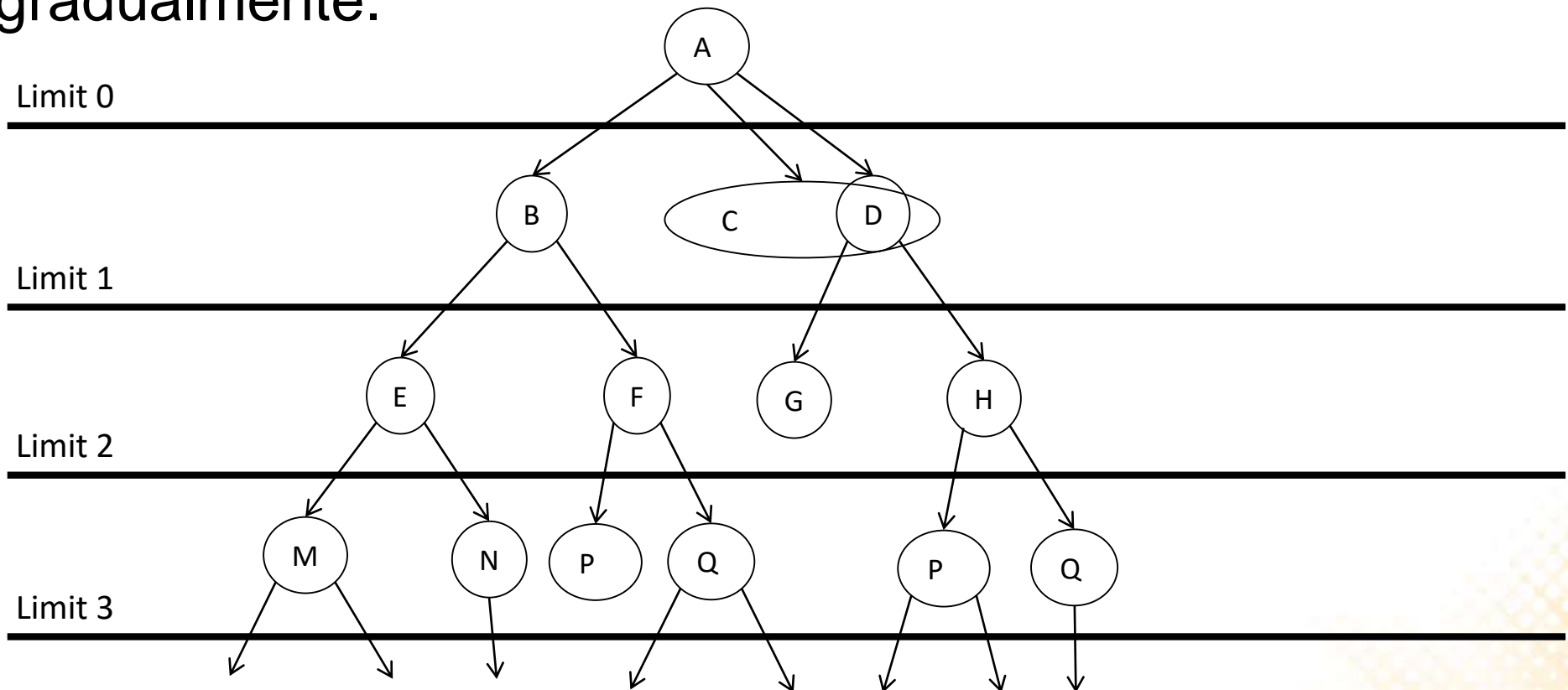
Limite  $l=3$





# Busca de Aprofundamento Iterativo

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



# Busca em Aprofundamento Iterativo

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth = 0 to  $\infty$  do
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
    if result  $\neq$  cutoff then return result
```

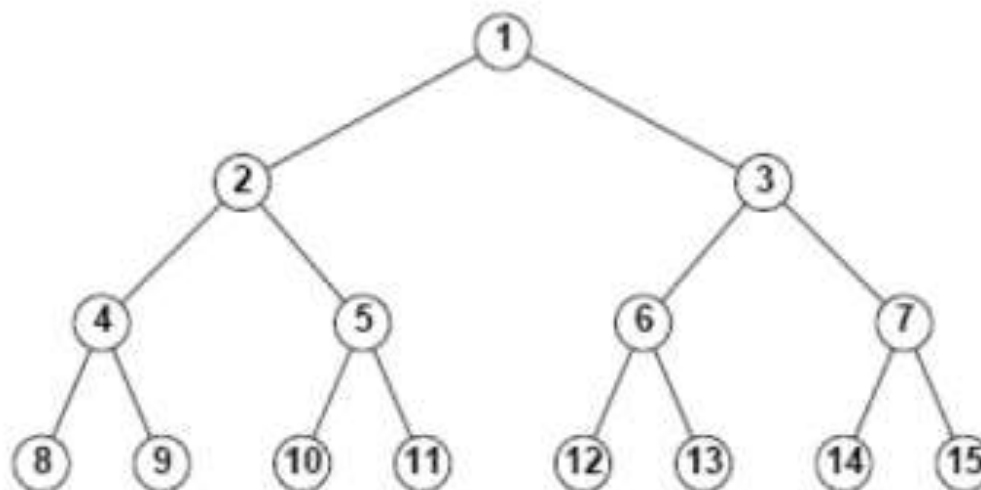
# Avaliação da Busca de Aprofundamento Iterativo

Critério	Avaliação
Completo	Sim, quando $b$ for finito = busca em largura
Otimidade	Sim, se custos das ações forem iguais
Tempo	$O(b^d)$ = busca em largura
Espacial	$O(b \cdot d)$ = busca em profundidade

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

# Exercícios

1. Dê um exemplo de problema em que a “busca em largura” funcionaria melhor do que a “busca em profundidade”. Dê um exemplo de problema em que a “busca em profundidade” funcionaria melhor do que a “busca em largura”. Justifique os exemplos.
2. Considerando a árvore abaixo qual o caminho percorrido para se chegar ao nó 11 utilizando as buscas em largura e profundidade?



# Leitura Complementar

- Russell, S. and Norvig, P. **Artificial Intelligence: a Modern Approach**, 3rd Edition, Prentice-Hall, 2009.
- **Capítulo 3: Resolução de Problemas por Meio de Busca**

