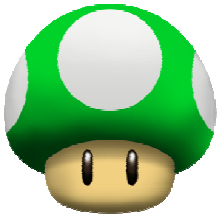
INF 1771 – Inteligência Artificial

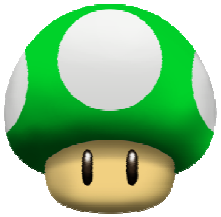
Aula 02 – Busca Cega

Edirlei Soares de Lima
<elima@inf.puc-rio.br>



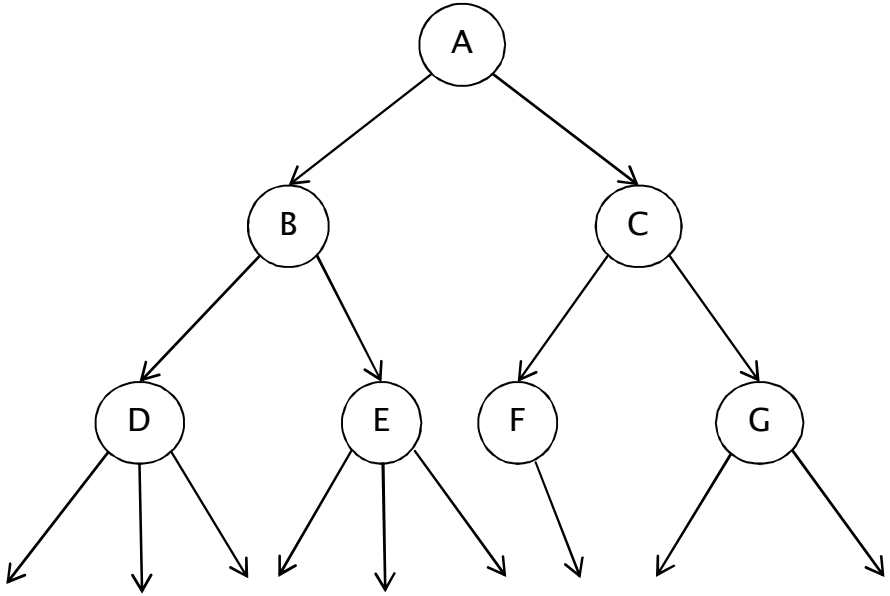
Busca Cega

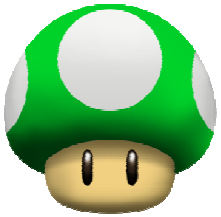
- **Algoritmos de Busca Cega:**
  - Busca em largura;
  - Busca de custo uniforme;
  - Busca em profundidade;
  - Busca com aprofundamento iterativo;



Busca em Largura

- **Estratégia:**
  - O nó raiz é expandido, em seguida todos os nós sucessores são expandidos, então todos próximos nós sucessores, e assim em diante.





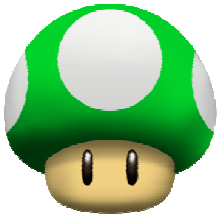
Busca em Largura

- Pode ser implementado com base no pseudocódigo da função "BuscaEmArvore" apresentado anteriormente. Utiliza-se uma estrutura de fila (first-in-first-out (FIFO)) para armazenar os nós das fronteira.

- Complexidade: $O(b^{d+1})$

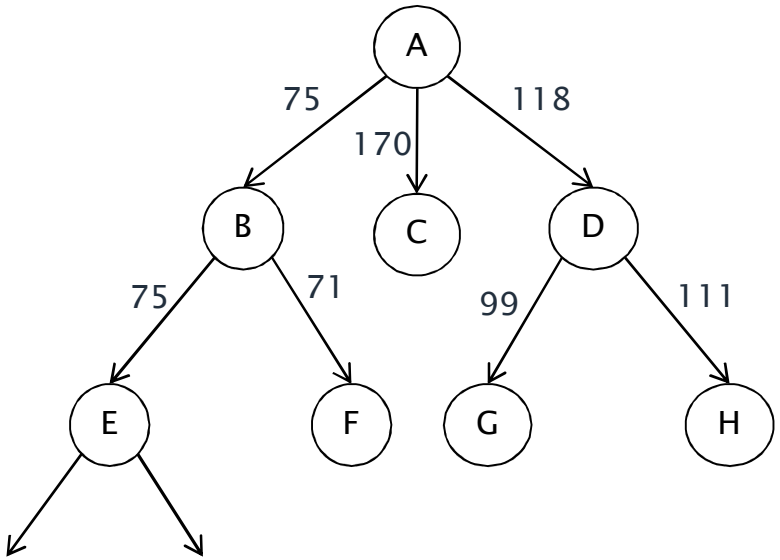
Profundidade (*d*)	**Nós**	**Tempo**	**Memória**
2	1100	0.11 ms	107 KB
4	111,100	11 ms	10.6 MB
6	10^7	1.1 seg	1 GB
8	10^9	2 min	103 GB
10	10^{11}	3 horas	10 TB
12	10^{13}	13 dias	1 PB
14	10^{15}	3.5 anos	99 PB

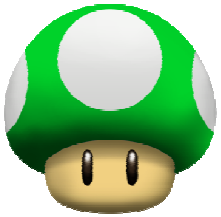
** Considerando o numero de folhas b = 10 e cada nó ocupando 1KB de memória*



Busca de Custo Uniforme

- **Estratégia:**
  - Expande sempre o nó de menor custo de caminho. Se o custo de todos os passos for o mesmo, o algoritmo acaba sendo o mesmo que a busca em largura.



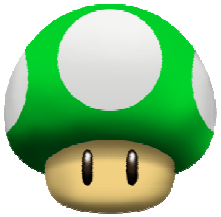


Busca de Custo Uniforme

- A primeira solução encontrada é a solução ótima se custo do caminho sempre aumentar ao logo do caminho, ou seja não existirem operadores com custo negativo.

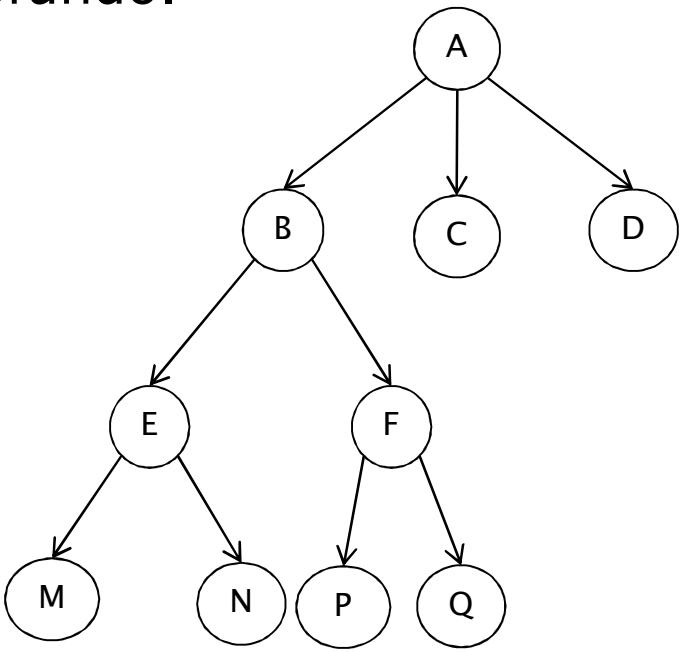
- Implementação semelhante a busca em largura. Adiciona-se uma condição de seleção dos nós a serem expandidos.

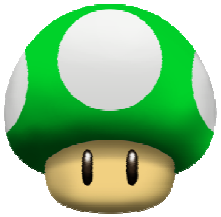
- Complexidade: $O(b^{1+(C/\alpha)})$
  - Onde:
    C = custo da solução ótima;
    a = custo mínimo de uma ação;



Busca em Profundidade

- **Estratégia:**
  - Expande os nós da vizinhança até o nó mais profundo.



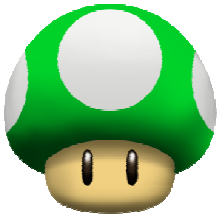


Busca em Profundidade

- Pode ser implementado com base no pseudocódigo da função “BuscaEmArvore” apresentado anteriormente. Utiliza-se uma estrutura de pilha last-in-first-out (LIFO) para armazenar os nós das fronteira.

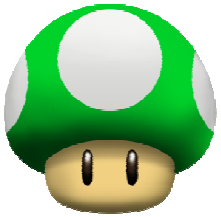
- Pode também ser implementado de forma recursiva.

- Consome pouca memória, apenas o caminho de nós sendo analisados precisa armazenado. Caminhos que já foram explorados podem ser descartados da memória.



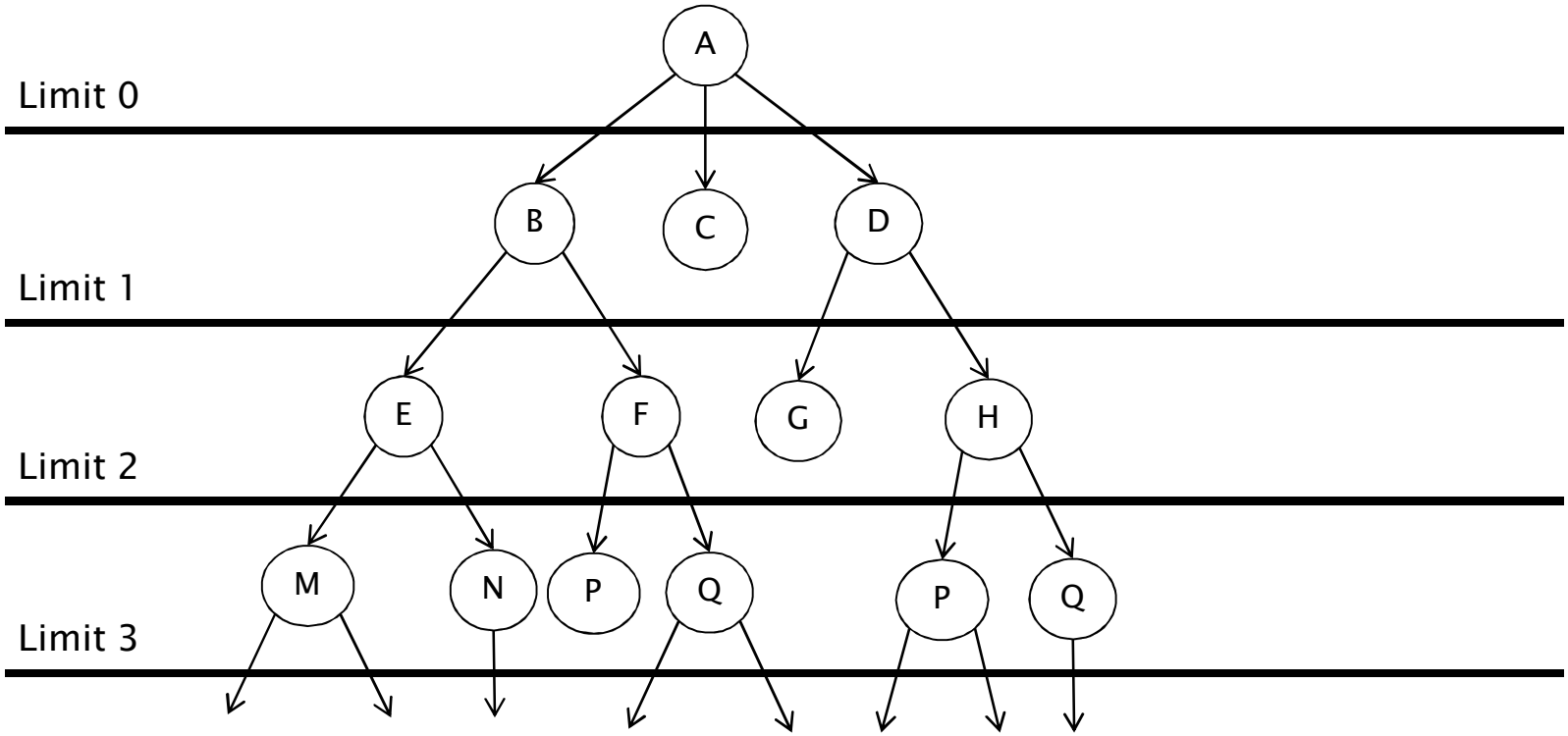
Busca em Profundidade

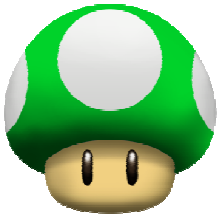
- Uso de memória pela **busca em largura** em uma arvore com 12 de profundidade: 1000 TB.
- Uso de memória pela **busca em profundidade** em uma arvore com 12 de profundidade: 118 KB.
- **Problema:** O algoritmo pode fazer uma busca muito longa mesmo quando a resposta do problema esta localizado a poucos nós da raiz da árvore.



Busca com Aprofundamento Iterativo

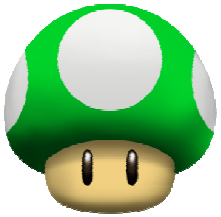
- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.





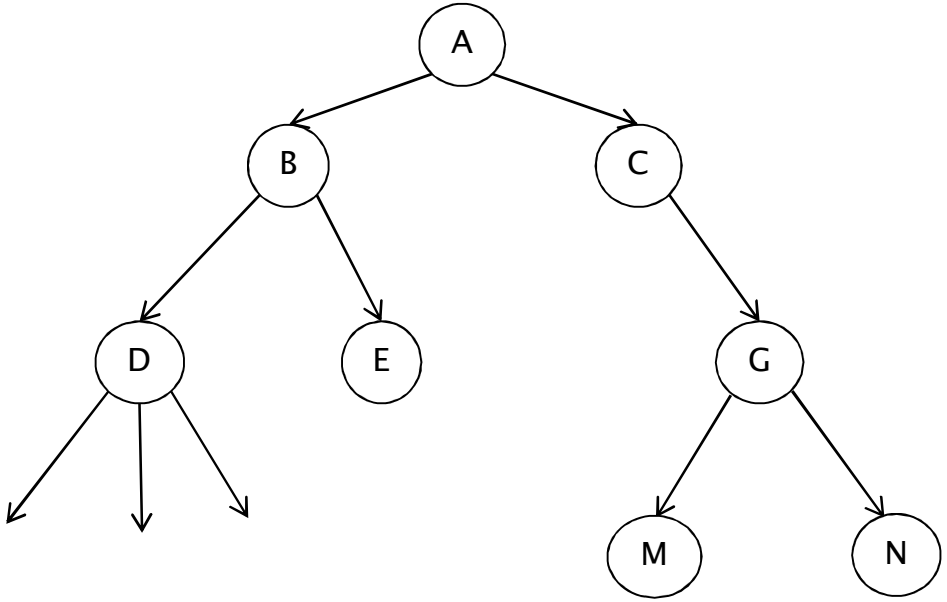
Busca com Aprofundamento Iterativo

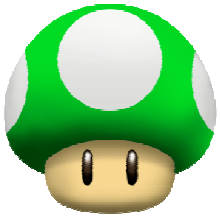
- Combina os benefícios da busca em largura com os benefícios da busca em profundidade.
- Evita o problema de caminhos muito longos ou infinitos.
- A repetição da expansão de estados não é tão ruim, pois a maior parte dos estados está nos níveis mais baixos.
- Cria menos estados que a busca em largura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



Busca Bidirecional

- **Estratégia:**
  - A busca se inicia ao mesmo tempo a partir do estado inicial e do estado final.

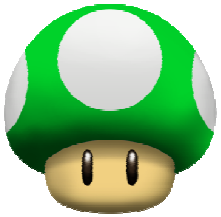




Comparação dos Metodos de Busca Cega

Criterio	Largura	Uniforme	Profundidade	Aprofundamento Iterativo	Bidirecional
Completo?	Sim [1]	Sim [1],[2]	Não	Sim [1]	Sim [1], [4]
Ótimo?	Sim [3]	Sim	Não	Sim [3]	Sim [3], [4]
Tempo	$O(b^{d+1})$	$O(b^{1+(C/\alpha)})$	$O(b^{m})$	$O(b^{d})$	$O(b^{d/2})$
Espaço	$O(b^{d+1})$	$O(b^{1+(C/\alpha)})$	$O(bm)$	$O(bd)$	$O(b^{d/2})$

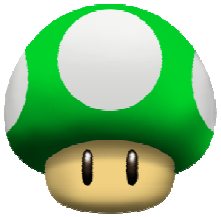
b = fator de folhas por nó.
d = profundidade da solução mais profunda.
m = profundidade máxima da árvore.
[1] completo se b for finito.
[2] completo se o custo de todos os passos for positivo.
[3] ótimo se o custo de todos os passos for idêntico.
[4] se ambas as direções usarem busca em largura.



Como evitar estados repetidos?

- Estados repetidos sempre vão ocorrer em problema onde os estados são reversíveis.

- Como evitar?
  - Não retornar ao estado "pai".
  - Não retorna a um ancestral.
  - Não gerar qualquer estado que já tenha sido criado antes (em qualquer ramo).
    - Requer que todos os estados gerados permaneçam na memória.



Exercícios

- Lista de Exercícios 1
  - http://edirlei.eternix.com.br/aulas/ia/ListaExercicios01.pdf