Busca não Informada

Inteligência Artificial

PCS3438

Escola Politécnica da USP

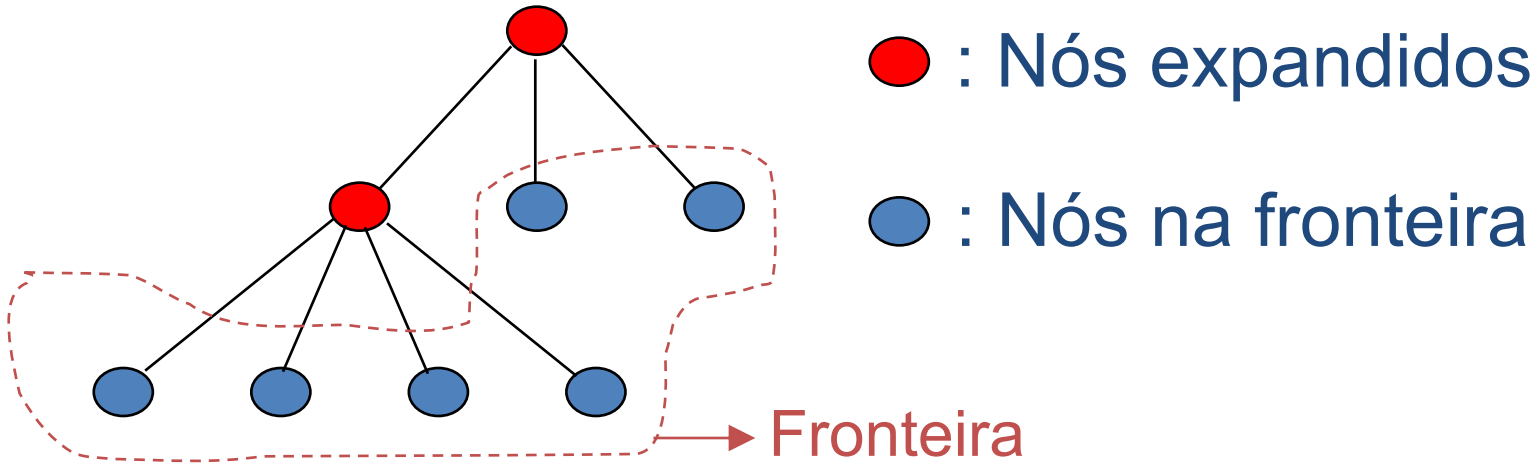
Engenharia de Computação (PCS)

Métodos de Busca

- **Busca cega** (ou busca não informada)
  - Não tem informação sobre qual sucessor é mais promissor para atingir a meta.
- **Busca heurística** (ou busca informada)
  - Possui informação (estimativa) de qual sucessor é mais promissor para atingir a meta.

Busca não informada

- A busca não informada (ou busca cega) não possui estimativas sobre qual sucessor é mais promissor para atingir a meta.
- Fronteira (ou borda): todos os nós **gerados** e ainda não **expandidos** (ou não **visitados**) da árvore de busca.



Medida de Desempenho na Busca

Desempenho de um algoritmo de busca:

- **Completo**: se existir uma solução, ela certamente é encontrada
- **Ótimo**: a busca encontra a solução de menor custo
- **Complexidade temporal**: quanto tempo demora para encontrar a solução
- **Complexidade espacial**: quanta memória é usada para realizar a busca

Medida de Desempenho na Busca

Em IA a árvore de busca é tipicamente infinita

A complexidade é expressa por:

- **b** – fator de ramificação (*branching*) ou número máximo de sucessores de um nó;
- **d** – profundidade (*depth*) do nó-meta mais próximo da raiz;
- **m** – comprimento máximo de um caminho no espaço de estados.

Estratégias de Busca Cega

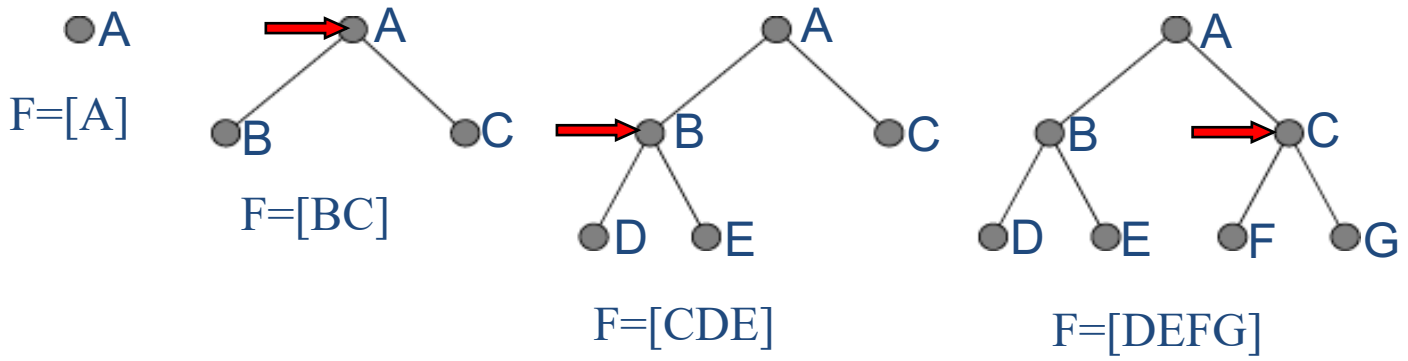
- **Busca em Largura**
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos

Busca em Largura ou em Extensão (BFS)

- Ordem de expansão dos nós:
  1. Nó raiz
  2. Todos os nós de profundidade 1
  3. Todos os nós de profundidade 2, etc…

Fronteira = **FIFO** (first-in-first-out)

➔ insere no fim da **fila**



BL (ou BFS)

Algorithm 1 Busca em Largura (problema)

```
nó ← um nó com ESTADO = problema.ESTADO-INICIAL
borda ← uma fila FIFO com nó como elemento único
explorado ← ∅
while borda ≠ ∅ do
  nó ← POP(borda) {escolhe o nó primeiro da fila}
  explorado ← explorado ∪ {nó.ESTADO}
  if problema.TESTE-DE-OBJETIVO(nó.ESTADO) then
    return SOLUÇÃO(nó)
  end if
  for cada ação em problema.AÇÕES(nó.ESTADO) do
    filho ← NÓ-FILHO(problema, nó, ação)
    if filho.ESTADO não está no explorado nem na borda then
      borda ← INSIRA(filho, borda)
    end if
  end for
end while
return erro
```

Note: teste ao visitar!

Põe na fila, na ordem de geração (mais antigos, primeiros)

Desempenho da busca em largura

Completa?

- Se b finito, é completa: se um nó-meta estiver a uma profundidade d, a busca em largura sempre irá encontrá-lo.

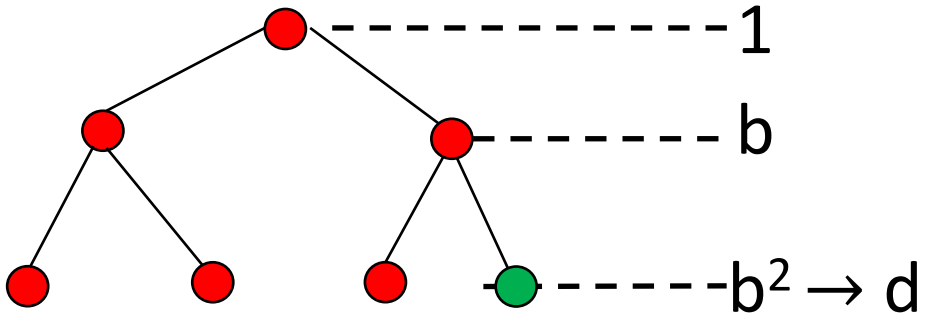
Ótima?

- Nem sempre – nó objetivo mais raso ≠ melhor nó objetivo
- É ótima se o custo do caminho for uma função não-decrescente da profundidade do nó (ex: todas ações têm mesmo custo)

Desempenho da busca em largura

- Complexidade de **tempo** e **espaço**
  - Meta em d, teste ao gerar, cada nó tem b filhos, no pior caso:

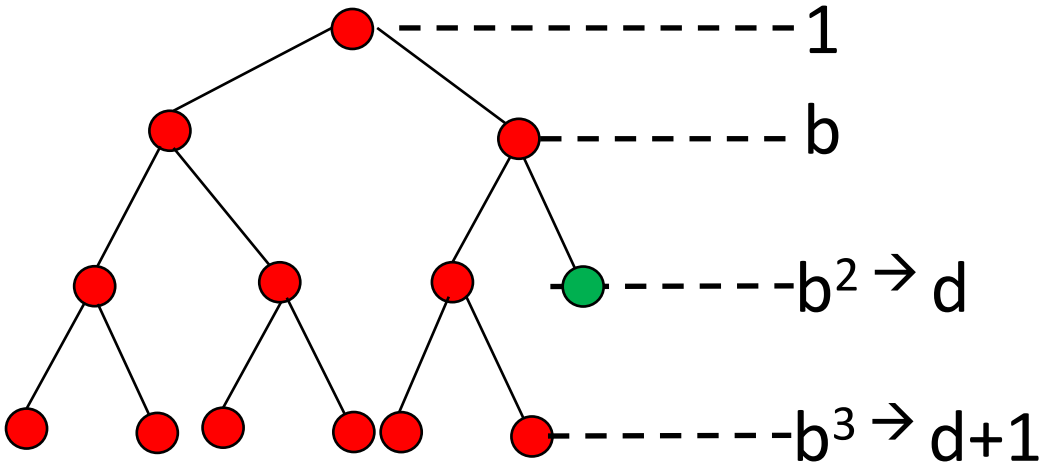
$1 + b + b^2 + b^3 + \ldots + b^d = \mathcal{O}(b^d)$



Desempenho da busca em largura

- Complexidade de **tempo** e **espaço**
  - Meta em d, teste ao visitar/expandir, cada nó tem b filhos, no pior caso:

$1 + b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b) = \mathcal{O}(b^{d+1})$



Estratégias de Busca Cega

- Busca em Largura
- **Busca de Custo Uniforme**
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos

Busca de Custo Uniforme (BCU)

- Modifica a busca em largura:
  - **Em vez de expandir o nó gerado primeiro, expande o nó *n* da fronteira com menor custo de caminho *g(n)* (da raiz ao nó)**

- Usa uma fila de prioridade ordenada por *g*
- Sempre testa objetivo ao visitar (não ao gerar)
  - Testa nós na borda e troca se atual tiver $g(n) < g_{ant}(n)$

Algorithm 2 Busca de Custo Uniforme (problema)

```
1: nó ← um nó com ESTADO = problema.ESTADO-INICIAL
2: nó.CUSTO-DE-CAMINHO ← 0
3: borda ← fila de prioridade ordenada pelo CUSTO-DE-CAMINHO, com nó
   como elemento único
4: explorado ← ∅
5: while borda ≠ ∅ do
6:    nó ← POP(borda) {escolhe o nó de menor custo na borda}
7:    explorado ← explorado ∪ {nó.ESTADO}
8:    if problema.TESTE-OBJETIVO(nó.ESTADO) then
9:       return SOLUÇÃO(nó)
10:   end if
11:   for cada ação em problema.AÇÕES(nó.ESTADO) do
12:      filho ← NÓ-FILHO(problema, nó, ação) {atualiza também o
         filho.CUSTO-DE-CAMINHO}
13:      if filho.ESTADO não está na borda nem no explorado then
14:         borda ← INSIRA(filho, borda) {coloca na fila em ordem}
15:      else
16:         if filho.ESTADO é um nó da borda com maior nó.CUSTO-DE-
            CAMINHO then
17:            eliminar aquele nó da borda
18:            borda ← INSIRA(filho, borda)
19:         end if
20:      end if
21:   end for
22: end while
23: return erro
```

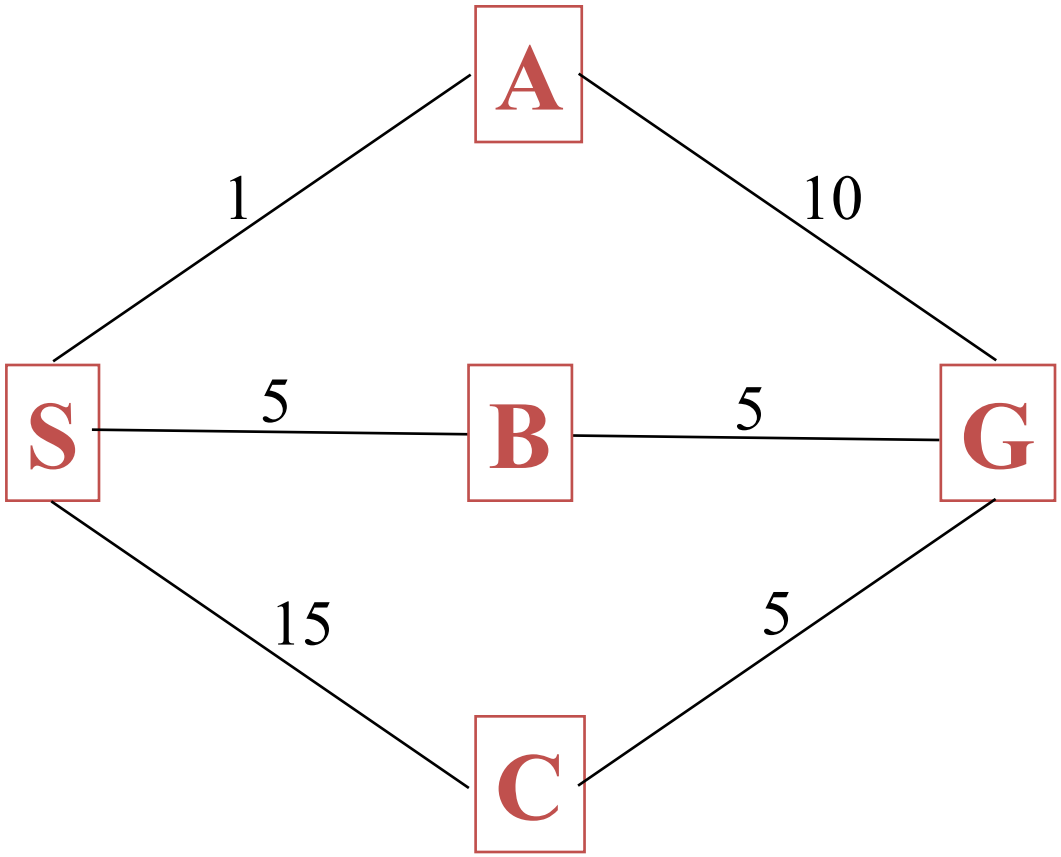
Teste ao visitar!

Na fila, em ordem!

Troca na borda (na fila, em ordem).

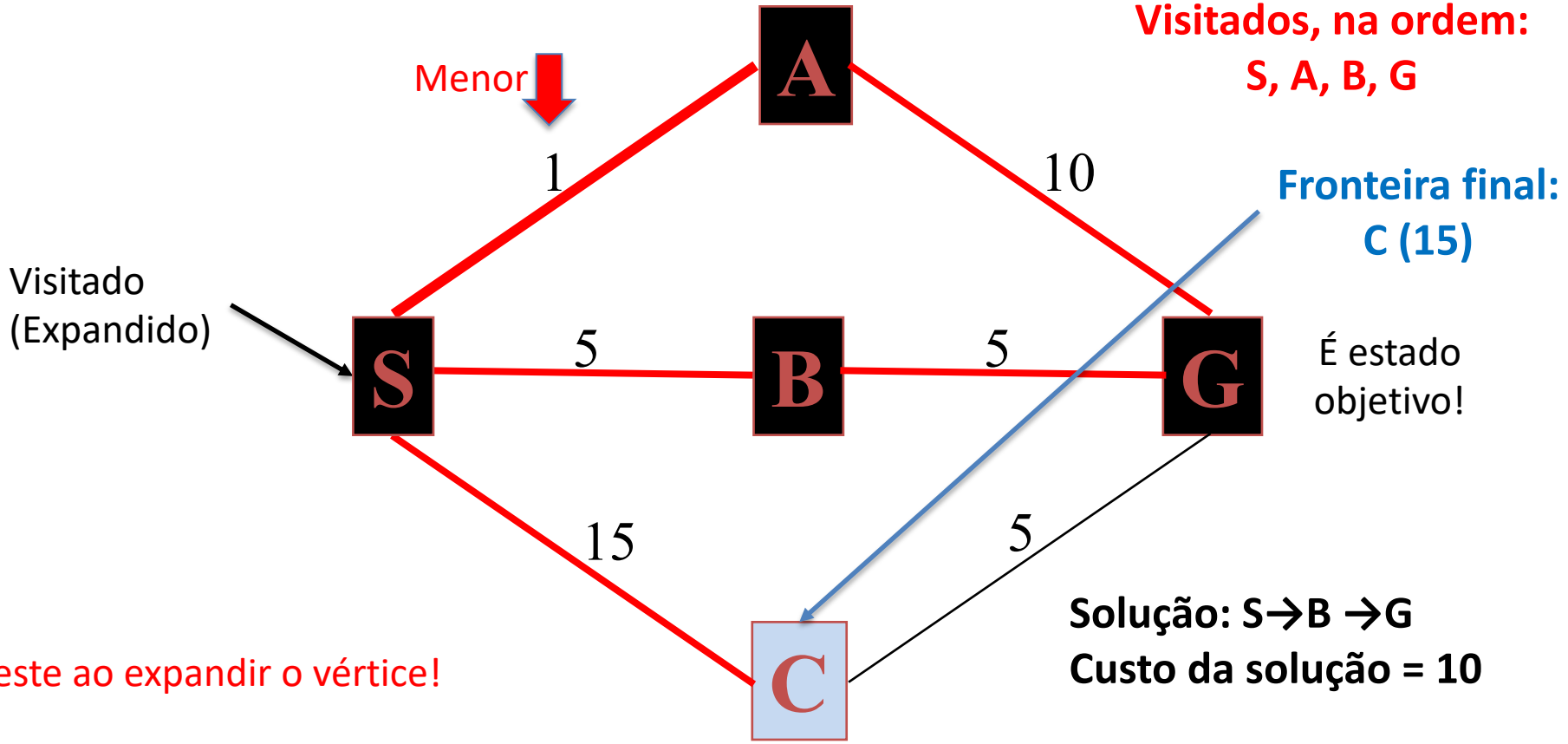
Busca de Custo Uniforme

- **Exemplo**: gerar a árvore de busca usando busca de custo uniforme, partindo de S e chegando a G. Mostre todos os estados da fronteira.



Busca de Custo Uniforme

- **Exemplo**: gerar a árvore de busca usando busca de custo uniforme, partindo de S e chegando a G. Mostre todos os estados da fronteira.



Desempenho da Busca de Custo Uniforme

- Completa? Só *se custo de cada ação* $\geq \varepsilon$, $\forall n$
  - ε *é uma constante pequena positiva*
    - *Loop* infinito: se existir um caminho com uma sequência infinita de ações de custo=0
- Ótima? *Só se g(sucessor(n)) > g(n)*
  - *custo* ***no mesmo caminho*** *sempre cresce (i.e., não tem ação com custo negativo ou 0)*
- Complexidade de tempo
  - *C* = custo da solução ótima (custo de cada ação* $\geq \varepsilon$*)*

➔ Pior caso: $\mathcal{O}(b^{\lceil C^* \rceil / \varepsilon})$, o que pode ser bem maior que b^d

Estratégias de Busca Cega

- Busca em Largura
- Busca de Custo Uniforme
- **Busca em Profundidade**
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos

Busca em Profundidade (DFS)

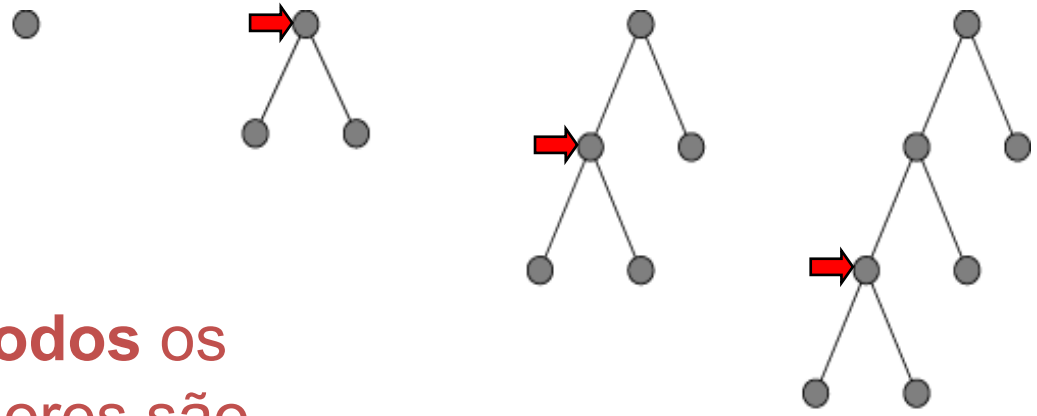
- Ordem de expansão dos nós:
  1. Nó raiz
  2. Primeiro nó de profundidade 1
  3. Primeiro nó de profundidade 2, etc…

Fronteira = **LIFO (last-in-first-out)** ➔ insere na **pilha**

– Versões: busca em grafo / em árvore (não são ótimas)

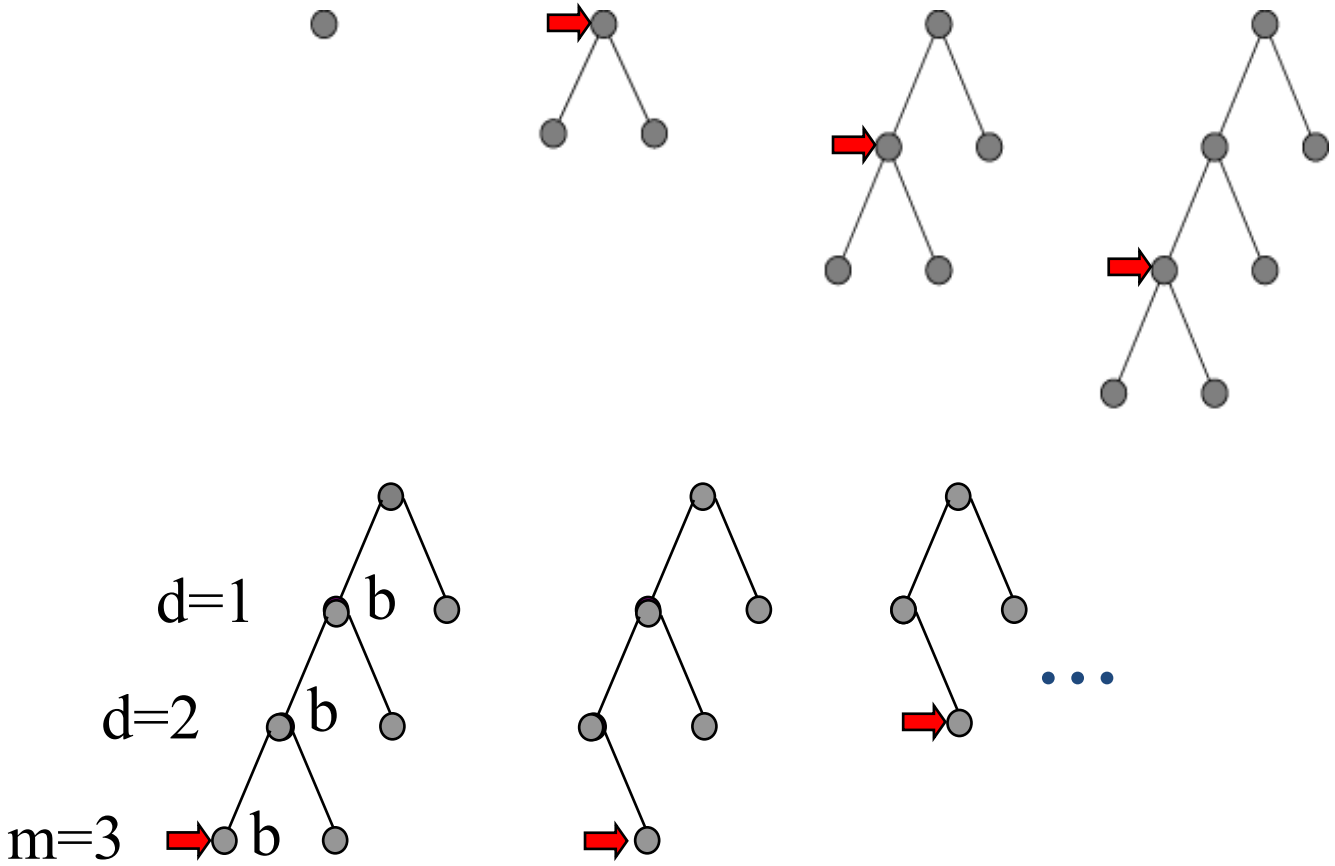
- Em grafo: evita estados repetidos e caminhos redundantes (usa mais memória) – é completa se espaço de estados finito
- Em árvore: não é completa (pode ficar em loop); mas usa pouca memória

DFS com busca em árvore



Aqui, **todos** os sucessores são gerados durante a expansão/visita.

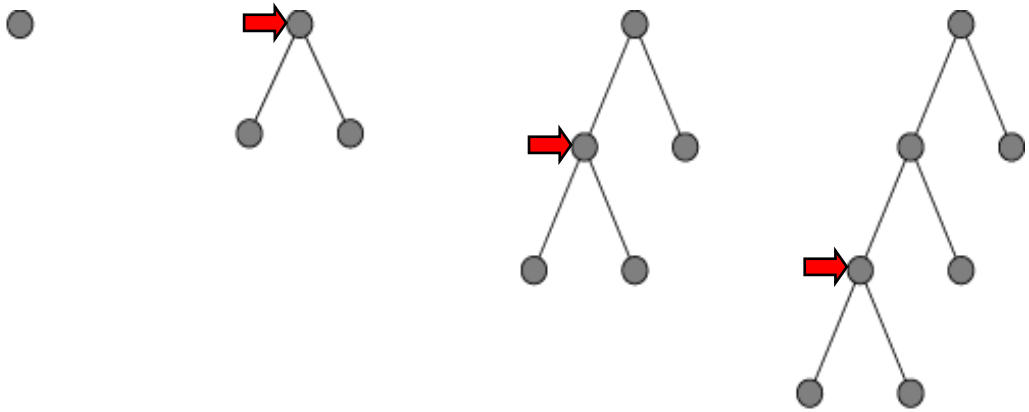
DFS com busca em árvore



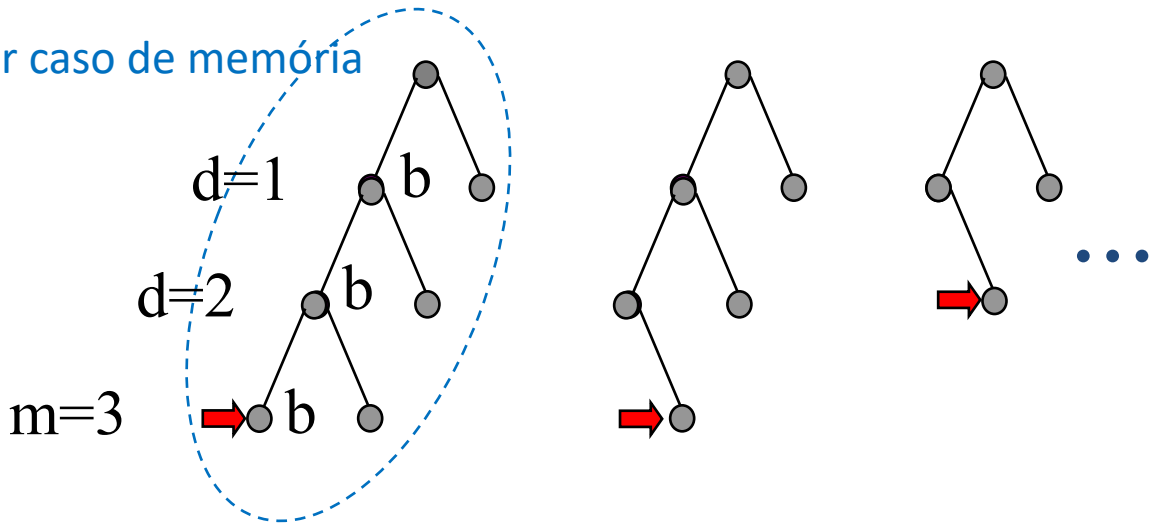
Neste exemplo, nós com profundidade 3 não têm sucessores.

Nós sem sucessores e já expandidos são "apagados".

DFS com busca em árvore



Pior caso de memória



Para espaço de estados com fator de ramificação b e **profundidade máxima m** (m pode ser >> d), requer apenas **bm+1** de memória no pior caso: $\mathcal{O}$(bm)

Neste exemplo, nós com profundidade 3 não têm sucessores.

Nós sem sucessores e já expandidos são "apagados".

Desempenho da Busca em Profundidade

- Complexidade espacial: $O(bm)$, no pior caso
- Complexidade temporal: $O(b^m)$, no pior caso

$$b^0 + b^1 + b^2 + b^3 + \ldots = O(b^m)$$

  - Para problemas com várias soluções, esta estratégia pode ser mais rápida do que a busca em largura.
  - Esta estratégia deve ser evitada quando as árvores geradas são muito *profundas* (**m** muito grande) ou geram *caminhos infinitos*.

Variante: **Busca em Profundidade com Retrocesso (DFS)** (*backtracking search*)

- Parecida com BP, mas **somente UM sucessor é gerado em cada iteração**
  - na BP, <u>todos</u> os sucessores são gerados na expansão do nó pai
- Portanto, requer só $\mathcal{O}$**(m)** de memória
  - BP requer $\mathcal{O}$(bm) de memória
- **Restrição**: deve ser capaz de retornar ao pai e criar o novo sucessor

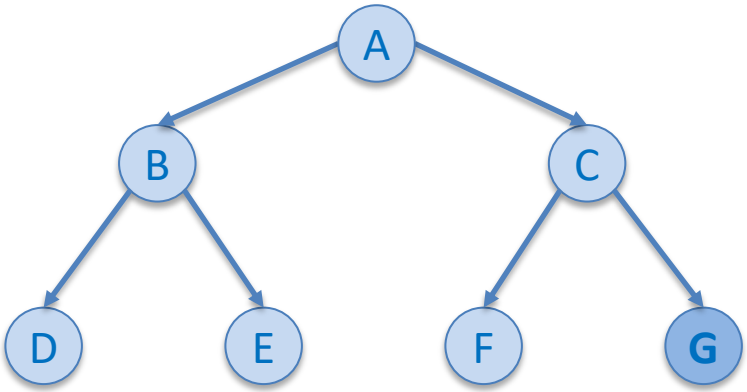
Estratégias de Busca Cega

- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- **Busca em Profundidade Limitada**
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- Evitando Estados Repetidos

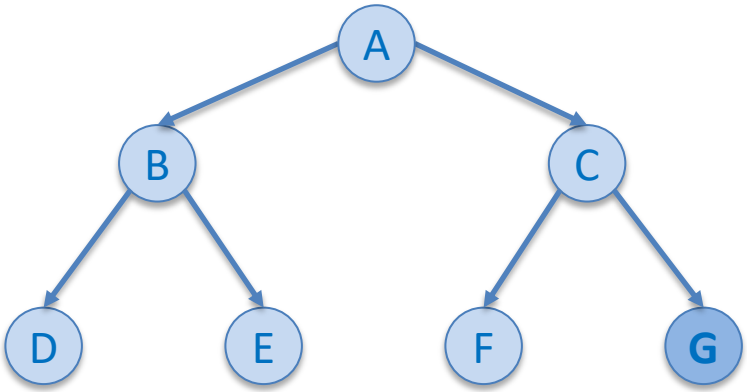
Busca em Profundidade Limitada

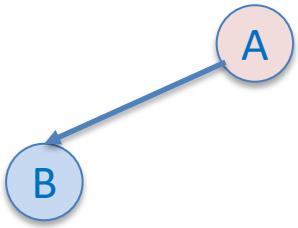
- Evita o problema de árvores não limitadas ao impor um limite máximo (l) de profundidade para os caminhos gerados.
  - O domínio do problema estabelece a profundidade limite.
  - Problema: definir limite l adequado!
- Completa? Somente se $l \geq d$.
- Ótima? Não, exceto se l = d e custo dos passos for igual
- Complexidade espacial: $\mathcal{O}(bl)$
- Complexidade temporal: $\mathcal{O}(b^l)$ no pior caso.

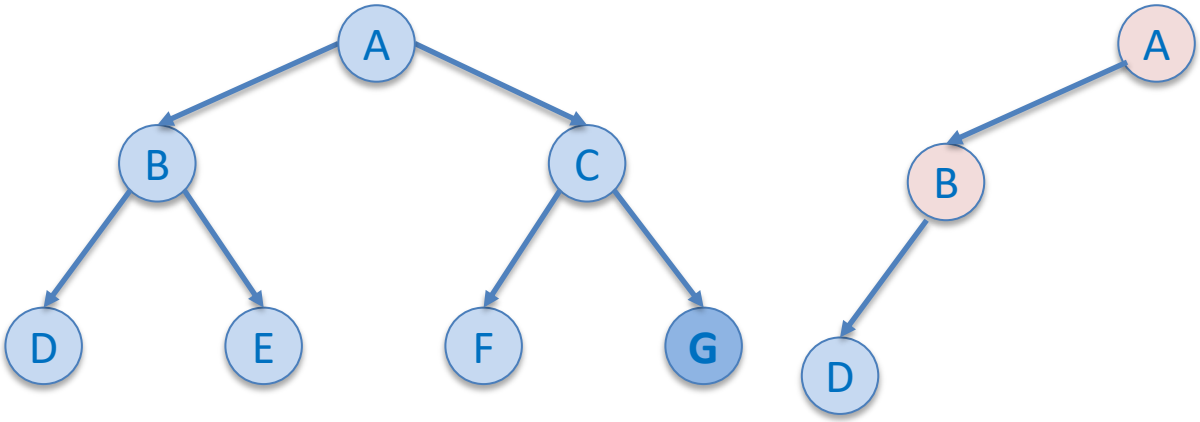
BP é caso particular de BPL, com l = ∞

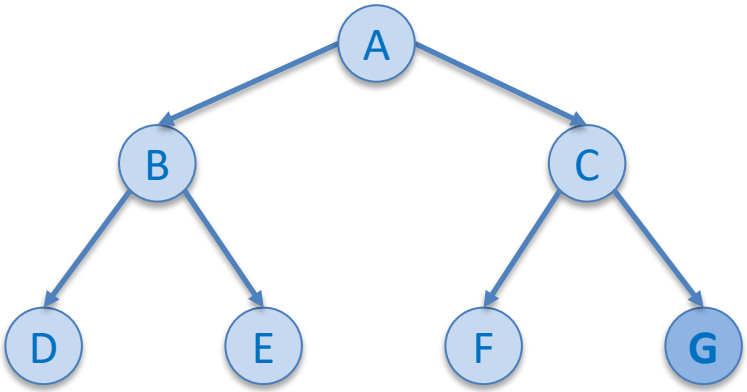


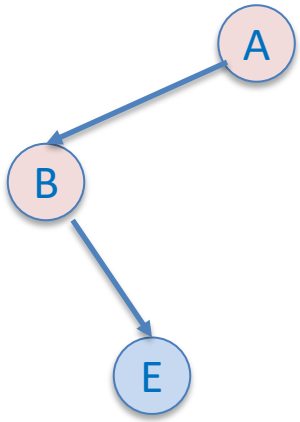
Limite l = 2

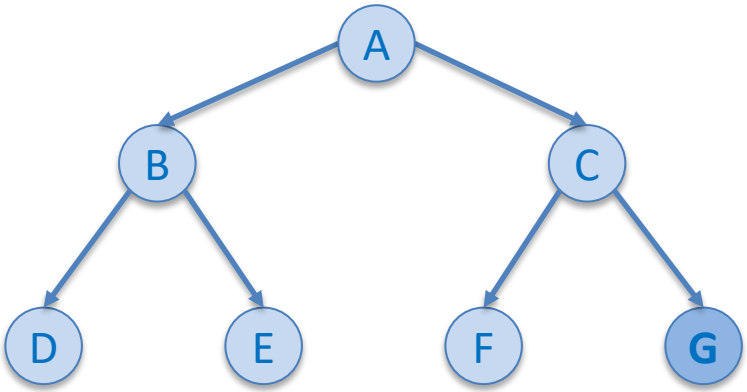

A
B
C
D
E
F
G

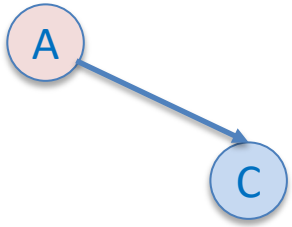

A
B

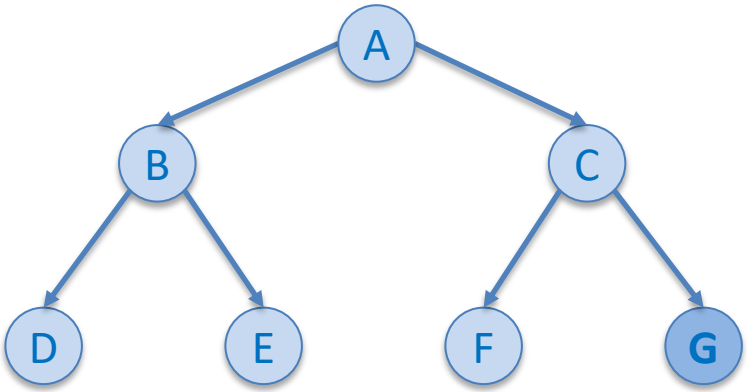

A
B
C
D
E
F
G
A
B
D

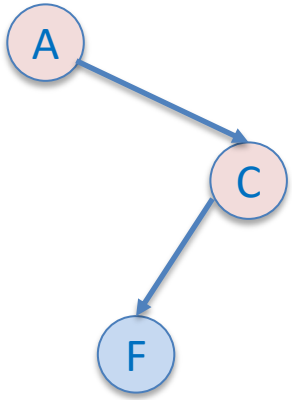

A
B
C
D
E
F
G

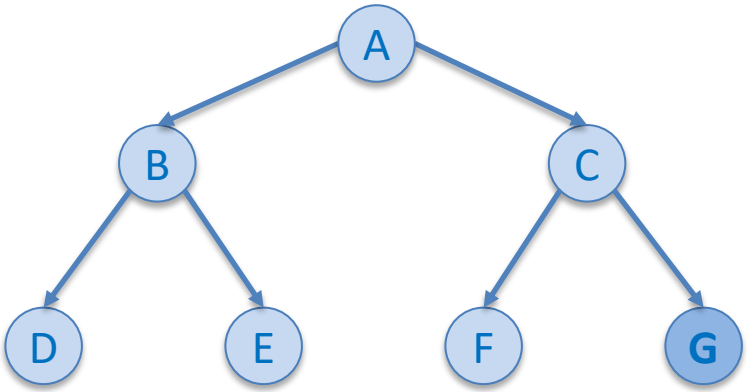

A
B
E

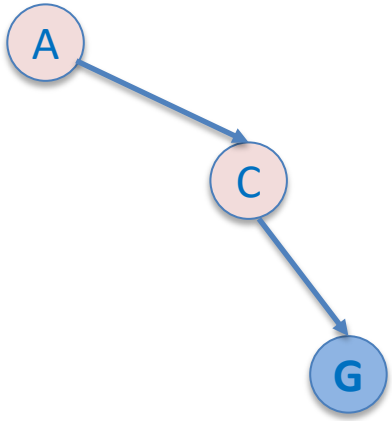

A
B
C
D
E
F
G


A
C


A
B
C
D
E
F
G


A
C
F





Responde G

Estratégias de Busca Cega

- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- **Busca em Profundidade com Aprofundamento Iterativo**
- Busca Bidirecional
- Evitando Estados Repetidos

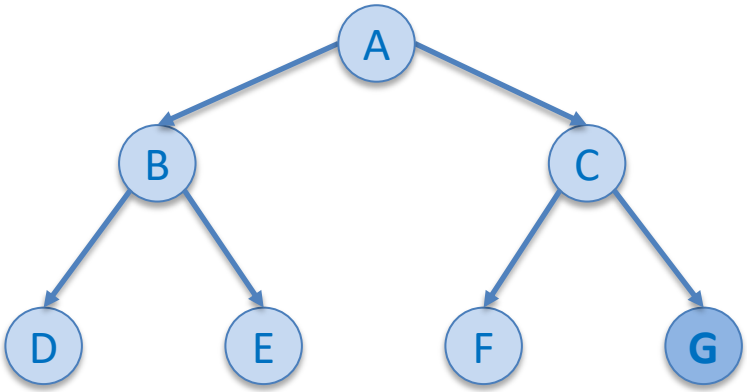
Busca com Aprofundamento Iterativo (BAI)

- Tenta limites com valores crescentes, partindo de zero, até encontrar a primeira solução (em d).
  - Combina vantagens da busca em largura (BL) com as da busca em profundidade (BP).
  - **Em geral, é a estratégia preferida de busca cega para quando o espaço de estados é muito grande e a profundidade da solução d é desconhecida.**

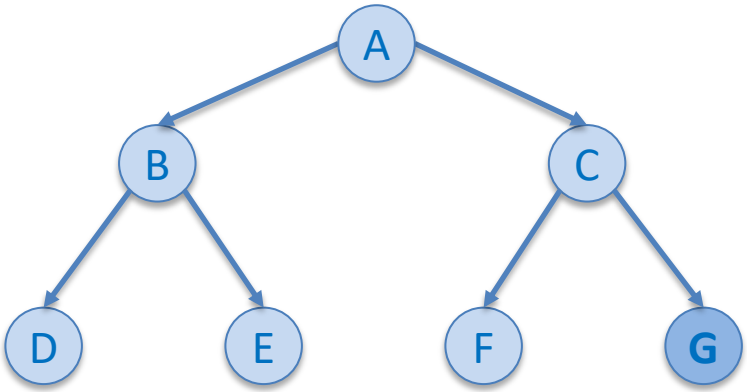
Algorithm 5 BAI(problema): Busca por Aprofundamento Iterativo

```
1: for limite = 0 até ∞ do
2:     resultado ← BPL(problema, limite)
3:     if resultado ≠ corte then
4:         return resultado
5:     end if
6: end for
```

Chamada inicial: Nível 0

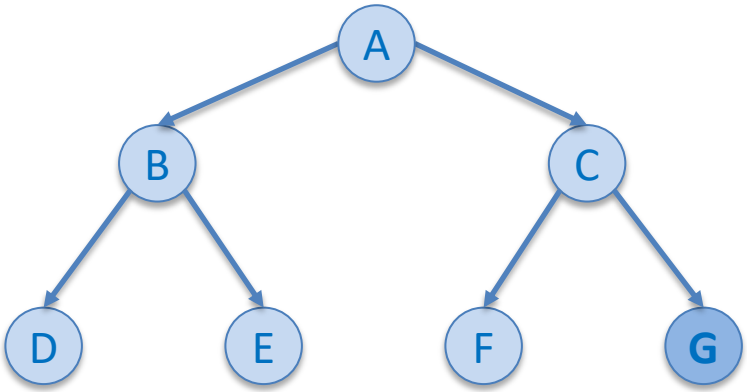


Chamada inicial: Nível 0

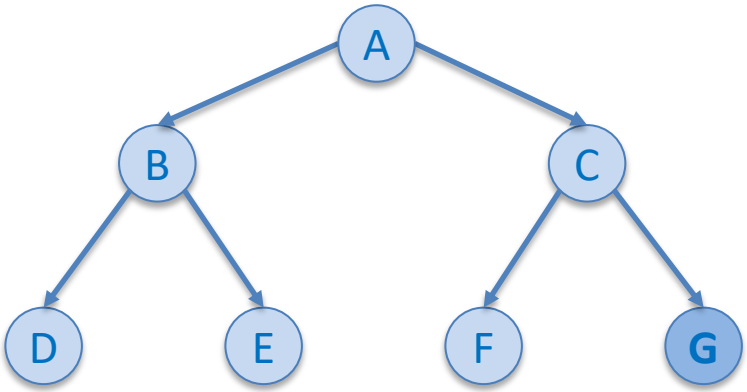


Nível 0: A não é meta!
Chama BAI novamente, agora com nível 1

Chamada: Nível 1



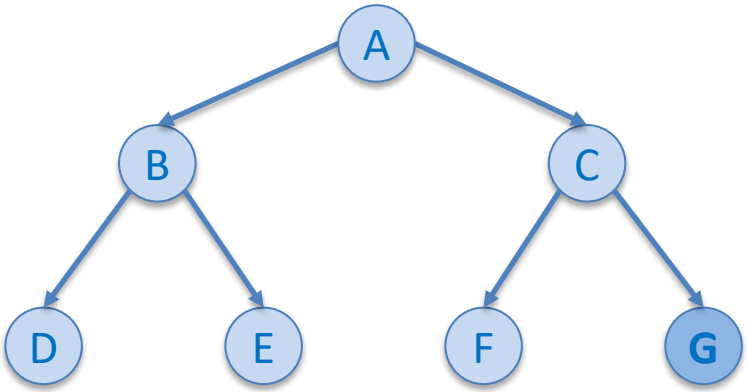
Chamada: Nível 1

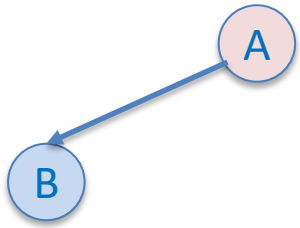


Recria o Nível 0: A não é meta!

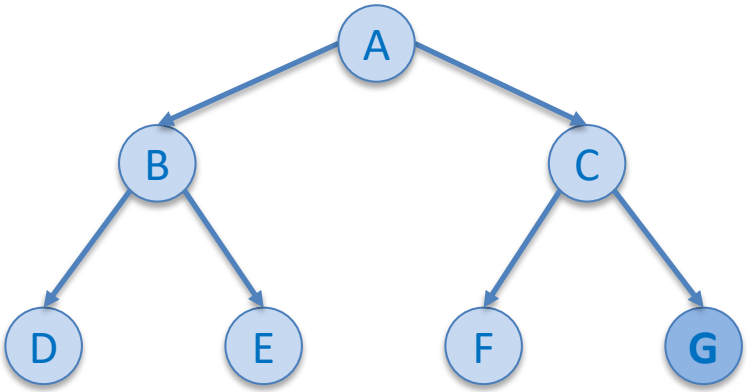
➔ Descobre os filhos.

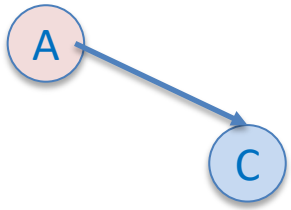
Chamada: Nível 1





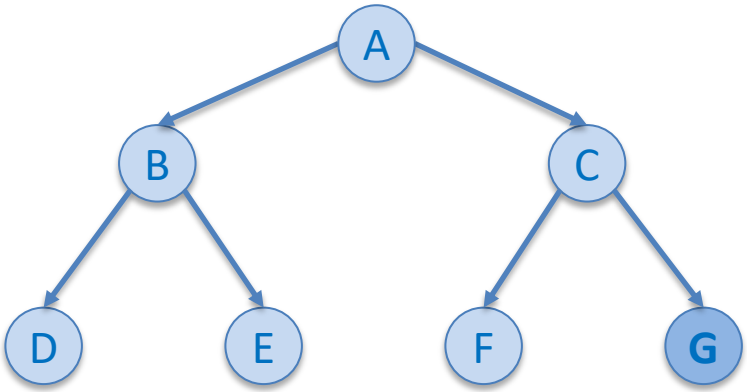
Chamada: Nível 1



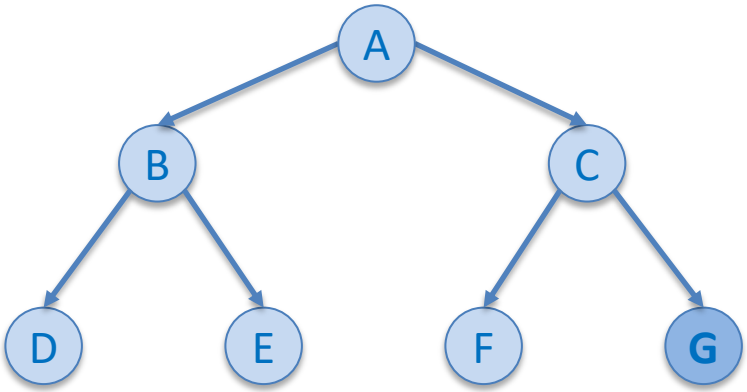


Nível 1: nenhum vértice é a meta.
Chama BAI novamente com nível 2.

Chamada: Nível 2



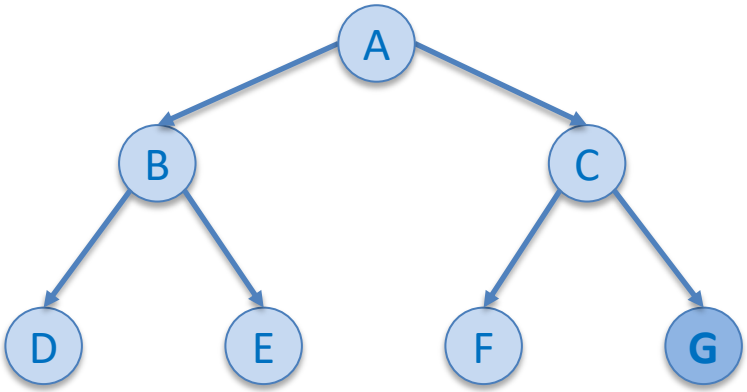
Chamada: Nível 2

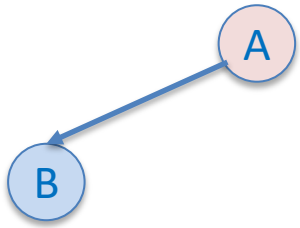


Recria o Nível 0: A não é meta!

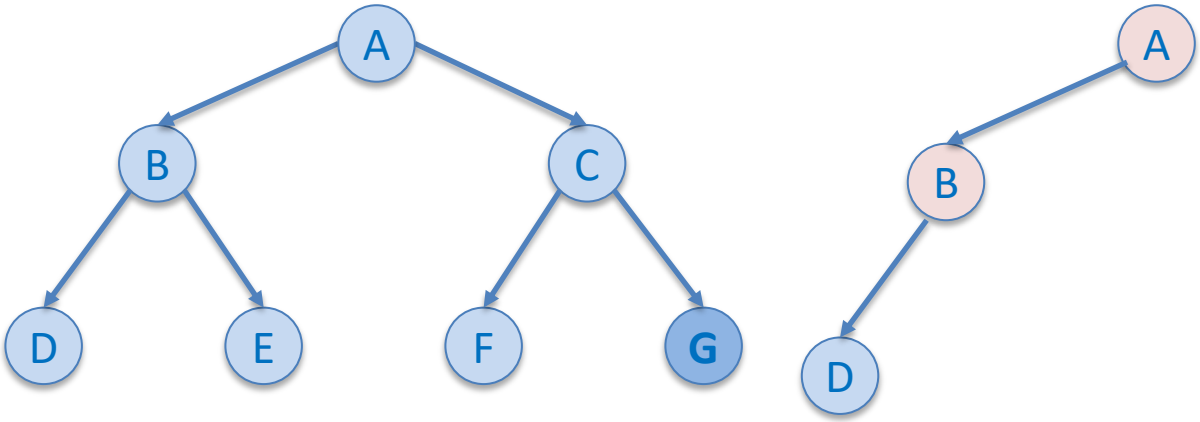
➔ Descobre os filhos.

Chamada: Nível 2



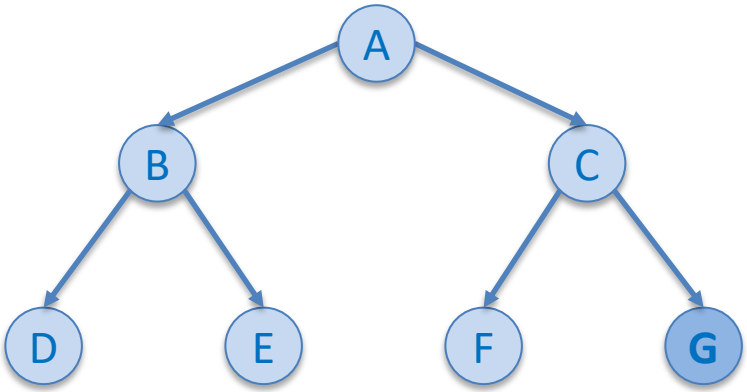


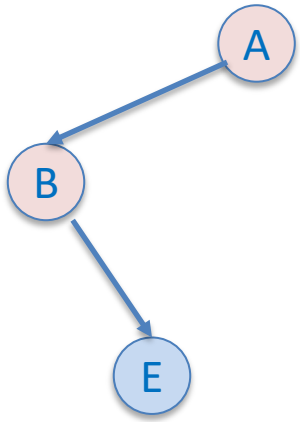
Chamada: Nível 2



Recria o Nível 0, 1: A e B não são a meta
Nível 2: D não é meta!
➔ Próximos filhos de B.

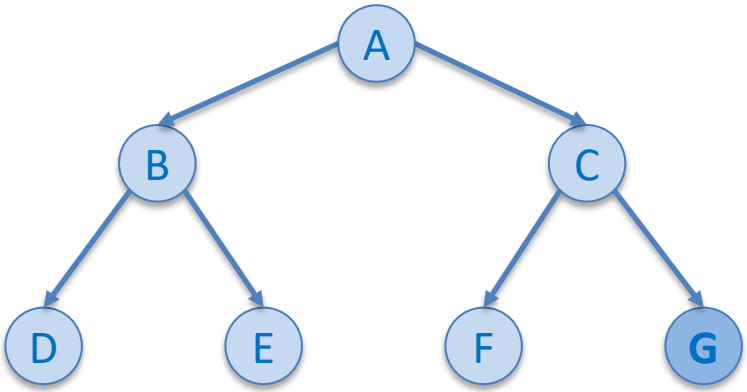
Chamada: Nível 2

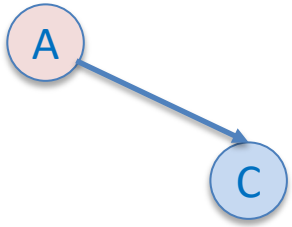




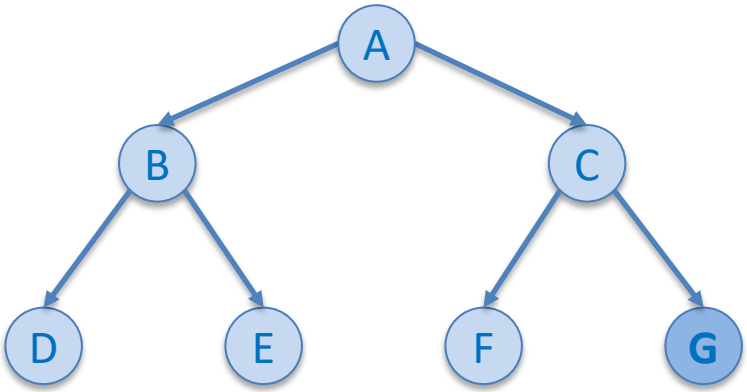
Nenhum filho de B é meta.
Volta para os outros filhos de A.

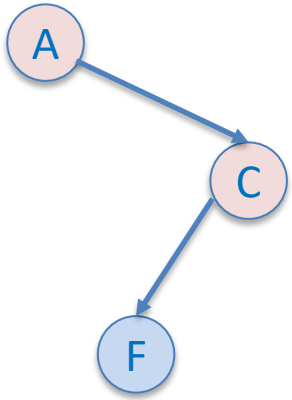
Chamada: Nível 2



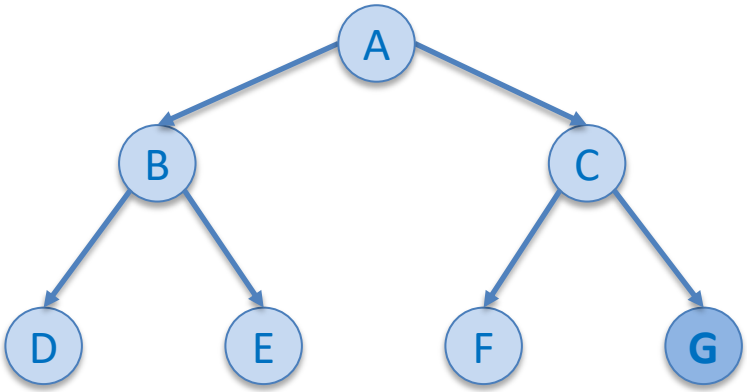


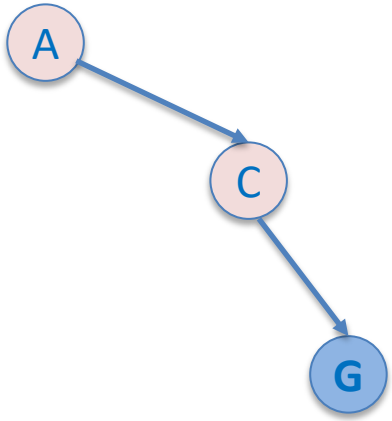
Chamada: Nível 2





Chamada: Nível 2





Encontrou a meta: Responde G

Desempenho da BAI

- **Completa**? Sim se b for finito (idem BFS).
- **Ótima**? Sim se o custo do caminho for uma função crescente com a profundidade do nó e custo dos passos for igual (idem BFS).
- **Complexidade espacial**: $\mathcal{O}(bd)$ – idem BPL (DFS com limite) com l=d
- **Complexidade temporal**: nós na profundidade da menor solução (d) são gerados 1 vez, em d-1 são gerados 2 vezes, ... na profundidade 1 são gerados d vezes:

$$(d)b + (d-1)b^2 + (d-2)b^3 + \ldots + (1)b^d = \mathcal{O}(b^d)$$

Estratégias de Busca Cega

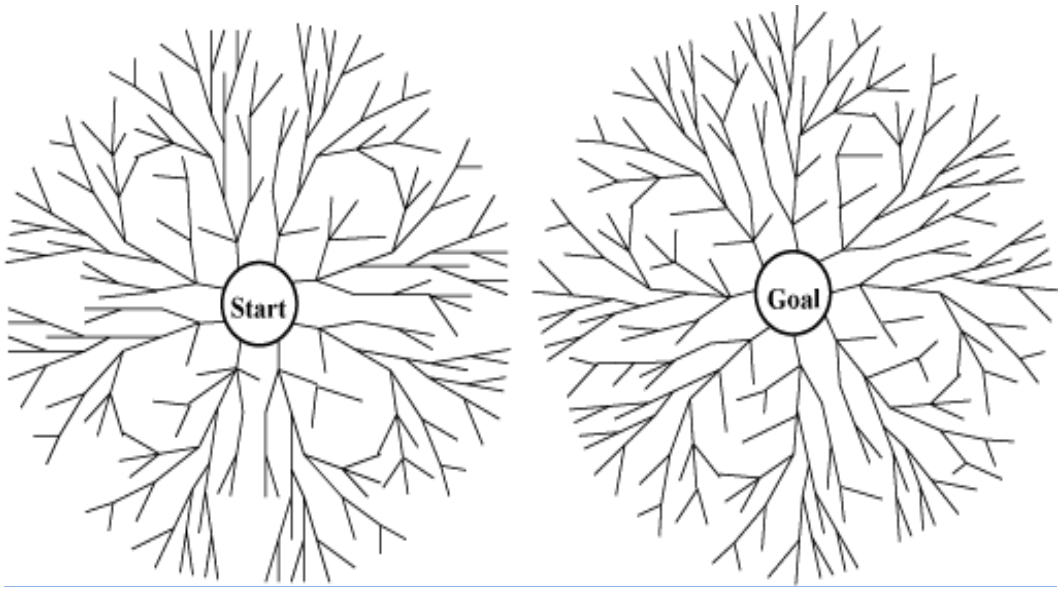
- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- **Busca Bidirecional**
- Evitando Estados Repetidos

Busca Bidirecional

- Busca em duas direções (duas buscas simultâneas):
  - para frente, a partir do nó inicial, e
  - para trás, a partir do nó final (objetivo)
- A busca para quando o nó a ser expandido por uma busca se encontra na fronteira da outra busca.
- **Motivação:**

$$b^{d/2} + b^{d/2} < b^{d}.$$

Ou: a área de dois círculos de raio R é menor que a área de um círculo de raio 2R



Busca Bidirecional

- Para BL nas duas direções: $\mathcal{O}(b^{d/2})$ no tempo e no espaço. Completeza e otimalidade: idem BL.
  - É possível utilizar *estratégias* diferentes em cada direção da busca (podendo sacrificar desempenho)

- **Encadeamento reverso** só é possível se todas as ações no espaço de estados forem **reversíveis**.
- Outro problema: quando há **vários estados-meta** ou quando é muito difícil computar os estados-meta pelo teste de término (ex. estados para cheque-mate).

Resumo Comparativo

Critério	BFS	BCU	DFS	BPL	BAI	Bidirecional (se aplicável)
Completa?	Sim[a]	Sim[a,b]	Não	Não	Sim[a]	Sim[a,d]
Ótima?	Sim[c]	Sim	Não	Não	Sim[c]	Sim[c,d]
Espaço	$\mathcal{O}(b^d)$	$\mathcal{O}(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$\mathcal{O}(bm)$	$\mathcal{O}(bl)$	$\mathcal{O}(bd)$	$\mathcal{O}(b^{d/2})$
Tempo	$\mathcal{O}(b^d)$	$\mathcal{O}(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$\mathcal{O}(b^m)$	$\mathcal{O}(b^l)$	$\mathcal{O}(b^d)$	$\mathcal{O}(b^{d/2})$

m >>>> d

l = d

a: se b for finito

b: se custo do passo > ε, com ε positivo

c: se custo dos passos forem todos idênticos

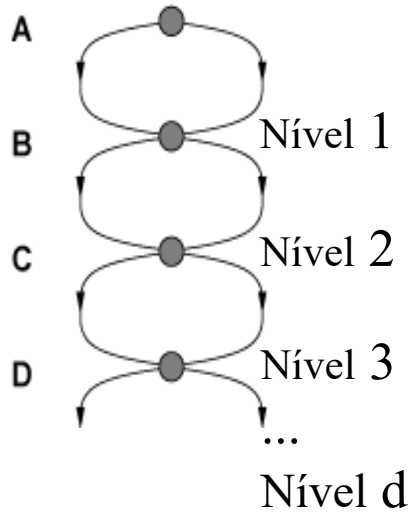
d: se ambos os sentidos usam BFS

Estratégias de Busca Cega

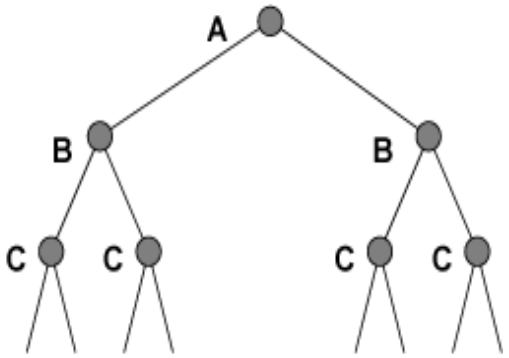
- Busca em Largura
- Busca de Custo Uniforme
- Busca em Profundidade
- Busca em Profundidade Limitada
- Busca em Profundidade com Aprofundamento Iterativo
- Busca Bidirecional
- **Evitando Estados Repetidos**

Evitando Estados Repetidos

Espaço de estados



Árvore de busca



Exemplo:

Número de estados= **d+1;**

2^d caminhos na árvore de busca (combinações possíveis de caminhos de A ao último estado, no nível d).

Ideia: podar (*prune*) estados repetidos, para gerar apenas a parte da árvore que corresponde ao grafo do espaço de estados (que é finito!)

Evitando Estados Repetidos

- **Problema**: deve armazenar <u>todos</u> nós gerados!
  - Além da lista de fronteira (também chamada de ***open list***), os algoritmos precisam da lista de nós visitados / expandidos (***closed list***)
  - Cada nó gerado é comparado com aqueles da *closed/open list*: se for repetido, **descarta aquele de caminho com custo pior.**
  - Pode ser implementado mais eficientemente com *hash tables*
  - DFS, BPL e BAI: perdem propriedade de complexidade linear no espaço.

Resumo

- Vimos como modelar um problema como busca de solução para o agente baseado em objetivos
- Vimos como formular o problema e várias estratégias para buscar a solução no espaço de estados
- Podemos flexibilizar as aplicações dando possíveis soluções para os casos de problemas **conformantes** (sem sensores), problemas **contingenciais** (alteram algo no ambiente durante a execução) e de exploração.

Ref: Russel,S.; Norvig, P. Inteligência Artificial. Tradução da 3ª. Ed. 2013. Cap. 3.

Próxima aula

Como melhorar a eficiência da busca com o uso de informação