GRAFOS

Algoritmos de Pesquisa

Grafos

Propriedades

- Algumas propriedades simples de grafos são fáceis de determinar, pois não dependendo da ordem pela qual se examinam as arestas
  - por exemplo: o grau de todos os vértices
- Outras propriedades, como as associadas a caminhos, é necessário identificá-las através de pesquisa realizada de vértice em vértice ao longo das arestas do caminho
- A maioria dos algoritmos que estudaremos usa este modelo abstrato básico
- Torna-se então necessário analisar o essencial dos algoritmos de pesquisa em grafos e as suas propriedades estruturais
- Pesquisar em grafos é equivalente a percorrer labirintos, sendo necessário
  - marcar pontos já visitados
  - ser-se capaz de recuar, num caminho efetuado, até ao ponto de partida

Algoritmos de pesquisa

Definição

- Dado um grafo G = (V, A) e um vértice S de G, um algoritmo de pesquisa explora o grafo passando por todos os vértices alcançáveis a partir de S
- Serão estudados dois tipos de algoritmos de pesquisa em grafos
  - pesquisa em Largura Primeiro (BFS – "Breadth-first-search")
    - usando uma **EAD Fila** (FIFO)
  - pesquisa em Profundidade Primeiro (DFS – "Depth-first-search")
    - usando uma **EAD Pilha** (LIFO) ou a **recursividade**
  - estes algoritmos designam-se também por algoritmos de travessia em grafos

Algoritmo de Pesquisa em Largura Primeiro (BFS)

Propriedades

- O algoritmo de pesquisa em largura primeiro
  - calcula a distância (menor número de arestas) de S a cada vértice de G
  - produz uma árvore (subgrafo de G) com raiz em S contendo todos os vértices alcançáveis a partir de S, denominada de **Árvore de Pesquisa em Largura** (**APL**)
  - nessa árvore o caminho da raiz S a cada vértice corresponde ao caminho mais curto entre os dois vértices
  - aplicado a grafos orientados e não orientados

Propriedades

- Utiliza a seguinte estratégia para efectuar a travessia do grafo
  - todos os vértices à distância k de S são visitados antes de qualquer vértice à distância k+1
- O algoritmo pinta os vértices (inicialmente brancos) de cinzento ou preto
  - um vértice **branco** ainda não foi descoberto
  - um vértice **cinzento** já foi visitado mas pode ter adjacentes ainda não descobertos (brancos)
  - um vértice **preto** só tem adjacentes já descobertos (cinzentos ou pretos)
- Os cinzentos correspondem à fronteira entre descobertos e não descobertos
- A árvore de travessia em largura é expandida atravessando-se a lista de adjacências de cada vértice cinzento **u**
  - para cada vértice adjacente **v** acrescenta-se à árvore a aresta (**u**, **v**)

Algoritmo

Entrada: grafo G e vértice inicial S

Saída: arrays 1D de inteiros **Pai**, **Cor** e **Dist**

algoritmo BFS:
  para (cada vértice **u** $\in$ V) **repetir**
    Cor[u] $\leftarrow$ branco
    Dist[u] $\leftarrow \infty$
    Pai[u] $\leftarrow$ NULO
  fim_para
  Cor[S] $\leftarrow$ cinzento
  Dist[S] $\leftarrow$ 0
  Q $\leftarrow$ **criarFila**()
  Q $\leftarrow$ **juntar**(S, Q)
  ...

Algoritmo (cont)

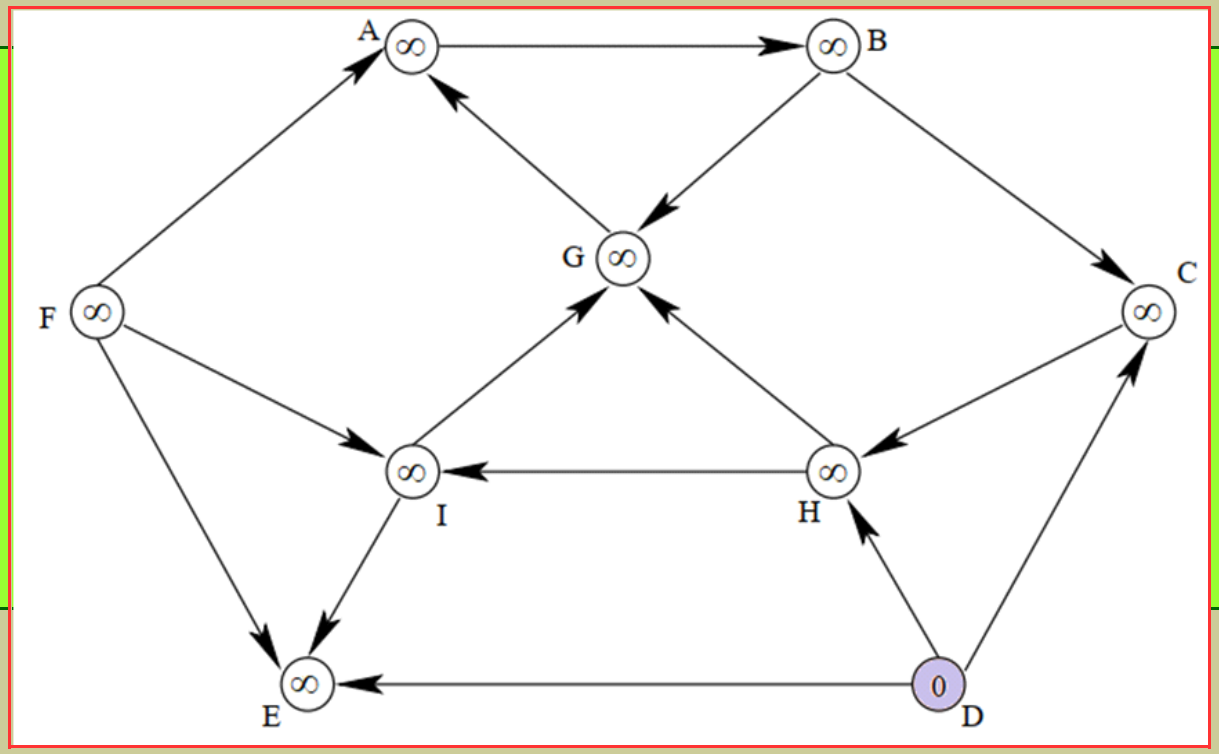
```
  ...
  enquanto (filaVazia(Q) = falso) repetir
     u ← frente(Q)
     Q ← remover(Q)
     para (cada vértice v adjacente ao vértice u) repetir
        se (Cor[v] = branco) então
           Q ← juntar(v, Q)
           Cor[v] ← cinzento
           Dist[v] ← Dist[u] + 1
           Pai[v] ← u
        fim_se
     fim_para
     Cor[u] ← preto
  fim_enquanto
fim_algoritmo
```

Estruturas de dados do algoritmo

- Utiliza-se uma fila de espera (estrutura linear FIFO) – ver EAD "Fila"
- Outras estruturas de dados
  - array 1D **Cor** para guardar as cores dos vértices,
  - array 1D **Dist** para as distâncias ao vértice S,
  - array 1D **Pai** para o pai de cada vértice na árvore construída
- Normalmente utiliza-se uma representação por listas de adjacências para guardar os dados do grafo
- A árvore construída não é única, pois depende da ordem pela qual são visitados os vértices adjacentes de um determinado vértice
- Como nem todos os vértices podem ser alcançáveis a partir do nó S, o grafo de pesquisa em G pode ser uma floresta composta por várias **árvores de pesquisa em largura**

Exemplo (S = D)

u ← Frente(Q) = D
Q ← [ ]

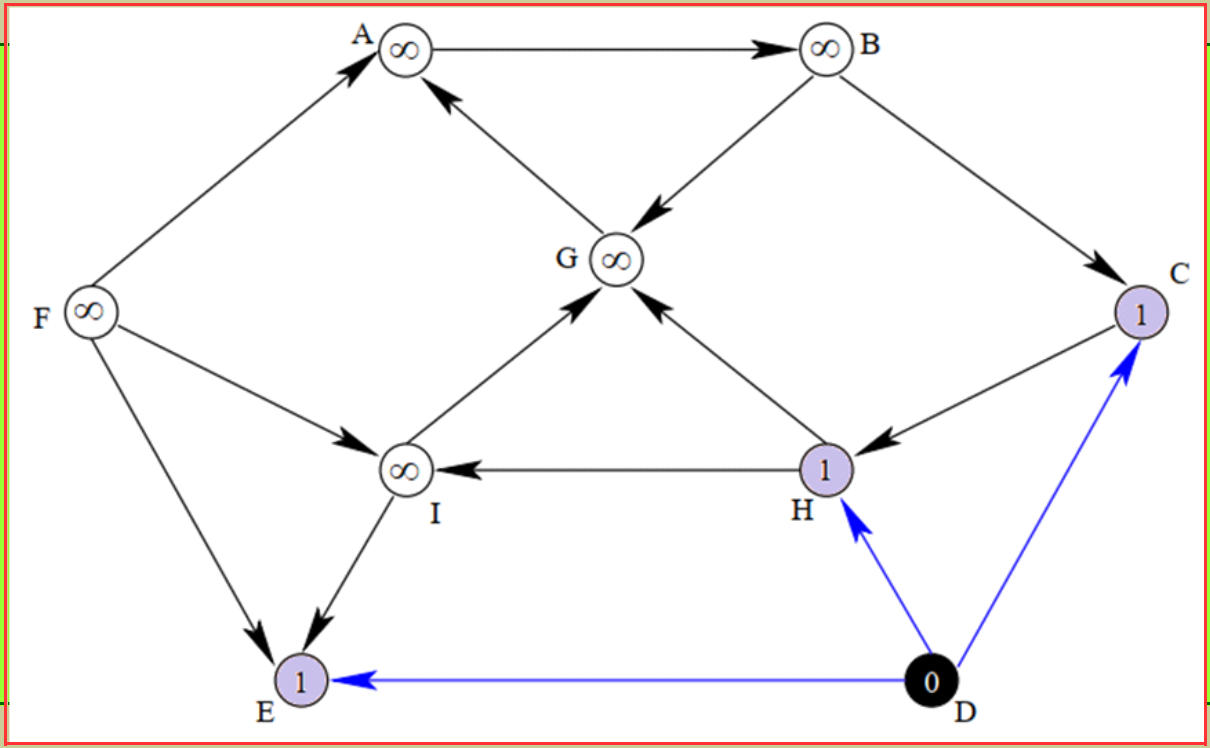


para (cada vértice **u** ∈ V) **repetir**
  Cor[u] ← branco
  Dist[u] ← ∞
  Pai[u] ← NULO
fim_para
Cor[S] ← cinzento
Dist[S] ← 0
Q ← **criarFila**()
Q ← **juntar**(S, Q)

Q ← [D]
Pai[D] ← NULO

Exemplo (S = D)

u ← Frente(Q) = D
Q ← [ ]

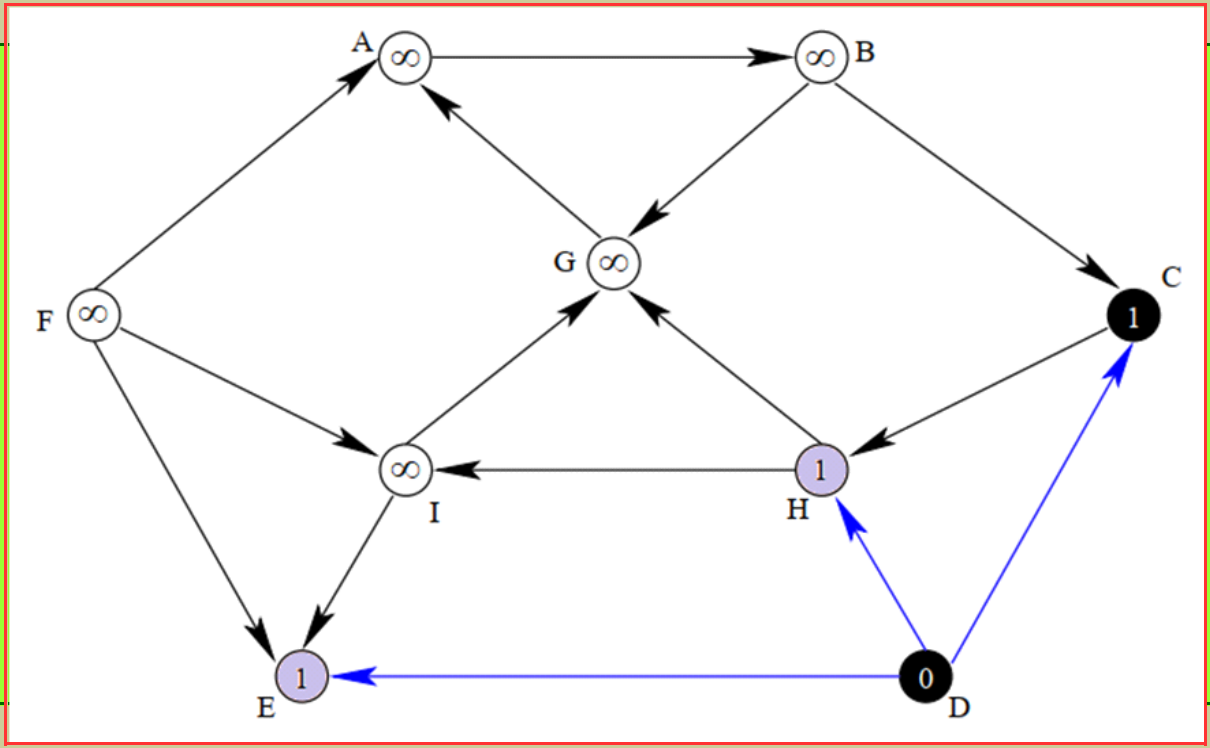



```
enquanto (filaVazia(Q) = falso) repetir
   u ← frente(Q)
   Q ← remover(Q)
   para (cada vértice v adjacente a u) repetir
      se (Cor[v] = branco) então
         Q ← juntar(v, Q)
         Cor[v] ← cinzento
         Dist[v] ← Dist[u] + 1
         Pai[v] ← u
      fim_se
   fim_para
   Cor[u] ← preto
fim_enquanto
```

Q ← [C, E, H]
Pai[C] ← D; Pai[E] ← D; Pai[H] ← D

Exemplo (S = D)

u ← Frente(Q) = C
Q ← [E, H]

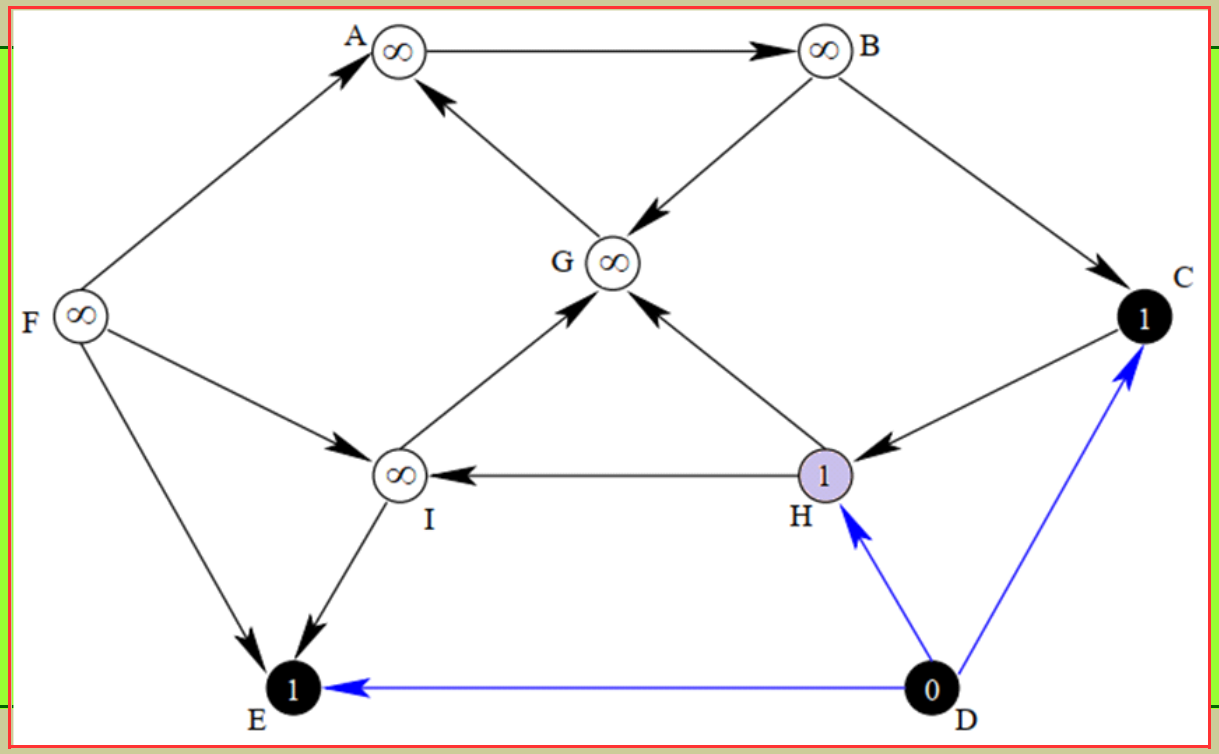



```
enquanto (filaVazia(Q) = falso) repetir
   u ← frente(Q)
   Q ← remover(Q)
   para (cada vértice v adjacente a u) repetir
      se (Cor[v] = branco) então
         Q ← juntar(v, Q)
         Cor[v] ← cinzento
         Dist[v] ← Dist[u] + 1
         Pai[v] ← u
      fim_se
   fim_para
   Cor[u] ← preto
fim_enquanto
```

Q = [E, H]

Exemplo (S = D)

u ← Frente(Q) = E
Q ← [H]

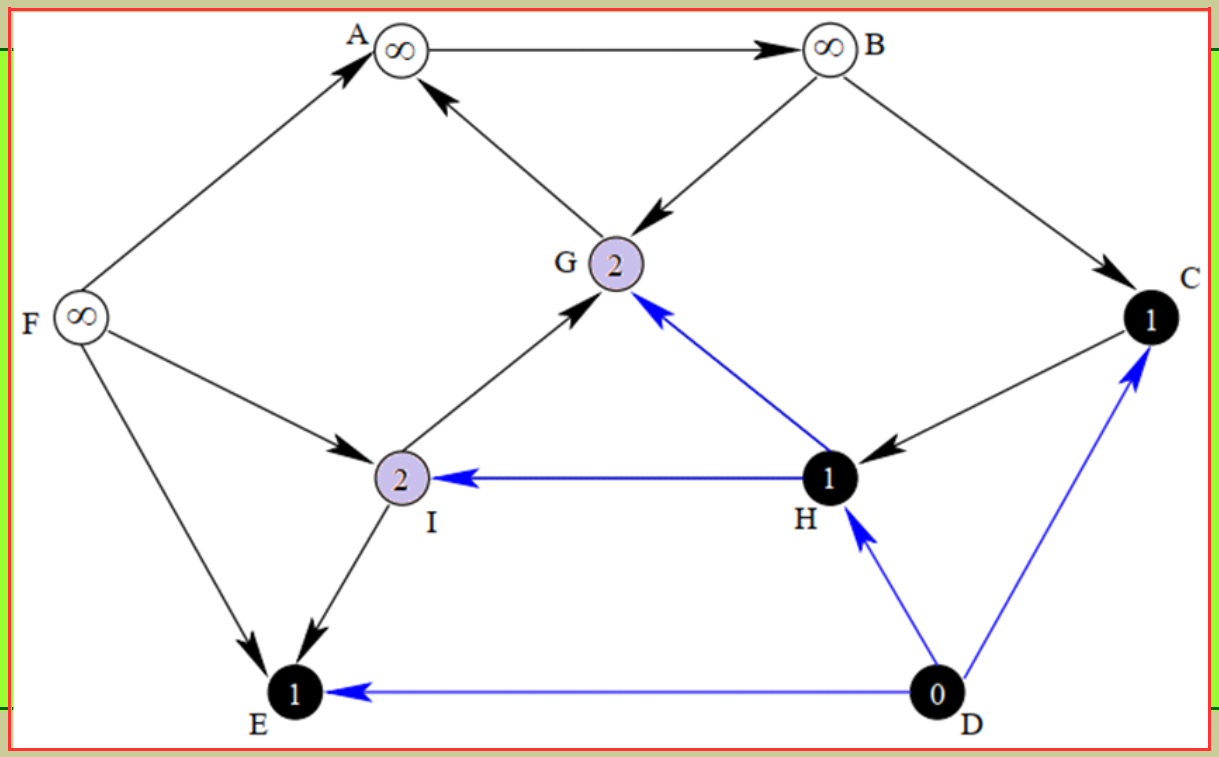



```
enquanto (filaVazia(Q) = falso) repetir
   u ← frente(Q)
   Q ← remover(Q)
   para (cada vértice v adjacente a u) repetir
      se (Cor[v] = branco) então
         Q ← juntar(v, Q)
         Cor[v] ← cinzento
         Dist[v] ← Dist[u] + 1
         Pai[v] ← u
      fim_se
   fim_para
   Cor[u] ← preto
fim_enquanto
```

Q = [H]

Exemplo (S = D)

u ← Frente(Q) = H
Q ← [ ]

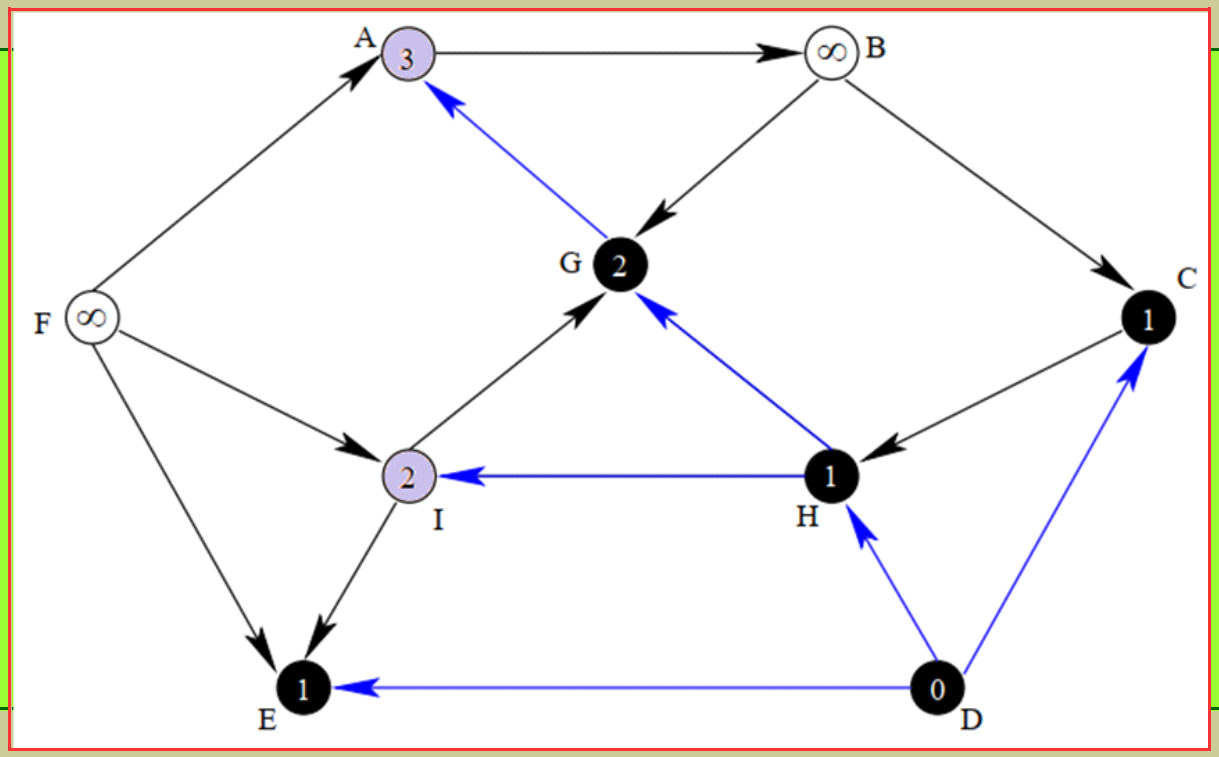



```
enquanto (filaVazia(Q) = falso) repetir
   u ← frente(Q)
   Q ← remover(Q)
   para (cada vértice v adjacente a u) repetir
      se (Cor[v] = branco) então
         Q ← juntar(v, Q)
         Cor[v] ← cinzento
         Dist[v] ← Dist[u] + 1
         Pai[v] ← u
      fim_se
   fim_para
   Cor[u] ← preto
fim_enquanto
```

Q ← [G, I]
Pai[G] ← H; Pai[I] ← H

Exemplo (S = D)

u ← Frente(Q) = G
Q ← [I]

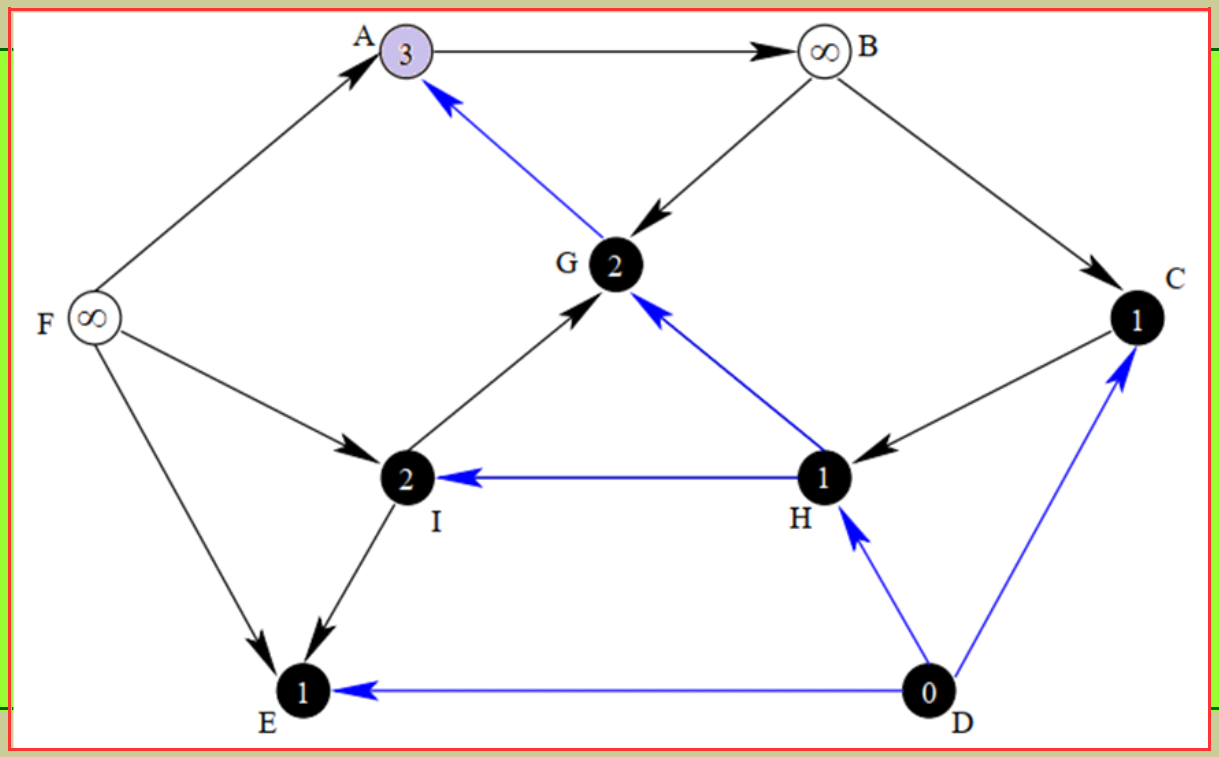



```
enquanto (filaVazia(Q) = falso) repetir
   u ← frente(Q)
   Q ← remover(Q)
   para (cada vértice v adjacente a u) repetir
      se (Cor[v] = branco) então
         Q ← juntar(v, Q)
         Cor[v] ← cinzento
         Dist[v] ← Dist[u] + 1
         Pai[v] ← u
      fim_se
   fim_para
   Cor[u] ← preto
fim_enquanto
```

Q ← [I, A]
Pai[A] ← G

Exemplo (S = D)

u ← Frente(Q) = I
Q ← [A]

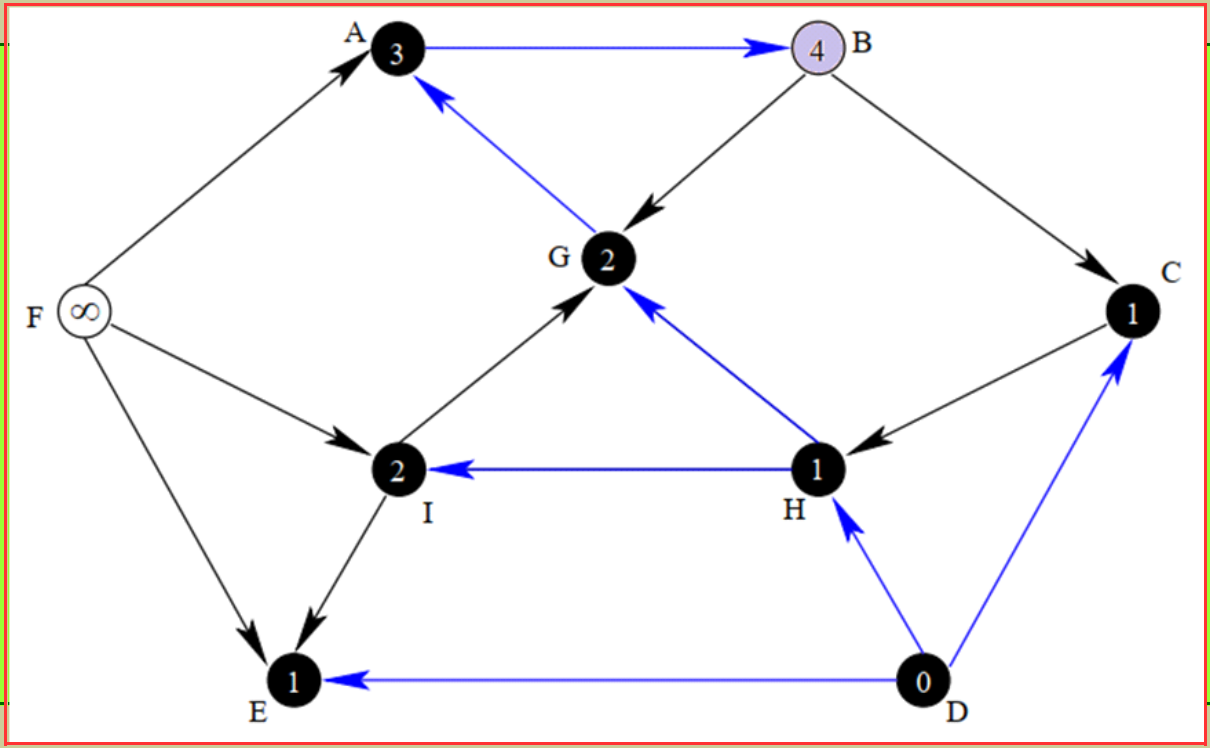



```
enquanto (filaVazia(Q) = falso) repetir
   u ← frente(Q)
   Q ← remover(Q)
   para (cada vértice v adjacente a u) repetir
      se (Cor[v] = branco) então
         Q ← juntar(v, Q)
         Cor[v] ← cinzento
         Dist[v] ← Dist[u] + 1
         Pai[v] ← u
      fim_se
   fim_para
   Cor[u] ← preto
fim_enquanto
```

Q = [A]

Exemplo (S = D)

u ← Frente(Q) = A
Q ← [ ]

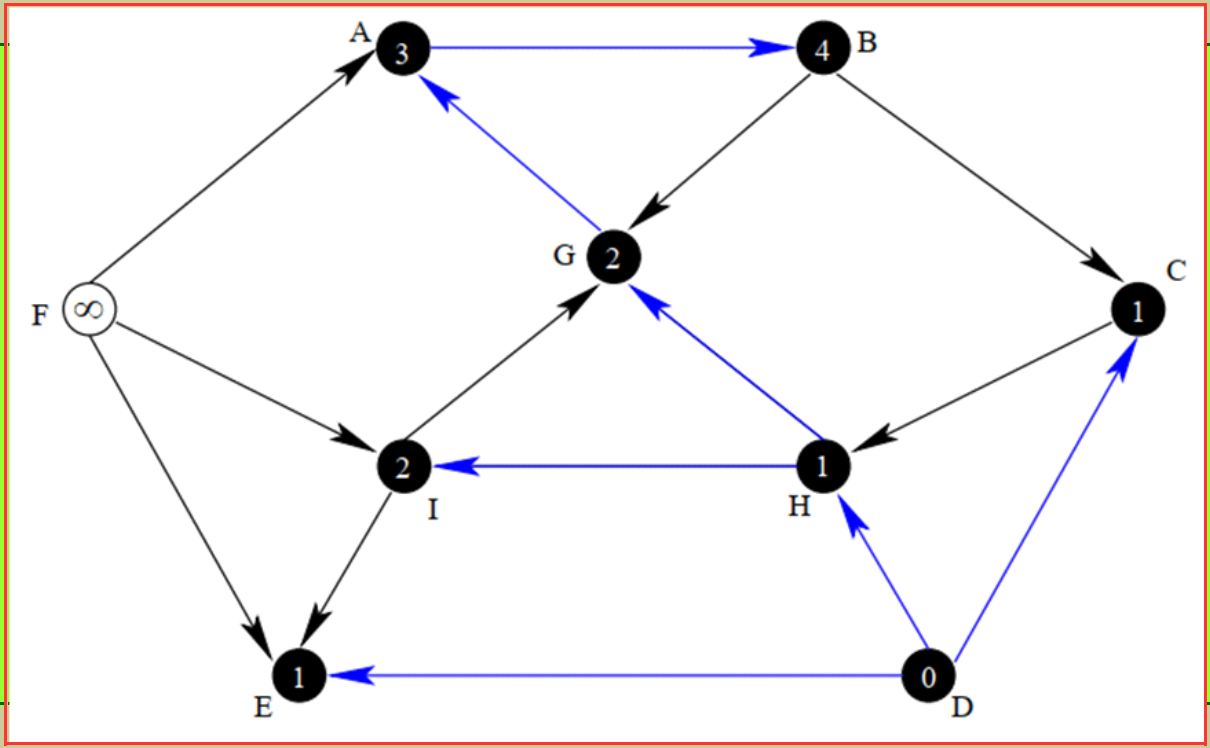



```
enquanto (filaVazia(Q) = falso) repetir
   u ← frente(Q)
   Q ← remover(Q)
   para (cada vértice v adjacente a u) repetir
      se (Cor[v] = branco) então
         Q ← juntar(v, Q)
         Cor[v] ← cinzento
         Dist[v] ← Dist[u] + 1
         Pai[v] ← u
      fim_se
   fim_para
   Cor[u] ← preto
fim_enquanto
```

Q ← [B]
Pai[B] ← A

Exemplo (S = D)

u ← Frente(Q) = B
Q ← [ ]




```
enquanto (filaVazia(Q) = falso) repetir
   u ← frente(Q)
   Q ← remover(Q)
   para (cada vértice v adjacente a u) repetir
      se (Cor[v] = branco) então
         Q ← juntar(v, Q)
         Cor[v] ← cinzento
         Dist[v] ← Dist[u] + 1
         Pai[v] ← u
      fim_se
   fim_para
   Cor[u] ← preto
fim_enquanto
```

Q = [ ]

Exemplo (resultados)

- Como o vértice F não foi alcançável a partir do vértice D, o algoritmo deve ser novamente aplicado à parte do grafo com os vértices ainda não alcancáveis
- Em geral, num grafo não fortemente ligado, é necessário iniciar uma nova pesquisa em cada componente ligado, para garantir que todos os vértices são alcançados
  - o algoritmo deve ser aplicado sucesivamente aos nós que não são alcançáveis para determinar novas outras APL's, formando todas elas uma floresta
- A árvore de pesquisa em largura do grafo (a **azul** no exemplo) fica definida pelo array **Pai**
  - Pai[C] = D; Pai[E] = D; Pai[H] = D
  - Pai[G] = H; Pai[I] = H
  - Pai[A] = G
  - Pai[B] = A

Árvore de Pesquisa em Largura

- Dado G = (V, A) e um array **Pai** dos antecessores de cada vértice,

  define-se $G_{pai} = (V_{Pai}, A_{Pai})$, o sub-grafo dos antecessores de G, como se segue

  $$V_{pai} = \{ v \in V \mid Pai[v] \neq NULL \} \cup \{ S \}$$

  $$A_{pai} = \{ (Pai[v], v) \mid v \in V_{pai} - \{ S \} \}$$

- Este grafo é uma **árvore de pesquisa em largura** de G a partir de **S** se
  - V_{Pai} for o conjunto de vértices alcançáveis a partir de **S**, e
  - para cada **v** $\in V_{Pai}$ o caminho mais curto (em G) de **S** até **v** pertencer a G_{Pai}

- Teorema (Para G orientado ou não orientado)

  O algoritmo de pesquisa em largura constrói um array **Pai** tal que $\mathbf{G_{Pai}}$ é uma **árvore de pesquisa em largura** de G

Algoritmo de Pesquisa em Profundidade Primeiro (DFS)

Propriedades

- Este algoritmo utiliza a seguinte estratégia para efectuar a travessia do grafo
  - as próximas arestas a explorar têm origem no mais recente vértice descoberto que ainda tenha vértices adjacentes não explorados
- Assim, quando todos os vértices adjacentes ao vértice **v** tiverem sido explorados, o algoritmo recua (“backtracks”) para explorar vértices com origem no vértice a partir do qual **v** foi descoberto
- Serão estudados **duas** versões deste algoritmo que percorre todos os vértices do grafo, uma usando uma EAD Pilha e a outra a recusrsividade, que consiste em
  - depois de terminada a pesquisa com origem em **S**, serão feitas novas pesquisas para descobrir os vértices não alcançáveis a partir de **S**
- O grafo de pesquisa em G determinado pode ser uma floresta composta por várias **árvores de pesquisa em profundidade (APP)**

Propriedades

- Os algoritmos usam a coloração (branco, cinzento e preto) para garantir que cada vértice pertence a uma única árvore; estas árvores são disjuntas
- O algoritmo 1 usa uma EAD Pilha para guardar os nós por onde passa
- O algoritmo 2 usa a recursividade
- Ambos os algoritmos usam as mesmas estruturas de dados
  - array 1D **Cor** para guardar as cores dos vértices,
  - array 1D **Dist** para as distâncias ao vértice S,
  - array 1D **Pai** para o pai de cada vértice na árvore construída
- Normalmente utiliza-se uma representação por listas de adjacências para guardar os dados do grafo

Algoritmo 1

Entrada: grafo G e vértice inicial S
Saída: arrays 1D de inteiros **Pai**, **Cor** e **Dist**

algoritmo DFS:
  para (cada vértice **u** $\in$ V) **repetir**
    Cor[u] $\leftarrow$ branco
    Dist[u] $\leftarrow \infty$
    Pai[u] $\leftarrow$ NULO
  fim_para
  Cor[S] $\leftarrow$ cinzento
  Dist[S] $\leftarrow$ 0
  P $\leftarrow$ **criarPilha**()
  P $\leftarrow$ **push**(S, P)
  ...

Algoritmo 1 (cont)

```
  ...
  enquanto (pilhaVazia(P) = falso) repetir
     u ← topo(P)
     P ← pop(P)
     se (Cor[u] ≠ preto) então
        para (cada vértice v adjacente ao vértice u) repetir
           se (Cor[v] = branco  ou  Cor[v] = cinzento) então
              P ← push(v, P)
              Cor[v] ← cinzento
              Dist[v] ← Dist[u] + 1
              Pai[v] ← u
           fim_se
        fim_para
        Cor[u] ← preto
     fim_se
  fim_enquanto
fim_algoritmo
```

Algoritmo 2

- Este algoritmo é composto por 2 partes
  - parte 1
    - atribui valores iniciais às estruturas de dados
    - chama o algoritmo da parte 2
  - parte 2
    - implementa um método recursivo traduzindo o algoritmo de pesquisa propriamente dito
- Parte 1 do algoritmo
  - atribuir valores iniciais às estruturas de dados: Pai, Dist e Cor
  - inicia a pesquisa no vértice S e nos restantes vértices não alcançáveis a partir de S
  - chama o algoritmo da parte 2
- Parte 2 do algoritmo
  - aplica o algoritmo de pesquisa

Algoritmo 2 – parte 1

Entrada: grafo G e vértice inicial S
Saída: arrays 1D de inteiros **Pai**, **Cor** e **Dist**

```
algoritmo pesquisaDFS:
  para (cada vértice u ∈ V) repetir
      Cor[u] ← branco
      Dist[u] ← ∞
      Pai[u] ← NULO
  fim_para
  Cor[S] ← cinzento
  Dist[S] ← 0
  DFS(G, S, Cor, Dist, Pai)
  para (cada vértice u ∈ V: Cor[u] = branco) repetir
      Cor[u] ← cinzento
      Dist[u] ← 0
      DFS(G, u, Cor, Dist, Pai)
  fim_para
fim_algoritmo
```

Algoritmo 2 – parte 2

Entrada**:** grafo G, vértice u, arrays Cor, Dist e Pai
Saída: arrays Cor, Dist e Pai atualizados

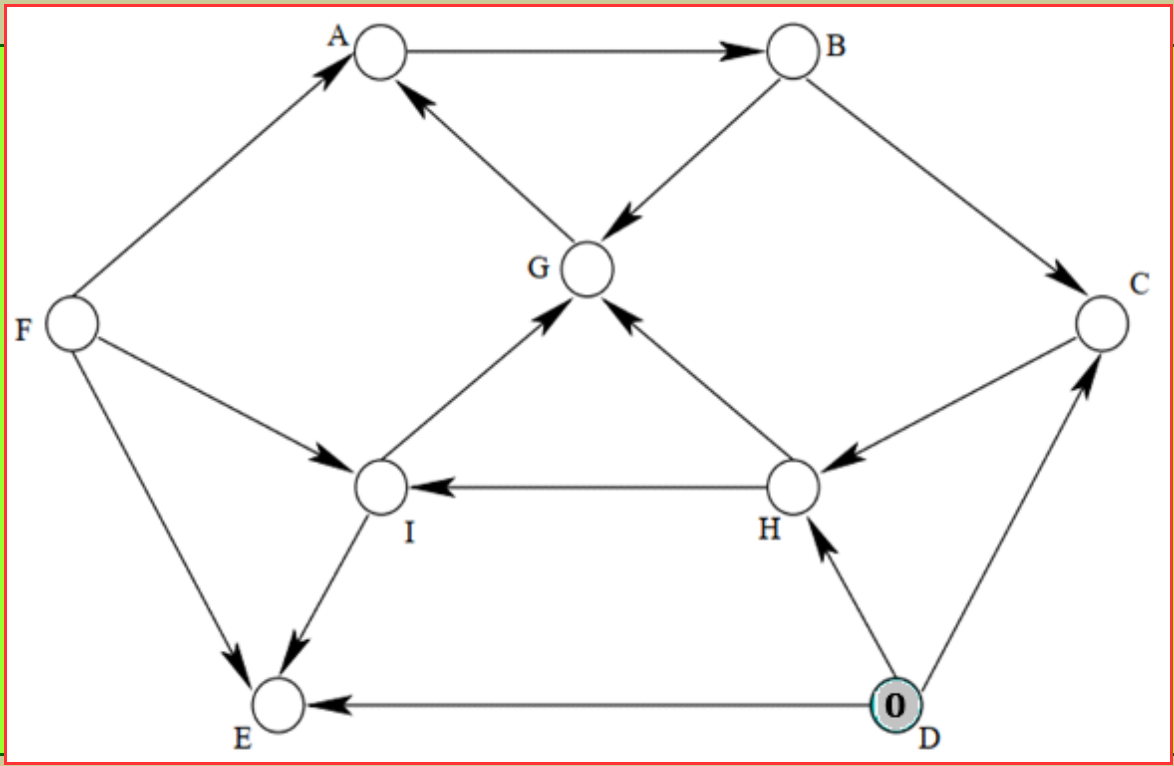
```
algoritmo DFS:
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Observações:

- No fim da execução do algoritmo **pesquisaDFS** todo o vértice **u** de G tem valores atribuídos às estruturas de dados: Pai[u], Dist[u] e Cor[u]
- Os resultados produzidos pelo algoritmo (floresta gerada e valores nas estruturas de dados) não são únicos, pois dependem da ordem pela qual são percorridos os vértices nos diversos ciclos
- No entanto, nas aplicações típicas deste algoritmo, os diversos resultados possíveis são equivalentes

Algoritmo 2 (exemplo) - pesquisaDFS(G, D)

Cor[u] ← branco, para todo o vértice u de G
Dist[u] ← 0, para todo o vértice u de G
Pai[u] ← NULO, para todo o vértice u de G
Cor[**D**] ← cinzento; *Dist[**D**] = 0*; *Pai[**D**] = NULO*
chamada de **DFS(**G, **D**, Cor, Dist, Pai**)**




```
algoritmo pesquisaDFS(G, S)
  para (cada u ∈ V) repetir
    Cor[u] ← branco
    Dist[u] ← 0
    Pai[u] ← NULO
  fim_para
  Cor[S] ← cinzento
  DFS(G, S, Cor, Dist, Pai)
  para (cada u ∈ V: Cor[u] = branco) repetir
    Cor[u] ← cinzento
    Dist[u] ← 0
    DFS(G, u, Cor, Dist, Pai)
  fim_para
fim_algoritmo
```

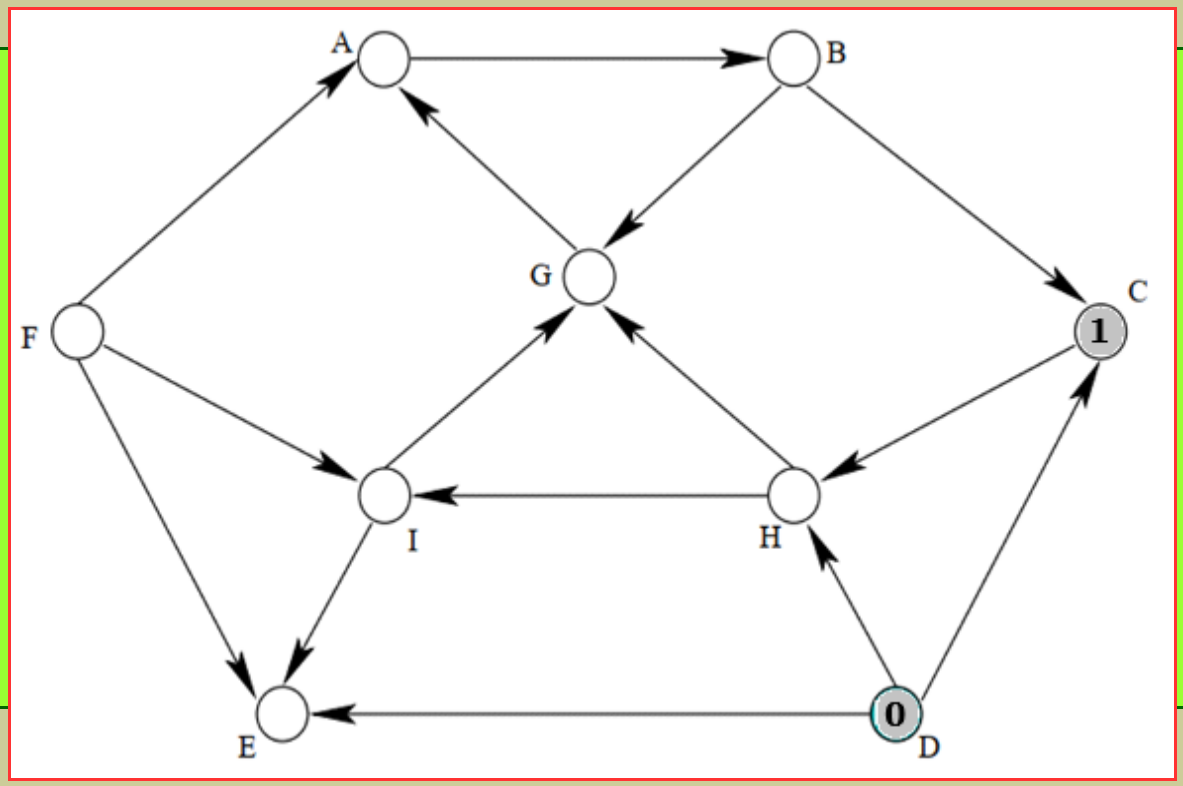
Algoritmo 2 (exemplo) - DFS(G, G, Cor, Dist, Pai)

Dist[D] = 0

vértice adjacente a D: **C** (Cor[C] = branco)

Cor[**C**] ← cinzento; Dist[**C**] ← Dist[D] + 1 = 1; Pai[**C**] ← D ⇒ **(D, C) pertence à APP**

chamada de **DFS(**G, **C**, Cor, Dist, Pai**)**




```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

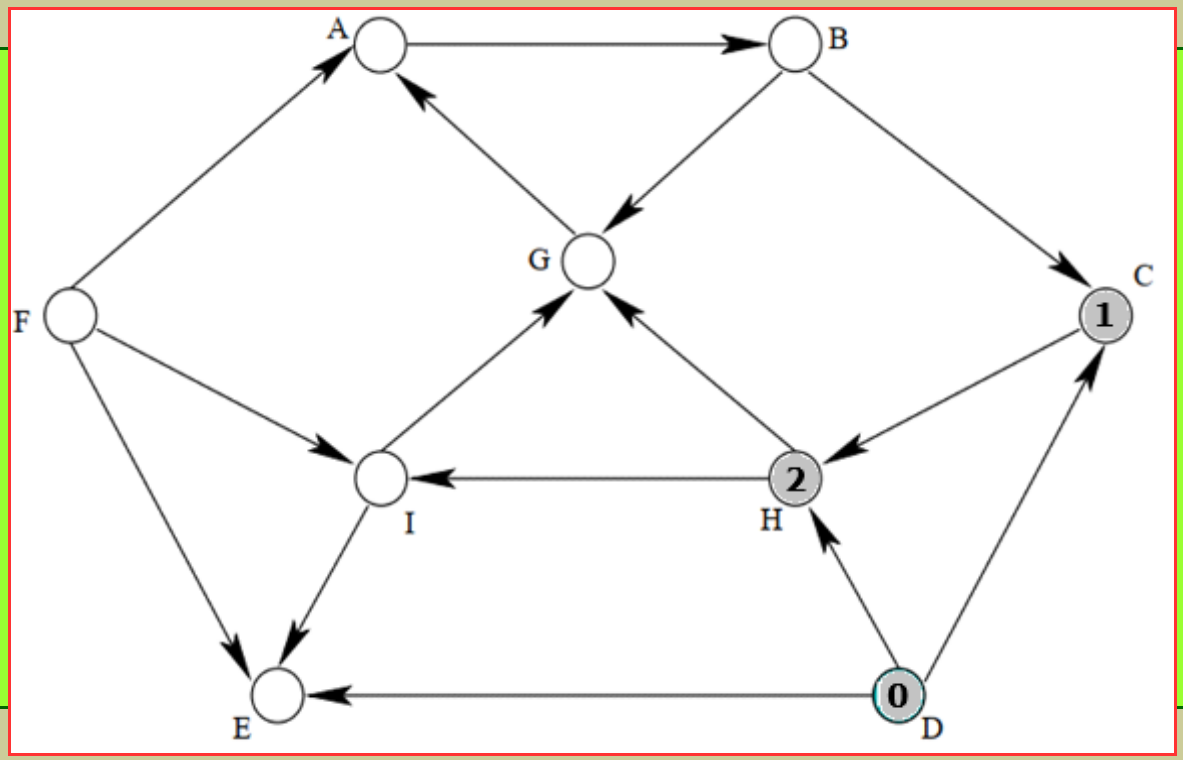
Algoritmo 2 (exemplo) - DFS(G, G, Cor, Dist, Pai)

Dist[C] = 1

vértice adjacente a C: **H** (Cor[H] = branco)

Cor[**H**] ← cinzento; Dist[**H**] ← Dist[C] + 1 = 2; Pai[**H**] ← C ⇒ **(C, H) pertence à APP**

chamada de **DFS(**G, **H**, Cor, Dist, Pai**)**




```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

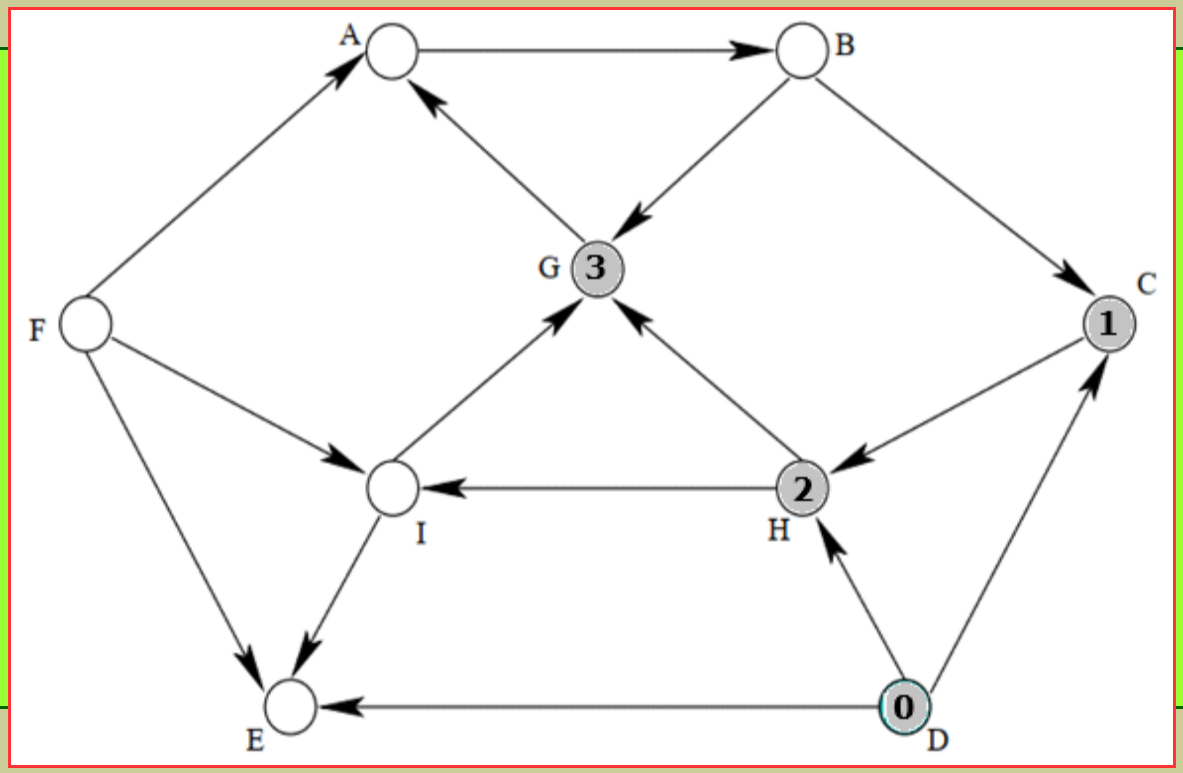
Algoritmo 2 (exemplo) - DFS(G, G, Cor, Dist, Pai)

Dist[H] = 2

vértice adjacente a H: **G** (Cor[G] = branco)

Cor[**G**] ← cinzento; Dist[**G**] ← Dist[H] + 1 = 3; Pai[**G**] ← H ⇒ **(H, G) pertence à APP**

chamada de **DFS(**G, **G**, Cor, Dist, Pai**)**




```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

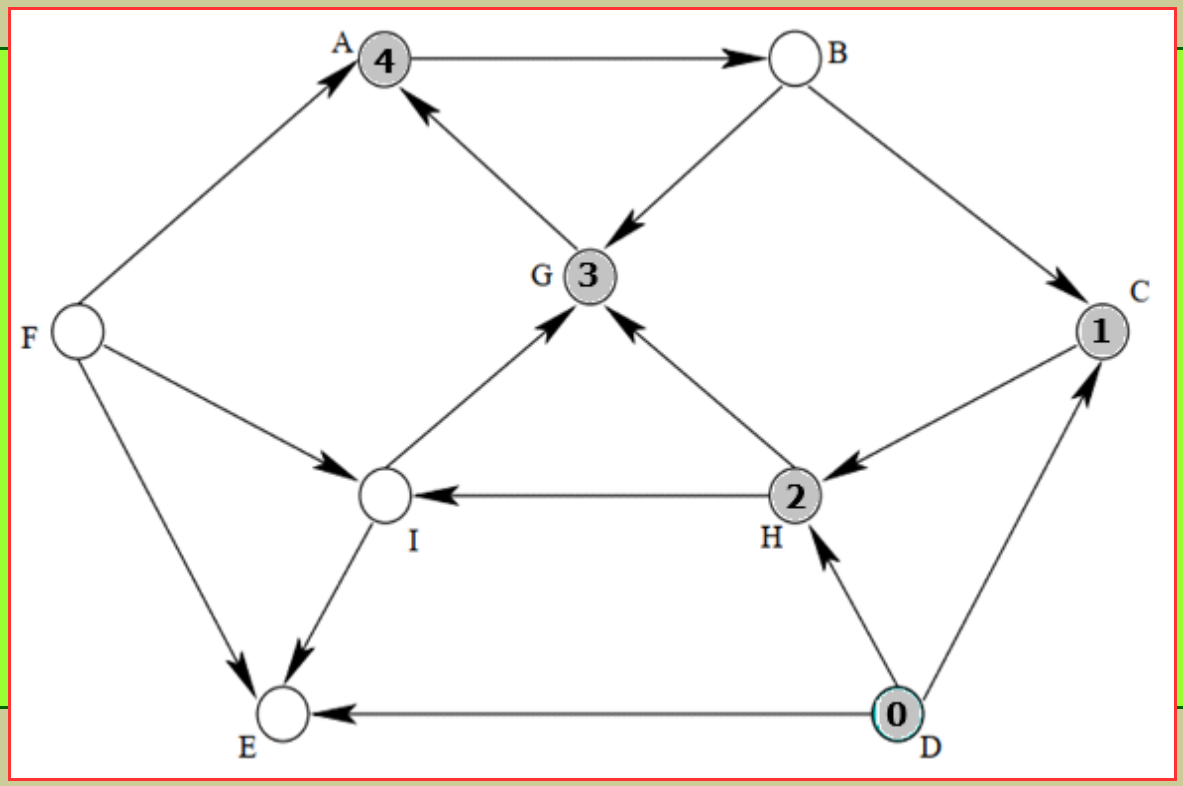
Algoritmo 2 (exemplo) - DFS(G, G, Cor, Dist, Pai)

Dist[G] = 3

vértice adjacente a G: **A** (Cor[A] = branco)

Cor[**A**] ← cinzento; Dist[**A**] ← Dist[G] + 1 = 4; Pai[**A**] ← G ⇒ **(G, A) pertence à APP**

chamada de **DFS(**G, **A**, Cor, Dist, Pai**)**




```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

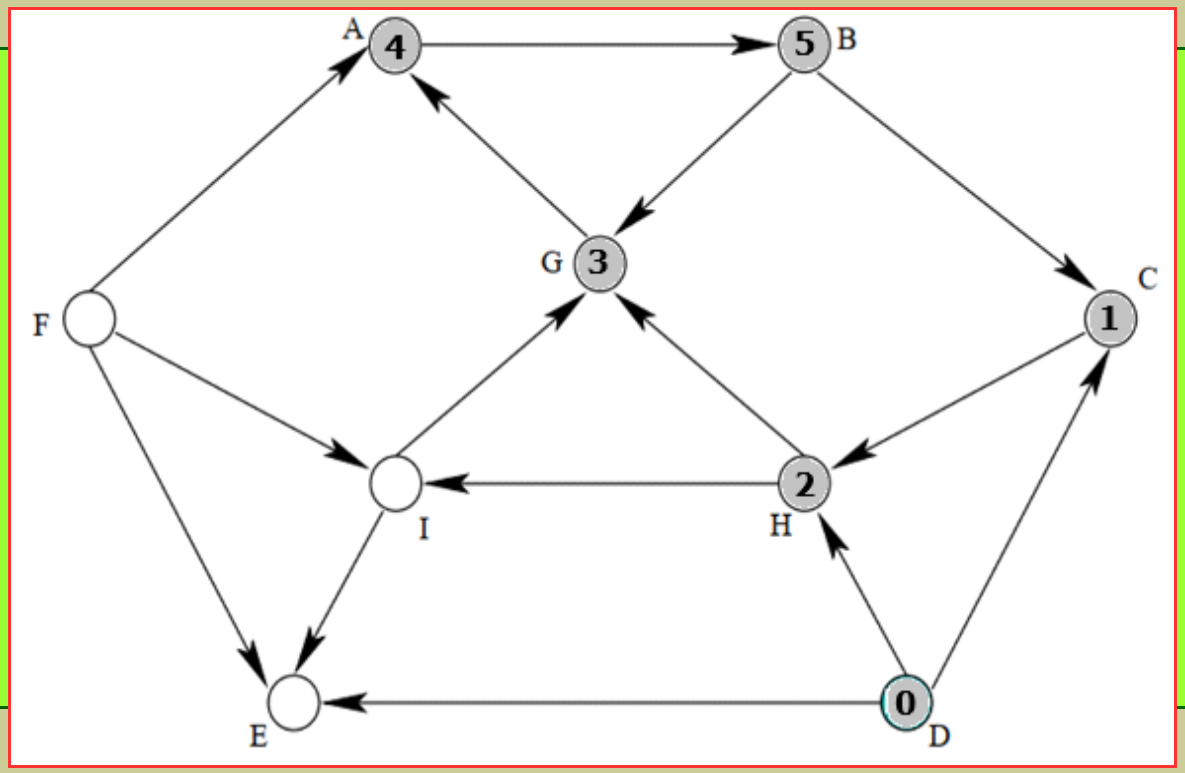
Algoritmo 2 (exemplo) - DFS(G, A, Cor, Dist, Pai)

Dist[A] = 4

vértice adjacente a A: **B** (Cor[B] = branco)

Cor[**B**] ← cinzento; Dist[**B**] ← Dist[A] + 1 = 5; Pai[**B**] ← A ⇒ **(A, B) pertence à APP**

chamada de **DFS(**G, **B**, Cor, Dist, Pai**)**



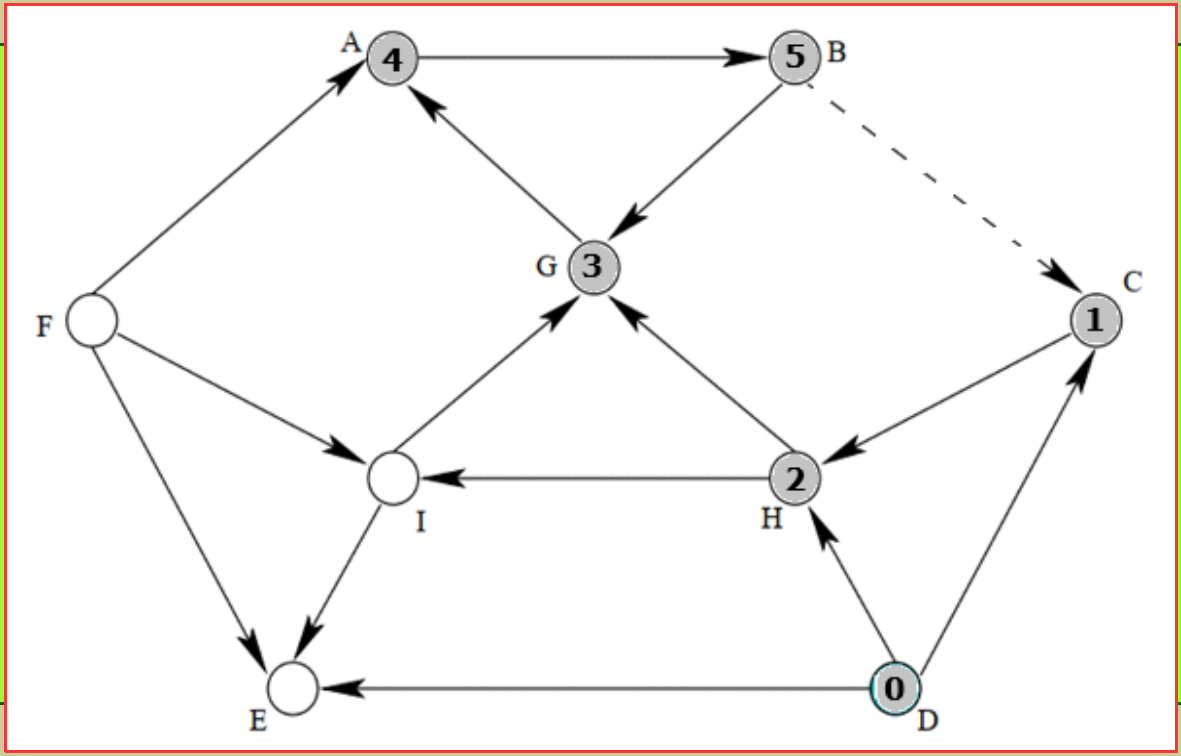

```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, B, Cor, Dist, Pai)

Dist[B] = 5

vértice adjacente a B: **C** (Cor[C] = cinzento (não branco))

passar ao próximo adjacente



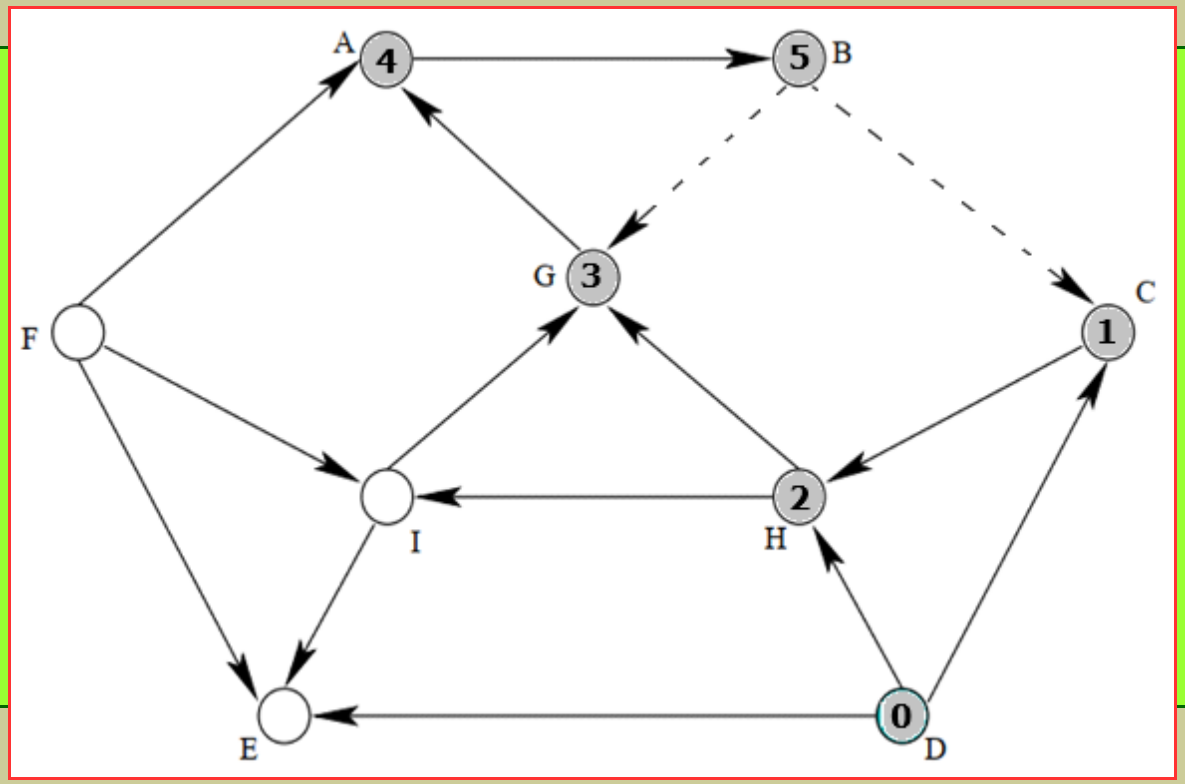

```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, B, Cor, Dist, Pai)

Dist[B] = 5

vértice adjacente a B: **G** (Cor[G] = cinzento (não branco))

passar ao próximo adjacente

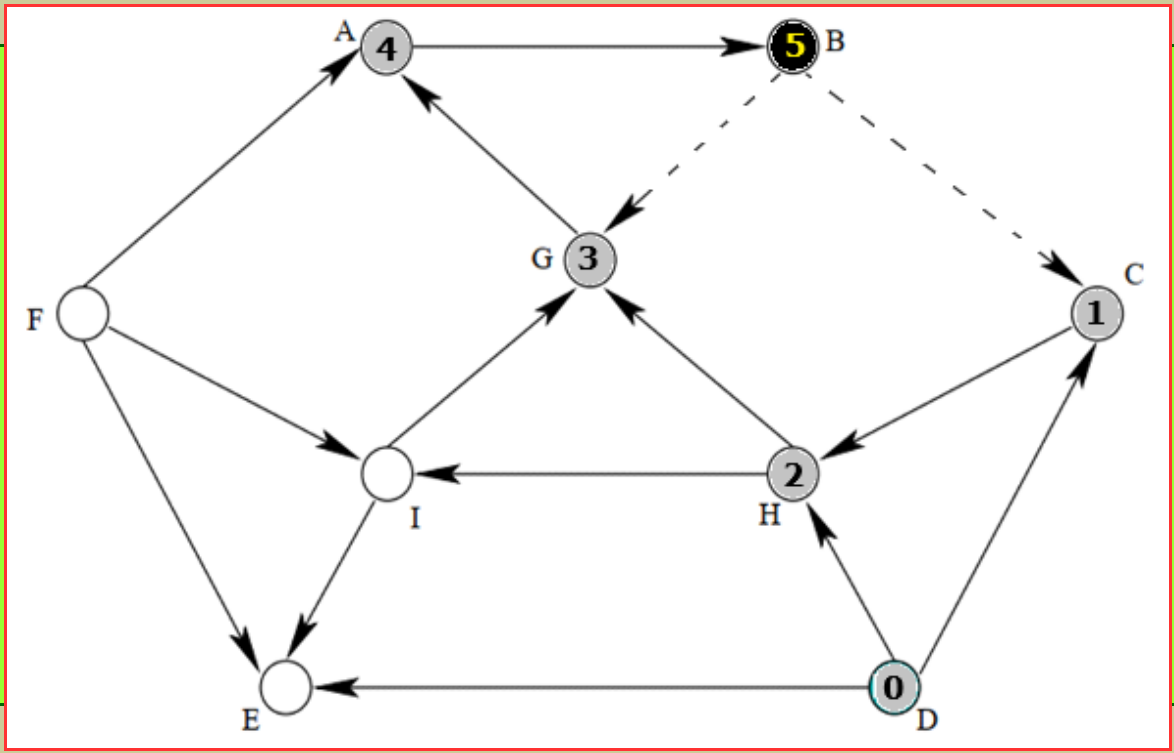



```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, B, Cor, Dist, Pai)

todos os vértices adjacentes de B tratados

Cor[**B**] ← preto

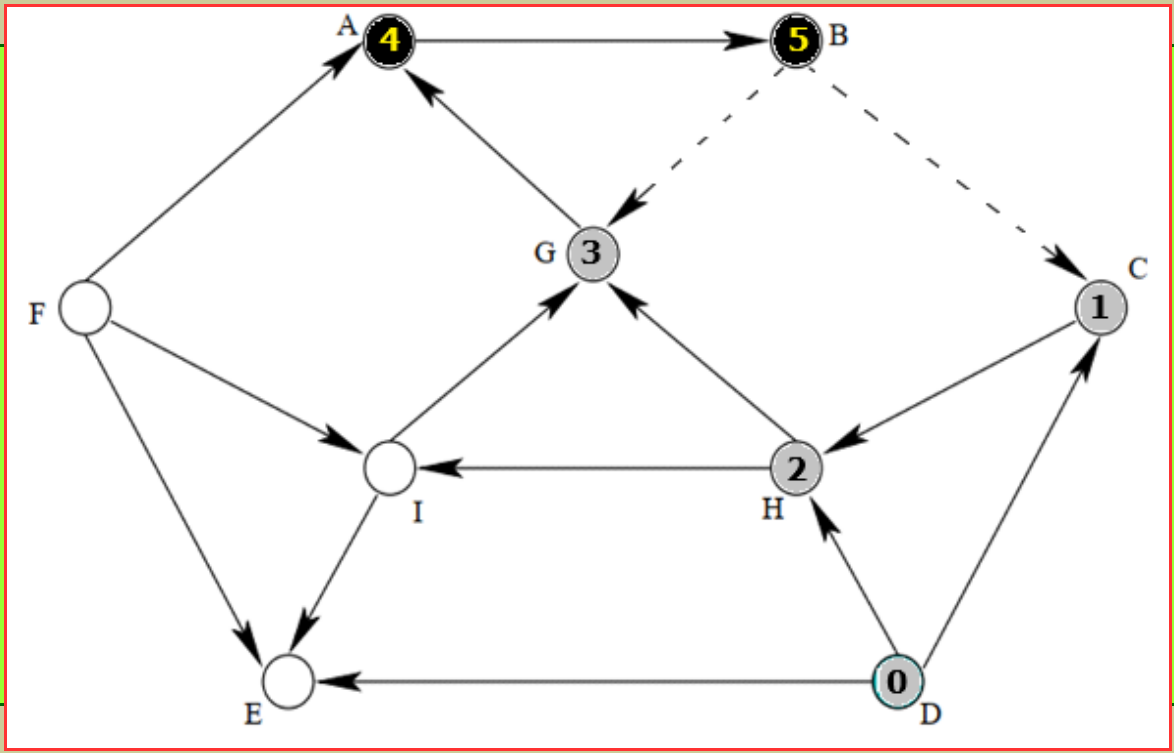



```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, A, Cor, Dist, Pai) { recuar para A = Pai[B] }

todos os vértices adjacentes de A tratados

Cor[**A**] ← preto

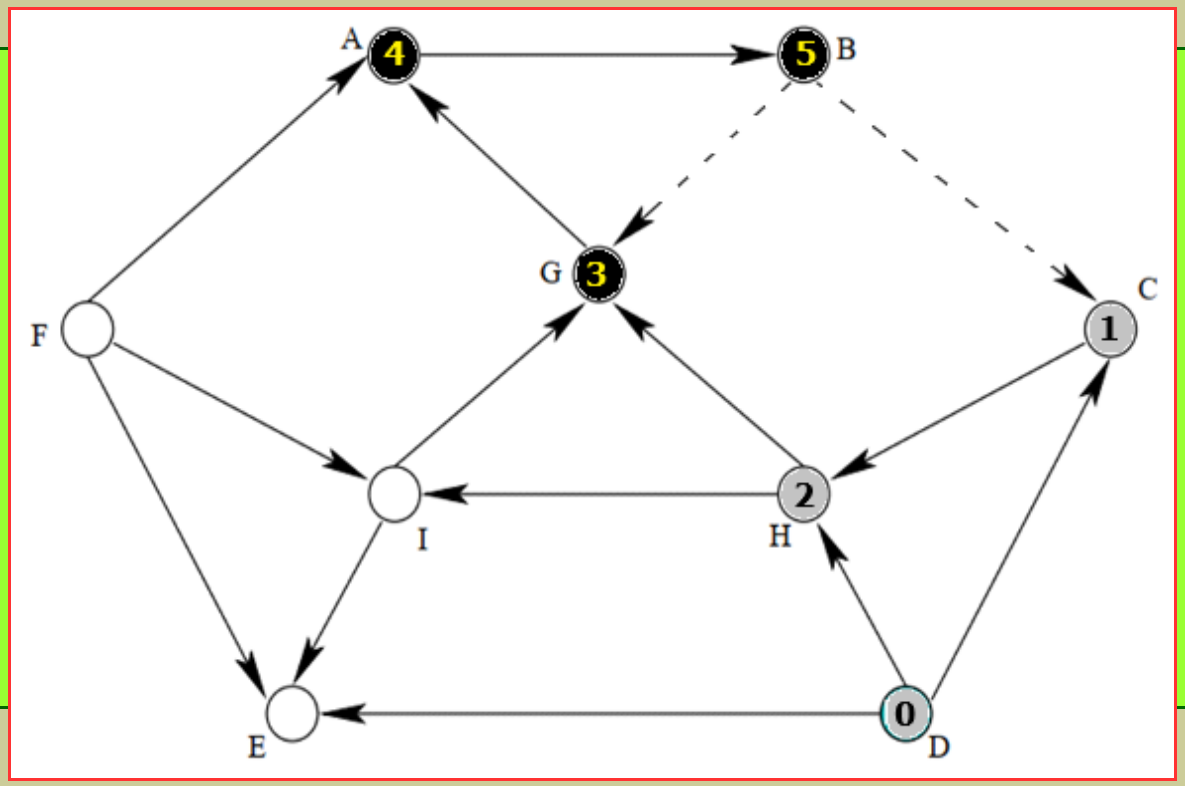



```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, G, Cor, Dist, Pai) { recuar para G = Pai[A] }

todos os vértices adjacentes de G tratados

Cor[**G**] ← preto




```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

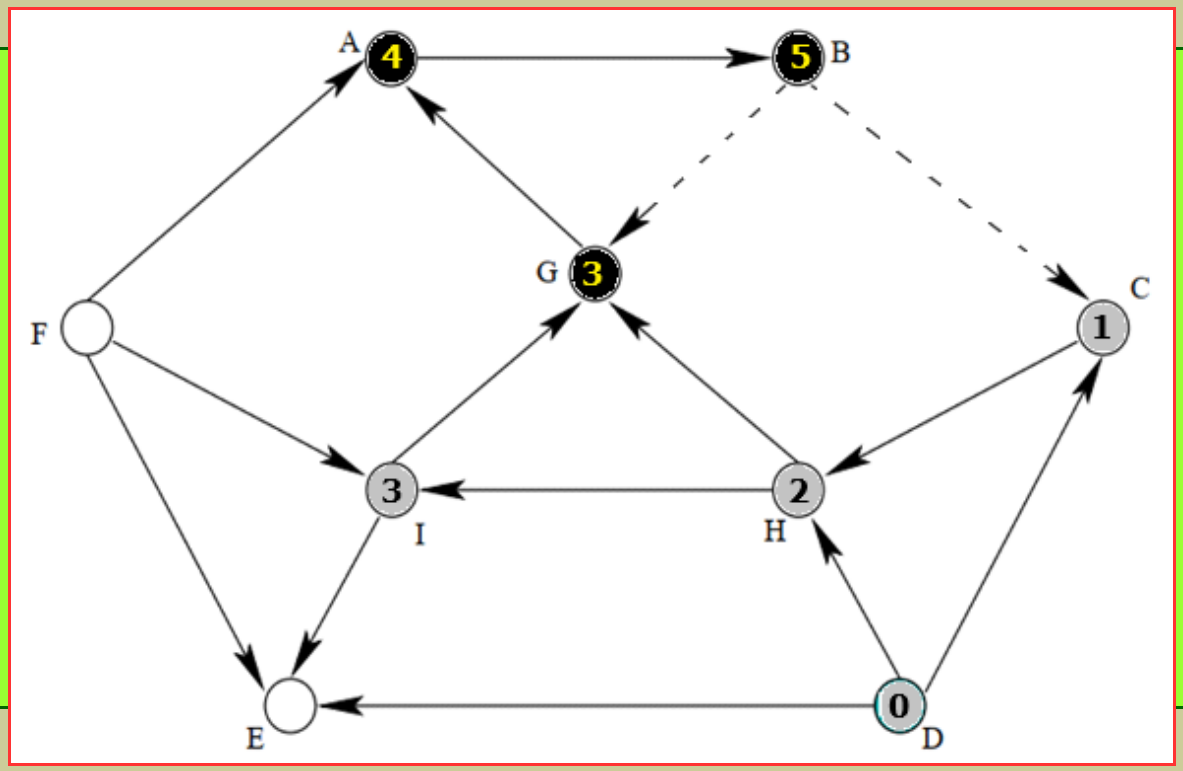
Algoritmo 2 (exemplo) - DFS(G, H, Cor, Dist, Pai) { recuar para H = Pai[G] }

Dist[H] = 2

vértice adjacente a H: **I** (Cor[I] = branco)

Cor[**I**] ← cinzento; Dist[**I**] ← Dist[H] + 1 = 3; Pai[**I**] ← H ⇒ **(H, I) pertence à APP**

chamada a **DFS(**G, **I**, Cor, Dist, Pai**)**




```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

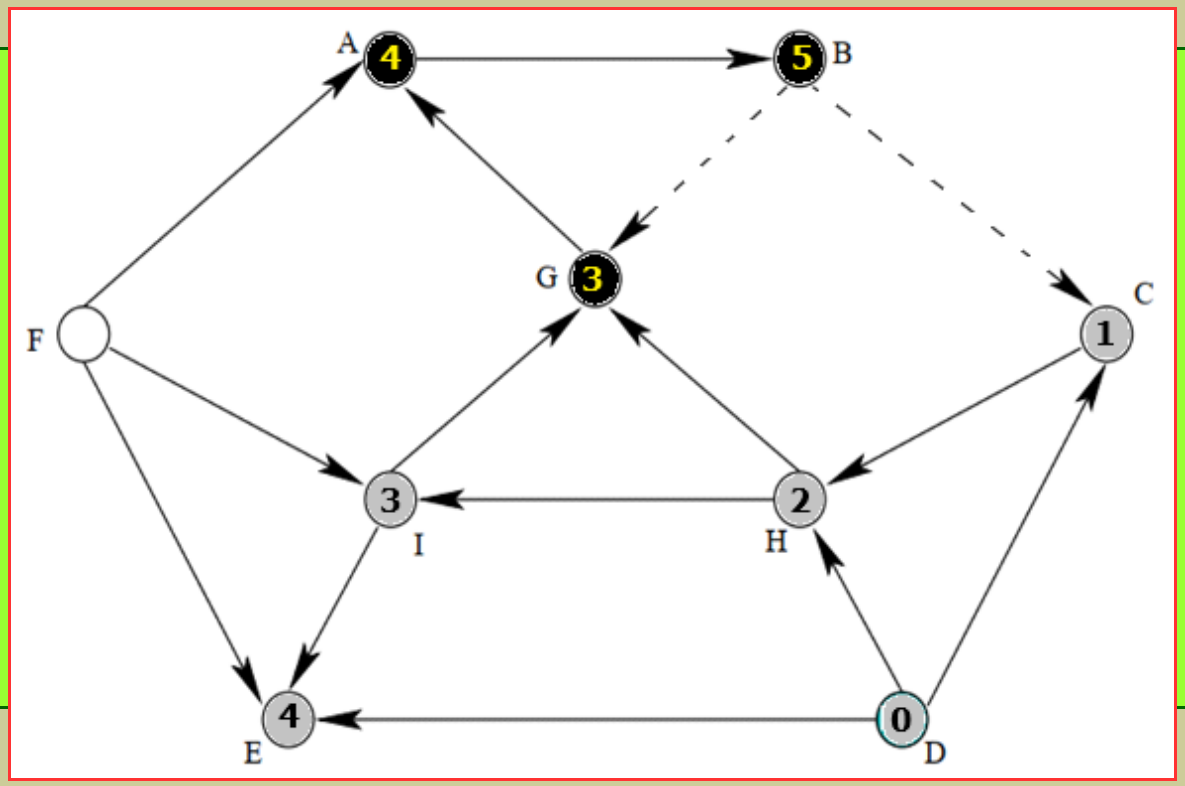
Algoritmo 2 (exemplo) - DFS(G, I, Cor, Dist, Pai)

Dist[I] = 3

vértice adjacente a I: **E** (Cor[E] = branco)

Cor[**E**] ← cinzento; Dist[**E**] ← Dist[I] + 1 = 4; Pai[**E**] ← I ⇒ **(I, E) pertence à APP**

chamada a **DFS(**G, **E**, Cor, Dist, Pai**)**

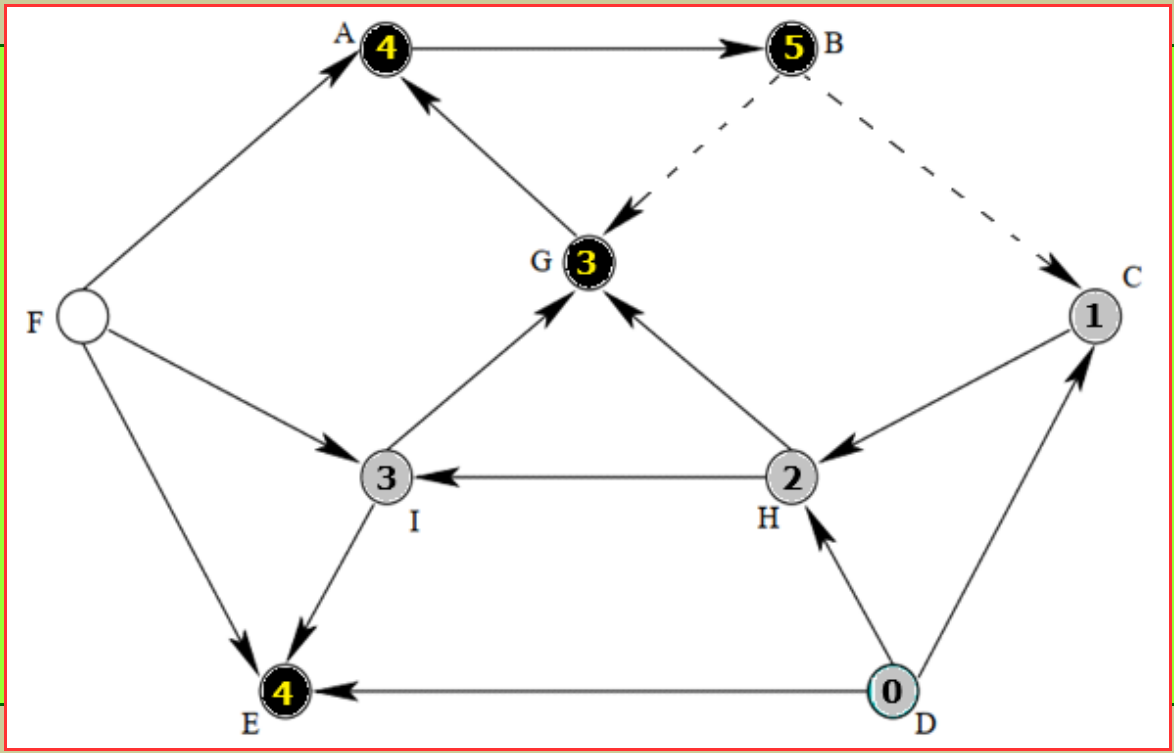



```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, E, Cor, Dist, Pai)

todos os vértices adjacentes de E tratados

Cor[**E**] ← preto



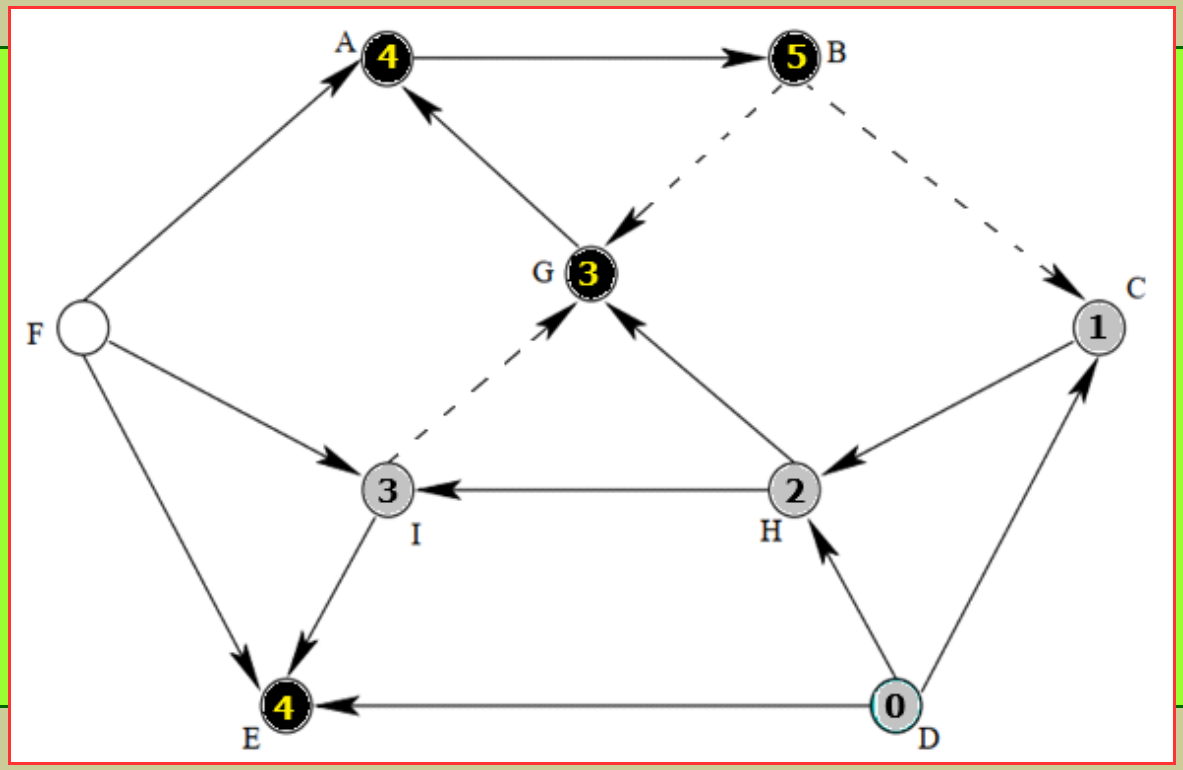

```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, I, Cor, Dist, Pai) { recuar para I = Pai[E] }

Dist[I] = 3

vértice adjacente a I: **G** (Cor[G] = preto (não branco))

passar ao próximo adjacente



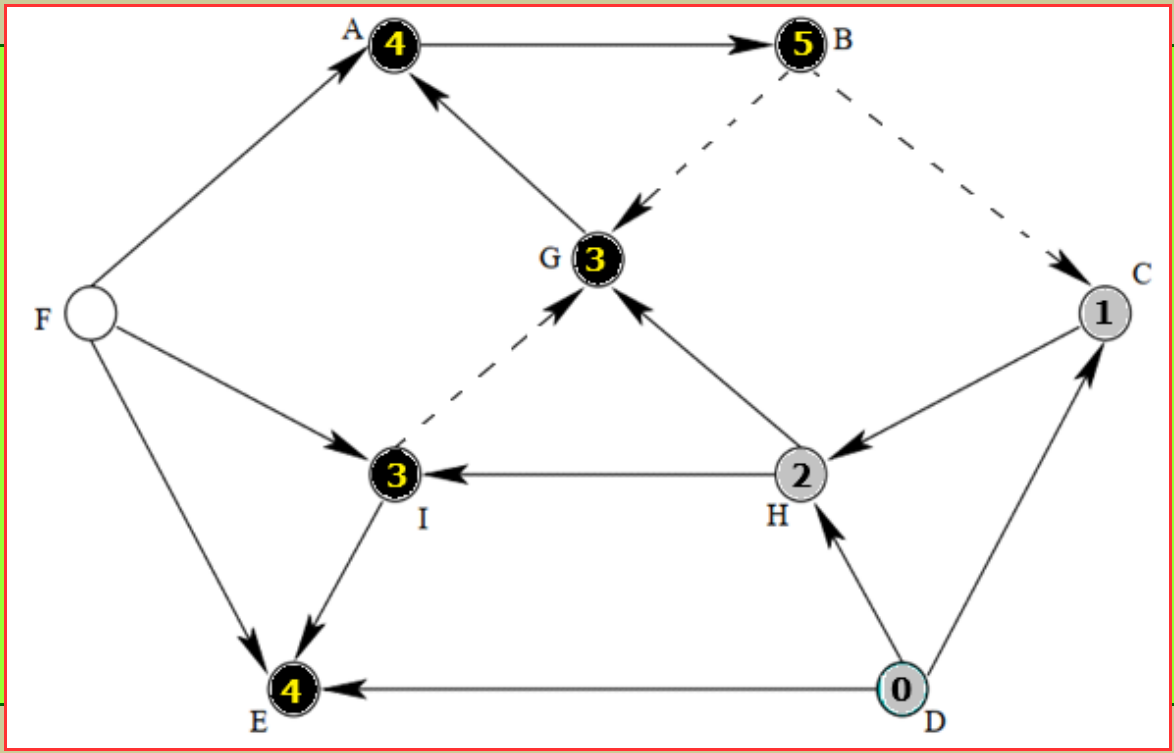

```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, I, Cor, Dist, Pai)

Dist[I] = 3

todos os vértices adjacentes de I tratados

Cor[**I**] ← preto



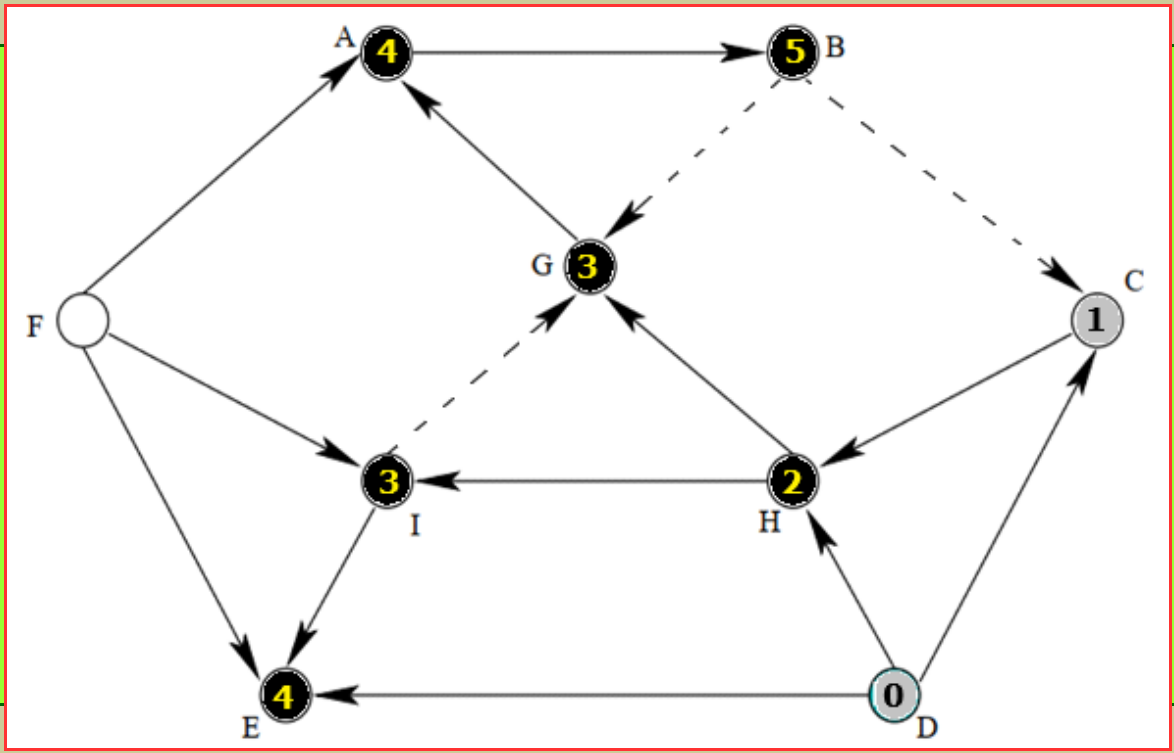

```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, H, Cor, Dist, Pai) { recuar para H = Pai[I] }

Dist[H] = 2

todos os vértices adjacentes de H tratados

Cor[**H**] ← preto



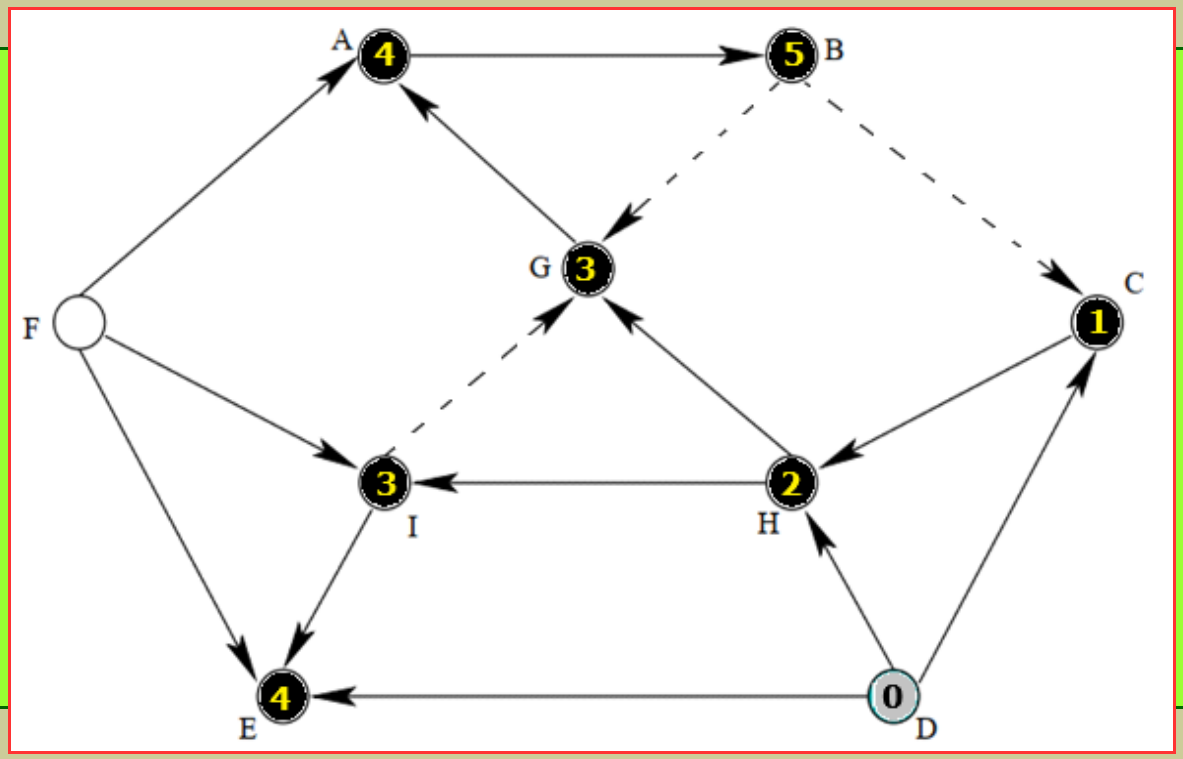

```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, C, Cor, Dist, Pai) { recuar para C = Pai[H] }

Dist[C] = 1

todos os vértices adjacentes de C tratados

Cor[**C**] ← preto



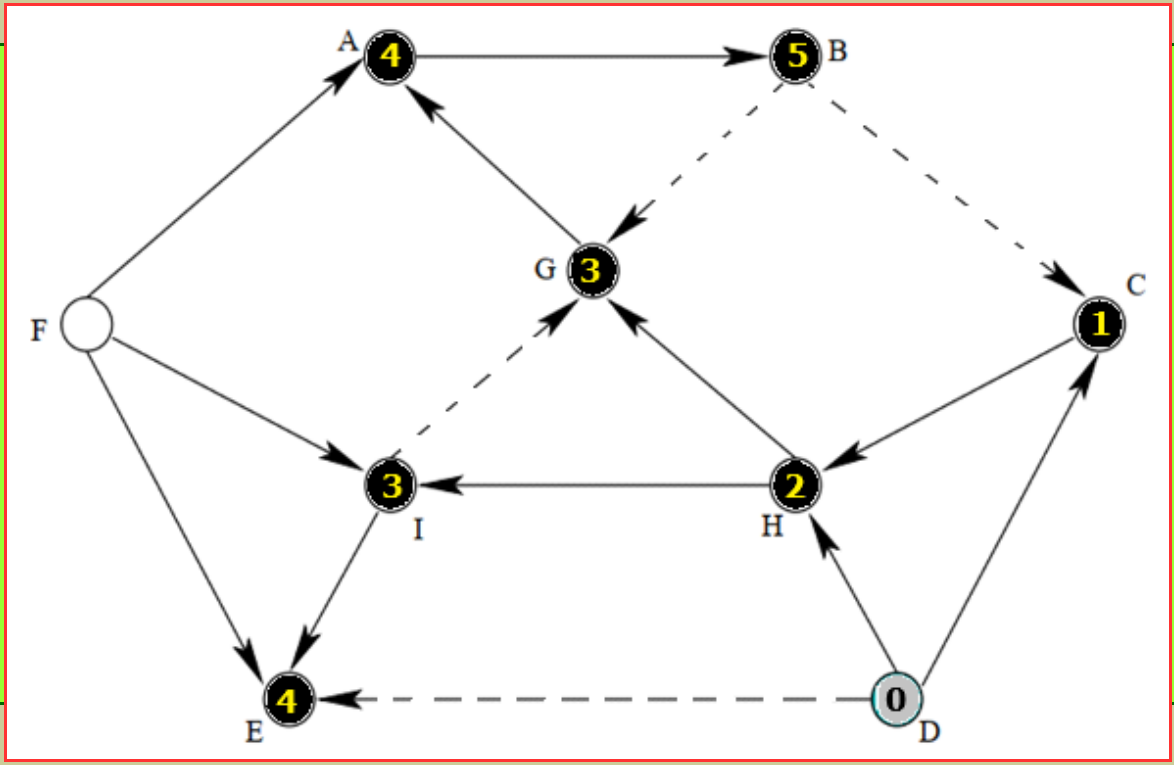

```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, D, Cor, Dist, Pai) { recuar para D = Pai[C] }

Dist[D] = 0

vértice adjacente a D: **E** (Cor[E] = preto (não branco))

passar ao próximo adjacente



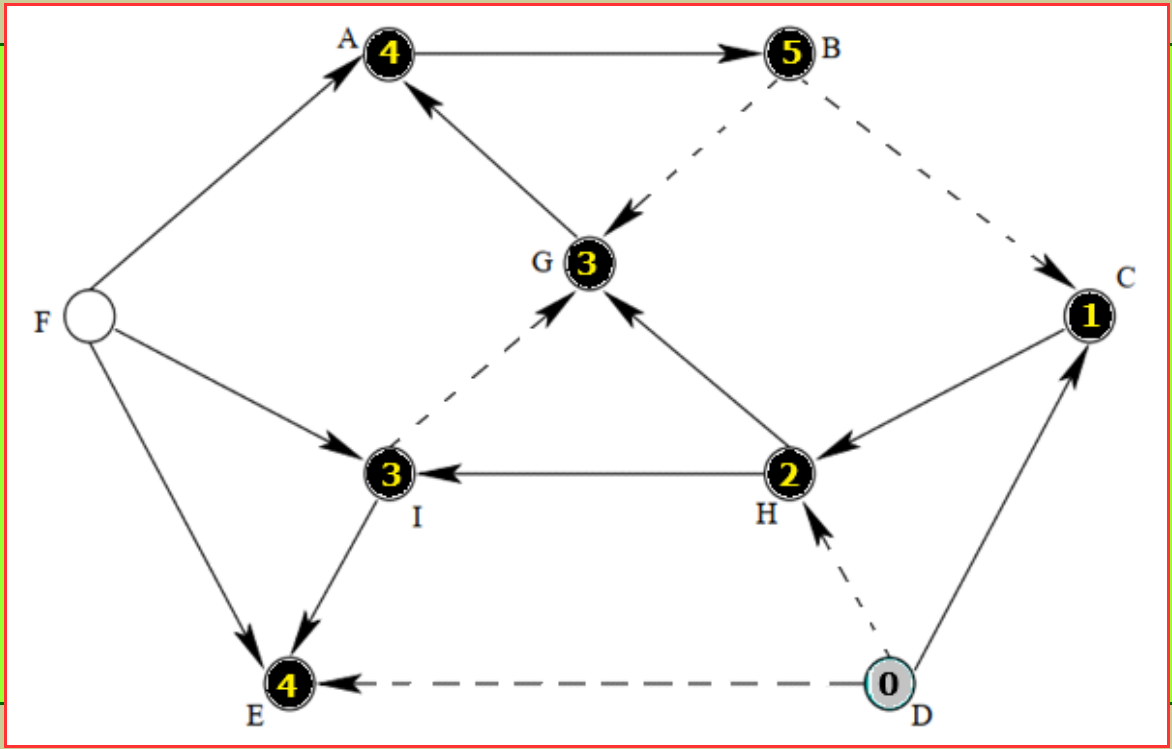

```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, D, Cor, Dist, Pai)

Dist[D] = 0

vértice adjacente a D: **H** (Cor[H] = preto (não branco))

passar ao próximo adjacente

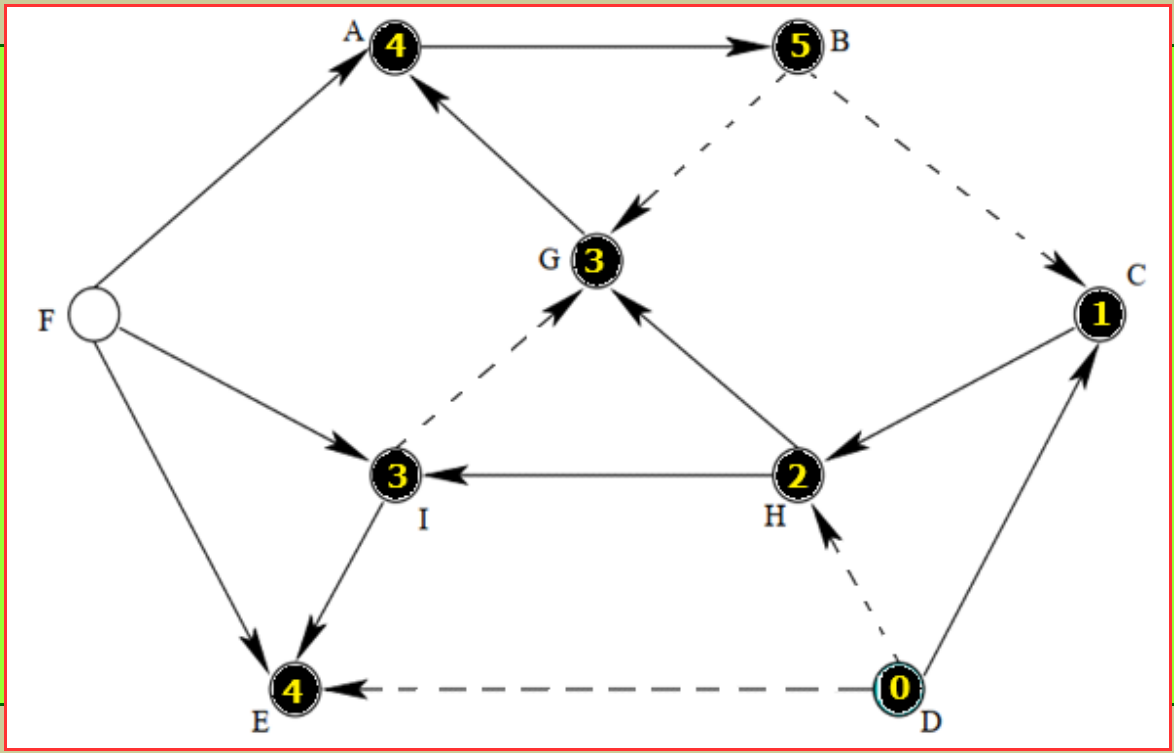



```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - DFS(G, D, Cor, Dist, Pai)

todos os vértices adjacentes de D tratados

Cor[**D**] ← preto




```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

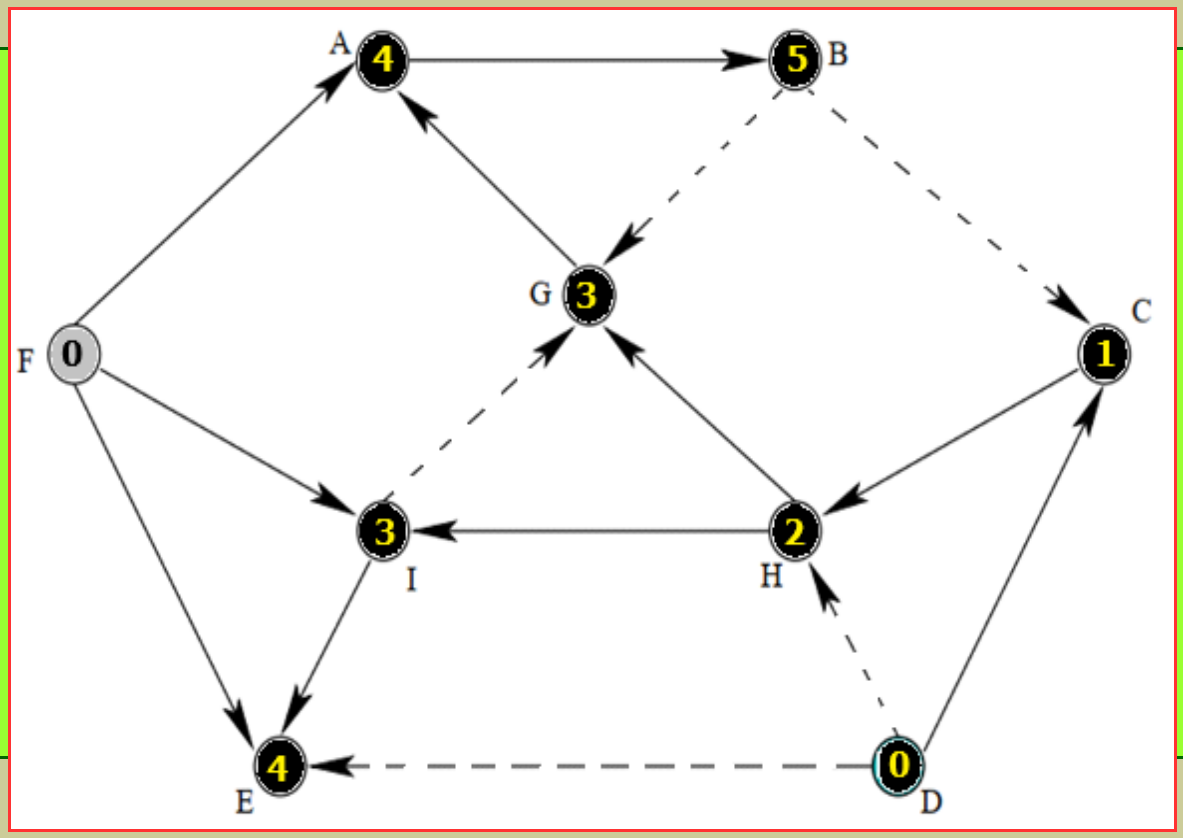
- Determinada a APP com início na vértice D (na figura): F não pertence à APP

Algoritmo 2 (exemplo) - pesquisaDFS(G, S), com u = F

Cor[**F**] = branco; *Dist[F] = 0*; *Pai[F] = NULO*

Cor[**F**] ← cinzento

chamada de **DFS(**G, **F**, Cor, Dist, Pai**)**




```
algoritmo pesquisarDFS(G, S)
  para (cada u ∈ V) repetir
    Cor[u] ← branco
    Dist[u] ← 0
    Pai[u] ← NULO
  fim_para
  Cor[S] ← cinzento
  DFS(G, S, Cor, Dist, Pai)
  para (cada u ∈ V: Cor[u] = branco) repetir
    Cor[u] ← cinzento
    Dist[u] ← 0
    DFS(G, u, Cor, Dist, Pai)
  fim_para
fim_algoritmo
```

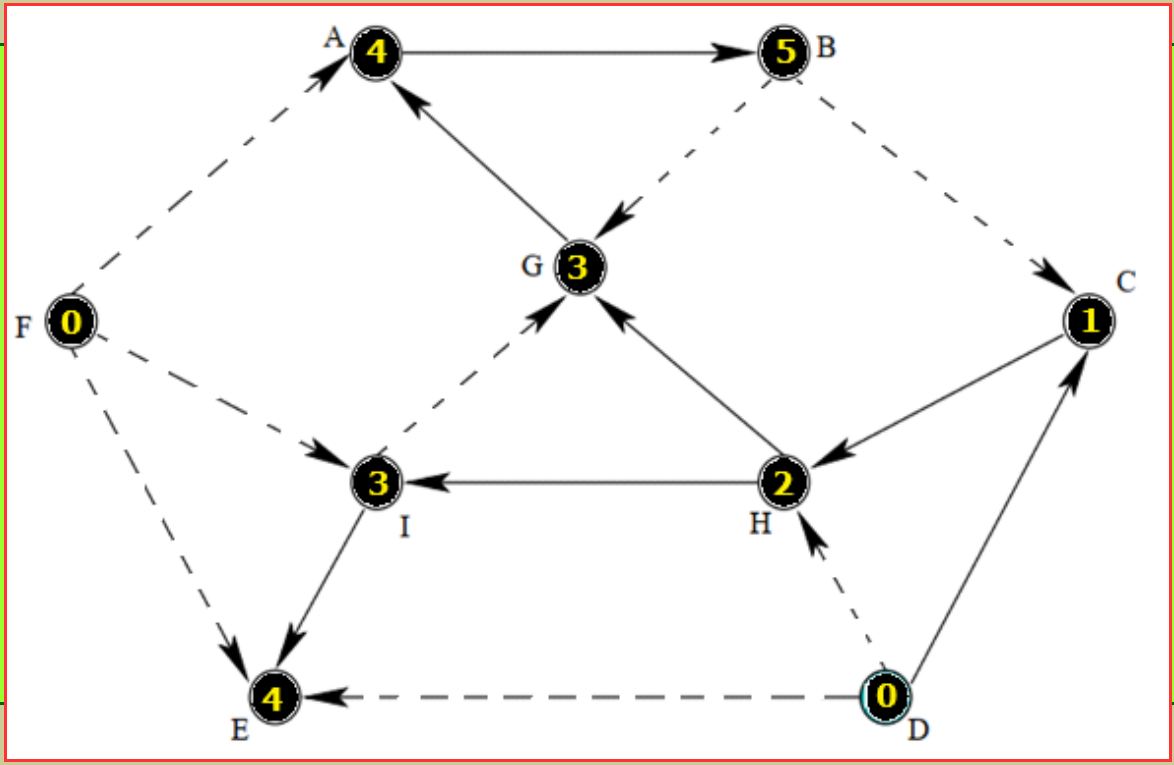
Algoritmo 2 (exemplo) - DFS(G, F, Cor, Dist, Pai)

Dist[F] = 0

vértices adjacentes a F: **A**, **E** e **I** (todos com Cor = preto (não branco))

todos os vértices adjacentes de F tratados

Cor[**F**] = preto




```
algoritmo DFS(G, u, Cor, Dist, Pai)
  para (cada vértice v adjacente a u) repetir
    se (Cor[v] = branco) então
      Cor[v] ← cinzento
      Dist[v] ← Dist[u] + 1
      Pai[v] ← u
      DFS(G, v, Cor, Dist, Pai)
    fim_se
  fim_para
  Cor[u] ← preto
fim_algoritmo
```

Algoritmo 2 (exemplo) - ordem das chamadas recursivas

D → C → H → G → A → B → A → G → H → I → E → I → H → C → D
F