Inteligência Artificial

Estratégias de Busca Parte 2

Prof. Jefferson Morais

Na Aula Anterior

- Tivemos na aula anterior uma introdução sobre buscas
- Aprendemos um pouco sobre desempenho
- E vimos os algoritmos de busca em profundidade e busca em largura

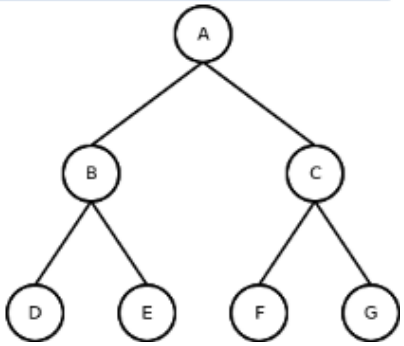
- Agora vamos ver outros algoritmos de busca, ainda na categoria da Busca Sem Informação

Sem Informação: Busca Em Profundidade Limitada

- Consiste em limitar a busca em profundidade até o nível **l**
- Nós na profundidade **l** são tratados como folhas
- A busca em profundidade é um caso especial onde **l = ∞**

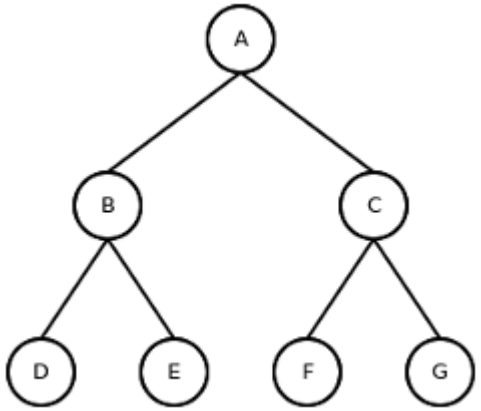
```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

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? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

Dois tipos de falha:
- ***failure***: nenhuma solução
- ***cutoff***: nenhuma solução dentro do limite de profundidade



Sem Informação: Busca Em Profundidade Limitada

- Desempenho
  - Completa: pode ser incompleta, caso l < d
  - Ótima: não, com **d < l** recai na mesma situação da busca em profundidade. Lembrando que **d** é a profundidade da solução mais rasa.
  - Complexidade de tempo: **$O(b^l)$**
  - Complexidade de espaço: **O(bl)**

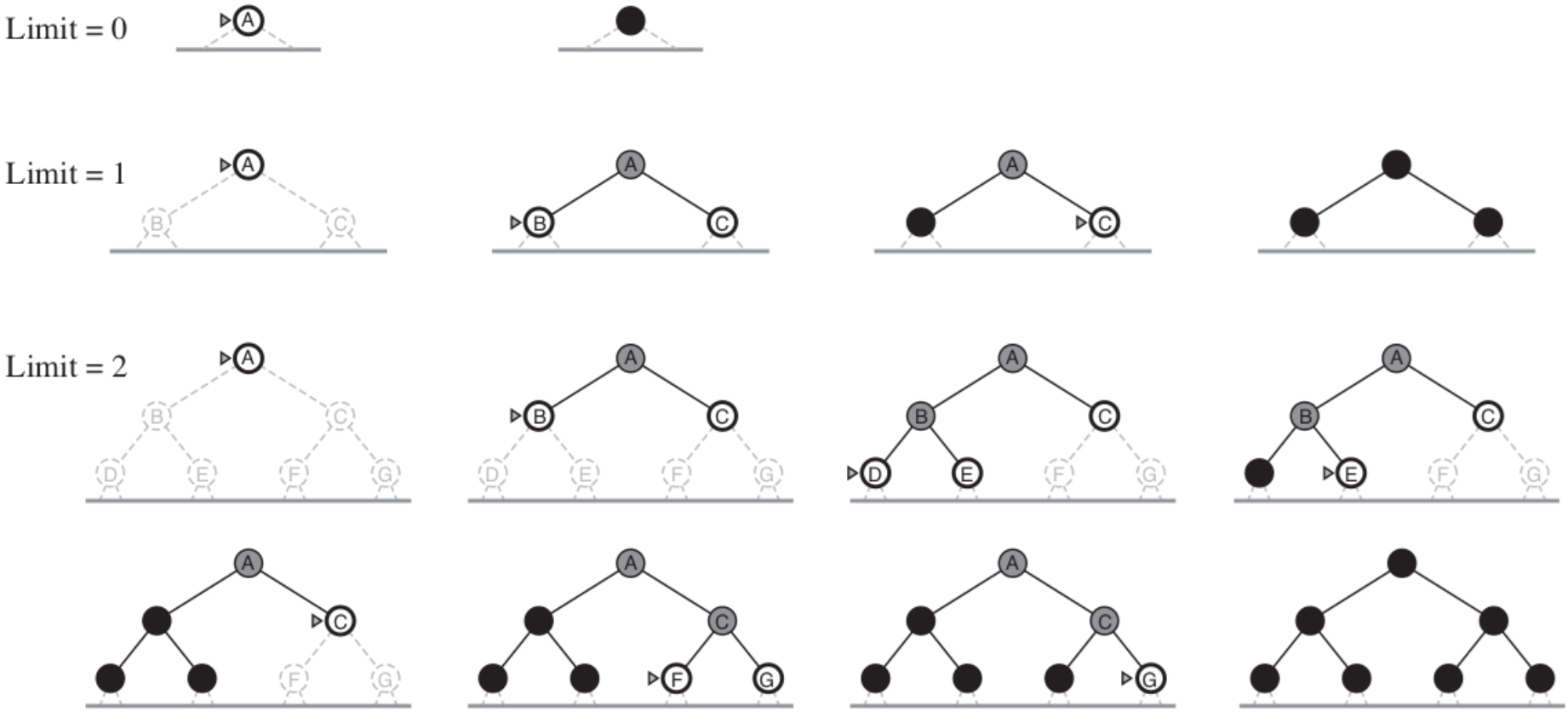


Sem Informação: Busca Em Profundidade Iterativa

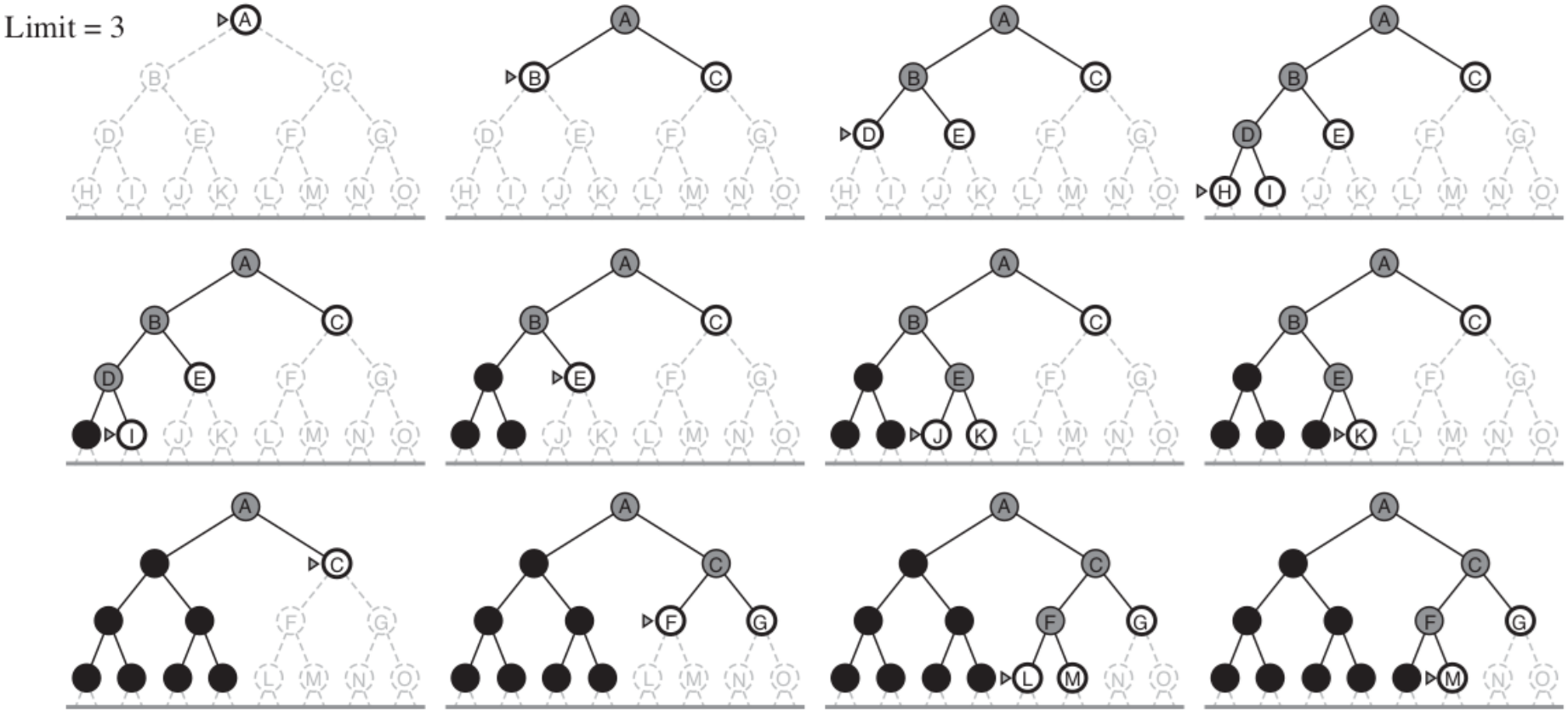
- É uma **estratégia geral** da busca em profundidade
- Esse algoritmo encontra o melhor limite de profundidade **l** automaticamente
- O algoritmo trabalha aumentando gradualmente o limite até encontrar um objetivo
- Observação: esta estratégia torna o algoritmo ótimo, pois o algoritmo garante sempre **l = d**

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

Sem Informação: Busca Em Profundidade Iterativa



Sem Informação: Busca Em Profundidade Iterativa



Sem Informação: Busca Em Profundidade Iterativa

- Pode parecer desperdício, pois os estados são gerados várias vezes

- Custo não é muito alto, pois os nós do nível inferior (profundidade d) são gerados uma vez, os do penúltimo são gerados duas vezes, e assim por diante...

- O total de nós gerados é

  $$\mathbf{N(BPI) = (d)b + (d - 1)b^2 + \ldots (2)b^{d-1} + (1)b^d = O(b^d)}$$

  (mesma da busca em largura em termos **assintóticos**)

- Há um custo extra em gerar os níveis mais altos múltiplas vezes, mas não é grande
- Ex.: se b = 10 e d = 5, os números são
  - N(BPI) = 50 + 400 + 3.000 + 20.000 + 100.000 = 123.450
  - N(BL) = 10 + 100 + 1.000 + 10.000 + 100.000 = 111.100

Sem Informação: Busca Em Profundidade Iterativa

- Desempenho
  - Completo: sim, quando **b** é finito
  - Ótimo: sim, quando o custo de caminho é uma função não decrescente da profundidade do nó
  - Complexidade de tempo: **$O(b^d)$**
  - Complexidade de espaço: **O(bd)**
- Em geral a BFI é o método de busca sem informação **preferido** quando o espaço de busca é **grande** e a profundidade da solução é **desconhecida**

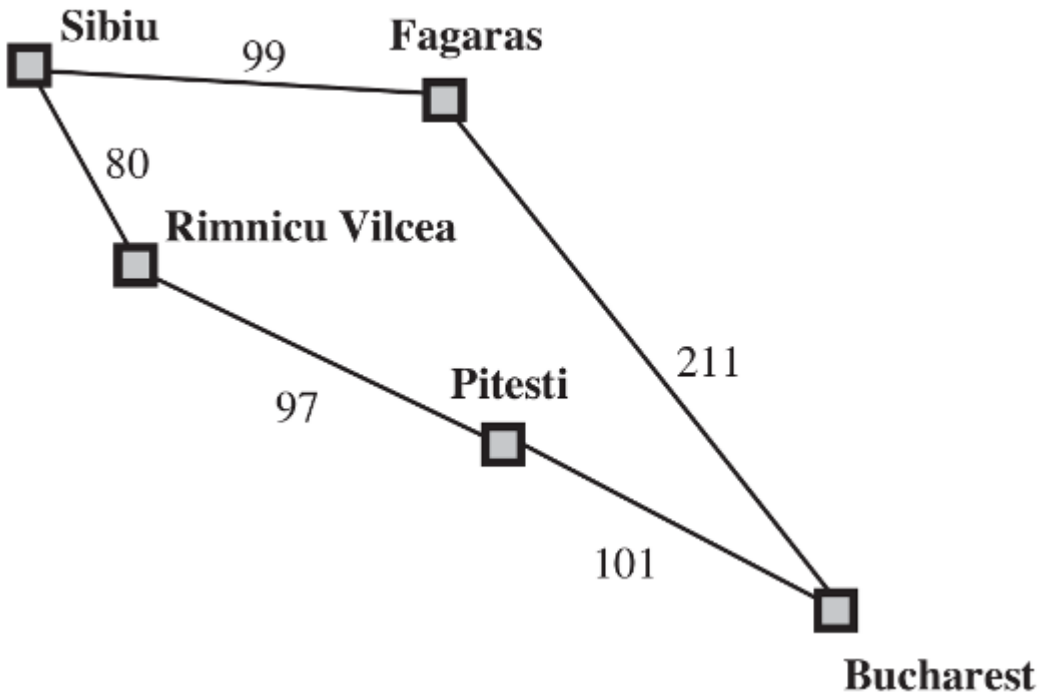
Sem Informação: Busca de Custo Uniforme

- Em um grafo ponderado, a busca de custo uniforme é ótima para qualquer função de custo do passo
- Em vez de expandir o nó mais raso, a busca de custo uniforme expande o nó n com o custo de caminho g(n) mais baixo
- É utilizada uma fila com prioridade na borda

Início: Sibiu

Objetivo: Bucareste

g(n) = custo da raiz até o nó **n.**



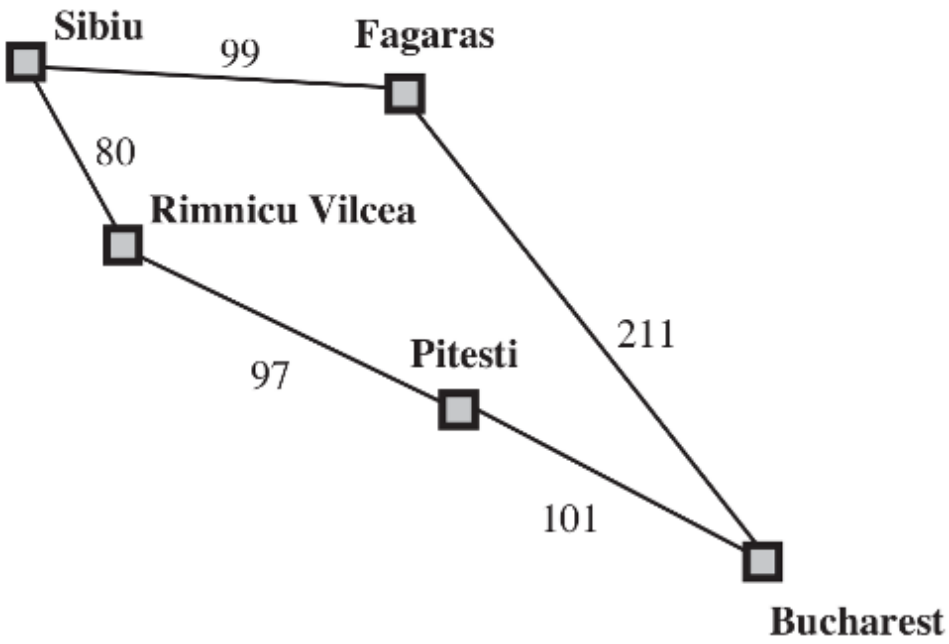
Sem Informação: Busca de Custo Uniforme

- Sucessores de Sibiu: Vilcea (80) e Fagaras (99), nesta ordem expande
  - **Vilcea** → Pitesti

    80 + 97 = 177
  - **Fagaras** → Bucareste

    99 + 211 = 310
- Nó objetivo já na borda (Bucareste), mas a busca de custo uniforme se mantém
- **Pitesti** → Bucareste

  80 + 97 + 101 = 278
- Esse custo é melhor que o antigo, portanto é a solução



Sem Informação: Busca de Custo Uniforme

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

Sem Informação: Busca de Custo Uniforme

- Desempenho
  - Completa: sim
  - Ótima: sim, quando um nó n é selecionado para expansão, o caminho ideal para esse nó foi encontrado, ou seja, o algoritmo expande os nós na ordem de seu custo de caminho ótimo
  - Complexidade de tempo e espaço: é orientada por custos de caminhos em vez de profundidades com $\mathbf{O(b^{1+\lfloor C^*/\varepsilon \rfloor})}$
    - **C***: custo de encontrar a solução ótima
    - **ε**: cada ação custa pelo menos **ε**

- Esse algoritmo não se importa com o número de passos que um caminho tem, mas apenas com o seu custo total

Sem Informação: Busca de Custo Uniforme

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$
Optimal?	Yes[c]	Yes	No	No	Yes[c]

- **b** é o fator de ramificação
- **d** é a profundidade da solução mais rasa
- **m** é a profundidade máxima da árvore
- **i** é o limite de profundidade
- [a] completa se b é finito;
- [b] completa se o custo do passo é positivo
- [c] ótima se os custos dos passos são todos idênticos

Sem Informação: Busca de Custo Uniforme

- **Próxima Aula:**

Busca **Com** Informação