DCC192

2025/1

UFMG

Desenvolvimento de Jogos Digitais

A20: IA — Pathfinding I

Prof. Lucas N. Ferreira

Plano de aula

- **Representação de Mapas em Jogos**
- **Algoritmos de Busca Não-Informados**
  - **Busca em profundidade**
  - **Busca em largura**
- Algoritmos de Busca Informados
  - Heurísticas
  - Greedy Best-First Search
  - A*

Pathfinding em Jogos

Em muitos jogos, precisamos mover objetos do jogos de maneira “inteligente”, incluindo utilizando o menor caminho, desviando de obstáculos, seguindo outros objetos, etc.

Warcraf 1: Pathfinding para movimentação de unidades com o mouse

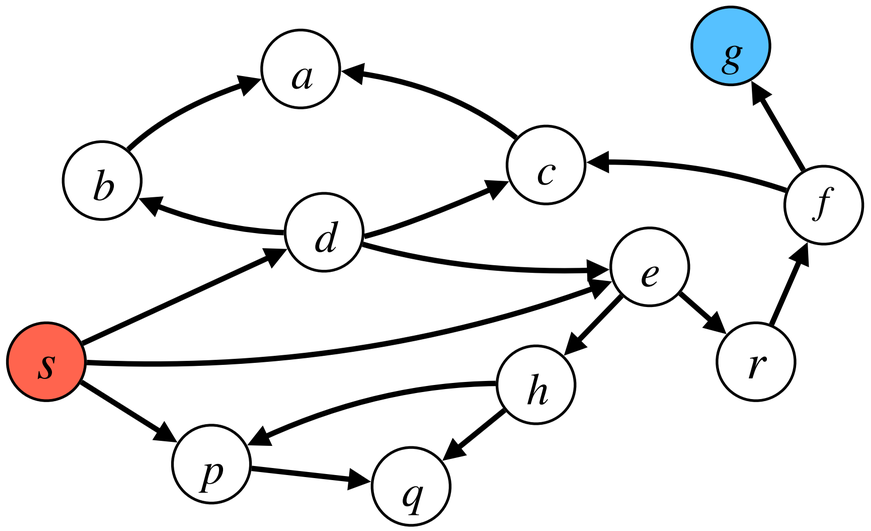
Problemas de Busca no Espaço de Estados

Pathfinding é formalizado em IA como um **problema de busca no espaço de estados**:

- Conjunto de estados S, chamado de **espaço de estados**
- **Estado inicial** $s \in S$
- **Estado final** $g \in S$
- **Função de ações** $A(s)$ que retorna o conjunto finito de ações possíveis em s
- **Modelo de transição** $T(s, a)$, uma função que retorna um novo estado s' resultado da aplicação da ação a no estado s
- **Função custo de ação** $C(s, a, s')$ que retorna o custo numérico da aplicação da ação a no estado s para alcançar o estado s'

Espaço de Estados

Em problemas de busca, o **espaço de estados** é geralmente representado como um grafo, onde os **vértices são os estados** e as **arestas são as ações:**



- O conjunto de vértices é igual ao de estados S
- O conjunto de arestas A contém as ações
- Uma **solução** (caminho) é uma sequência de estados $C = \{s, s_1, s_2, \ldots, g\}$ tal que $(s_i, s_{i+1}) \in A$, para todo s_i na sequência S
- Um **caminho C é ótimo**, também denotado como C^*, se não existe nenhum outro caminho entre s e g com custo menor que C

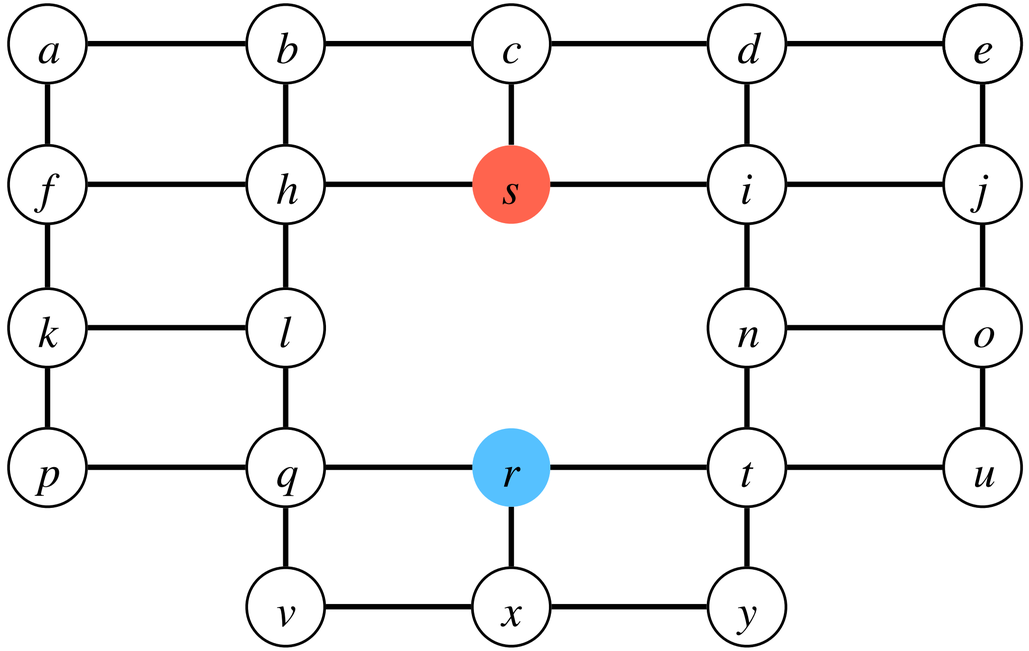
Pathfinding: Espaços de Estados

As posições dos objetos em jogos constuman ser contínuas. Precisamos discretizar esse espaço para fazer buscas eficientes. A técnica mais comum é utilizar um grid:

- Conjunto de estados S: cada célula livre é um estado
- **Estado inicial** $s \in S$: a coordenada (i, j) atual da unidade
- **Estado final** $g \in S$: a coordenada destino clicada pelo jogador
- **Função de ações** $A(s)$: direções (up, down, left, ...) das coordenadas vizinhas de s que estão livres
- **Modelo de transição** $T(s, a)$: a célula s' vizinha de s alcançada pela movimentação na direaçao de a
- **Função custo de ação** $C(s, a, s')$: um valor real pré-definido de se caminhar na grama vs. nas pedras (opcional)

Pathfinding: Espaços de Estados

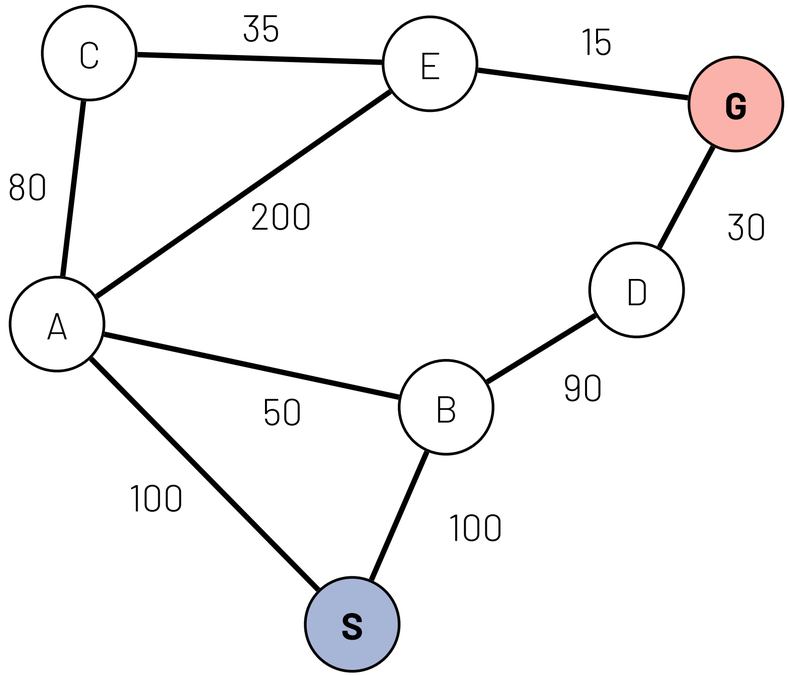
As posições dos objetos em jogos constuman ser contínuas. Precisamos discretizar esse espaço para fazer buscas eficientes. A técnica mais comum é utilizar um grid:



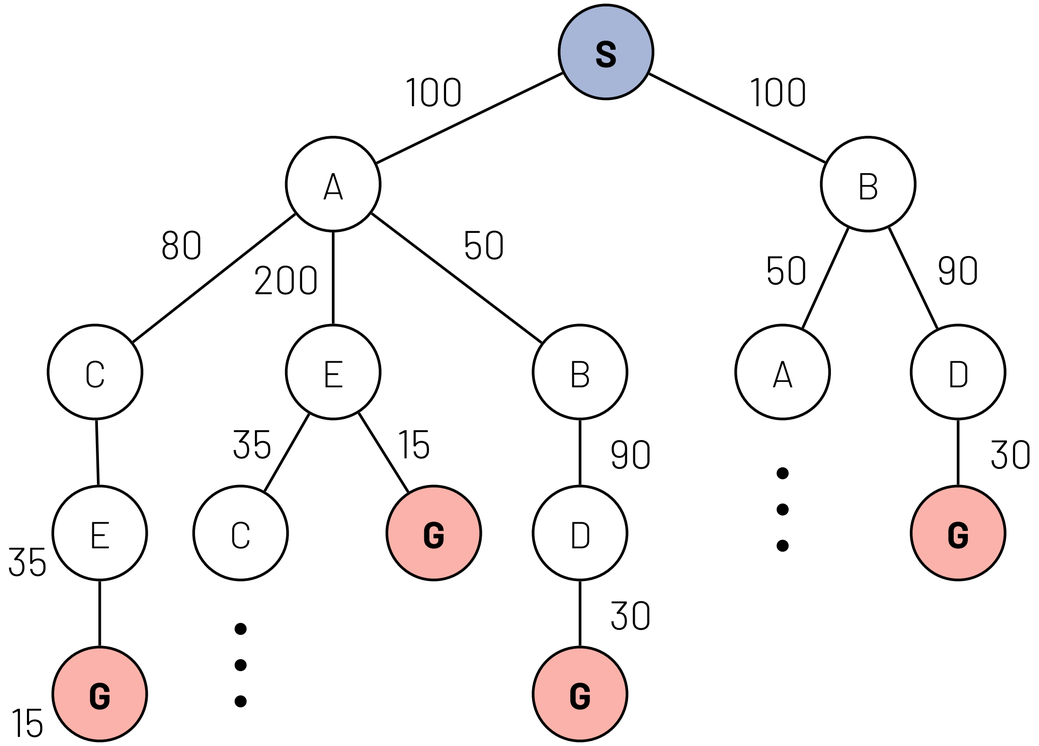
Cada célula vazia é um vértice, onde células adjacentes sem obstáculo entre si são conectadas via uma aresta

Árvore de Busca

Em IA, os grafos do espaço de estados geralmente não são armazenados em memória, pois, na prática, costumam ser muito grandes.



Ao invés disso, podemos aplicar a função do modelo de transição $T(s, a)$ para gerar uma **árvore de busca**, começando pelo estado inicial.

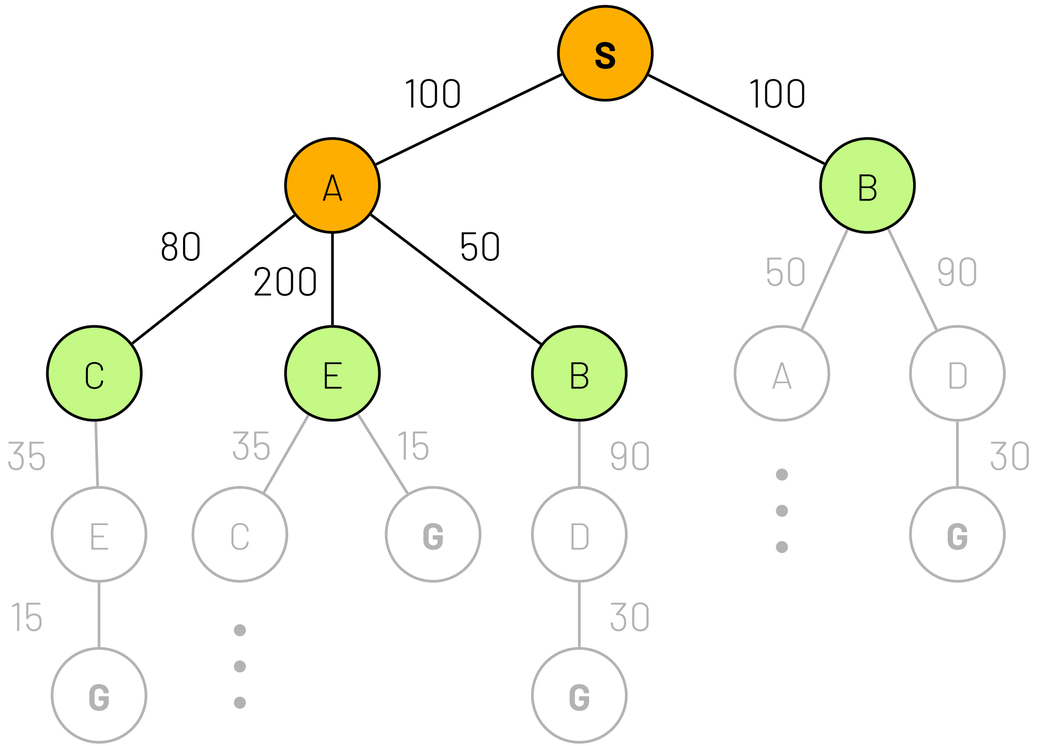


*Dizemos que os grafos representados por um estado inicial e uma função sucessora são **implícitos**.*

Árvore de Busca

Definições:

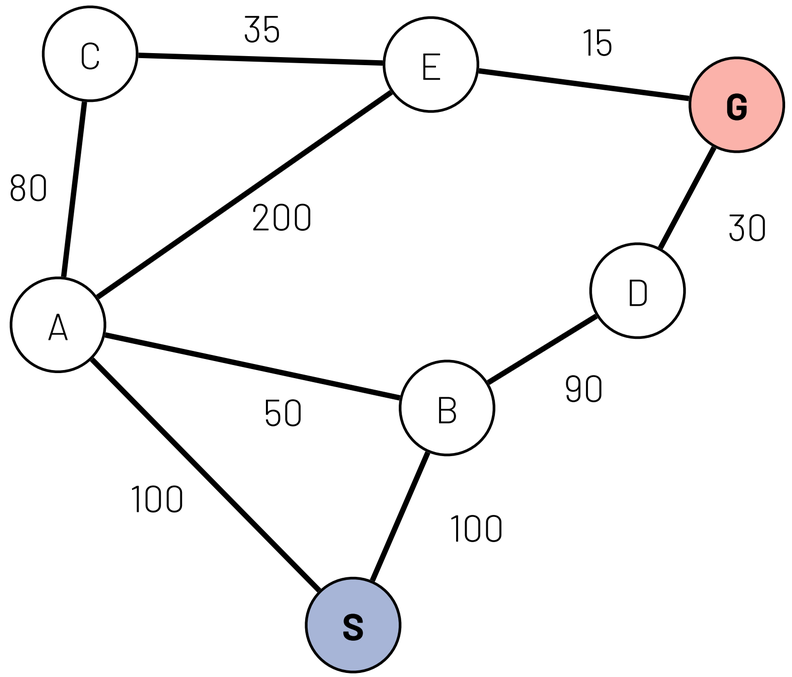
- O estado inicial é a **raiz** da árvore;
- Um nó é **expandido** quando um algoritmo considera as ações para aquele estado s chamando a função $A(s)$;
- Um nó é **gerado** quando seu pai é expandido;
- O conjunto de caminhos produzidos até o momento é chamado de **fronteira**;
- Um **ciclo** ocorre quando um nó aparece múltiplas vezes em um mesmo caminho
  (e.g., S, A, E, C, A, E, G)

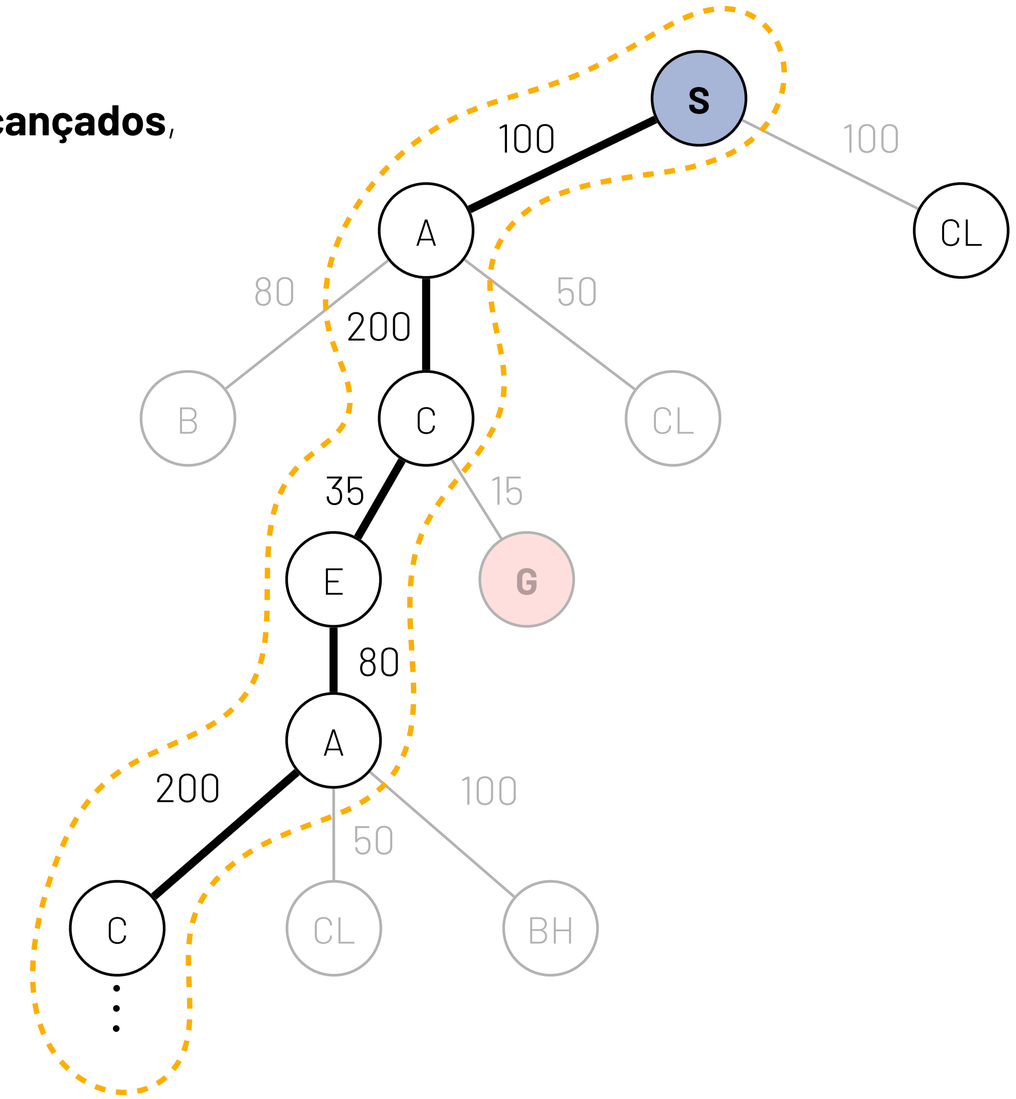


Ciclos e caminhos redundantes

Ciclos são caminhos redundantes geram **árvores de busca infinitas**:

- Podemos evitar ciclos durante a busca com uma tabela de nós **alcançados**, expandindo apenas aqueles que:
  - Ainda não foram visitados ou;
  - Estão sendo visitados por um caminho melhor.





Algoritmos de busca (que estudaremos)

- **Busca sem informação**

  Não possuem informação sobre a distância entre um determinado estado e e o estado final g

  - Busca em largura (Breath-first search - BFS) — assume que ações todas tem o mesmo custo
  - Busca em profundidade (Depth-first search- DFS) — assume que ações todas tem o mesmo custo
  - Busca de custo uniforme (Algoritmo de Dijkstra) — assume ações com custo diferentes
- **Busca informada**

  Possuem informação sobre a distância entre um determinado nó e o estado final

  - Busca gulosa de melhor escolha
  - Algoritmo A*

Algoritmo genérico de busca em árvore

Os algoritmos de busca em árvore seguem a mesma estrutura geral:

```
def busca-arvore(s, g, A, T, C):
 1. fronteira = [s] # Inicializar a fronteira com o estado inicial s
 2. alcancado = {s} # Marcar nó inicial como visitado
 3. custo[s] = 0    # Inicializar custo do estado inicial
 4. while fronteira não estiver vazia:
 5.     n = fronteira.pop() # Escolher um nó da fronteira para expandir
 6.     if n == g:          # Verificar se o nó n escolhido é o estado final g
 7.         return caminho entre s e g
 8.     for filho in T(n, A(n)):                  # Expandir o nó n escolhido usando função de ações A
 9.          custo_filho = custo[n] + C(n, filho) # Calcular custo de chegar até o filho por n
10.          if filho not in alcancado or custo_filho < custo[filho]:
11.             fronteira.append(filho)
12.             alcancado.append(filho)
13.             custo[filho] = custo_filho
```

A principal diferença entre os algoritmos é a **estratégia de expansão** do nó `n`; Usamos diferentes **estruturas de dados** para implementar essas estratégias.

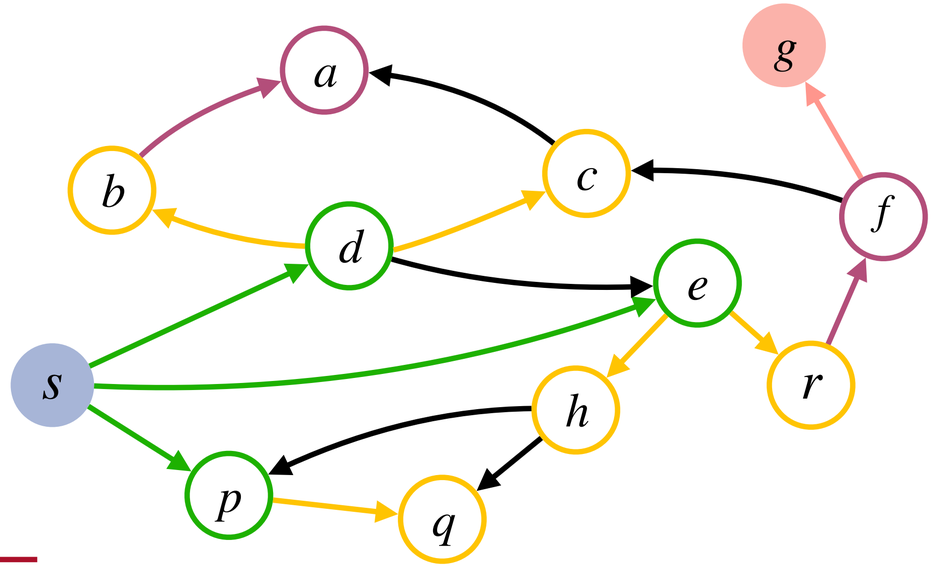
`alcancado` é uma tabela hash (dicionário em python) utilizada para evitar ciclos.

Busca em Largura

Fronteira é uma fila (FIFO)

Expandir o nó mais raso primeiro

- Nós a uma aresta de distância (d, e, p)
- Nós a duas arestas de distância (b, c, h, r, q)
- Nós a três arestas de distância (a, f)
- ...



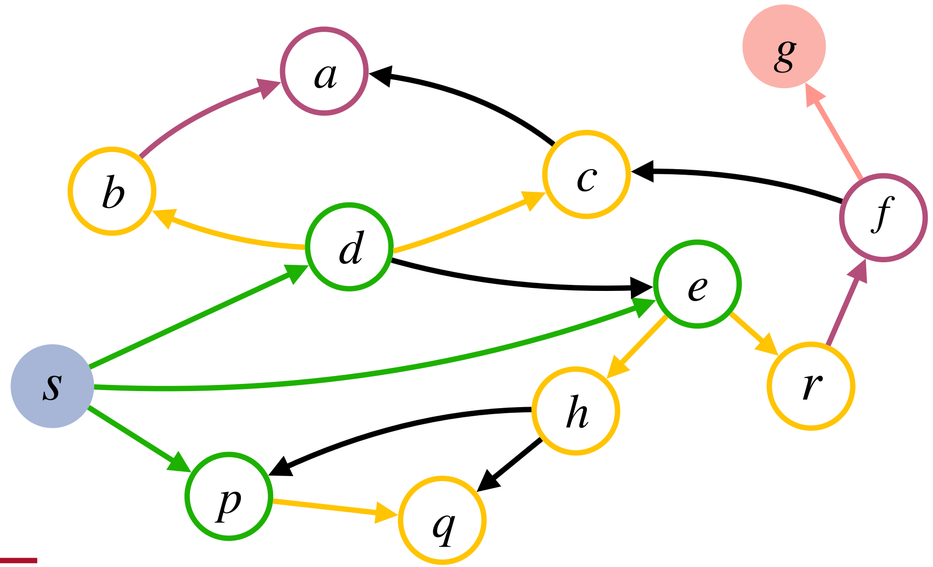
Tempo	Nó	Fronteira (fila)	Alcançado
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

Busca em Largura

Fronteira é uma fila (FIFO)

Expandir o nó mais raso primeiro

- Nós a uma aresta de distância (d, e, p)
- Nós a duas arestas de distância (b, c, h, r, q)
- Nós a três arestas de distância (a, f)
- ...



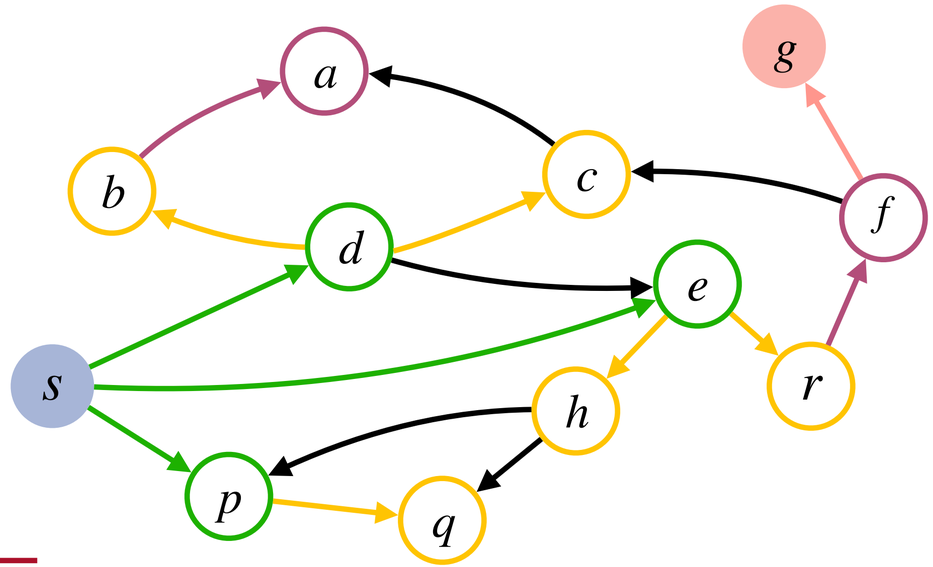
Tempo	Nó	Fronteira (fila)	Alcançado
1	s	[~~d~~, e, p]	{s, d, e, p}
2	d	[e, p, b, c]	{d, e, p, b, c}
3	e	[~~p~~, b, c, h, r]	{d, e, p, b, c, h, r}
4	p	[~~b~~, c, h, r, q]	{d, e, p, b, c, h, r, q}
5	b	[~~c~~, h, r, q, a]	{d, e, p, b, c, h, r, q, a}
6	c	[~~h~~, r, q, a]	{d, e, p, b, c, h, r, q, a}
7	h	[~~r~~, q, a]	{d, e, p, b, c, h, r, q, a}
8	r	[~~q~~, a, f]	{d, e, p, b, c, h, r, q, a}
9	q	[~~a~~, f]	{d, e, p, b, c, h, r, q, a}
10	a	[~~f~~, g]	{d, e, p, b, c, h, r, q, a}
11	f	[~~g~~]	{d, e, p, b, c, h, r, q, a}
12	g	[ ]	{d, e, p, b, c, h, r, q, a}

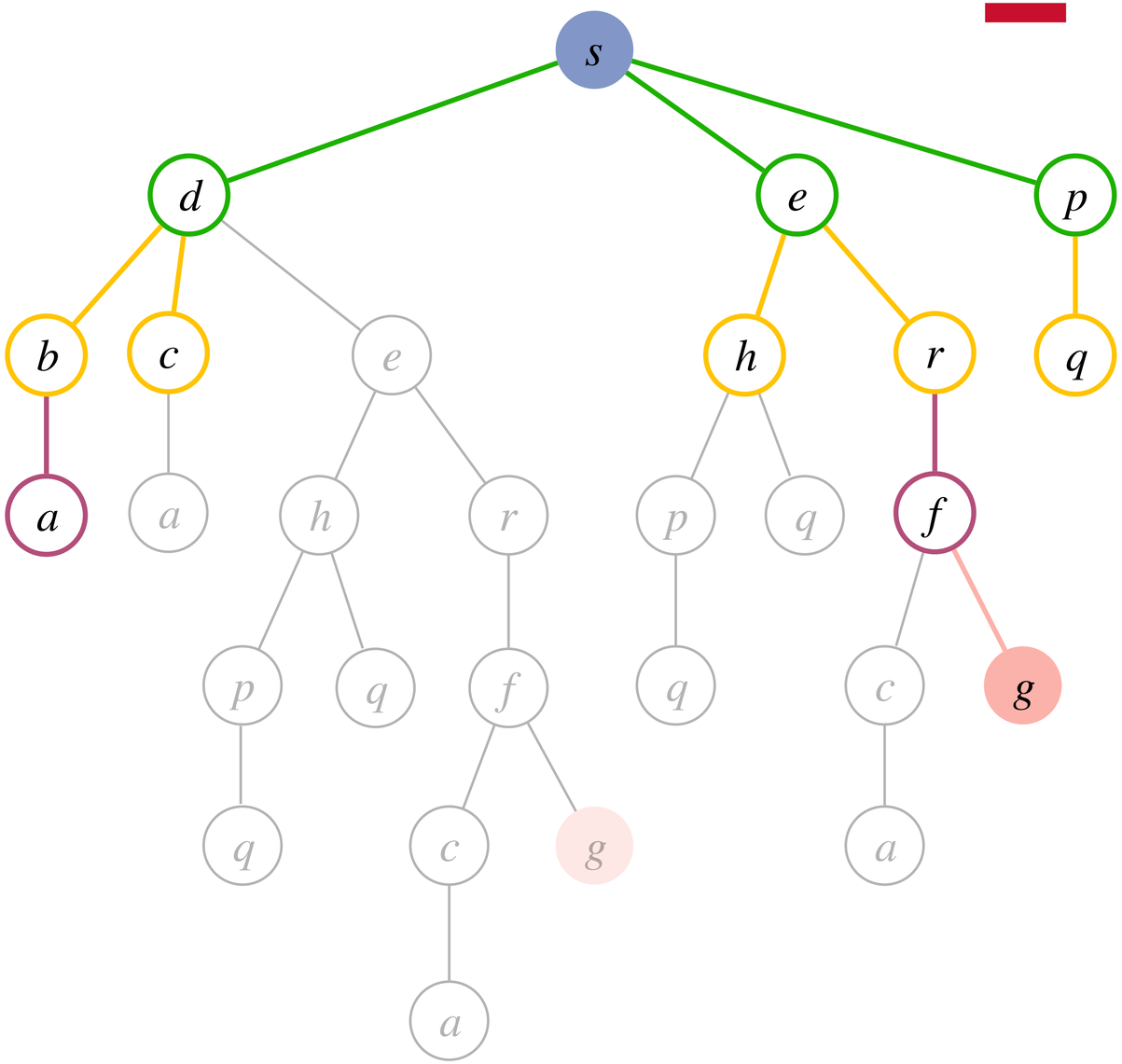
Busca em Largura

Fronteira é uma fila (FIFO)

Expandir o nó mais raso primeiro

- Nós a uma aresta de distância (d, e, p)
- Nós a duas arestas de distância (b, c, h, r, q)
- Nós a três arestas de distância (a, f)
- ...





Implementação da Busca em Largura

```
def BFS(s, g, A, T, C):
 1. fila = [s]
 2. alcancado = {s}
 3. custo[s] = 0
 4. while fila não estiver vazia:
 5.     n = fila.pop(0)          # Escolher o primeiro nó da fila para expandir
 6.     if n == g:               # Verificar se o nó n escolhido é o estado final g
 7.         return caminho entre s e g
 8.     for filho in T(n, A(n)):   # Expandir o nó n escolhido usando função de ações A
 9.         custo_filho = custo[n] + C(n, filho) # Calcular custo de chegar até o filho por n
 10.        if filho not in alcancado or custo_filho < custo[filho]:
 11.            fila.append(filho)
 12.            alcancado.append(filho)
 13.            custo[filho] = custo_filho
```

Na BFS, não é necessário manter os custos

Propriedade da Busca em Largura

- Complexidade de tempo

  Explora todos os nós acima da solução mais rasa — seja d a profundidade da solução mais rasa, a complexidade de tempo é $O(b^d)$

- Complexidade de espaço

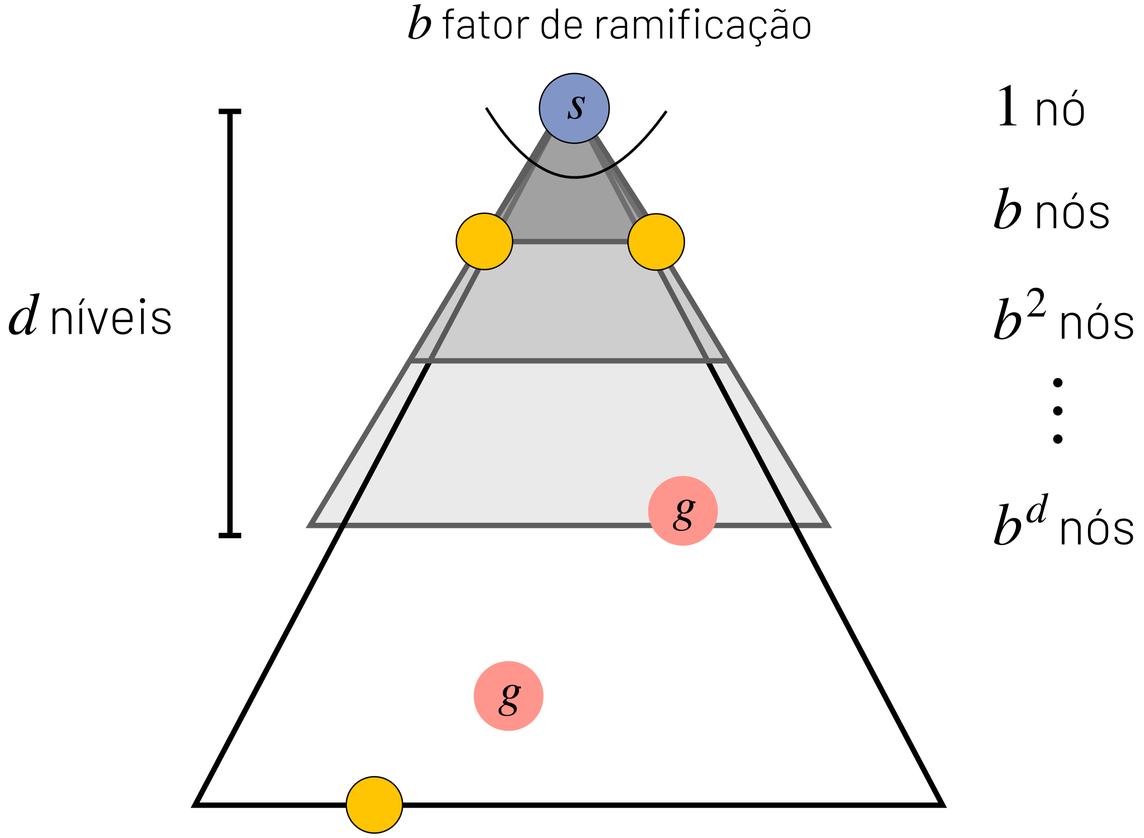
  Armazena todos os nós até o nível d antes da solução g, portanto complexidade de espaço é $O(b^d)$

- Completo

  Sim. Se existe uma solução, então d é finito e a BFS vai encontrá-la

- Ótimo

  Sim, mas apenas se os custos forem uniformes (iguais a 1)

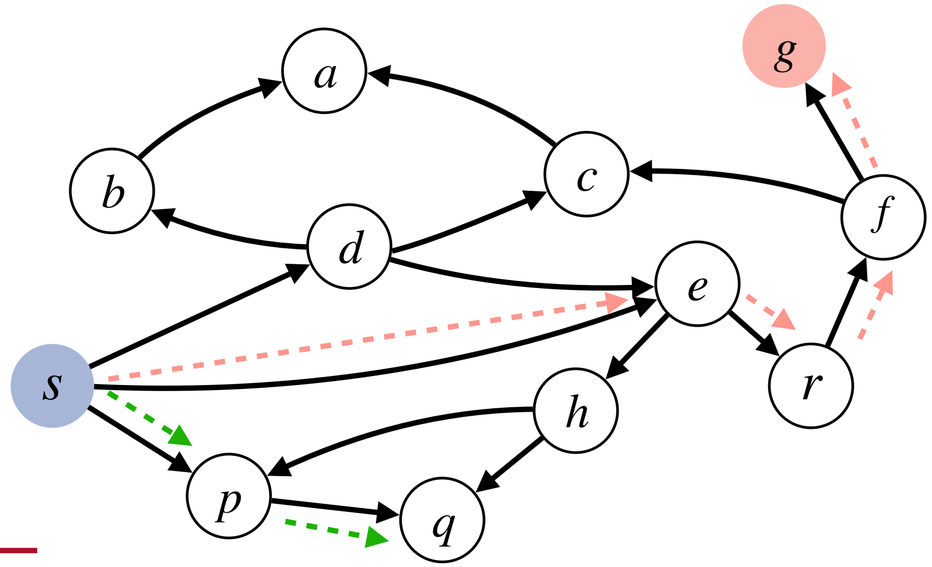


Busca em Profundidade

Fronteira é uma pilha (LIFO)

Expandir o nó mais profundo primeiro

- Nós do primeiro caminho
- Nós do segundo caminho
- Nós do terceiro caminho
- ...



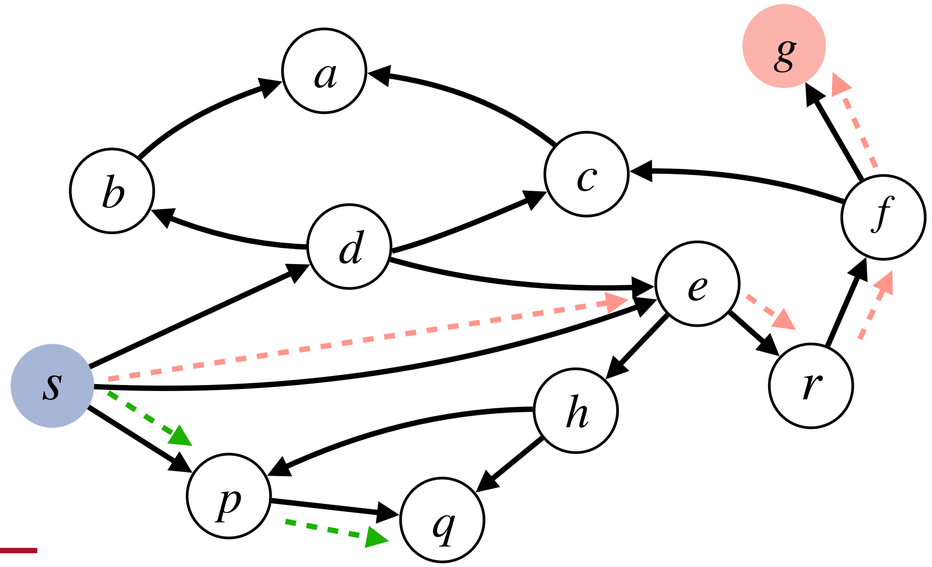
Tempo	Nó	Fronteira (pilha)	Alcançado
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

Busca em Profundidade

Fronteira é uma pilha (LIFO)

Expandir o nó mais profundo primeiro

- Nós do primeiro caminho
- Nós do segundo caminho
- Nós do terceiro caminho
- ...



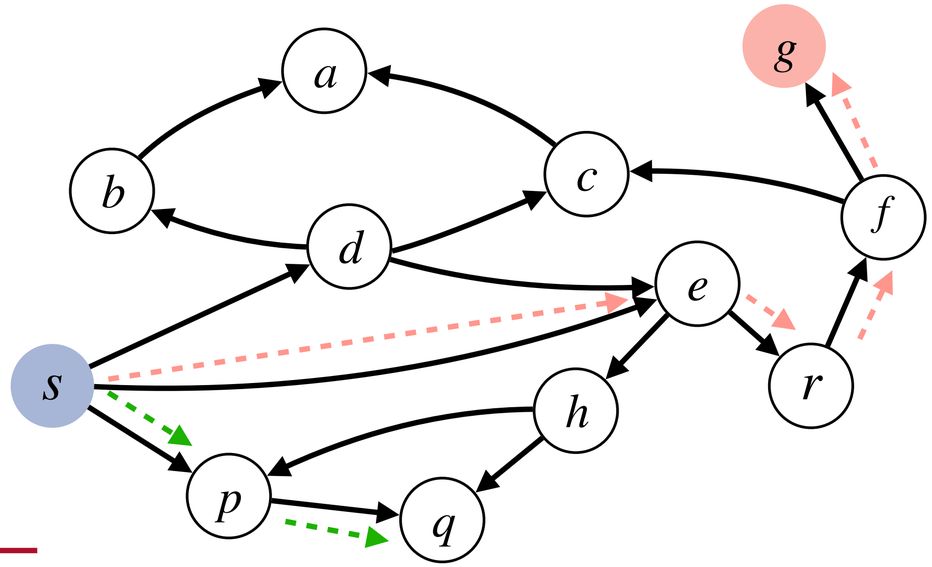
Tempo	Nó	Fronteira (pilha)	Alcançado
1	s	[d, e, ~~p~~]	{s, d, e, p}
2	p	[d, e, ~~q~~]	{s, d, e, p, q}
3	q	[d, ~~e~~]	{s, d, e, p, q}
4	e	[d, h, ~~r~~]	{s, d, e, p, q, h, r}
5	r	[d, h, ~~f~~]	{s, d, e, p, q, h, r, f}
6	f	[d, h, ~~g~~]	{s, d, e, p, q, h, r, f}
7	g	[d, h]	{s, d, e, p, q, h, r, f}
8			
9			
10			
11			
12			

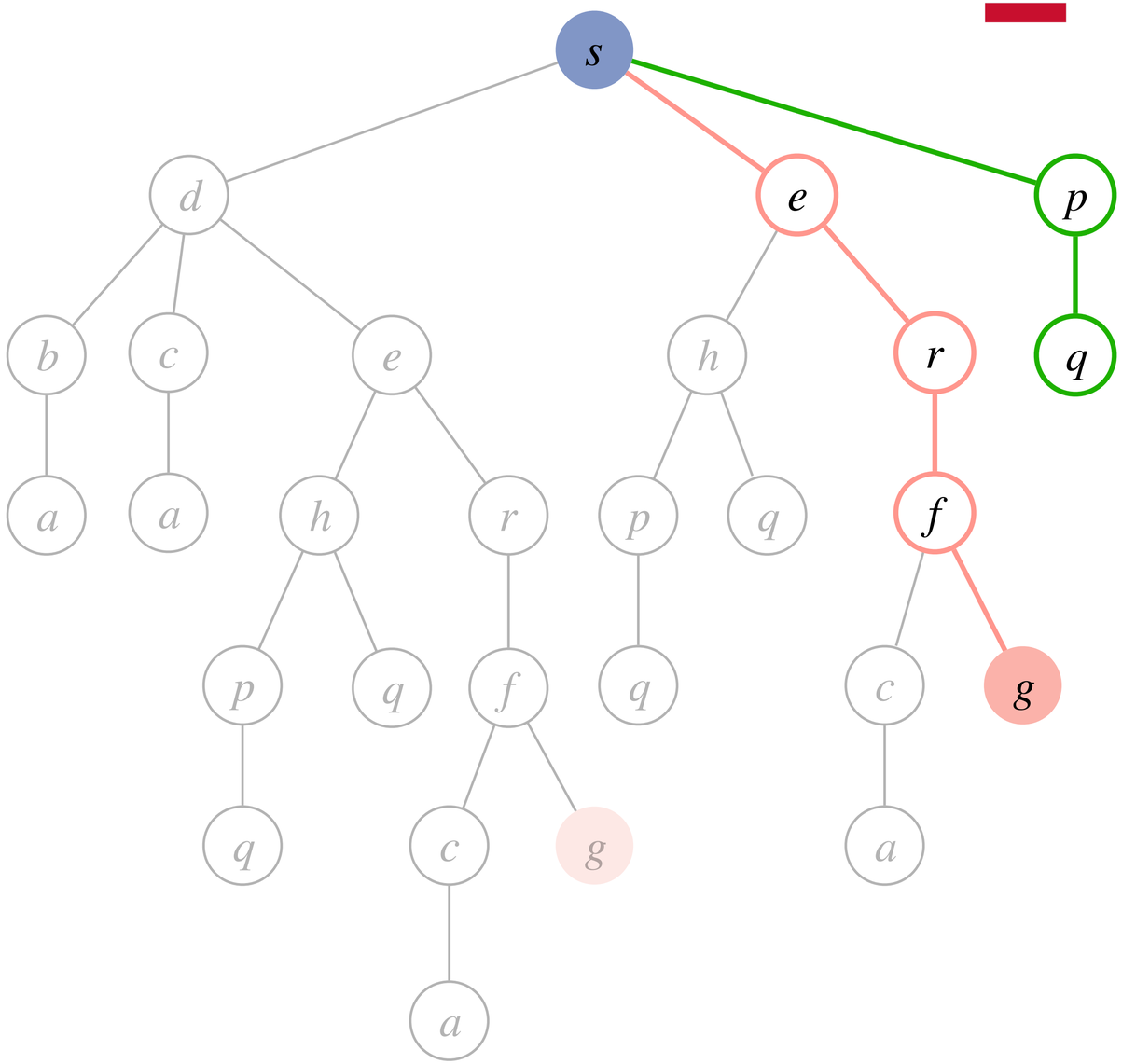
Busca em Profundidade

Fronteira é uma pilha (LIFO)

Expandir o nó mais profundo primeiro

- Nós do primeiro caminho
- Nós do segundo caminho
- Nós do terceiro caminho
- ...





Busca em Profundidade

```
def DFS(s, g, A, T, ~~C~~):
 1. pilha = [s]
 2. alcancado = {s}
~~3. custo[s] = 0~~
 4. while pilha não estiver vazia:
 5.     n = pilha.pop()          # Escolher o último nó da fila para expandir
 6.     if n == g:               # Verificar se o nó n escolhido é o estado final g
 7.         return caminho entre s e g
 8.     for filho in T(n, A(n)):   # Expandir o nó n escolhido usando função de ações A
~~9.         custo_filho = custo[n] + C(n, filho)~~ ~~# Calcular custo de chegar até o filho por n~~
 10.        if filho not in alcancado ~~or custo_filho < custo[filho]~~:
 11.            pilha.append(filho)
 12.            alcancado.append(filho)
~~13.            custo[filho] = custo_filho~~
```

Na DFS, a **fronteira** é uma **pilha (LIFO)**

Na DFS, também não é necessário manter os custos

Propriedade da Busca em Profundidade

- Complexidade de tempo

  No pior caso, terá que processar a árvore toda. Seja m a profundidade total (finita) da árvore, a complexidade de tempo é $O(b^m)$

- Complexidade de espaço

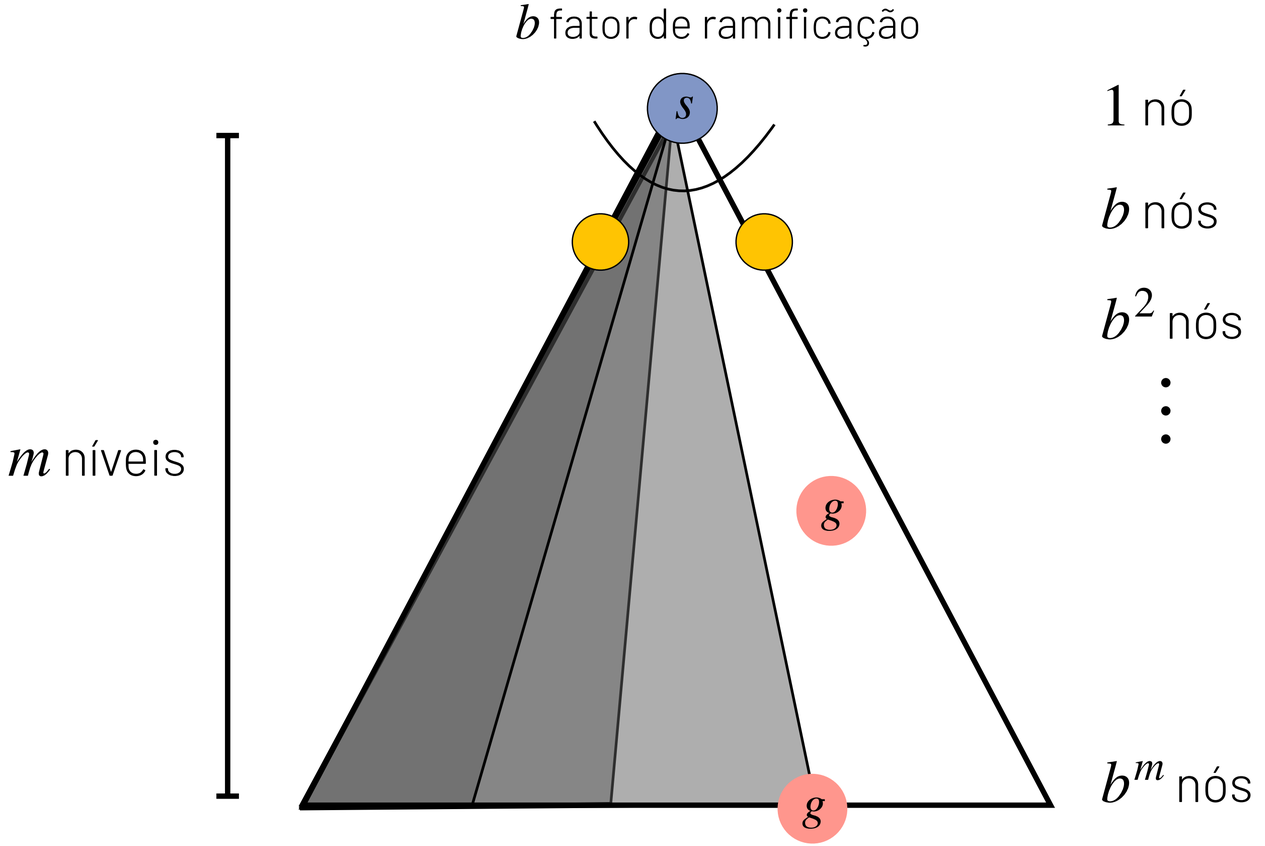
  Armazena os irmãos de cada nó no caminho até a raiz, portanto complexidade de espaço é $O(bm)$

- Completo

  Apenas se a árvore de busca for finita (ou usando verificação de ciclos).

- Ótimo

  Não, ele a solução mais à esquerda (recursivo) ou à direita (iterativo), independente de custo



Próxima aula

A20: IA — Pathfinding II

- Representação de Mapas em Jogos
- Algoritmos de Busca Não-Informados
  - Busca em profundidade
  - Busca em largura
- **Algoritmos de Busca Informados**
  - **Heurísticas**
  - **Greedy Best-First Search**
  - **A***