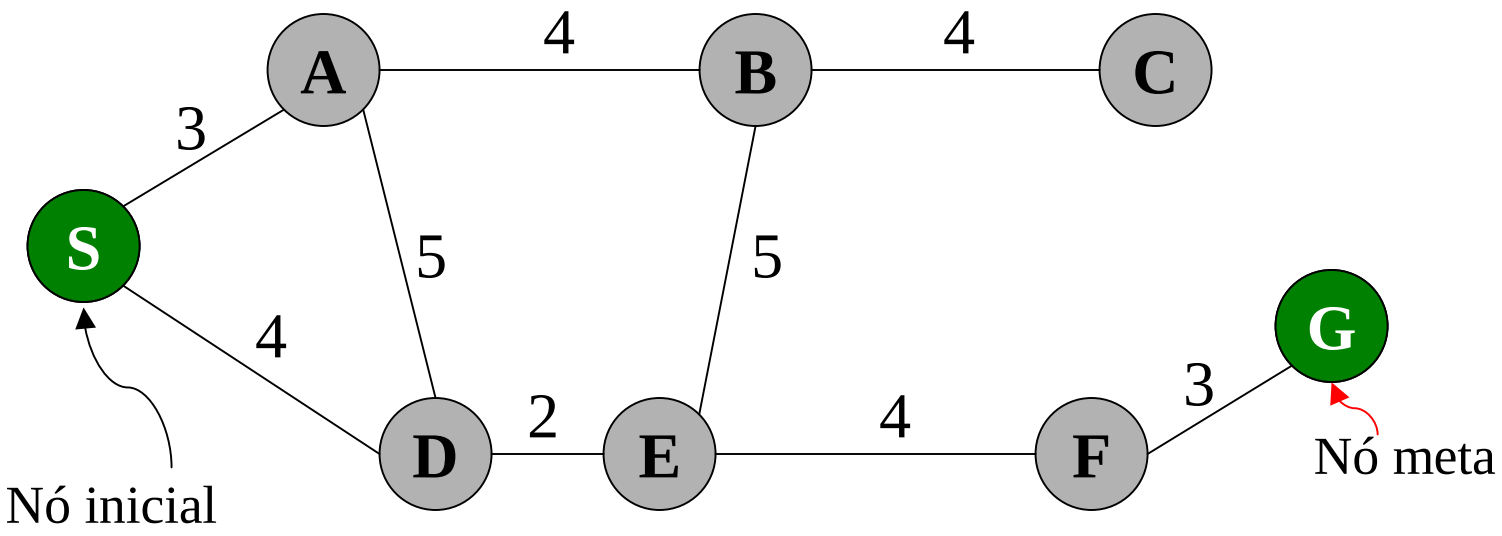
EA 072 Inteligência Artificial
em Aplicações Industriais

Programação de Algoritmos de Busca Não Informados

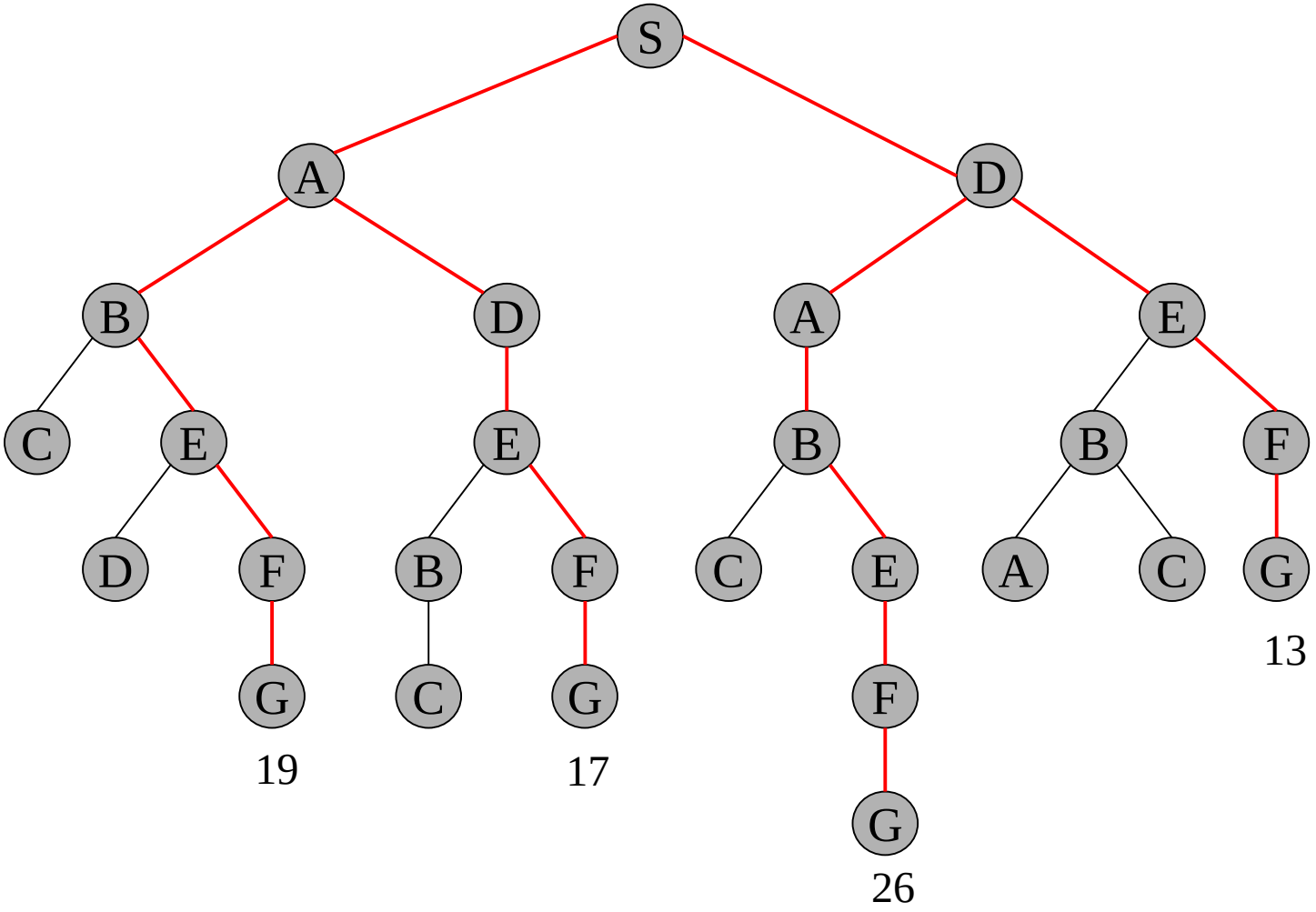
Busca Não Informada em Grafos

Exemplo: caminho entre origem (nó inicial) e destino (nó meta)



Caminho entre S e G ?

Resolvendo o Problema de Busca em Grafos



A maneira óbvia de resolver o problema é enumerar todos os caminhos possíveis.
Em geral, esta estratégia é inviável.

Busca em Profundidade (depth-first search)

function DEPTH_FIRST_SEARCH (*problem*) **returns** a solution, or failure

GENERAL_SEARCH (*problem*, ENQUEUE_AT_FRONT)

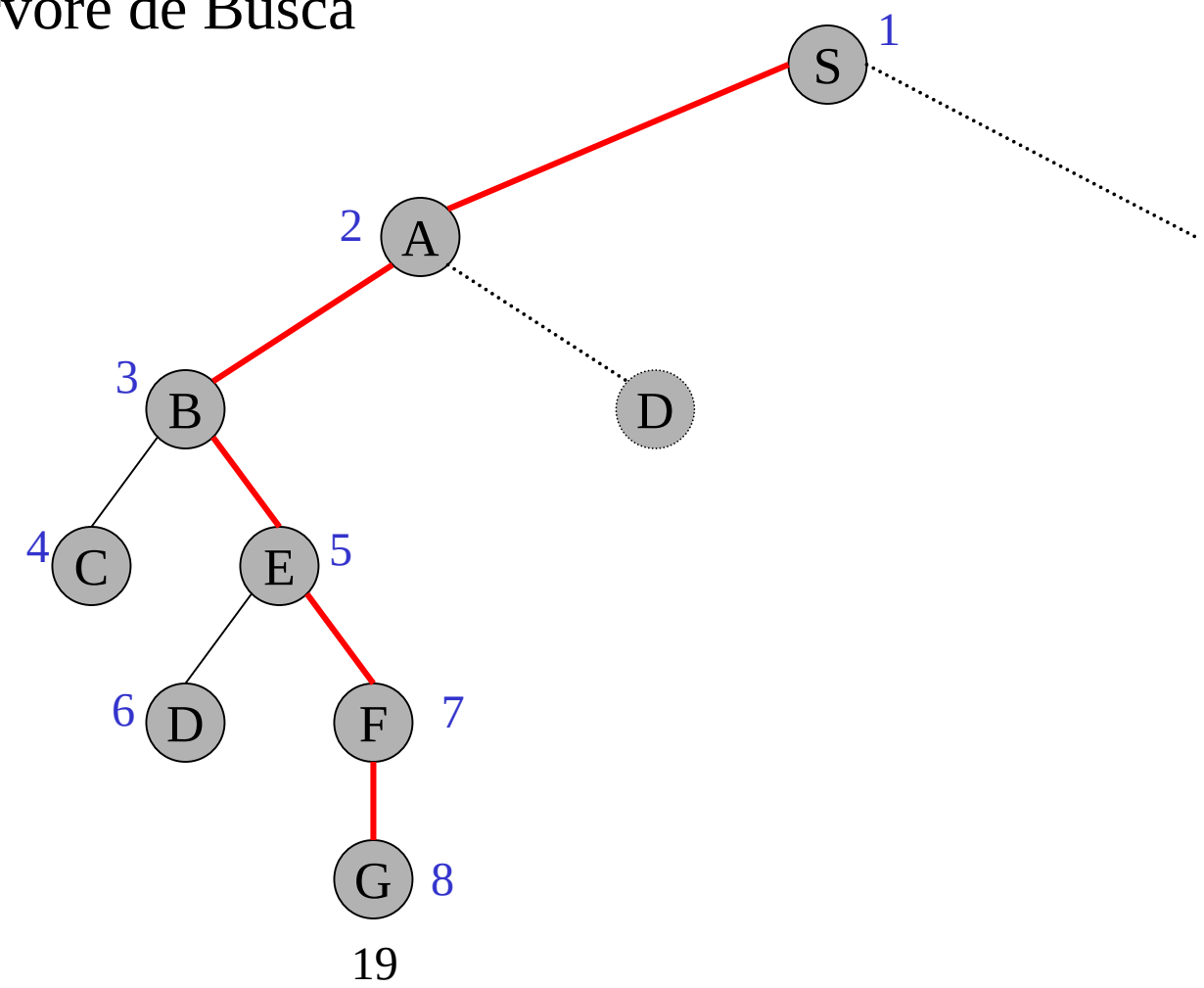
function GENERAL_SEARCH (*problem*, QUEUING_FN) **returns** a solution, or failure

nodes ← MAKE_QUEUE (MAKE_NODE (INITIAL_STATE [*problem*]))

loop do
if *nodes* is empty **then return** failure
node ← REMOVE_FRONT (*nodes*)
if GOAL_TEST [*problem*] applied to STATE (*node*) succeeds
then return *node*
nodes ← QUEUING_FN (*nodes*, EXPAND (*node*, OPERATORS [*problem*]))
end

Busca em Profundidade (depth-first search)

- Árvore de Busca



Busca em Largura (breadth-first search)

function BREADTH_FIRST_SEARCH (*problem*) **returns** a solution, or failure

GENERAL_SEARCH (*problem*, ENQUEUE_AT_END)

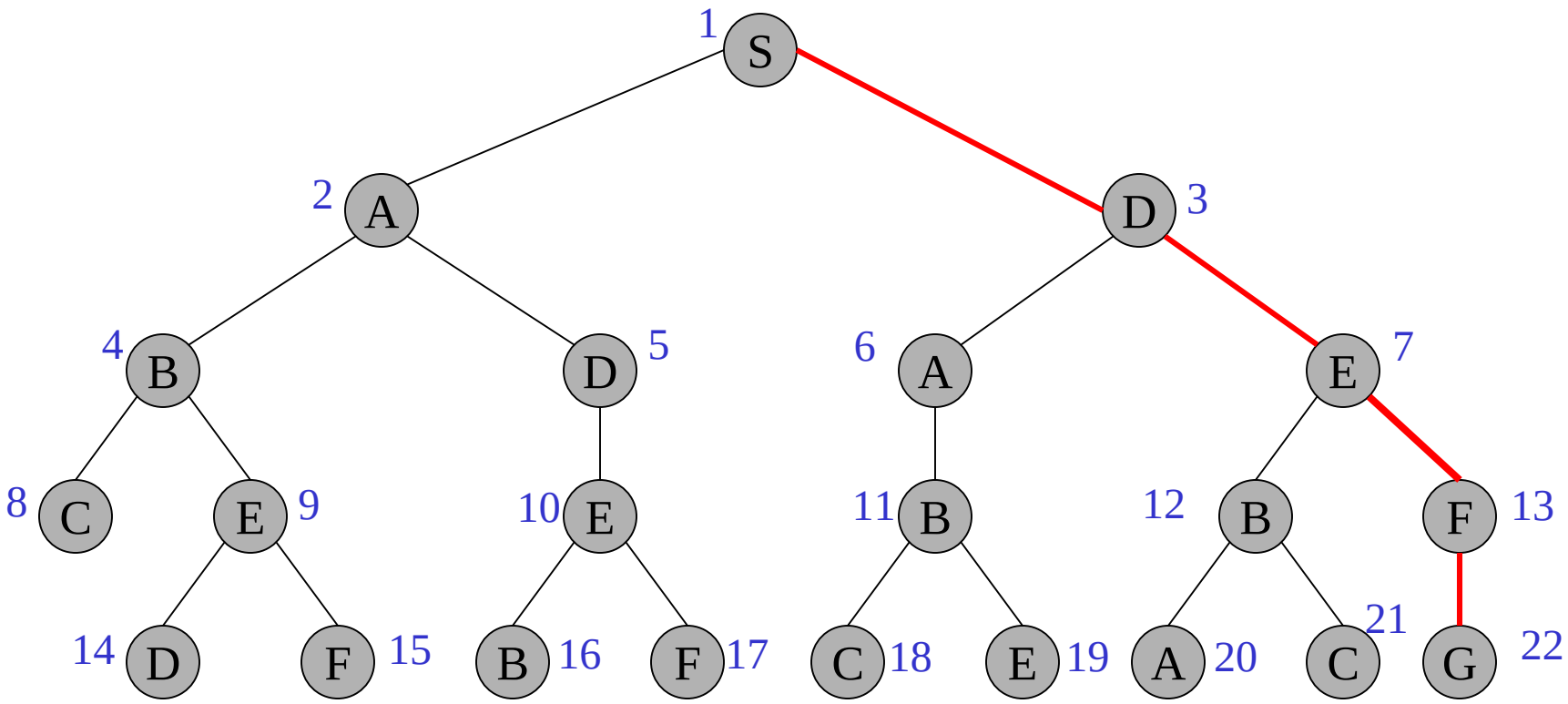
function GENERAL_SEARCH (*problem*, QUEUING_FN) **returns** a solution, or failure

nodes ← MAKE_QUEUE (MAKE_NODE (INITIAL_STATE [*problem*]))

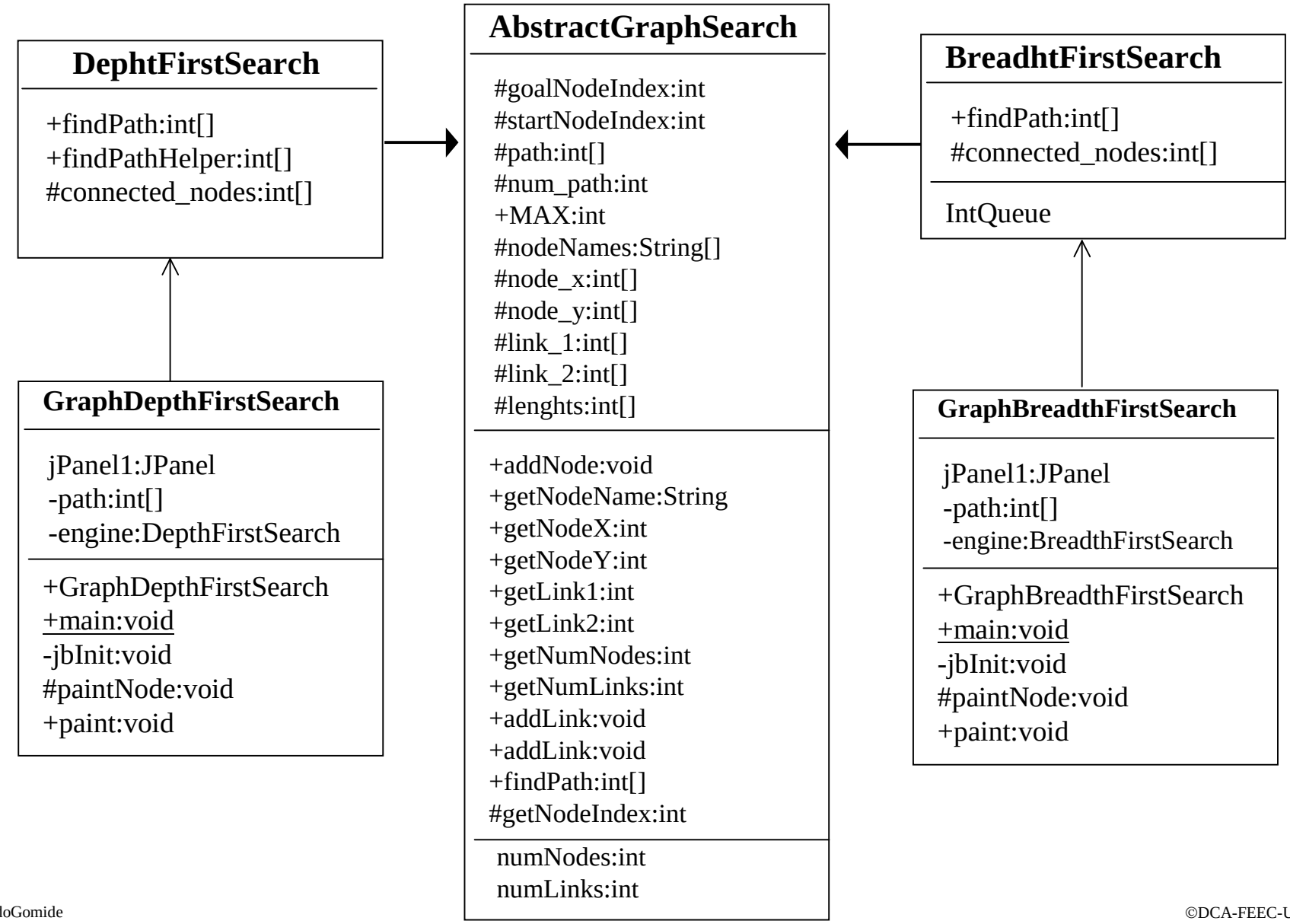
loop do
if *nodes* is empty **then return** failure
node ← REMOVE_FRONT (*nodes*)
if GOAL_TEST [*problem*] applied to STATE (*node*) succeeds
then return *node*
nodes ← QUEUING_FN (*nodes*, EXPAND (*node*, OPERATORS [*problem*]))
end

Busca em Largura (breadth-first search)

- Árvore de Busca



Implementação dos Algoritmos em Java



Implementação dos Algoritmos em Java

- **AbstractGraphSearch**: classe que aloca os dados requeridos pelas classes derivadas;

  – Métodos:

  addNode(String name, int x, int y) – adiciona um novo nó;
  addLink(int n1, int n2) – adiciona um link bidirecional entre os nós indexados por n1 e n2. (indíces começam em zero);
  addLink(String n1, String n2) – adiciona um link bidirecional entre nós especificados por seus nomes;
  getNumNodes() – retorna o número de nós;
  getNumLinks() – retorna o número de links;
  getNodeName(int index) – retorna o nome do nó;
  getNodeIndex(String name) – retorna o índice do nó, dado seu nome;
  getNodeX(int index) – retorna a coordenada X do nó de índice index;
  getNodeY(int index) – retorna a coordenada Y do nó de índice index;
  getLink1(int index) – retorna o nó armazenado na posição index de Link1;
  getLink2(int index) – retorna o nó armazenado na posição index de Link2;

  – Método abstrato:

  public int[] findPath(int start_node, int goal_node)

Implementação dos Algoritmos em Java

- **DepthFirstSearch**: classe que aloca os dados requeridos pelas classes derivadas;

  – Métodos:

  int[] findPath(int start_node, int goal_node) – retorna um vetor com os indices dos nós do caminho encontrado;

  int [] findPathHelper(int [] path, int num_path, int goal_node) – método que auxilia findPath, fazendo a busca recursiva nos nós do caminho;

  int [] connected_nodes(int [] path, int num_path) – encontra todos os nós conectados ao último nó do caminho que não estão no caminho.

Implementação dos Algoritmos em Java

- **BreadthFirstSearch**: classe que aloca os dados requeridos pelas classes derivadas;

  – Métodos:

  int[] findPath(int start_node, int goal_node) – retorna um vetor com os indices dos nós do caminho encontrado;

  int [] connected_nodes(int [] path, int num_path) – encontra todos os nós conectados ao último nó do caminho que não estão no caminho.

  class IntQueue() – métodos para manipulação da fila;

Implementação dos Algoritmos em Lisp

- Entrada dos dados do grafo

```
;;;; DADOS

(setf (get 's 'neighbors) '(a d)
   (get 'a 'neighbors) '(s b d)
   (get 'b 'neighbors) '(a c e)
   (get 'c 'neighbors) '(b)
   (get 'd 'neighbors) '(s a e)
   (get 'e 'neighbors) '(b d f)
   (get 'f 'neighbors) '(e))
   (get 'g 'neighbors) '(f))

(setf (get 's ‘coordinates) '(0 3)
    (get 'a 'coordinates) '(3 3)
    (get 'b 'coordinates) '(7 3)
   (get 'c 'coordinates) '(11 3)
    (get 'd 'coordinates) '(0 7)
    (get 'e 'coordinates) '(2 7)
    (get 'f 'coordinates) '(6 7))
    (get ‘g 'coordinates) '(10 7))
```

Implementação dos Algoritmos em Lisp

- Busca em Profundidade

```
(defun extend (path)
(print (reverse path))                          ;Imprime o caminho.
(mapcar #'(lambda (new-node) (cons new-node path)) ;Cria novos caminhos.
      (remove-if #'(lambda (neighbor) (member neighbor path))
                 (get (first path) 'neighbors))))


(defun depth-first (start finish &optional
                           (queue (list (list start))))
  (cond ((endp queue) nil)                    ; Verifica se a pilha está vazia.
        ((eq finish (first (first queue)))    ; Verifica se o nó meta foi atingido.
         (reverse (first queue)))             ; Retorna o caminho.
        (t (depth-first                       ; Tenta novamente.
        start
        finish
       (append (extend (first queue))     ; Novos caminhos no inicio da fila.
               (rest queue))))))          ; Retira o caminho expandido da fila.
```

Implementação dos Algoritmos em Lisp

- Busca em Profundidade – comportamento de queue

Início
((S))
Primeira chamada recursiva
((A S) (D S))
Segunda chamada recursiva
((B A S) (D A S) (D S))
Terceira chamada recursiva
((C B A S) (E B A S) (D A S) (D S))
Quarta chamada recursiva
((E B A S) (D A S) (D S))
Quinta chamada recursiva
((D E B A S) (F E B A S) (D A S) (D S))
Sexta chamada recursiva
((F E B A S) (D A S) (D S))
Sétima chamada recursiva
((G F E B A S) (D A S) (D S))

Implementação dos Algoritmos em Lisp

- Busca em Largura

```
(defun extend (path)
   (print (reverse path))                          ; Imprime o caminho.
   (mapcar #'(lambda (new-node) (cons new-node path)) ; Cria novos caminhos.
         (remove-if #'(lambda (neighbor) (member neighbor path))
               (get (first path) 'neighbors))))


(defun breadth-first (start finish &optional
                           (queue (list (list start))))
  (cond ((endp queue) nil)                    ;Verifica se a fila está vazia.
       ((eq finish (first (first queue)))     ;Verifica se o nó meta foi atingido.
        (reverse (first queue)))              ;Retorna o caminho.
      (t (breadth-first                       ;Tenta novamente.
      start
      finish
     (append (rest queue)     ; Retira o caminho expandido da fila.
           (extend(first queue))))))         ; Novos caminhos no final da fila.
```

Implementação dos Algoritmos em Lisp

- Busca em Largura – comportamento de queue

Início
((S))
Primeira chamada recursiva
((A S) (D S))
Segunda chamada recursiva
((D S) (B A S) (D A S))
Terceira chamada recursiva
((B A S) (D A S) (A D S) (E D S))
Quarta chamada recursiva
((D A S) (A D S) (E D S) (C B A S) (E B A S))
Quinta chamada recursiva
((A D S) (E D S) (C B A S) (E B A S) (E D A S))
Sexta chamada recursiva
((E D S) (C B A S) (E B A S) (E D A S) (B A D S))
Sétima chamada recursiva
((C B A S) (E B A S) (E D A S) (B A D S) (B E D S) (F E D S))
Oitava chamada recursiva
((E B A S) (E D A S) (B A D S) (B E D S) (F E D S))

Implementação dos Algoritmos em Lisp

- Busca em Largura – comportamento de queue

Nona chamada recursiva
((E D A S) (B A D S) (B E D S) (F E D S) (D E B A S) (F E B A S))
Décima chamada recursiva
((B A D S) (B E D S) (F E D S) (D E B A S) (F E B A S) (B E D A S) (F E D A S))
Décima primeira chamada recursiva
((B E D S) (F E D S) (D E B A S) (F E B A S) (B E D A S) (F E D A S) (C B A D S)
(E B A D S))
Décima segunda chamada recursiva
((F E D S) (D E B A S) (F E B A S) (B E D A S) (F E D A S) (C B A D S) (E B A D S)
(A B E D S) (C B E D S))

Observação

Este material refere-se às notas de aula do curso EA 072 Inteligência Artificial em Aplicações Industriais da Faculdade de Engenharia Elétrica e de Computação da Unicamp. Não substitui o livro texto, as referências recomendadas e nem as aulas expositivas. Este material não pode ser reproduzido sem autorização prévia dos autores. Quando autorizado, seu uso é exclusivo para atividades de ensino e pesquisa em instituições sem fins lucrativos.