

AULA 02

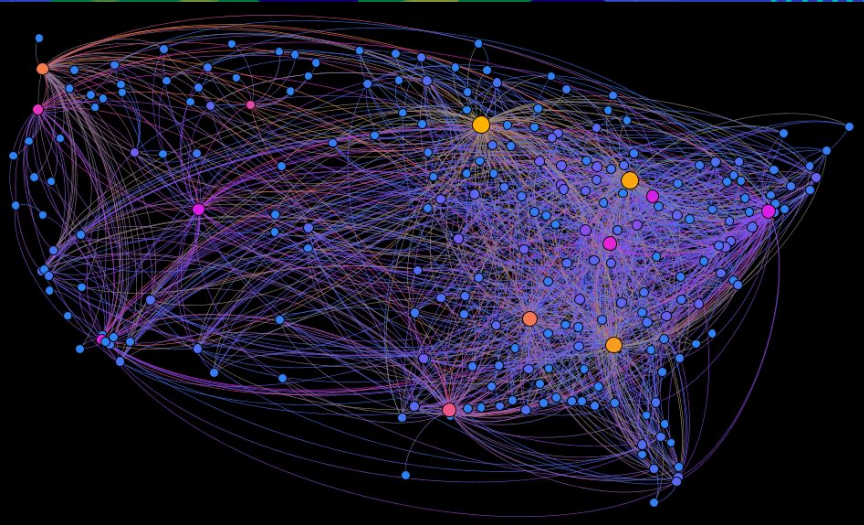
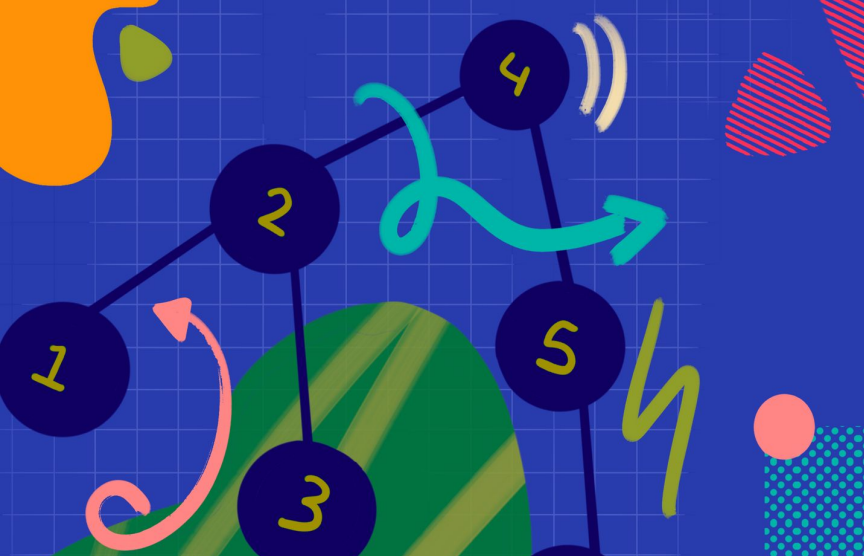
# Métodos de Busca Sem Informação

CAMILA LARANJEIRA

[mila.laranjeira@gmail.com](mailto:mila.laranjeira@gmail.com)



PUC Minas



# Agenda

- Modelos baseados em Estados
- Métodos de Busca
  - Profundidade (Depth-First Search)
  - Largura (Breadth-First Search)
  - Profundidade Iterativa
  - Custo Uniforme (Uniform Cost Search)

# Sobre os slides

Esses slides usam material de:

- José Augusto Baranauskas do Departamento de Computação e Matemática – FFCLRP-USP
- Lecture 5: Search 1 - Dynamic Programming, Uniform Cost Search | Stanford CS221: AI (Autumn 2019)
  - <https://youtu.be/alsgJJYrIXk>

# Enigma: Ajude o fazendeiro

Um **fazendeiro** quer cruzar o rio levando sua **cabra**, seu **lobo** e seu **repolho** premiado. Ele é o único que consegue pilotar o barco, e só pode levar um item a cada viagem. Mas cuidado! Não deixe o lobo sozinho com a cabra, e nem a cabra sozinha com o repolho premiado.

Quantas viagens serão necessárias?

- ☐ 4
- ☐ 5
- ☐ 6
- ☐ 7
- ☐ Sem solução

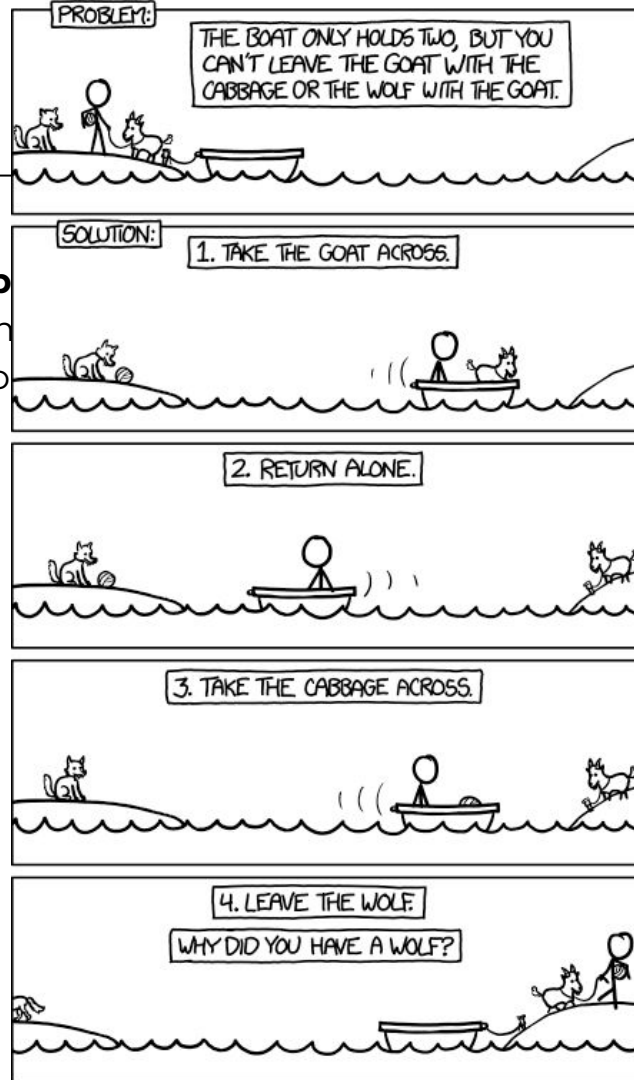


# Enigma: Ajude o fazendeiro

Um **fazendeiro** quer cruzar o rio levando sua **cabra**, seu **lobo** único que consegue pilotar o barco, e só pode levar um item. Se deixar o lobo sozinho com a cabra, e nem a cabra sozinha com o

Quantas viagens serão necessárias?

- ☒ 4 (ツ)
- ☐ 5
- ☐ 6
- ☐ 7
- ☐ Sem solução



# Enigma: Ajude o fazendeiro

Um **fazendeiro** quer cruzar o rio levando sua **cabra**, seu **lobo** e seu **repolho** premiado. Ele é o único que consegue pilotar o barco, e só pode levar um item a cada viagem. Mas cuidado! Não deixe o lobo sozinho com a cabra, e nem a cabra sozinha com o repolho premiado.

Quantas viagens serão necessárias?

- ☐ 4
- ☐ 5
- ☐ 6
- ☐ 7
- ☐ Sem solução



# Problemas baseados em Estados

Alguns problemas podem ser modelados como um conjunto de **ações** e transições entre **estados**.

- Um algoritmo inteligente deve encontrar o conjunto de estados que maximiza alguma medida de desempenho.

# Problemas baseados em Estados

Alguns problemas podem ser modelados como um conjunto de **ações** e transições entre **estados**.

- Um algoritmo inteligente deve encontrar o conjunto de estados que maximiza alguma medida de desempenho.
- Exemplo: quebra-cabeça-8

5	4	
6	1	8
7	3	2

Estado Inicial

1	2	3
8		4
7	6	5

Estado Final

- Quais os estados válidos?
- Quais as ações disponíveis?
- Quais os estados objetivo?
- Qual o custo de transição entre estados?



# Problemas baseados em Estados

- Exemplo: quebra-cabeça-8

5	4	
6	1	8
7	3	2

Estado Inicial

1	2	3
8		4
7	6	5

Estado Final

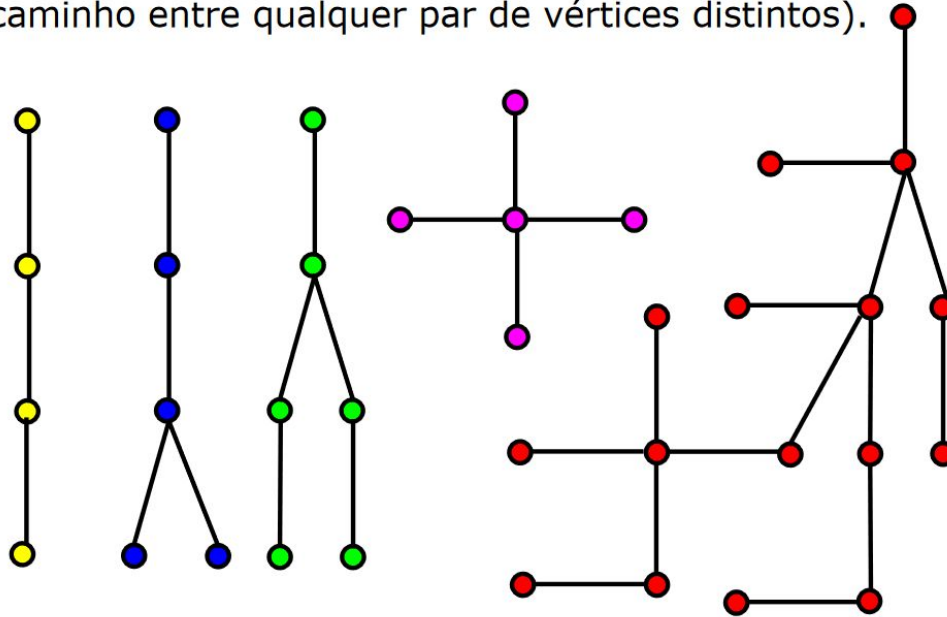
- Quais os estados válidos?
  - Cada número deve ocupar um espaço inteiro
- Quais as ações disponíveis?
  - Cima, baixo, esquerda, direita
- Quais os estados objetivo?
  - Números ordenados
- Qual o custo de transição entre estados?
  - Constante

Minimizar o número  
de transições/ações

# Árvore - Grafo acíclico

- Podemos explorar as ações e transições entre estados usando a estrutura de uma árvore

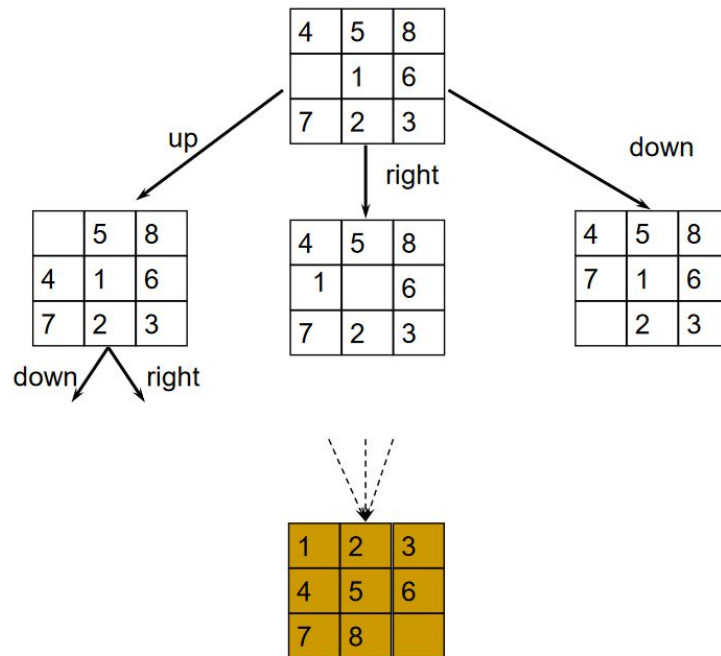
Grafo acíclico (não possui ciclos) e conexo (existe um caminho entre qualquer par de vértices distintos).



- Vértices: estados
- Arestas: transições causadas por ações
- Custo da ação define o peso da aresta.
- Folhas da árvore podem conter estados finais (soluções)

# Árvore - Grafo acíclico

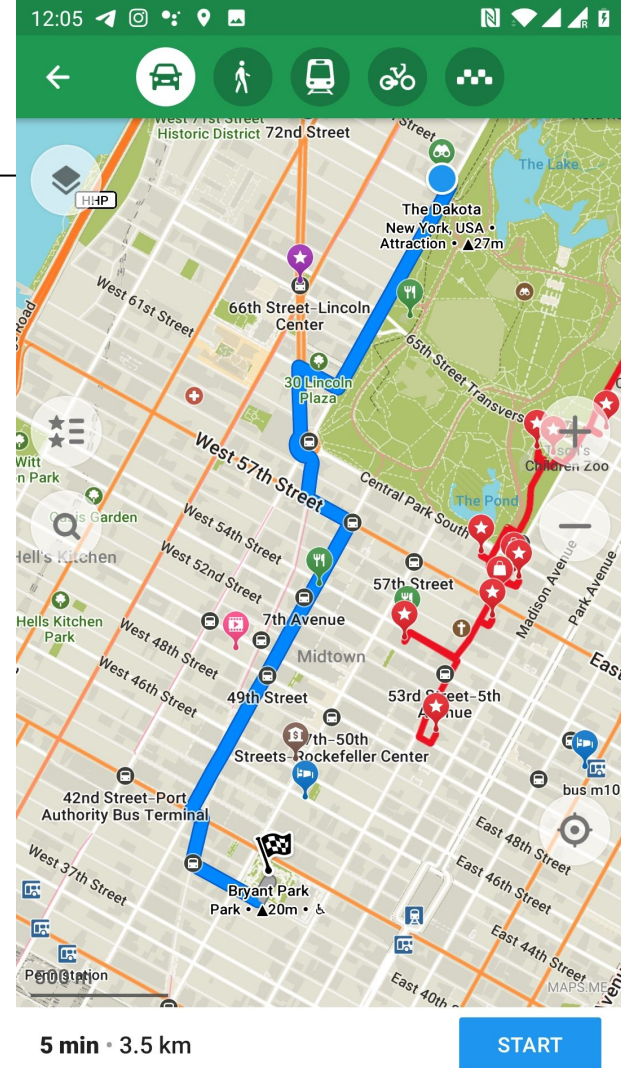
- Podemos explorar as ações e transições entre estados usando a estrutura de uma árvore



- Vértices: estados
- Arestas: transições causadas por ações
- Custo da ação define o peso da aresta.
- Folhas da árvore podem conter estados finais (soluções)

# Aplicações

- GPS - escolhendo rotas
- Custos de caminho
  - Mais rápido?
  - Mais curto?
  - Mais bonito?
- Ações
  - Seguir reto
  - Virar direita
  - Virar esquerda



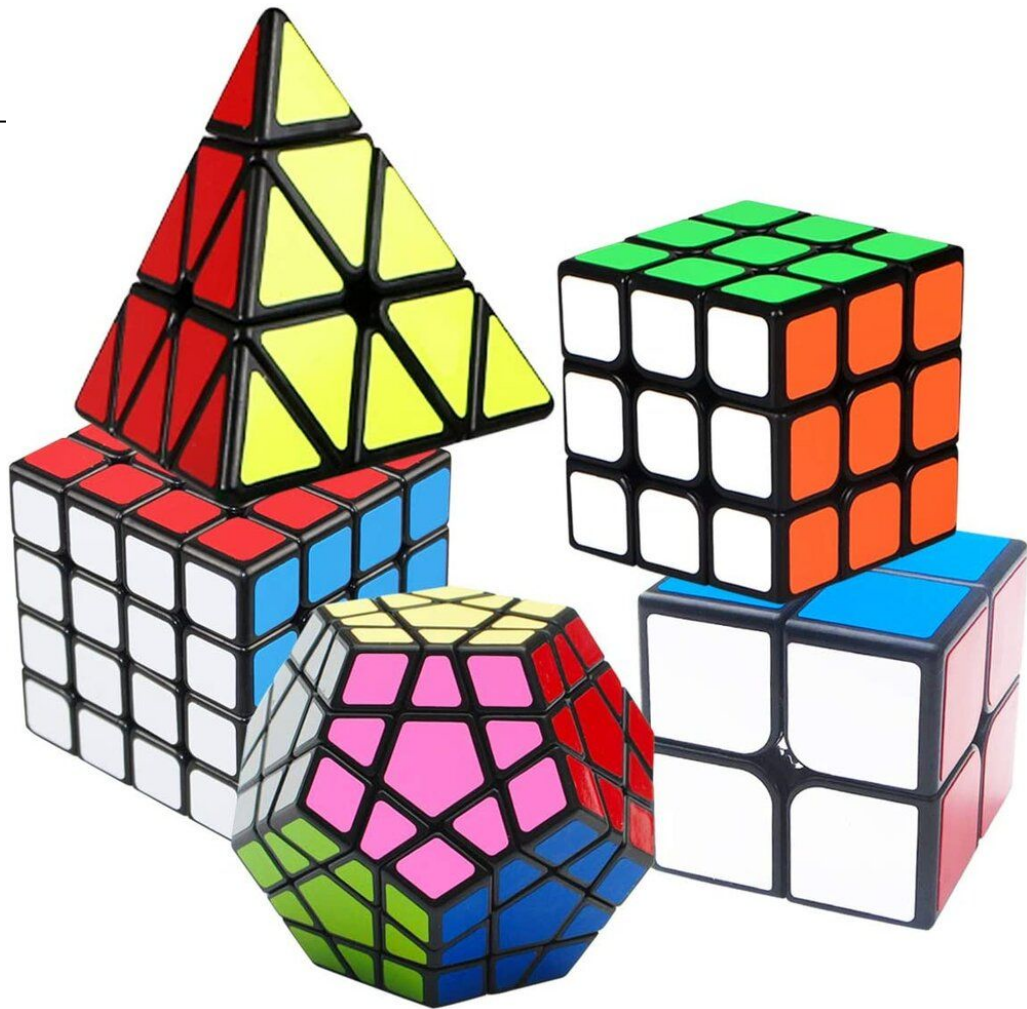
# Aplicações

- Robótica: movimento de um manipulador
- Custo
  - Mais rápido?
  - Mais econômico?
  - Mais seguro?
- Ações
  - Velocidade linear
  - Velocidade angular



# Aplicações

- Quebra-cabeças
- Custo
  - Constante
- Ações
  - Mover uma peça





# Problemas baseados em Estados



Farmer - Fazendeiro

Cabbage - Repolho

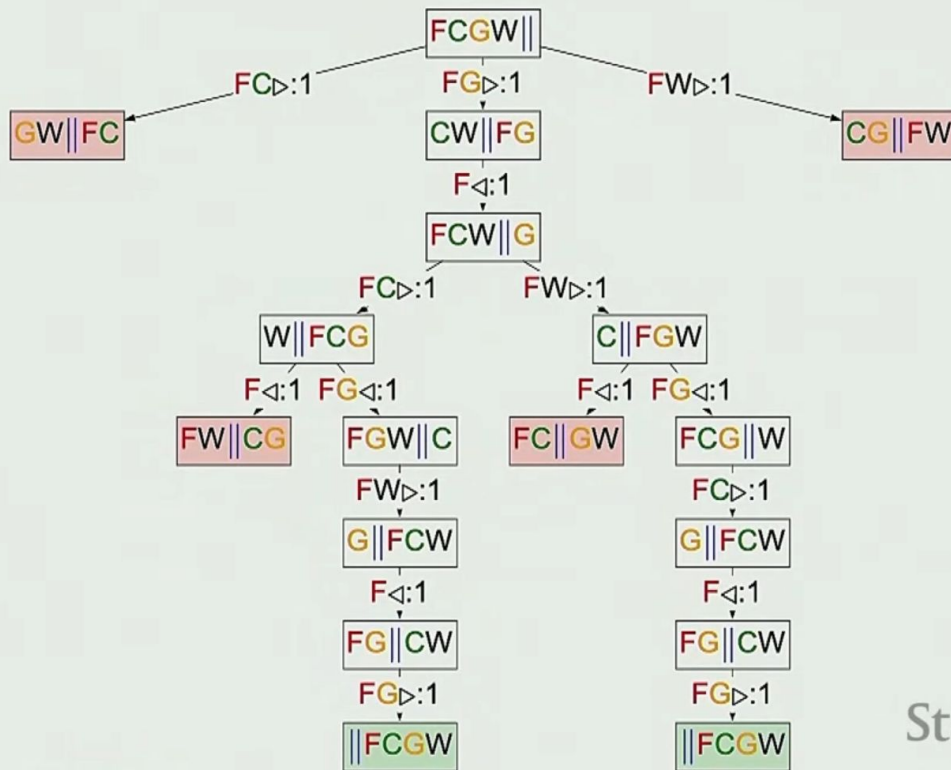
Goat - Cabra

Wolf - Lobo

Lecture 5: Search 1 - Dynamic  
Programming, Uniform Cost  
Search | Stanford CS221: AI  
(Autumn 2019)

<https://youtu.be/alsqJJYrIXk>

# Problemas baseados em Estados

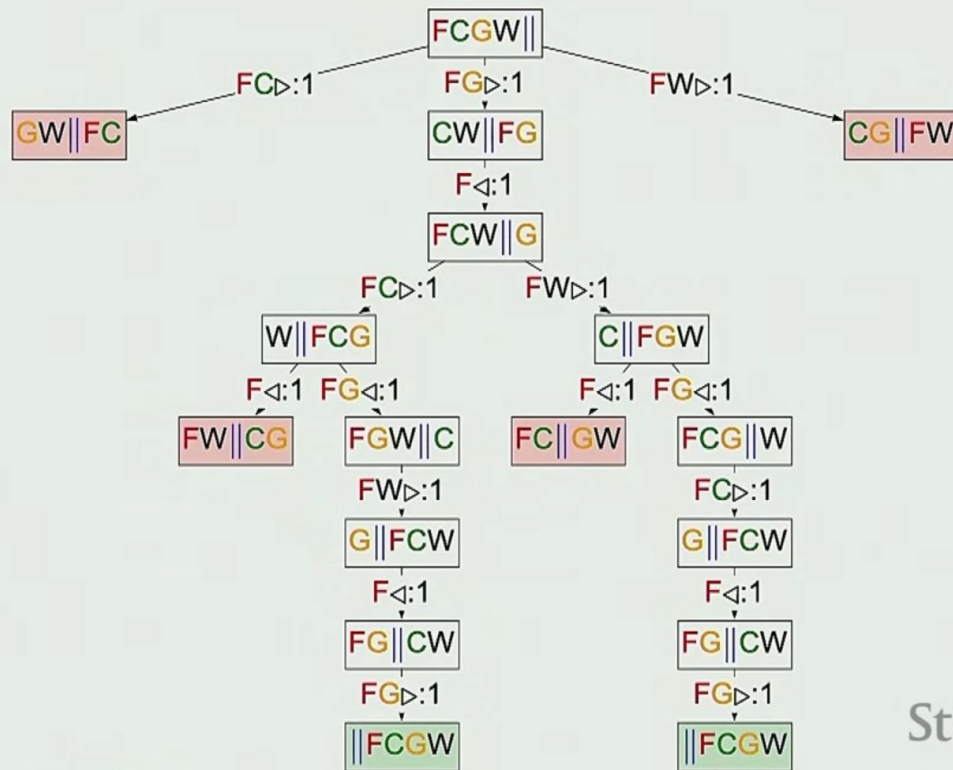


Farmer - Fazendeiro  
Cabbage - Repolho  
Goat - Cabra  
Wolf - Lobo

Stanford



# Problemas baseados em Estados



Quantas viagens serão necessárias?

☐ 4

☐ 5

☐ 6

☒ 7

☐ Sem solução

Stanford

# Algoritmos de busca no espaço de estados

O problema de navegar no espaço de busca então pode ser entendido como **gerar e manter uma árvore de busca** até que uma solução (nó final) ou caminho seja gerado.

- Busca sem informação (busca cega)
  - Somente geram sucessores e distinguem se é o objetivo ou não
- Busca com informação (busca heurística)
  - Sabem se um estado é mais promissor que outro, se está mais próximo da solução.

# Algoritmos de busca cega

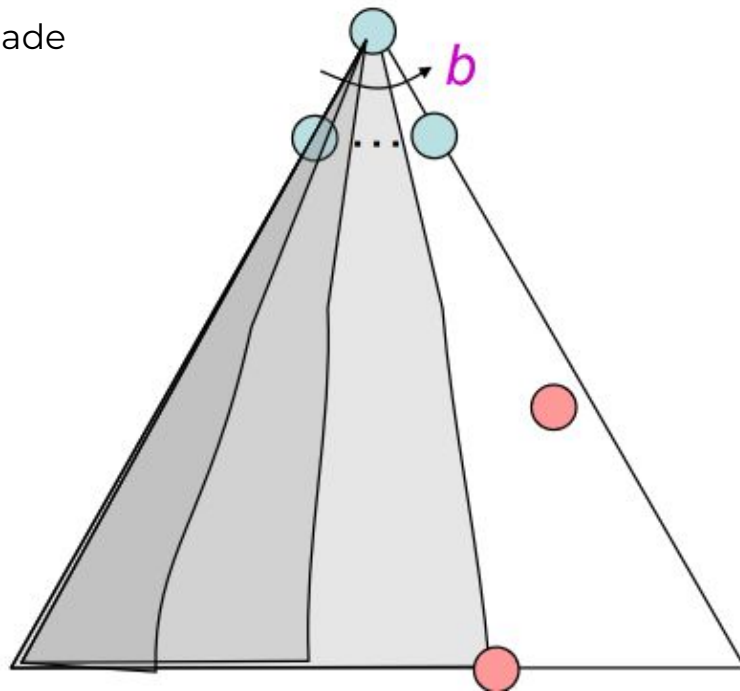
- Métodos de Busca
  - Profundidade (Depth-First Search)
  - Largura (Breadth-First Search)
  - Profundidade Iterativa
  - Custo Uniforme (Uniform Cost Search)

# Algoritmos de busca cega

Algoritmo	Custo	Tempo	Espaço
DFS-Backtracking			
DFS			
DFS-I			
BFS			

# DFS - Backtracking

- Depth-First Search: Busca em profundidade
- Se concentra em buscar primeiro na profundidade
- Dado um vértice  $v$  recentemente descoberto, explora todas as arestas saindo dele.
- Ao finalizar a exploração de  $v$ , “anda para trás” (“backtracks”) para explorar o restante
- A busca termina quando todos os vértices forem descobertos



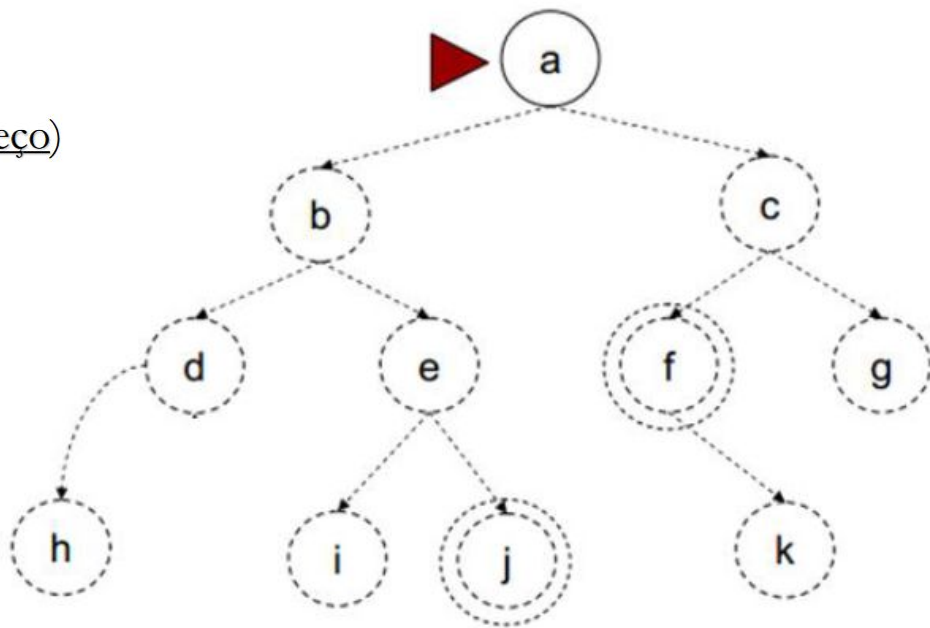
# DFS - Backtracking

**função Busca-em-Profundidade (*problema*)**

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)

*Inserir no início e remove do início,  
que estrutura de dados é essa?*

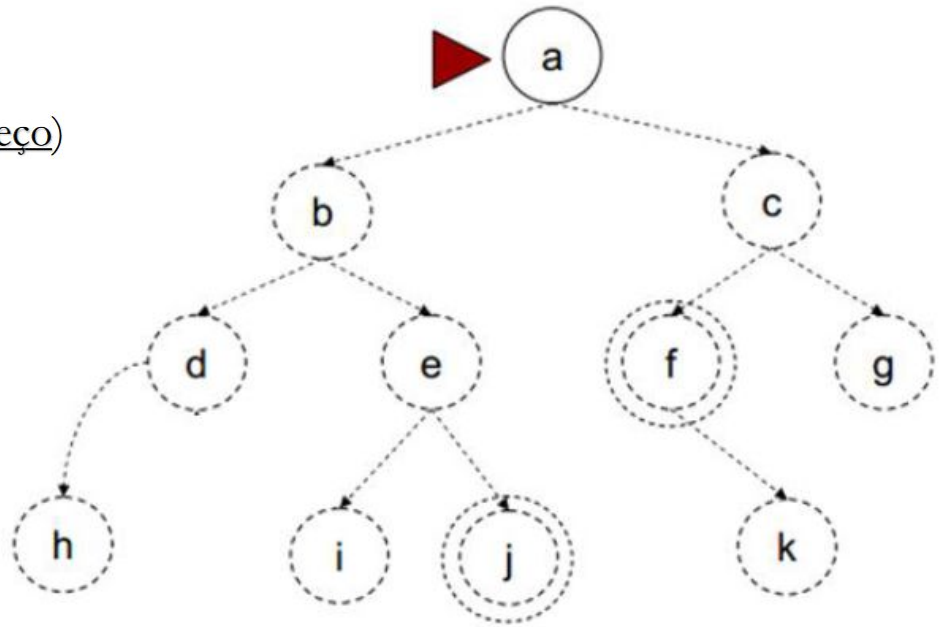


# DFS - Backtracking

**função** Busca-em-Profundidade (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)

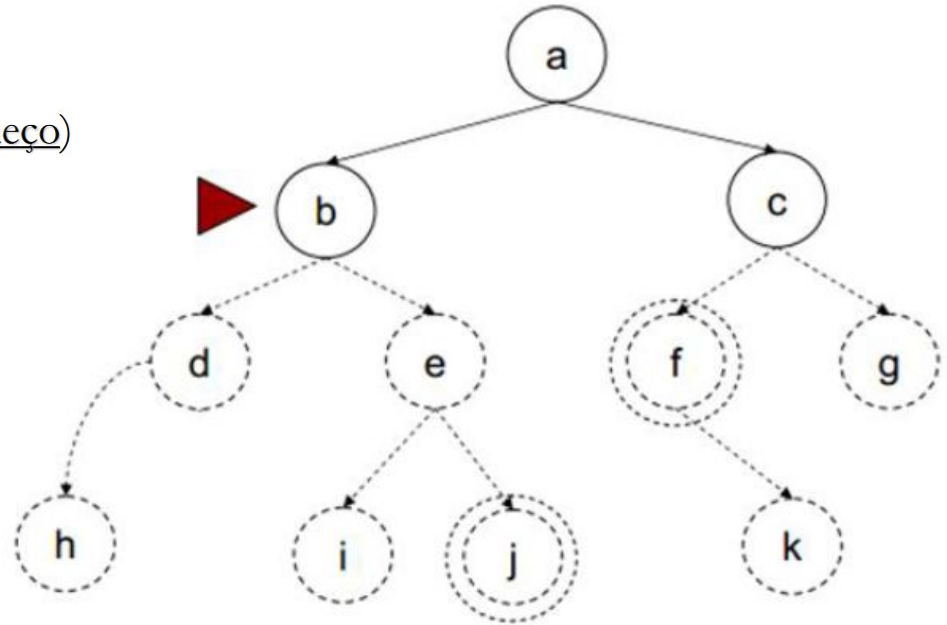


# DFS - Backtracking

**função** Busca-em-Profundidade (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)



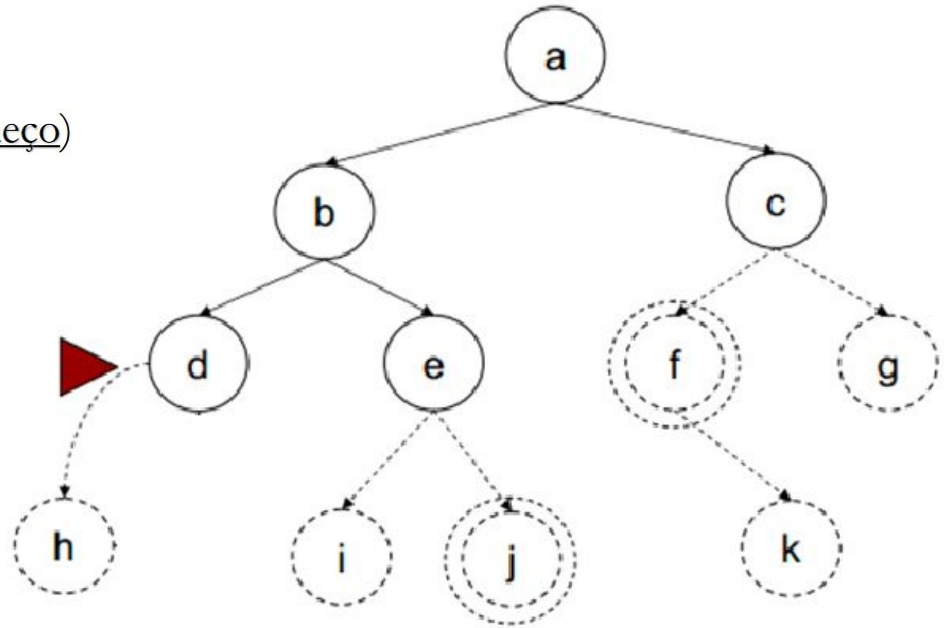


# DFS - Backtracking

**função** Busca-em-Profundidade (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)

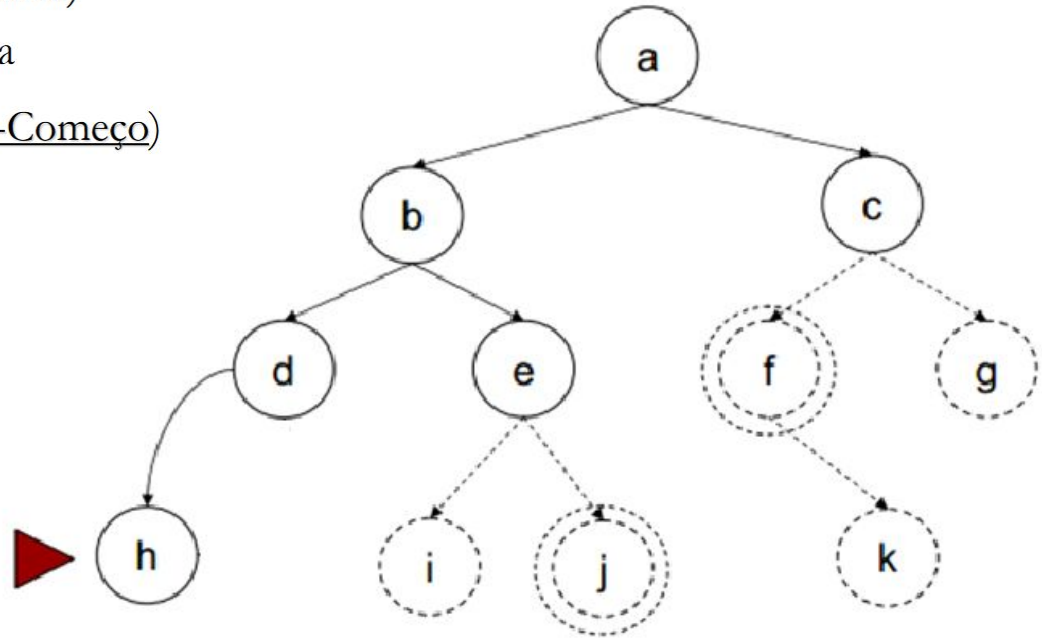


# DFS - Backtracking

**função** Busca-em-Profundidade (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)

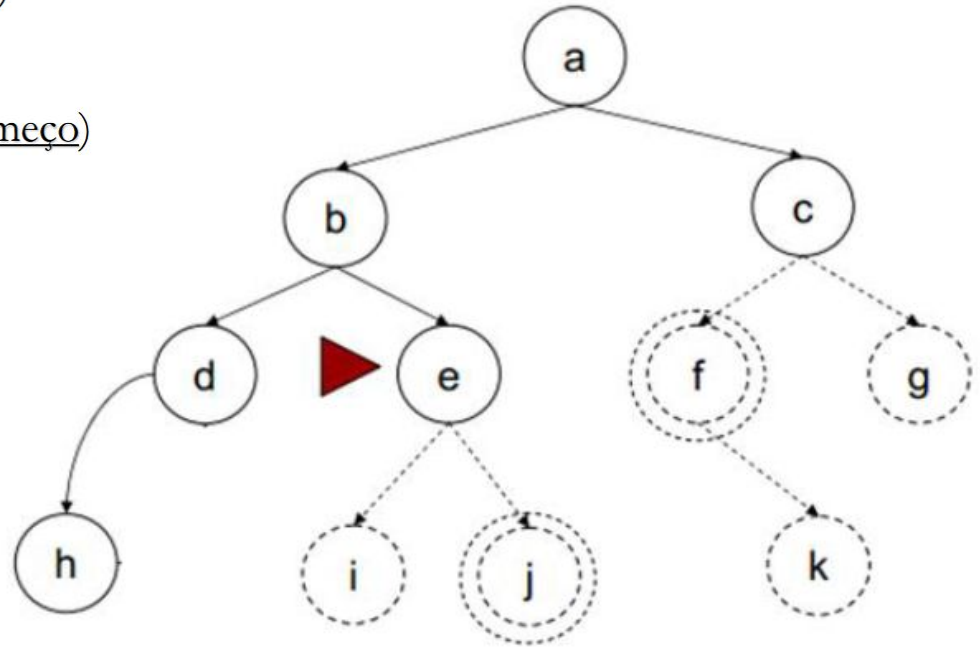


# DFS - Backtracking

**função** Busca-em-Profundidade (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)

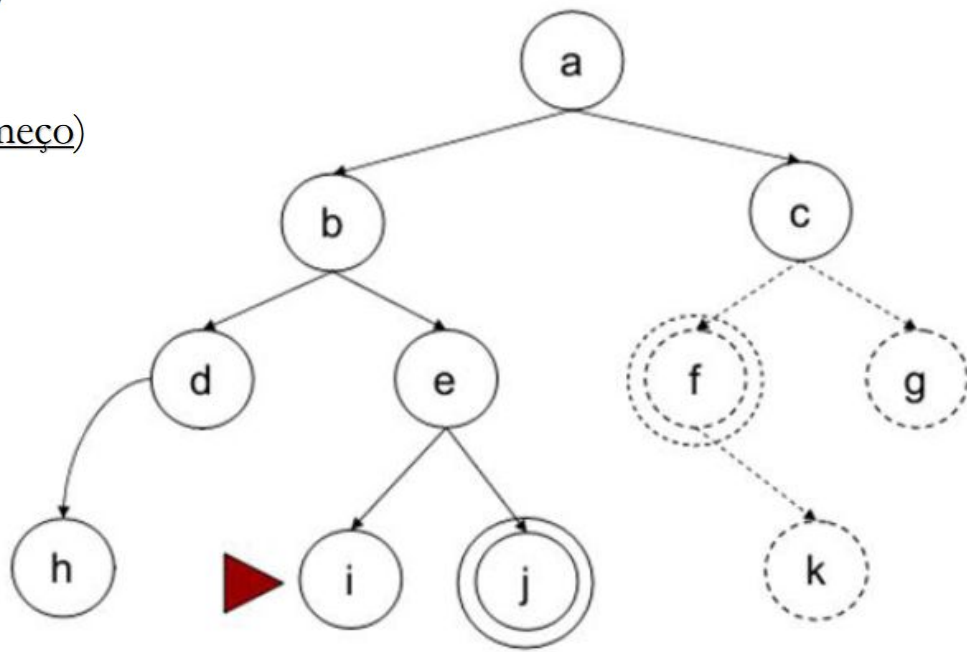


# DFS - Backtracking

**função** Busca-em-Profundidade (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)

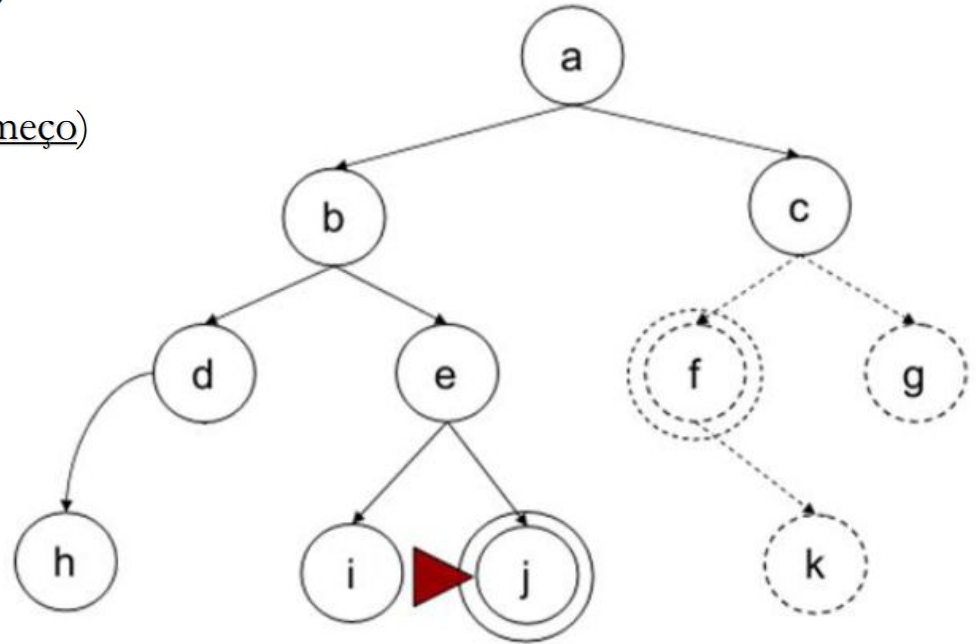


# DFS - Backtracking

**função** Busca-em-Profundidade (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)

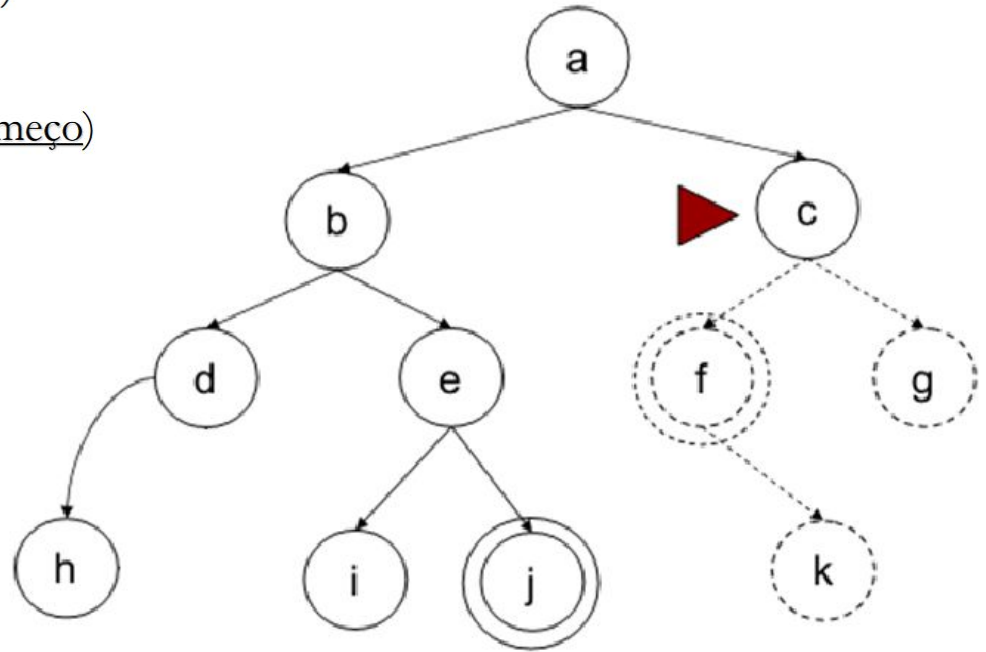


# DFS - Backtracking

**função** Busca-em-Profundidade (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)

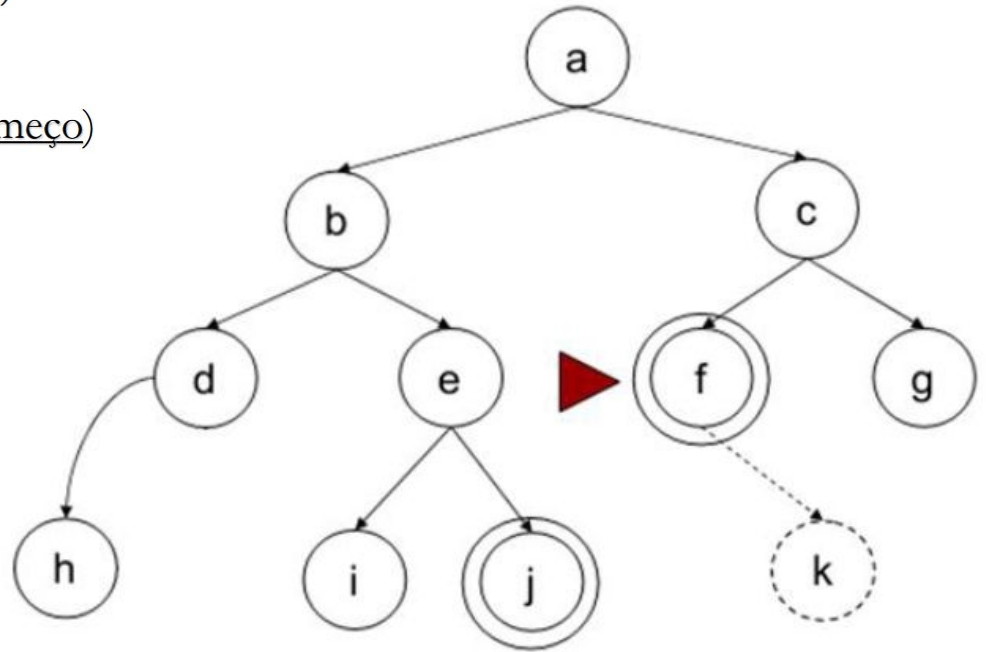


# DFS - Backtracking

**função** Busca-em-Profundidade (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)

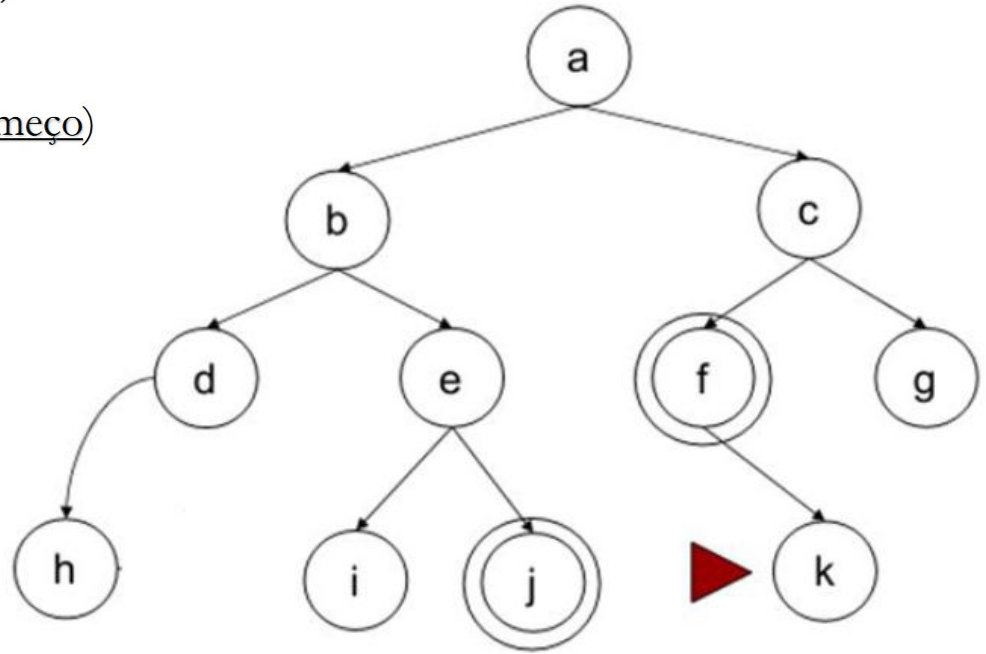


# DFS - Backtracking

**função** Busca-em-Profundidade (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)



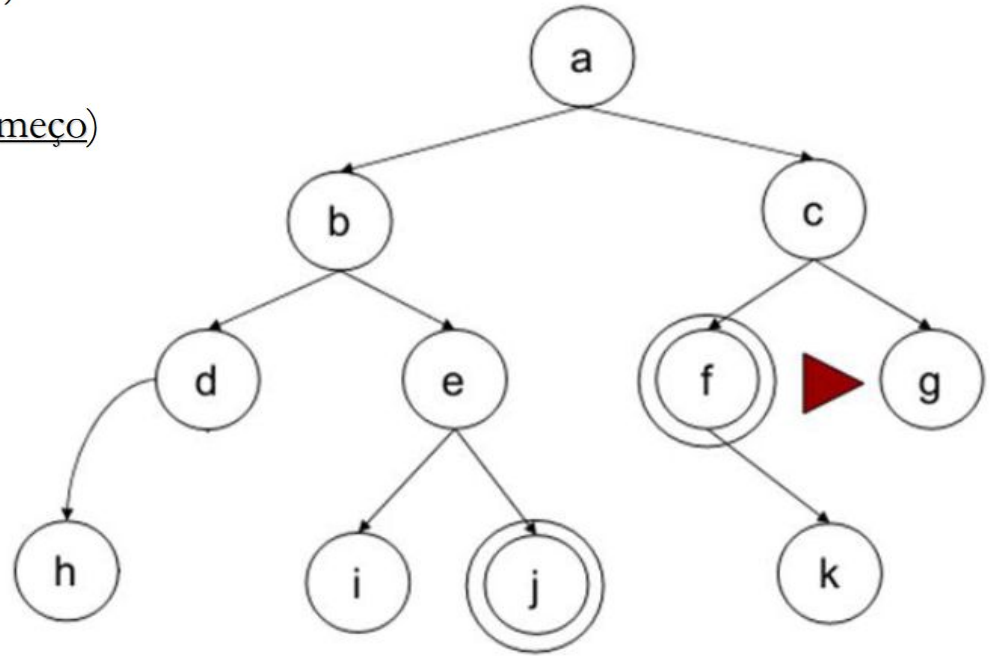


# DFS - Backtracking

**função** Busca-em-Profundidade (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)

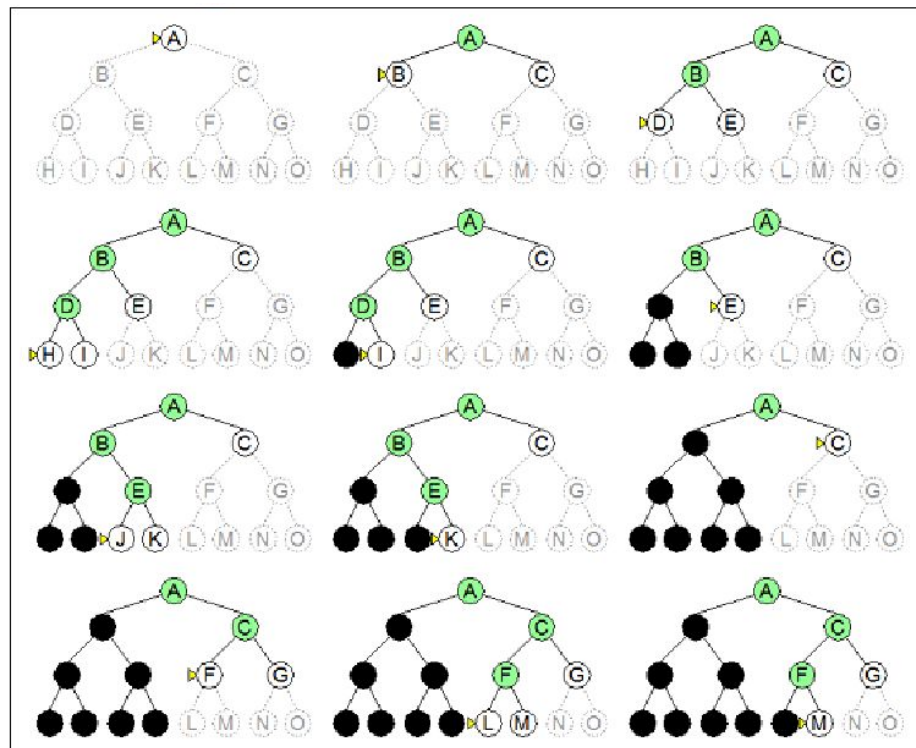


# DFS - Backtracking

função Busca-em-Profundidade (*problema*)

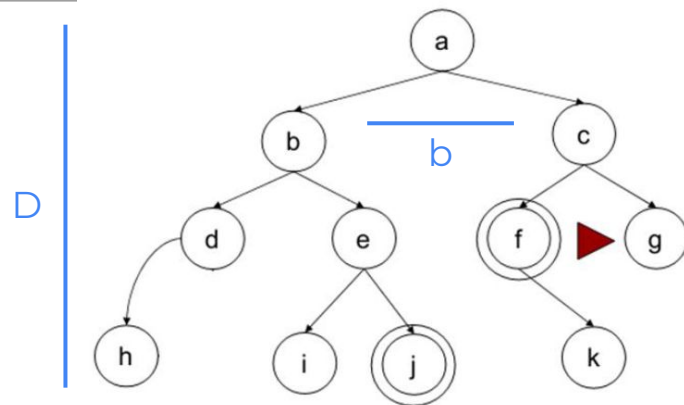
retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-no-Começo)



# Algoritmos de busca cega

Algoritmo	Custo	Tempo	Espaço
DFS-Backtracking	Qualquer	$O(b^D)$	$O(D)$
DFS			
DFS-I			
BFS			



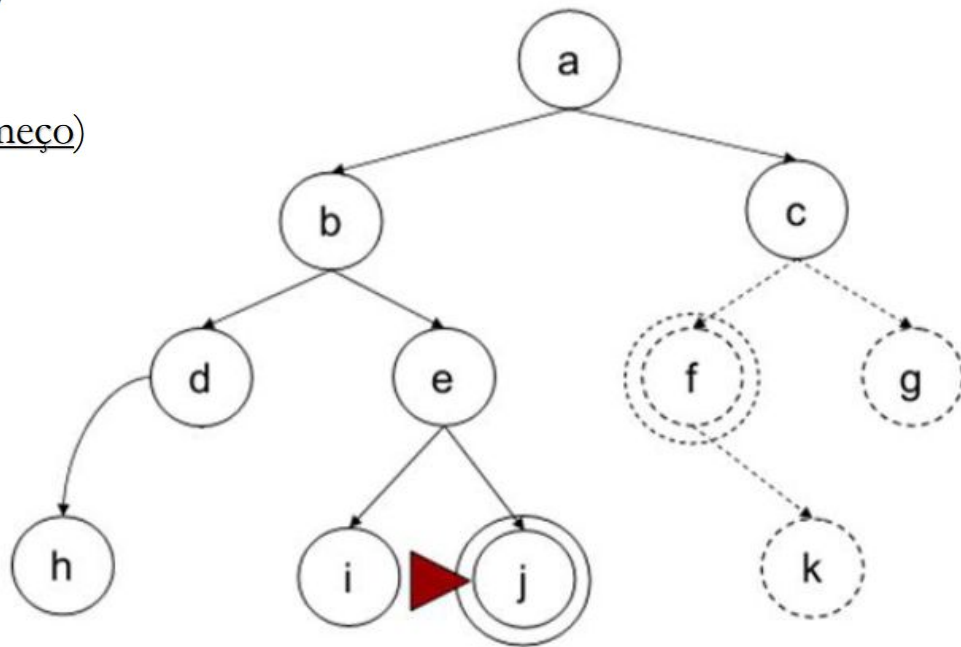
# DFS

**função Busca-em-Profundidade (*problema*)**

retorna uma solução ou falha

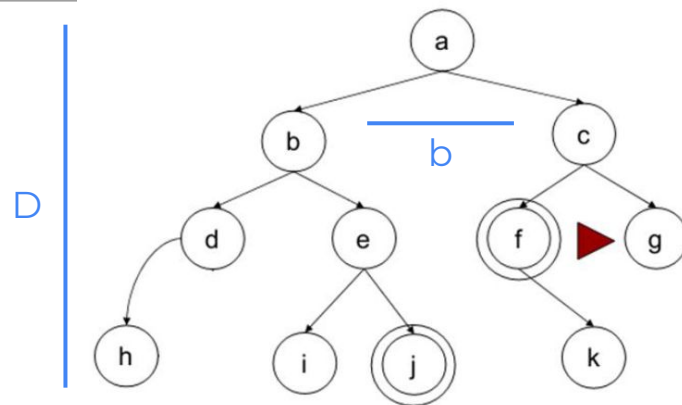
Busca-Genérica (*problema*, Inserir-no-Começo)

- **Assumindo custo 0 (zero)**  
podemos encerrar o algoritmo  
ao encontrar a primeira solução.
  - Sem backtrack



# Algoritmos de busca cega

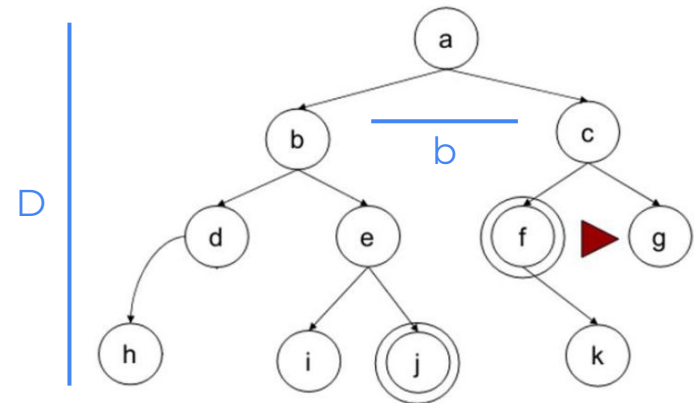
Algoritmo	Custo	Tempo	Espaço
DFS-Backtracking	Qualquer	$O(b^D)$	$O(D)$
DFS	$\emptyset$	$O(b^D)$	$O(D)$
DFS-I			
BFS			



# Algoritmos de busca cega

Algoritmo	Custo	Tempo	Espaço
DFS-Backtracking	Qualquer	$O(b^D)$	$O(D)$
DFS	$\emptyset$	$O(b^D)$	$O(D)$
DFS-I			
BFS			

A complexidade de tempo não melhora para o pior caso, mas sim para o caso médio e melhor caso

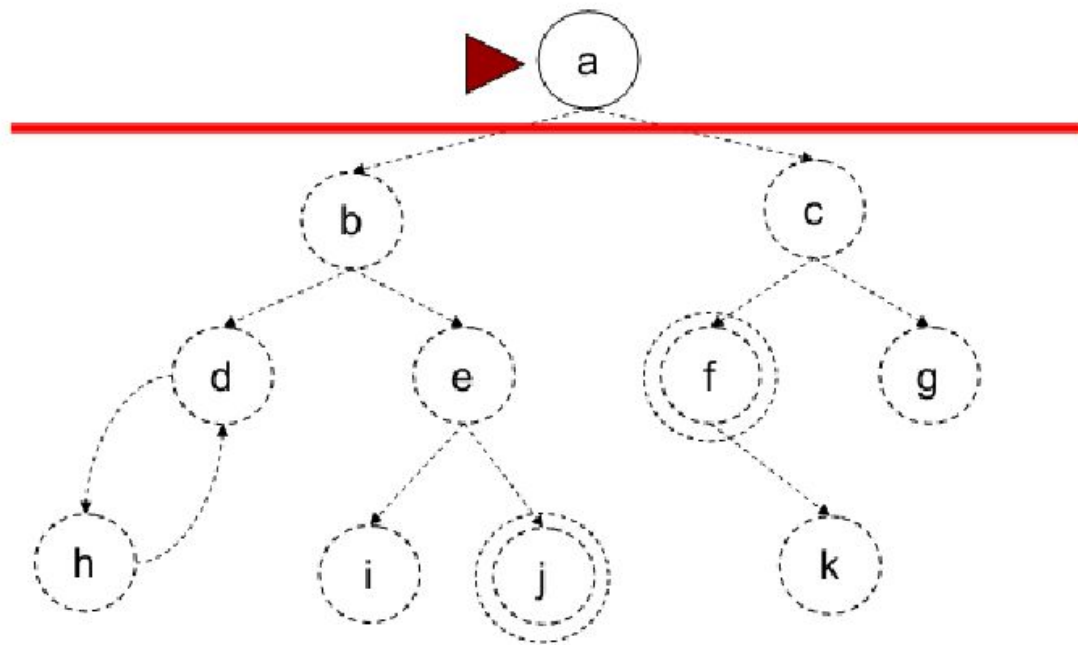


# DFS - Limitações

- Dependendo do problema, podem existir espaços de estado infinitos
  - Lembre-se, estamos gerando uma árvore de busca e descobrindo novos caminhos
- Em um caso menos extremo, o algoritmo pode perder muito tempo explorando regiões muito profundas e pouco promissoras
- Para torná-lo mais eficiente, podemos **limitar a profundidade da busca**.

# DFS Limitado

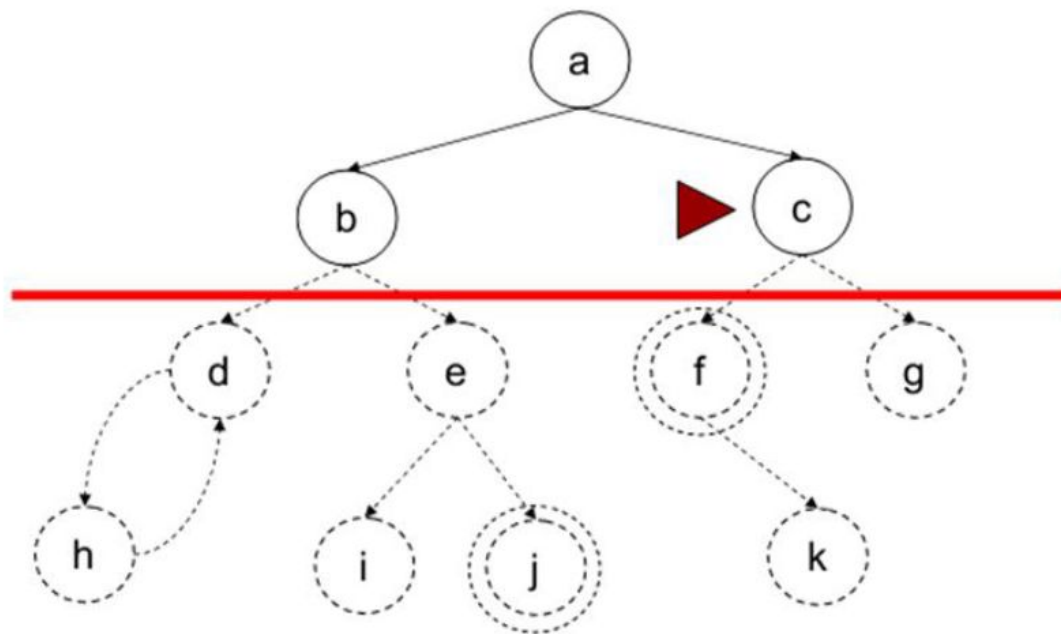
- Limite 1 = 0





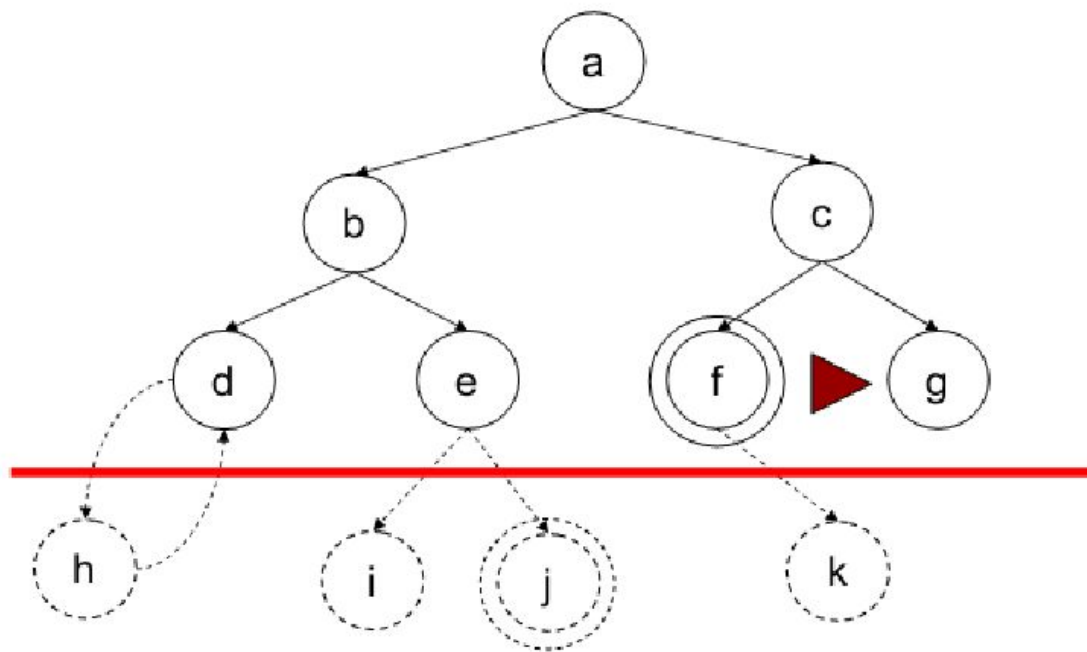
# DFS Limitado

- Limite 1 = 1



# DFS Limitado

- Limite 1 = 2

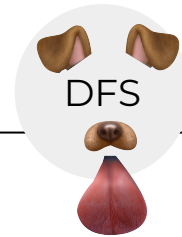


# DFS Limitado Iterativo

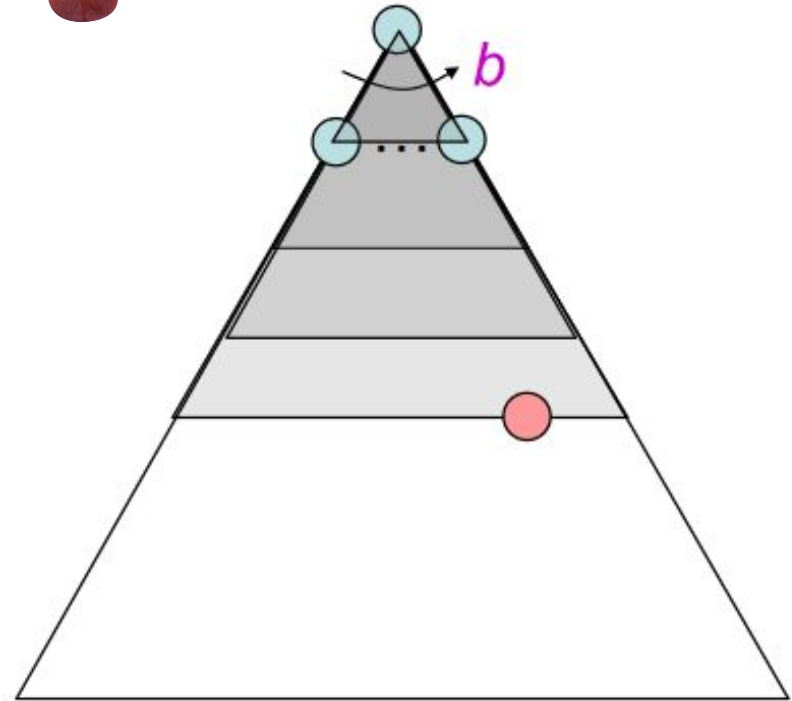
- É difícil estimar um limite razoável
  - Se for muito pequeno a busca falha
  - Se for muito grande, aumenta a complexidade
- Solução: Busca em profundidade **limitada iterativa**
  - Defina um limite razoável, tendendo para pequeno, e gradativamente o aumente.



# DFS Limitado Iterativo

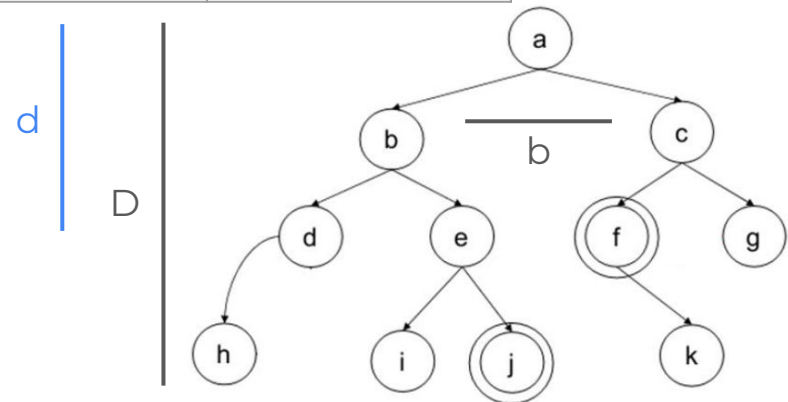


- É difícil estimar um limite razoável
  - Se for muito pequeno a busca falha
  - Se for muito grande, aumenta a complexidade
- Solução: Busca em profundidade **limitada iterativa**
  - Defina um limite razoável, tendendo para pequeno, e gradativamente o aumente.



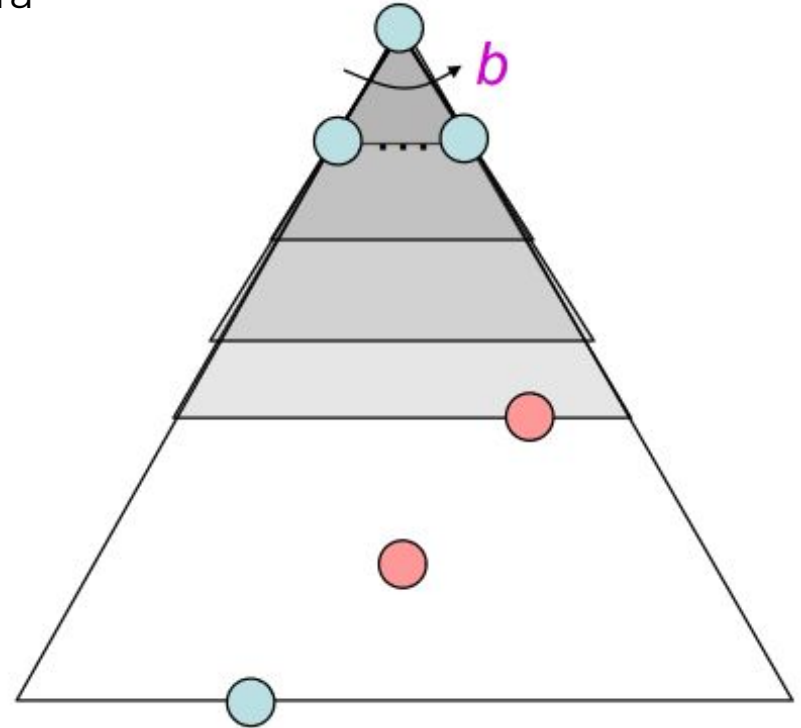
# Algoritmos de busca cega

Algoritmo	Custo	Tempo	Espaço
DFS-Backtracking	Qualquer	$O(b^D)$	$O(D)$
DFS	$\emptyset$	$O(b^D)$	$O(D)$
DFS-I	$c \geq \emptyset$	$O(b^d)$	$O(D)$
BFS			



# BFS

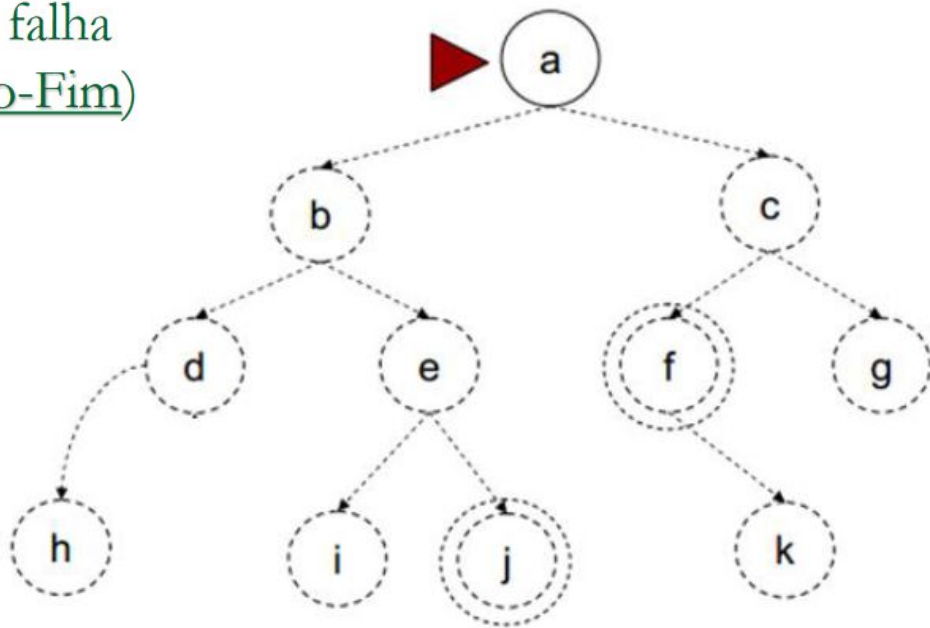
- Breadth-First Search: Busca em largura
- Se concentra em buscar primeiro na largura
- Visita todos os filhos de um vértice  $v$  qualquer antes de explorar novas fronteiras



# BFS

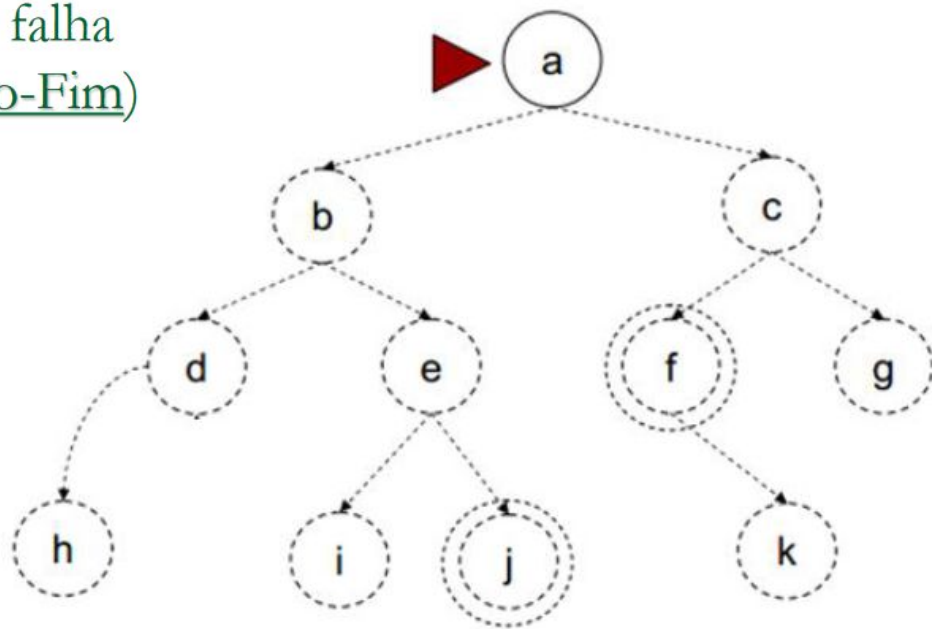
função Busca-em-Largura (*problema*)  
retorna uma solução ou falha  
Busca-Genérica (*problema*, Inserir-no-Fim)

*Inserir no final e remove do início,  
que estrutura de dados é essa?*



# BFS

função Busca-em-Largura (*problema*)  
retorna uma solução ou falha  
Busca-Genérica (*problema*, Inserir-no-Fim)

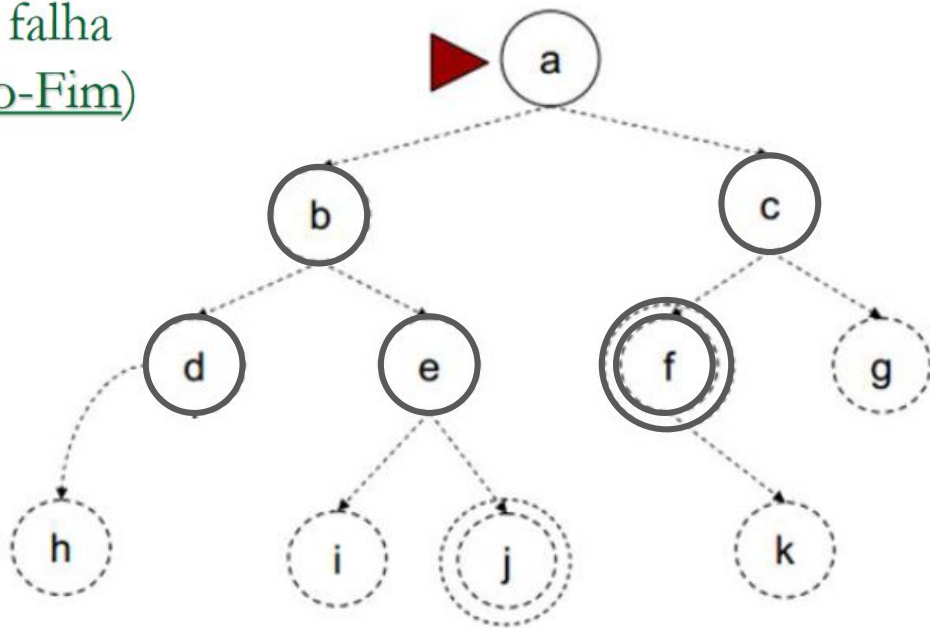




# BFS

função Busca-em-Largura (*problema*)  
retorna uma solução ou falha  
Busca-Genérica (*problema*, Inserir-no-Fim)

**Pára ou continua?**

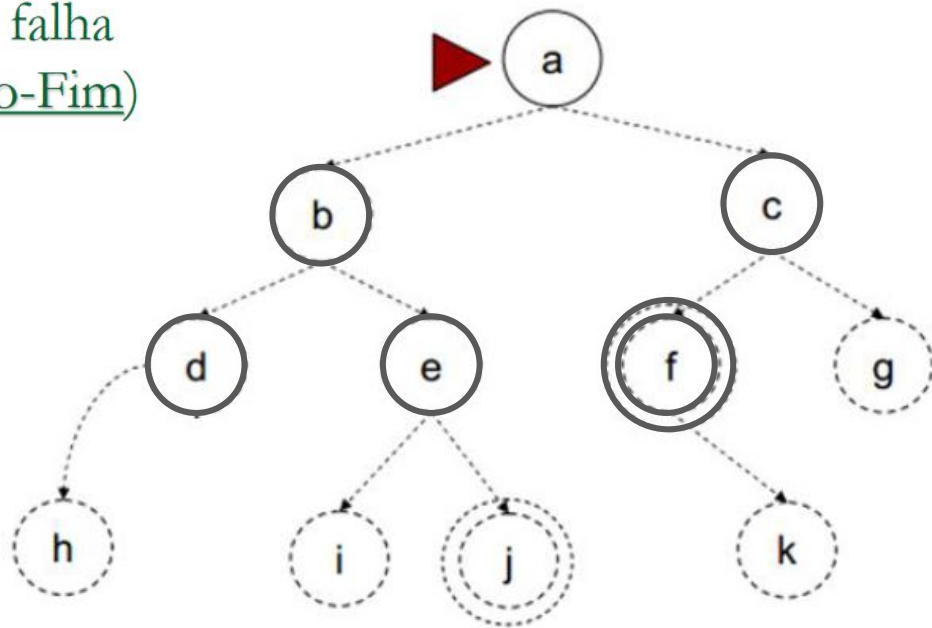


# BFS

função Busca-em-Largura (*problema*)  
retorna uma solução ou falha  
Busca-Genérica (*problema*, Inserir-no-Fim)

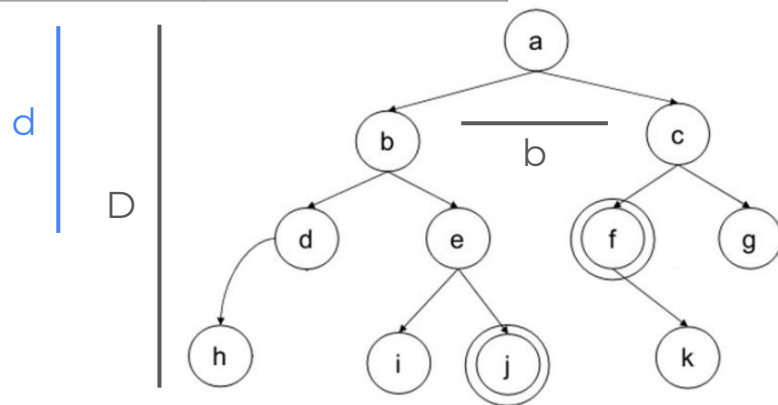
## Pára ou continua?

- Assumindo custo constante  $c > 0$ , soluções mais rasas serão as melhores.
- Para um custo qualquer, precisamos explorar o restante da árvore, pode ter um caminho mais barato.



# Algoritmos de busca cega

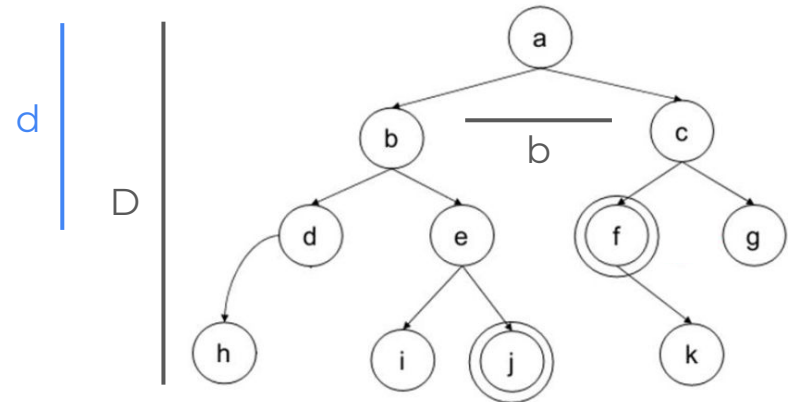
Algoritmo	Custo	Tempo	Espaço
DFS-Backtracking	Qualquer	$O(b^D)$	$O(D)$
DFS	$\emptyset$	$O(b^D)$	$O(D)$
DFS-I	$c \geq \emptyset$	$O(b^d)$	$O(D)$
BFS	1	$O(b^d)$	$O(b^d)$



# Algoritmos de busca cega

Algoritmo	Custo	Tempo	Espaço
DFS-Backtracking	Qualquer	$O(b^D)$	$O(D)$
DFS	$\emptyset$	$O(b^D)$	$O(D)$
DFS-I	$c \geq \emptyset$	$O(b^d)$	$O(D)$
BFS	1	$O(b^d)$	$O(b^d)$

Exige o registro de todo o grafo enquanto o explora.



# Algoritmos de busca cega

Tempo e espaço gastos na busca em largura:

$b = 10$

1000 nós por segundo

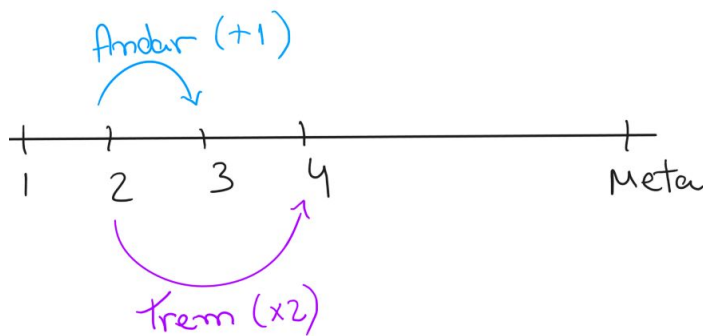
100 bytes por nó

Profundidade	Nodos	Tempo	Memória
0	1	1 milisegundo	100 bytes
2	111	0.1 segundo	11 kilobytes
4	11111	11 segundos	1 megabytes
6	$10^6$	18 minutos	111 megabytes
8	$10^8$	31 horas	11 gigabytes
10	$10^{10}$	128 dias	1 terabytes
12	$10^{12}$	35 anos	111 terabytes
14	$10^{14}$	3500 anos	11111 terabytes

# Problema do Trem mágico

- As ruas da cidade são numeradas de 1 a N, você precisa chegar em uma cidade M qualquer e possui duas escolhas:
  - Andar: cada ação te leva de  $s$  a  $s+1$
  - Pegar o trem mágico: te leva de  $s$  a  $s*2$
- Porém, você não pode ultrapassar a cidade M ao longo de seu trajeto

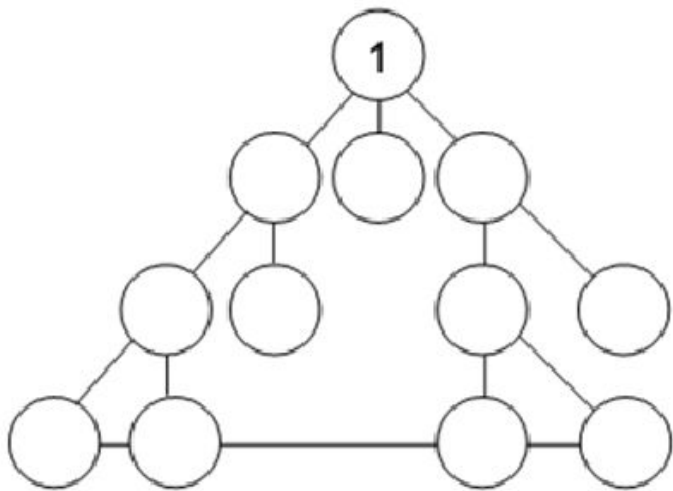
*Nada de ciclos nesse grafo!*



[busca\\_cega.ipynb](#)

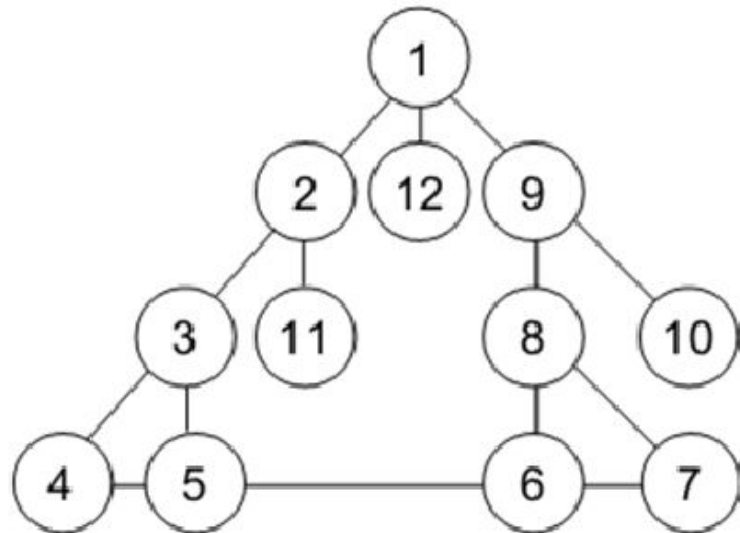
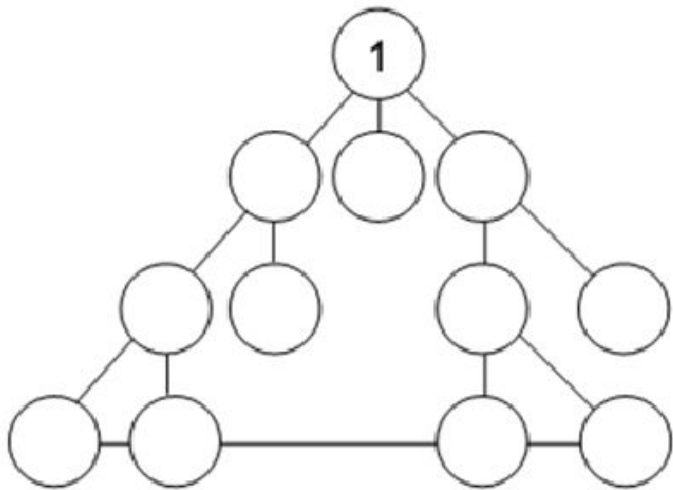
# Grafos cíclicos

- Se há risco de um nós ser revisitado, registre os nós visitados e não visite novamente
- Exemplo: busca em **profundidade**



# Grafos cíclicos


- Se há risco de um nós ser revisitado, registre os nós visitados e não visite novamente
- Exemplo: busca em **profundidade**






# Grafos cíclicos

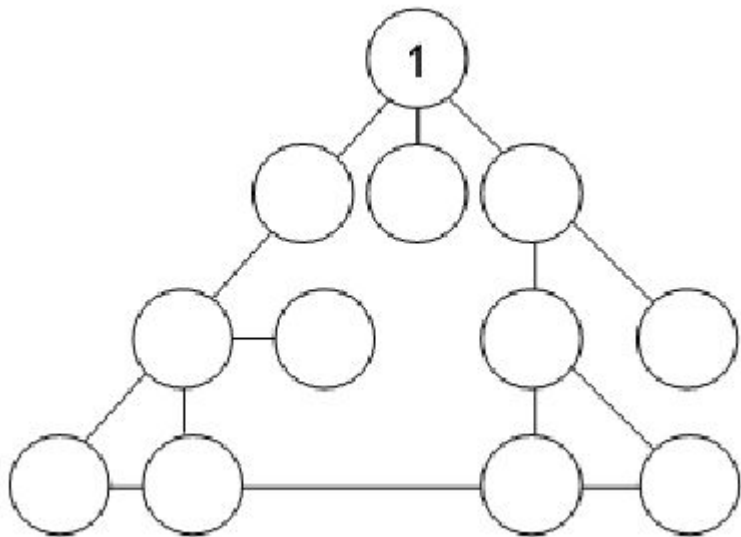
- Se há risco de um nós ser revisitado, registre os nós visitados e não visite novamente
- Exemplo: busca em **profundidade** (pilha ou recursão)

```
function Busca_Profundidade(Inicio, Alvo)
{
    empilha(Inicio)
    while nao pilhaVazia()
    {
        no = desempilha()
        foiVisitado(no)
        if no == Alvo
        {
            return no
        }
        for each Filho in Expande(no)
        {
             if nao Visitado(Filho)
            {
                empilha(Filho)
            }
        }
    }
}
```

```
function Busca_Profundidade(Inicio, Alvo)
{
    foiVisitado(Inicio)
    if Inicio == Alvo
    {
        return Inicio
    }
    for each Filho in Expande(Inicio)
    {
         if nao Visitado(Filho)
        {
            Busca_Profundidade(Filho, Alvo)
        }
    }
}
```

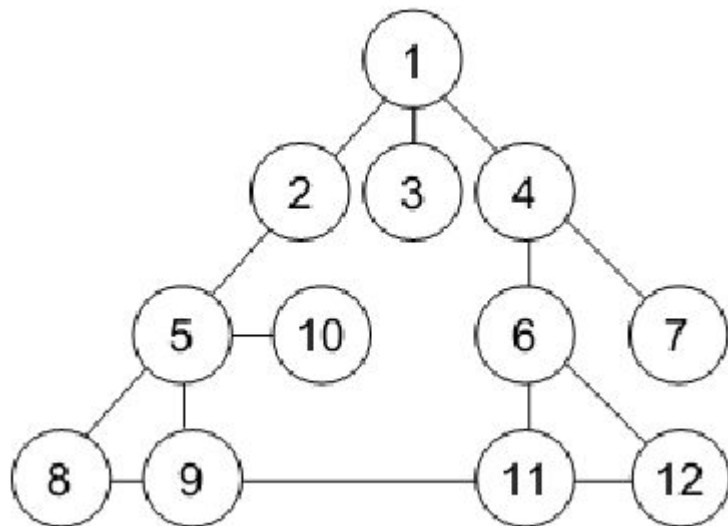
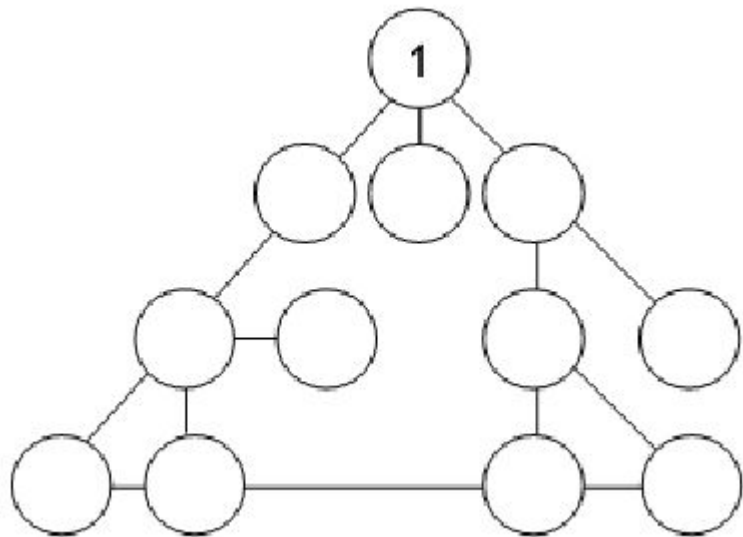
# Grafos cíclicos

- Basta registrar os nós visitados e não visitá-los novamente
- Exemplo: busca em **largura**



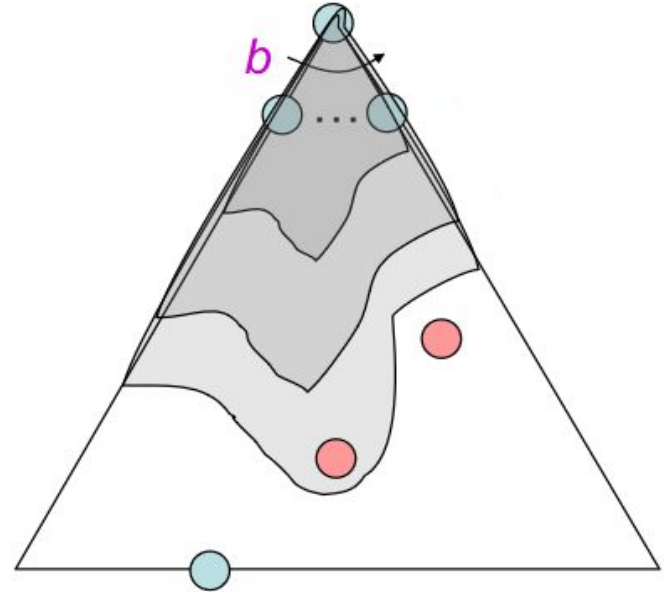
# Grafos cíclicos

- Basta registrar os nós visitados e não visitá-los novamente
- Exemplo: busca em **largura**



# Busca de custo uniforme

- Se assemelha à busca em largura, mas os caminhos são explorados de acordo com o seu custo
  - $g(n)$  - custo da raiz da busca até o nó  $n$
- É ótimo e completo para grafos de custo positivo

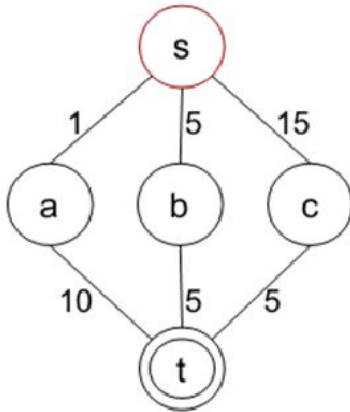


# Busca de custo uniforme

função Busca-de-Custo-Uniforme (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-Ordem-Crescente)



*Inserir de acordo com uma medida de prioridade e remove do início, que estrutura de dados é essa?*

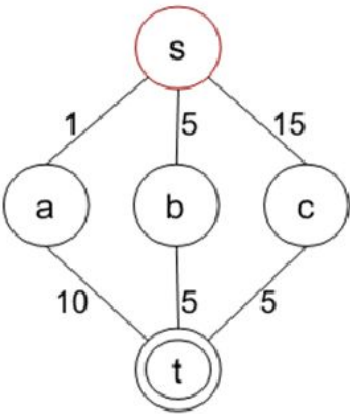
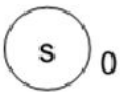
Nó	Fila ordenada pelo valor de g
S	

# Busca de custo uniforme

função Busca-de-Custo-Uniforme (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-Ordem-Crescente)



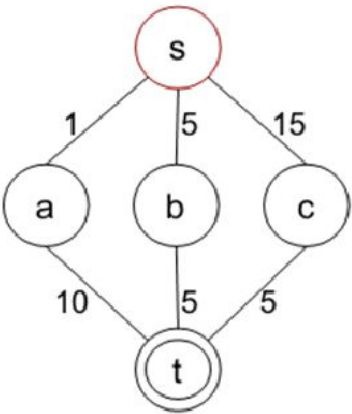
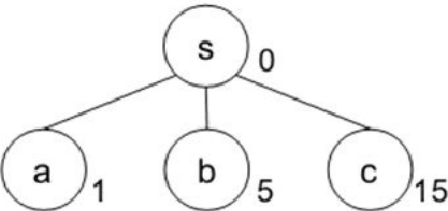
Nó	Fila ordenada pelo valor de g
S	

# Busca de custo uniforme

função Busca-de-Custo-Uniforme (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-Ordem-Crescente)



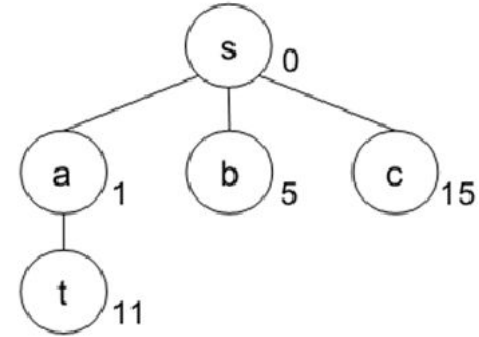
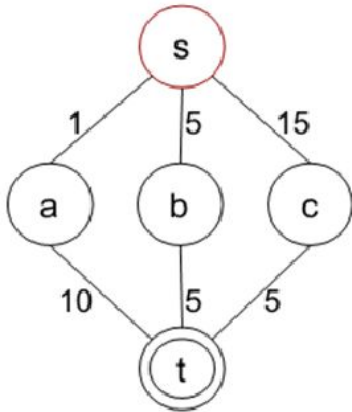
Nó	Fila ordenada pelo valor de g
S	[s,a]:1, [s,b]:5, [s,c]:15

# Busca de custo uniforme

função Busca-de-Custo-Uniforme (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-Ordem-Crescente)



Nó	Fila ordenada pelo valor de g
S	[s,a]:1, [s,b]:5, [s,c]:15
a	[s,b]:5, [s,a,t]:11, [s,c]:15

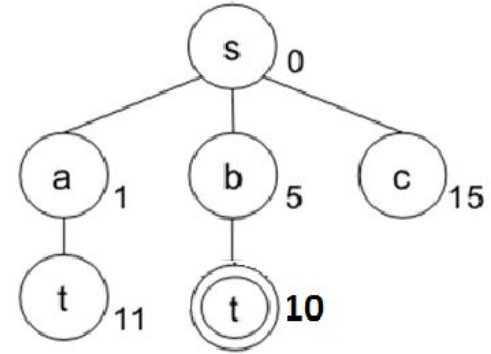
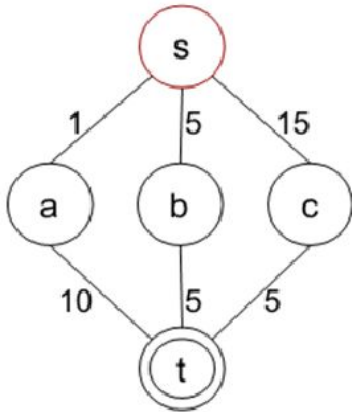


# Busca de custo uniforme

função Busca-de-Custo-Uniforme (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Inserir-Ordem-Crescente)



Nó	Fila ordenada pelo valor de g
S	[s,a]:1, [s,b]:5, [s,c]:15
a	[s,b]:5, [s,a,t]:11, [s,c]:15
b	[s,b,t]:10, [s,a,t]:11, [s,c]:15
t	Solução (início da fila): [s,b,t]

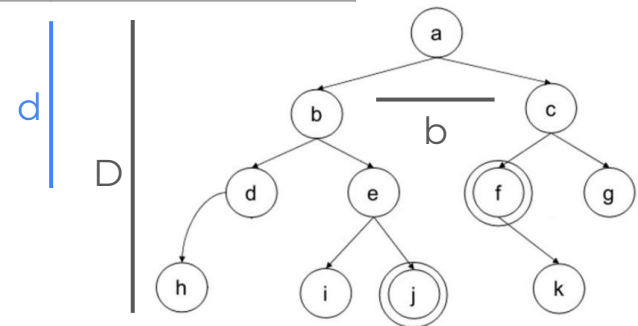
# Busca de custo uniforme

- Se assemelha à busca em largura, mas os caminhos são explorados de acordo com o seu custo
  - $g(n)$  - custo da raiz da busca até o nó  $n$
- É ótimo e completo para grafos de custo positivo
- Relação com outros algoritmos
  - Se os custos forem uma constante, torna-se idêntica à busca em largura
  - Se a heurística for uma função constante, é um caso particular do  $A^*$
  - Se continuarmos até que todos os nós sejam removidos da fila, é o algoritmo de Dijkstra

# Algoritmos de busca cega

Algoritmo	Custo	Tempo	Espaço
DFS-Backtracking	Qualquer	$O(b^D)$	$O(D)$
DFS	$\emptyset$	$O(b^D)$	$O(D)$
DFS-I	$c \geq \emptyset$	$O(b^d)$	$O(D)$
BFS	1	$O(b^d)$	$O(b^d)$
UCS	$c > \emptyset$	$O(b^{C^*/e})$	$O(b^{C^*/e})$

Se uma solução custa  $C^*$  e os custos na fila são pelo menos  $e$ , então a “profundidade efetiva” é aproximadamente  $C^*/e$



# Algoritmos de busca cega

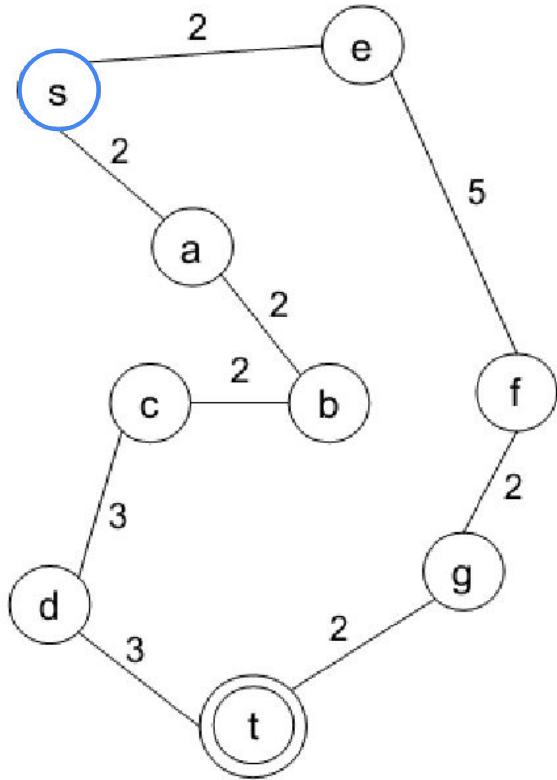
Algoritmo	Custo	Tempo	Espaço	Completo	Ótimo
DFS-Backtracking	Qualquer	$O(b^D)$	$O(D)$	Não	Não
DFS	$\emptyset$	$O(b^D)$	$O(D)$	Não	Não
DFS-I	$c \geq \emptyset$	$O(b^d)$	$O(D)$	Sim	Sim
BFS	1	$O(b^d)$	$O(b^d)$	Sim	Sim
UCS	$c > \emptyset$	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$	Sim	Sim

- **Completeness:** se houver solução, ela será encontrada
- **Ótimo:** encontra a melhor solução

# Algoritmos de busca cega

- DFS: incompleto, subótimo, eficiente em espaço
- DFS-l: completo, subótimo, eficiente em espaço
- BFS: completo, subótimo, proibitivo em espaço
- UCS: completo, ótimo, proibitivo em espaço

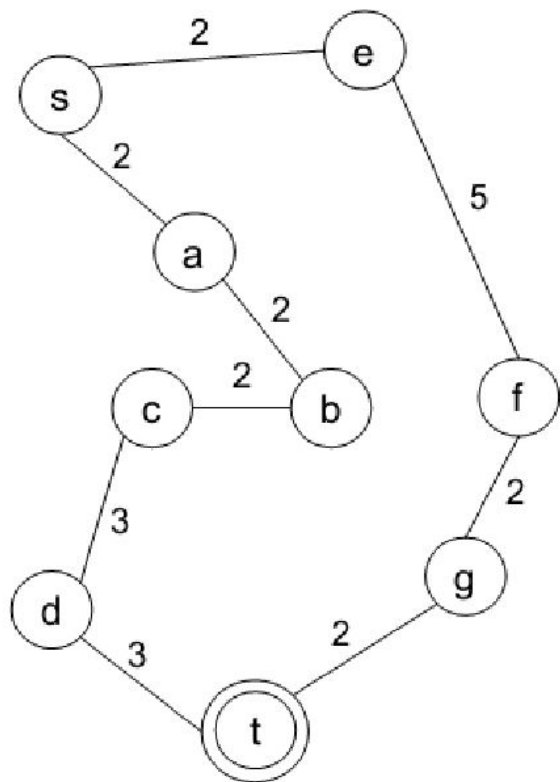
# Exercício



- Explore o espaço de estados ao lado usando o algoritmo de busca de custo uniforme

Nó	Fila ordenada pelo valor de g
S	[a,s]:2, [s,e]:2

# Exercício

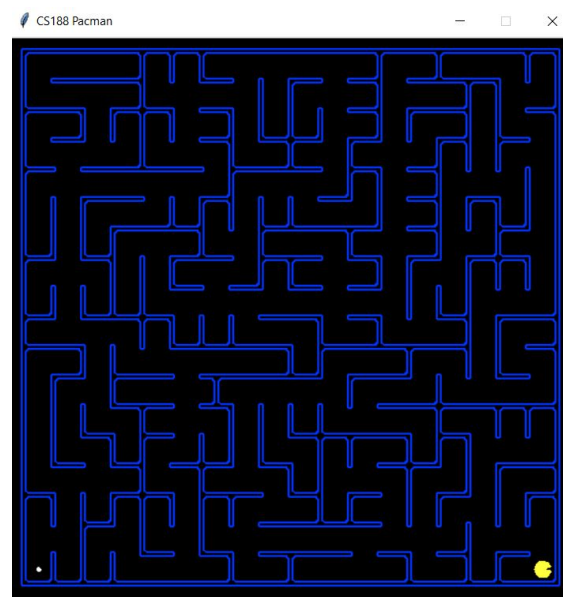
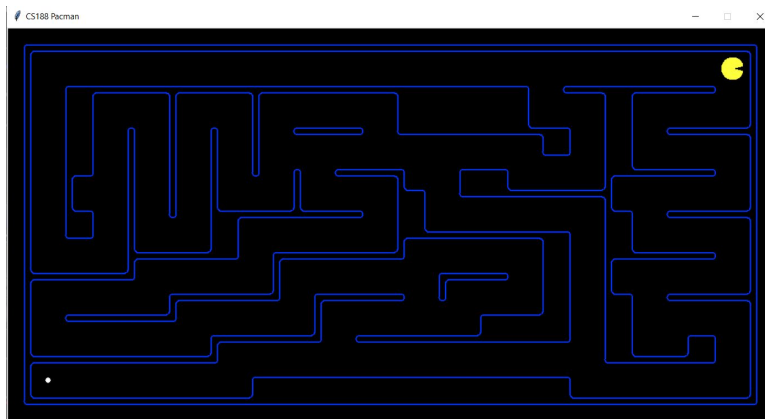


- Explore o espaço de estados ao lado usando o algoritmo de busca de custo uniforme

Nó	Fila ordenada pelo valor de g
S	[a,s]:2, [s,e]:2
A	[s,e]:2, [b,a,s]:4
E	[b,a,s]:4, [f,e,s]:7
B	[c,b,a,s]:6 [f,e,s]:7,
C	[f,e,s]:7, [d,c,b,a,s]:9
F	[d,c,b,a,s]:9, [g,f,e,s]:9
D	[g,f,e,s]:9, [t,d,c,b,a,s]:12
G	[t,g,f,e,s]:11 [t,d,c,b,a,s]:12,
T	

# Pacman

- Implemente a **busca em profundidade** e a **busca em largura** para solucionar um cenário limitado do Pacman, onde só há uma comida disponível na posição **(1,1)**.
  - tinyMaze
  - mediumMaze
  - bigMaze





# Pacman

- Você vai editar apenas as funções `depthFirstSearch` e `breadthFirstSearch` no arquivo [search.py](#)

```
75 def depthFirstSearch(problem: SearchProblem):
76     """
77     Search the deepest nodes in the search tree first.
78
79     Your search algorithm needs to return a list of actions that reaches the
80     goal. Make sure to implement a graph search algorithm.
81
82     To get started, you might want to try some of these simple commands to
83     understand the search problem that is being passed in:
84
85     print("Start:", problem.getStartState())
86     print("Is the start a goal?", problem.isGoalState(problem.getStartState()))
87     print("Start's successors:", problem.getSuccessors(problem.getStartState()))
88     """
89     """ YOUR CODE HERE """
90
91     util.raiseNotDefined()
92
93 def breadthFirstSearch(problem: SearchProblem):
94     """Search the shallowest nodes in the search tree first."""
95     """ YOUR CODE HERE """
96
97     util.raiseNotDefined()
```



# Pacman

- Seu código deve retornar a lista de ações necessárias para alcançar a comida

```
def tinyMazeSearch(problem):  
    """  
    Returns a sequence of moves that solves tinyMaze. For any other maze, the  
    sequence of moves will be incorrect, so only use this for tinyMaze.  
    """  
    from game import Directions  
    s = Directions.SOUTH  
    w = Directions.WEST  
    return [s, s, w, s, w, w, s, w]
```

- Você pode executar esse exemplo, digitando no terminal:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```



# Pacman

- Você vai editar apenas as funções `depthFirstSearch` e `breadthFirstSearch` no arquivo `search.py`
- Após implementadas as funções, você pode vê-las funcionando para o `tinyMaze` ao rodar:
  - `python pacman.py -l tinyMaze -p SearchAgent -a fn=depthFirstSearch`
  - `python pacman.py -l tinyMaze -p SearchAgent -a fn=breadthFirstSearch`
- Execute também para o `mediumMaze` e o `bigMaze`

# Pacman

- `print("Start:", problem.getStartState())`
  - Start: (5,5)
- `print("Is the start a goal?", problem.isGoalState(problem.getStartState()))`
  - Is the start a goal? False
- `print("Start's successors:", problem.getSuccessors(problem.getStartState()))`
  - Start's successors: [(‘South’, (5,4), 1), (‘West’, (4,5), 1)]

```
75 def depthFirstSearch(problem: SearchProblem):  
85     print("Start:", problem.getStartState())  
86     print("Is the start a goal?", problem.isGoalState(problem.getStartState()))  
87     print("Start's successors:", problem.getSuccessors(problem.getStartState()))  
88     ""  
89     """ YOUR CODE HERE """
```

# Pacman

- A biblioteca de funções `util.py` implementa as estruturas de dados necessárias para solucionar o problema
  - Stack - Pilha (DFS)
  - Queue - Fila (BFS)

```
133 class Stack:
134     "A container with a last-in-first-out (LIFO) queuing policy."
135     def __init__(self):
136         self.list = []
137
138     def push(self,item):
139         "Push 'item' onto the stack"
140         self.list.append(item)
141
142     def pop(self):
143         "Pop the most recently pushed item from the stack"
144         return self.list.pop()
145
146     def isEmpty(self):
147         "Returns true if the stack is empty"
148         return len(self.list) == 0
149
150 class Queue:
151     "A container with a first-in-first-out (FIFO) queuing policy."
152     def __init__(self):
153         self.list = []
154
155     def push(self,item):
156         "Enqueue the 'item' into the queue"
157         self.list.insert(0,item)
158
159     def pop(self):
160         """
161         Dequeue the earliest enqueued item still in the queue. This
162         operation removes the item from the queue.
163         """
164         return self.list.pop()
165
166     def isEmpty(self):
167         "Returns true if the queue is empty"
168         return len(self.list) == 0
169
```

# Referências

- <http://dcm.ffclrp.usp.br/~augusto/ia/>
- <http://adm-net-a.unifei.edu.br/phl/pdf/0036188.pdf>
- [http://www-usr.inf.ufsm.br/~pozzer/disciplinas/pj3d\\_busca.pdf](http://www-usr.inf.ufsm.br/~pozzer/disciplinas/pj3d_busca.pdf)
- <http://www.algoritmos.com/2009/08/busca-em-profundidade-dfs.html>
- <http://www.dca.fee.unicamp.br/~gomide/courses/EA072/transp/EA072BuscaBasicaProgramas.pdf>
- <http://professor.ufabc.edu.br/~leticia.bueno/classes/teoriagrafos/materiais/dfs.pdf>
- [http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/Grafos/index\\_grafos.html](http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/Grafos/index_grafos.html)
- <http://professor.ufabc.edu.br/~ronaldo.prati/InteligenciaArtificial/pratica1.html> - PAC-MAN
- [http://en.wikipedia.org/wiki/Breadth-first\\_search](http://en.wikipedia.org/wiki/Breadth-first_search)