

# CTC-12



## Projeto e Análise de Algoritmos

**Carlos Alberto Alonso Sanches**

CTC-12

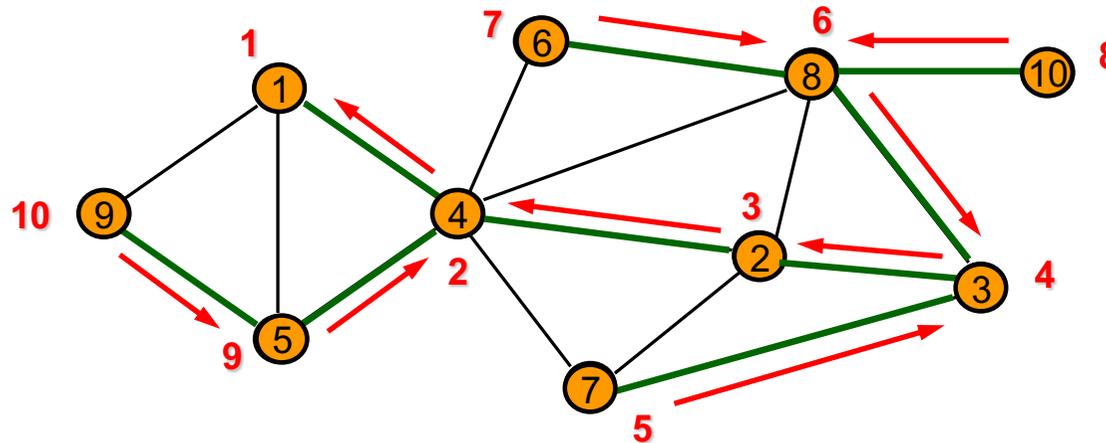


## 8) Algoritmos em grafos

Tarjan e similares

# Em profundidade (*depth-first search*)

- Tática: enquanto for possível, aprofundar-se no grafo; quando não for mais possível, recuar um nível.
- Exemplo (supomos armazenamento em ordem crescente):



- A exploração em profundidade possibilita a resolução eficiente de diversos problemas, como veremos adiante.

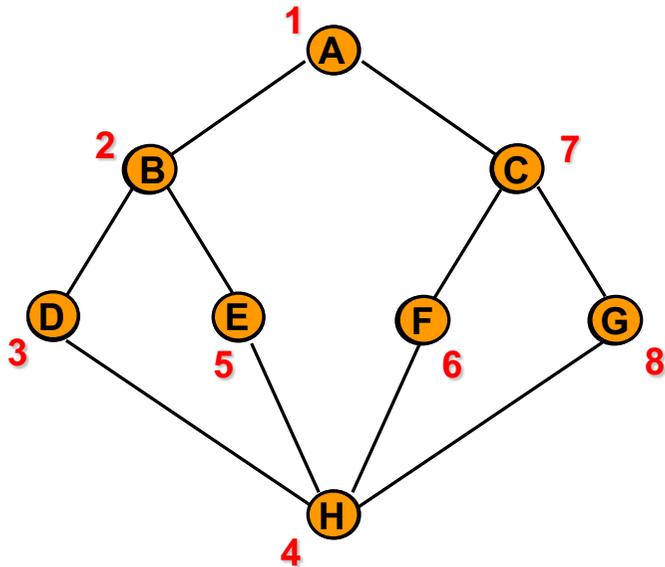
# Versão recursiva

```
int cont = 0;
desmarcar todos os vértices;

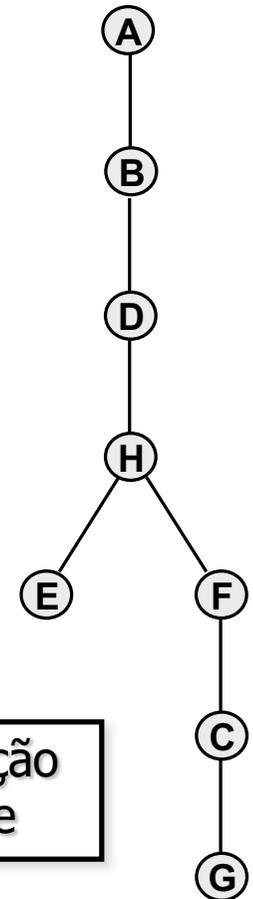
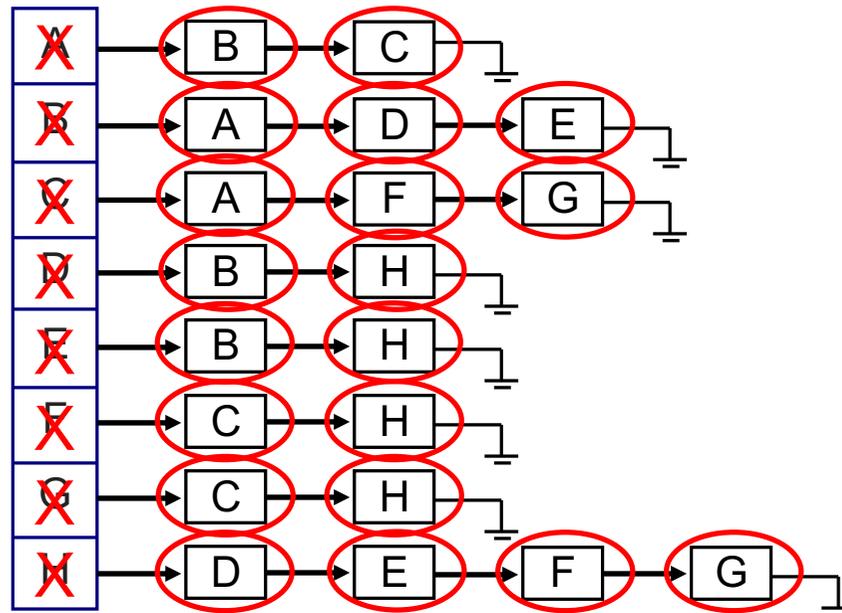
DFS(s) {
    marcar s;
    expl[s] = ++cont;
    // explorando s
    for <s,v> ∈ E {
        if v está desmarcado
            DFS(v);
    }
}
```

- Analogamente à exploração em largura, a complexidade de tempo também é  $\Theta(n+m)$ .

# Exemplo



- não visitado
- visitado



Árvore de exploração  
em profundidade

# Versão iterativa

```
DFS(s) {
  desmarcar todos os vértices;
  stack P;
  int cont = 0;
  marcar s;
  push(P,s);
  while (!isEmpty(P)) {
    curr = top(P);
    pop(P);
    expl[curr] = ++cont;
    // explorando curr
    for <curr,v> ∈ E {
      if v está desmarcado {
        marcar v;
        push(P,v);
      }
    }
  }
}
```

Pequena diferença em relação à versão recursiva: inicialmente, marca e empilha os vértices vizinhos; depois, numera-os à medida que são desempilhados

A ordem de empilhamento é diferente da versão recursiva: gerará outra árvore de exploração!

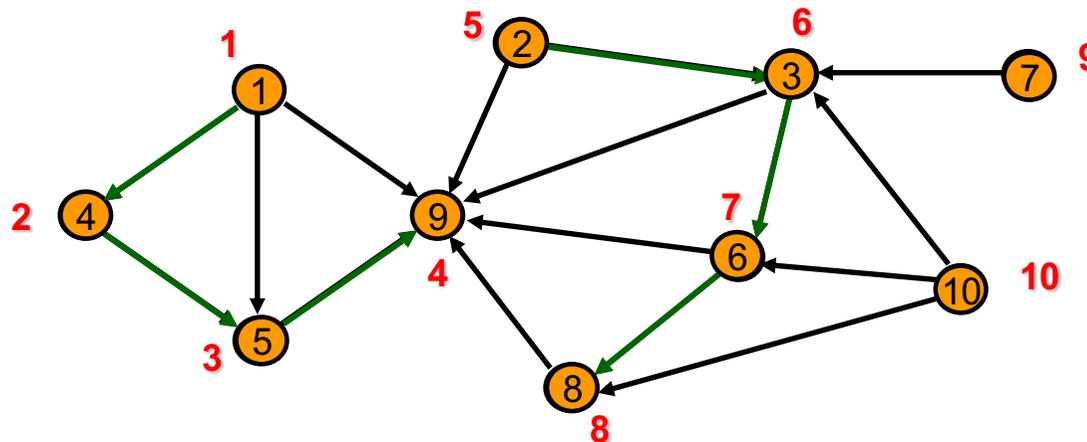
# Exploração em profundidade (digrafos)

```
int cont;
```

```
TravessiaDFS(s) {  
  desmarcar todos os vértices;  
  cont = 0;  
  DFS(s);  
  for v ∈ V {  
    if v está desmarcado  
      DFS(v);  
  }  
}
```

Código adicional

```
DFS(v) {  
  marcar v;  
  expl[v] = ++cont;  
  // explorando v  
  for <v,u> ∈ E {  
    if u está desmarcado  
      DFS(u);  
  }  
}
```



Tempo:  $\Theta(n+m)$

Solução análoga para grafos desconexos

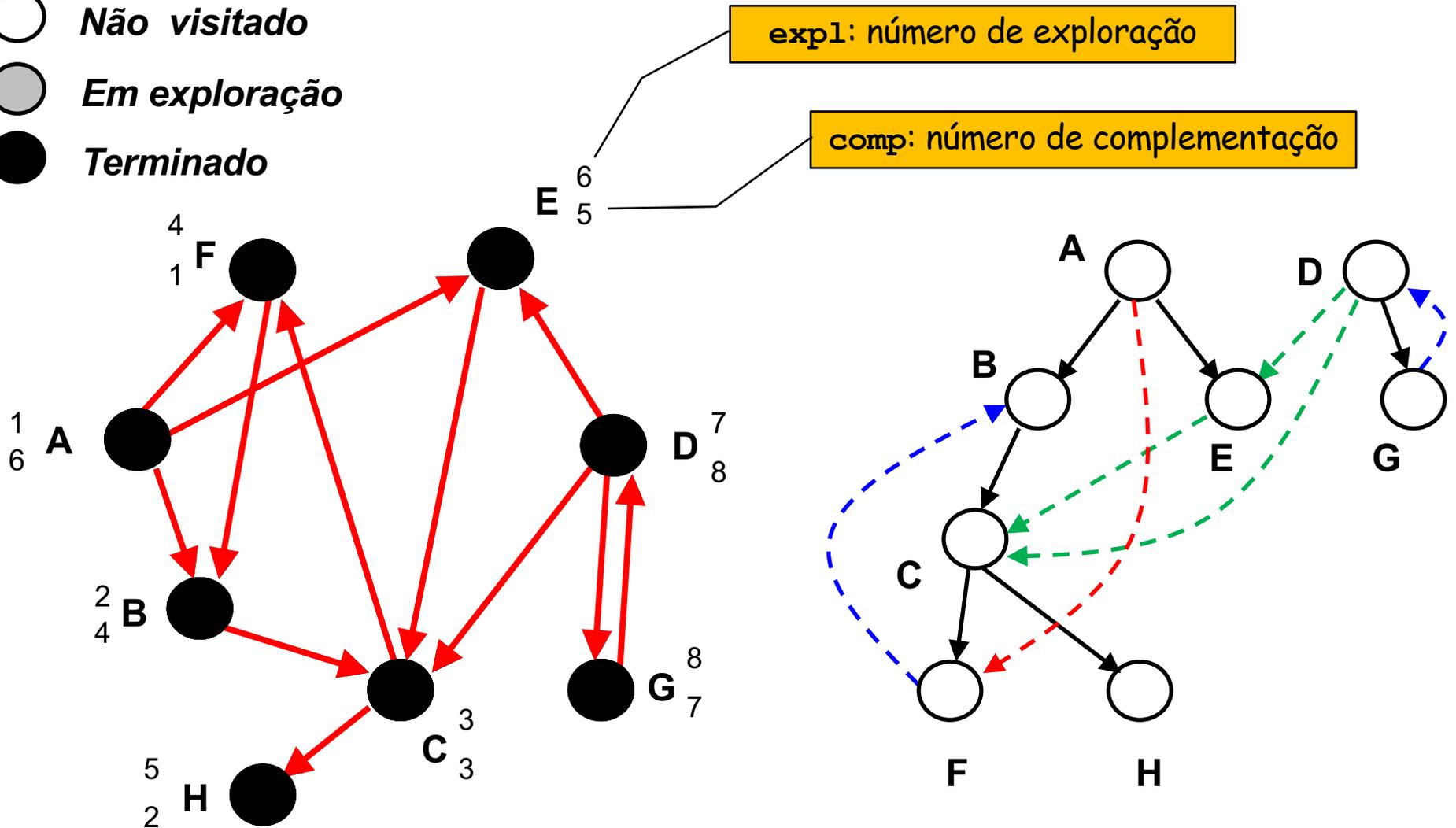
# Ideia de Tarjan (1972)



- Durante a *exploração em profundidade* de um digrafo, numerar seus vértices de acordo com o início e o término dessa exploração.
- As diferentes situações permitem estabelecer uma classificação dos arcos.

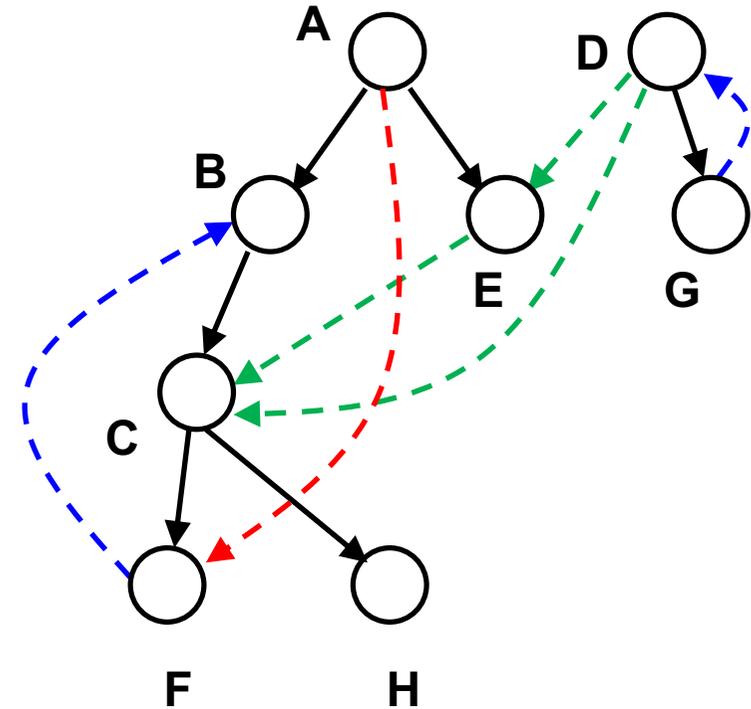
# Exemplo

- Não visitado
- ◐ Em exploração
- Terminado



# Classificação dos arcos

- Classificação do arco  $\langle v, u \rangle$ :
  - Árvore (T):  $u$  ainda não havia sido explorado, e será filho de  $v$  em  $T$  ( $expl[u]=0$ )
  - Retorno (B):  $u$  é antecessor de  $v$  em  $T$ , pois começou antes de  $v$  e ainda está em exploração ( $expl[u] < expl[v]$  e  $comp[u]=0$ )
  - Cruzamento (C):  $u$  está em outra árvore ou sub-árvore, pois começou antes de  $v$  e já foi explorado ( $expl[u] < expl[v]$  e  $comp[u] > 0$ )
  - Avanço (F):  $u$  é descendente de  $v$  em  $T$ , pois começou depois de  $v$  e já foi explorado ( $expl[u] > expl[v]$  e  $comp[u] > 0$ )



# Algoritmo de Tarjan

```
Tarjan(G) {
  int ce = 0;
  int cc = 0;
  for v ∈ V {
    expl[v] = 0;
    comp[v] = 0;
  }
  for v ∈ V
    if (expl[v] == 0)
      DFST(v);
}

DFST(v) {
  expl[v] = ++ce;
  for <v,u> ∈ E
    if (expl[u] == 0) {
      tipo[<v,u>] = T;
      DFST(u);
    }
    else if (expl[u] > expl[v])
      tipo[<v,u>] = F;
    else if (comp[u] > 0)
      tipo[<v,u>] = C;
    else tipo[<v,u>] = B;
  comp[v] = ++cc;
}
```

Complexidade de tempo:  $\Theta(n+m)$

# Exercício

- Considere um *grafo não orientado, sem laços e sem arestas repetidas*. Se aplicarmos nele o algoritmo de Tarjan, somente haverá arestas de árvore e de retorno.
- Por que neste caso não existem arestas de avanço e de cruzamento?

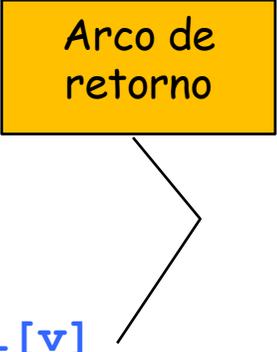
# Teste de aciclicidade

- Em certas aplicações, uma tarefa importante é o reconhecimento de um *digrafo acíclico* (conhecido como DAG).
  - Alguns exemplos: ordenação topológica (“temporização” entre os vértices), fluxo máximo, dependência de dados, otimização de rotas, otimização de circuitos digitais, etc.
- A exploração em profundidade pode nos dar a solução desse problema em tempo  $O(n+m)$ .
- Concretamente, basta uma variação do algoritmo de Tarjan: se um arco de retorno for encontrado durante a exploração, então o digrafo será cíclico.

# Algoritmo

```
Aciclicidade(G) {
    stack P;
    bool aciclico = true;
    int ce = 0;
    int cc = 0;
    for v ∈ V {
        expl[v] = 0;
        comp[v] = 0;
    }
    for v ∈ V
        if (expl[v] == 0)
            DFSA(v);
    if (aciclico)
        digrafo é acíclico
    else
        digrafo é cíclico
}
```

```
DFSA(v) {
    expl[v] = ++ce;
    push(P,v);
    for <v,u> ∈ E
        if (expl[u] == 0)
            DFSA(u);
        else
            if (expl[u] < expl[v]
                && comp[u] == 0) {
                aciclico = false;
                // ciclo está em P
                // desde o topo até u
                stop;
            }
    pop(P);
    comp[v] = ++cc;
}
```

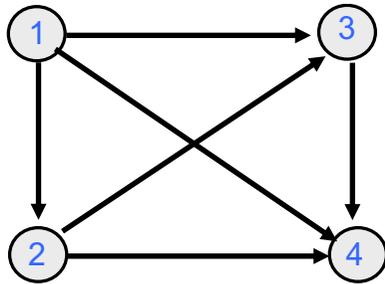


# Ordenação topológica

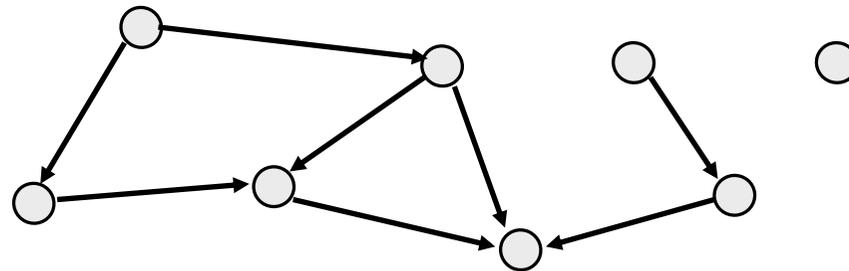
- Uma ordenação topológica de um digrafo  $G=(V,E)$  corresponde a uma bijeção  $f: V \rightarrow \{1, 2, \dots, n\}$  tal que, para todo arco  $(u,v) \in E$ ,  $f(u) < f(v)$ .
- Em outras palavras, deseja-se numerar os vértices de tal modo que, se houver em  $G$  um caminho de  $u$  até  $v$ , então o número de  $u$  será menor que o de  $v$ .
- É possível provar que um digrafo  $G$  admite uma ordenação topológica se e somente se for acíclico.
- Se os vértices forem alinhados de acordo com uma ordenação topológica, todos os arcos terão uma mesma direção.
- Alguns exemplos de aplicação: cálculos de fórmulas em planilhas, gerenciamento de tarefas em um projeto (diagramas PERT/CPM), ordem de compilação dos arquivos fontes, resolução de conflitos em bancos de dados, sequência de entregas em redes de transportes, etc.

# Exercício

- Encontre uma ordenação topológica para os digrafos abaixo.
  - Dica: utilize os graus de entrada dos vértices.



Uma única ordenação



Mais de uma ordenação

# Através dos graus de entrada

- Calcular o grau de entrada de todos os vértices.
- Começar com o conjunto de vértices que têm grau de entrada nulo.
- Para cada um desses vértices, dar um numeração baixa, eliminá-los do conjunto e descontá-los nos graus de entrada dos seus sucessores. Se algum desses sucessores passar a ter grau de entrada nulo, incluí-lo no conjunto.
- A execução termina quando esse conjunto se torna vazio.

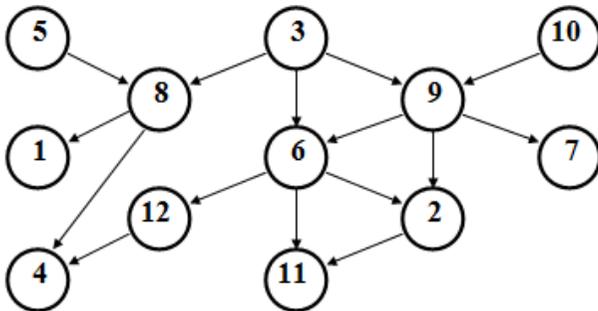
# Algoritmo

```
TopSort() {
    counter = 0;
    for v ∈ V
        calcular indegree(v);
    queue q;
    for v ∈ V
        if (indegree(v) == 0) q.enqueue(v);
    while (!q.isEmpty()) {
        v = q.dequeue();
        f[v] = ++counter;
        for <v,w> ∈ E
            if(--indegree(w) == 0) q.enqueue(w);
    }
    if (counter != n) print("Grafo é cíclico");
}
```

- Ao invés de fila, poderia ser utilizada uma pilha?

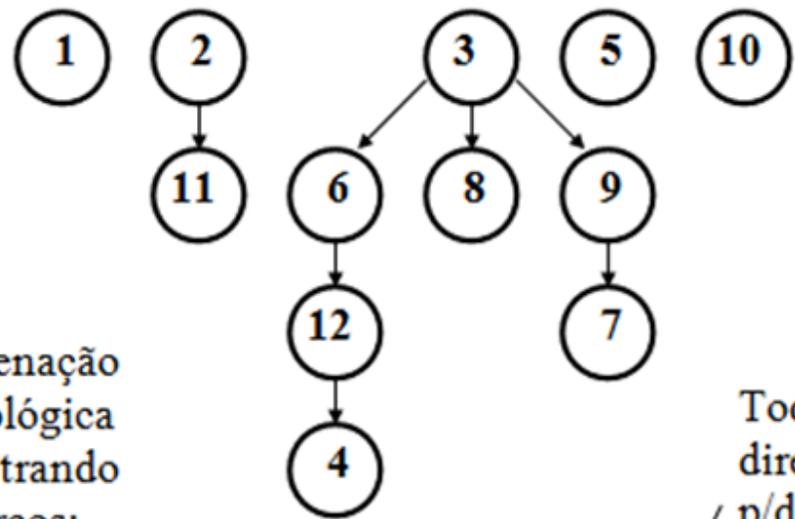
# Através da exploração em profundidade

- Considere o DAG abaixo:



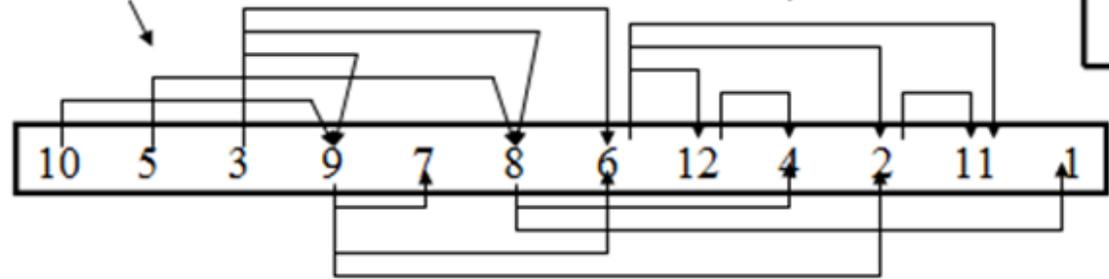
- Vamos fazer sua exploração em profundidade dando prioridade aos vértices de menor valor.

Sequência de término de exploração



Ordenação topológica mostrando os arcos:

Todas as direções p/direita



- 10
- 5
- 3
- 9
- 7
- 8
- 6
- 12
- 4
- 2
- 11
- 1

# Algoritmo

```
OrdemTopol(G) {  
    // supõe digrafo acíclico  
    int ce = 0;  
    int cc = 0;  
    for v ∈ V {  
        expl[v] = 0;  
        comp[v] = 0;  
    }  
    for v ∈ V  
        if (expl[v] == 0)  
            DFSOT(v);  
    for v ∈ V  
        f[v] = n-comp[v]+1;  
}
```

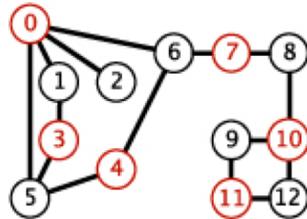
```
DFSOT(v) {  
    expl[v] = ++ce;  
    for <v,u> ∈ E  
        if (expl[u] == 0)  
            DFSOT(u);  
    comp[v] = ++cc;  
}
```

Usando uma pilha, seria possível imprimir os vértices já na ordem topológica. Como?

Complexidade de tempo:  $\Theta(n+m)$

# Bipartição (ou bicoloração) de vértices

- Um grafo  $G$  é bipartido (ou bicolorido) quando existe uma bipartição de seus vértices em subconjuntos disjuntos  $V_1$  e  $V_2$  tais que qualquer aresta de  $G$  possui uma extremidade em  $V_1$  e outra em  $V_2$ .
- Exemplo:



$$V_1 = \{0, 3, 4, 7, 10, 11\}$$

$$V_2 = \{1, 2, 5, 6, 8, 9, 12\}$$

- É possível demonstrar que um grafo admite bipartição se e somente se não tiver ciclos de tamanho ímpar.
- Alguns exemplos de aplicação: otimização do tráfego em redes de comunicação, agendamento de horários ou recursos, detecção de comunidades em redes sociais, otimização em cadeias de suprimentos (melhores parceiros), etc.
- Uma simples variação no algoritmo de exploração em profundidade permite encontrar uma bipartição de um grafo, se existir.
- O algoritmo a seguir produz uma bipartição (atribui "1" ou "2" ao número de exploração de cada vértice) ou informa que o grafo não pode ser bipartido.

# Algoritmo

Ambos pseudocódigos retornam 0 se não existir bipartição

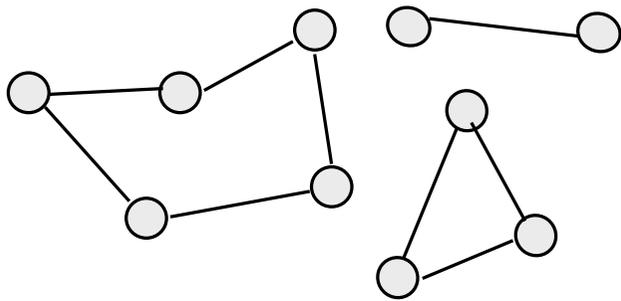
```
TarjanBP(G) {
  for v ∈ V
    expl[v] = 0;
  for v ∈ V
    if (expl[v] == 0)
      if (DFSBP(v,2) == 0)
        return 0;
  return 1;
}
```

```
DFSBP(v,color) {
  c = (color % 2) + 1;
  expl[v] = c; // troca cor
  for <v,u> ∈ E
    if (expl[u] == 0) {
      if (DFSBP(u,c) == 0)
        return 0; }
    else if (expl[u] == c)
      return 0;
  return 1;
}
```

Complexidade de tempo:  $\Theta(n+m)$

# Componentes conexas

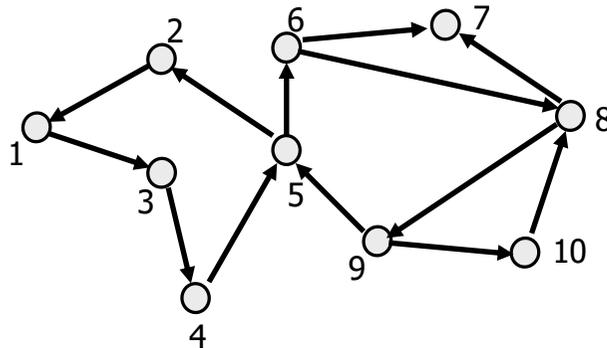
- Em um grafo, dizemos que dois vértices  $v$  e  $u$  estão conectados entre si se houver um caminho entre eles.
- Componentes conexas são subconjuntos maximais de vértices conectados entre si.
- Um grafo conexo possui uma única componente conexa, ou seja, todos seus vértices estão conectados entre si.
- Exemplo:



Grafo com 3  
componentes conexas

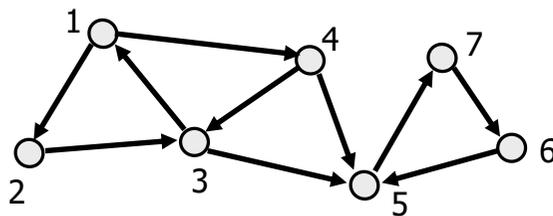
# Componentes fortemente conexas (SCC)

- Em um digrafo, dois vértices  $v_i$  e  $v_j$  estão conectados entre si se existem caminhos de  $v_i$  a  $v_j$  e de  $v_j$  a  $v_i$ . Desse modo, surge analogamente o conceito de componente fortemente conexa (CFC).
- Exemplos:



{1, 2, 3, 4, 5, 6, 8, 9, 10}

{7}



{1, 2, 3, 4}

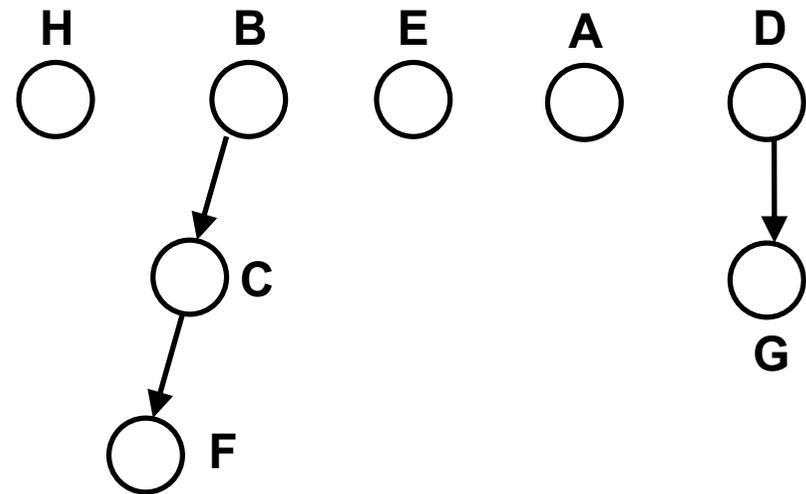
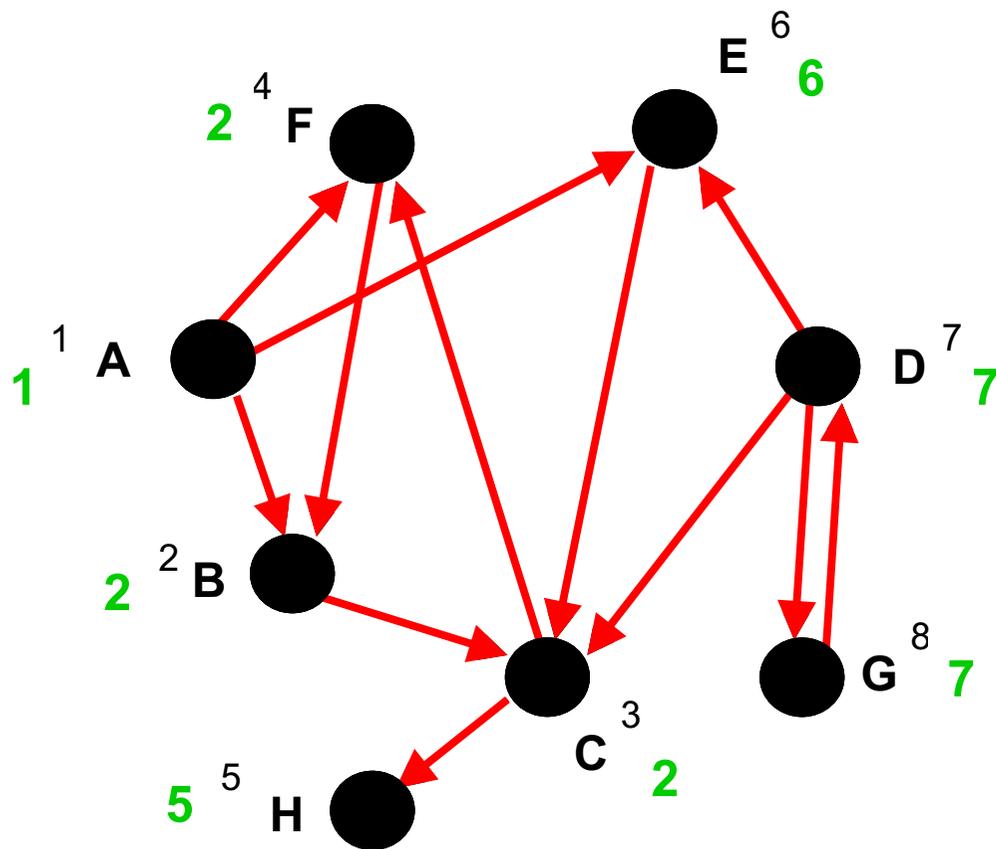
{5, 6, 7}

- É possível criar um digrafo acíclico (DAG) considerando cada CFC como um vértice e mantendo os arcos incidentes em CFCs distintas.
- Alguns exemplos de aplicação: possibilitar uma ordenação topológica, paralelização de laços, resolução de conflitos de concorrência, etc.

# Ideia do algoritmo

- É possível encontrar as componentes fortemente conexas de um digrafo através de uma variante do algoritmo de Tarjan.
- Considere a árvore  $T$  de exploração em profundidade e a numeração  $\text{expl}[v]$  para cada  $v \in V$ .
- Os vértices que estão em exploração são empilhados (permanecerão nessa pilha até que seja encontrada a sua componente conexa).
- Cada vértice  $v$  guardará  $\text{CFC}[v]$ , que é o menor número de exploração entre os vértices na pilha que atingir durante sua exploração. Desse modo, ficará automaticamente associado a uma componente conexa.
- Quando a exploração do vértice  $v$  terminar, se  $\text{expl}[v] = \text{CFC}[v]$  então todos os vértices na pilha (desde o topo até  $v$ ) pertencem a uma mesma componente, e podem ser desempilhados.

# Exemplo



- Importante: nem todos os vértices de uma mesma componente terminam com o mesmo valor de CFC.
- Exemplo: acrescente um arco  $\langle C, A \rangle$  nesse mesmo digrafo, e visite  $\langle C, F \rangle$  antes.

# Algoritmo

```
TarjanCFC(G) {  
  stack P;  
  int ce = 0;  
  for v ∈ V  
    expl[v] = 0;  
  for v ∈ V  
    if (expl[v] == 0)  
      DFSCFC(v);  
}
```

```
DFSCFC(v) {  
  expl[v] = ++ce;  
  push(P,v);  
  CFC[v] = expl[v];  
  for <v,u> ∈ E  
    if (expl[u] == 0) {  
      DFSCFC(u);  
      CFC[v] = min{CFC[v],CFC[u]};  
    }  
    else if (u ∈ P)  
      CFC[v] = min{CFC[v],expl[u]};  
  if (CFC[v] == expl[v])  
    do {  
      x = top(P);  
      pop(P);  
    } while (x != v);  
}
```

Arco de árvore

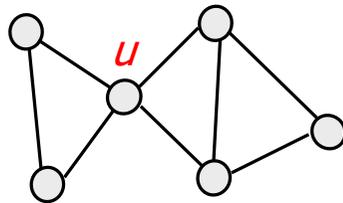
Teste em tempo constante com vetor de *flags*

Complexidade de tempo:  $\Theta(n+m)$

# Vértices e arestas de corte

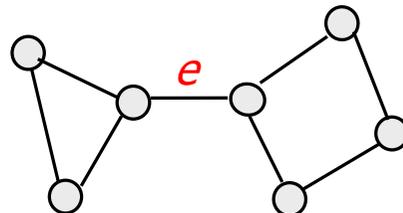
- Em um grafo  $G$ ,  $u \in V$  é chamado de vértice de corte (ou ponto de articulação) se a sua remoção desconecta  $G$ .

- Exemplo:



- Se uma componente conexa de um grafo não possui vértices de corte, ela é chamada de componente biconexa.
- Analogamente, uma aresta  $e$ , cuja remoção ocasiona a desconexão do grafo, recebe o nome de ponte (ou aresta de corte).

- Exemplo:



- A aplicação mais evidente deste algoritmo é detectar vulnerabilidades em redes de transmissão.

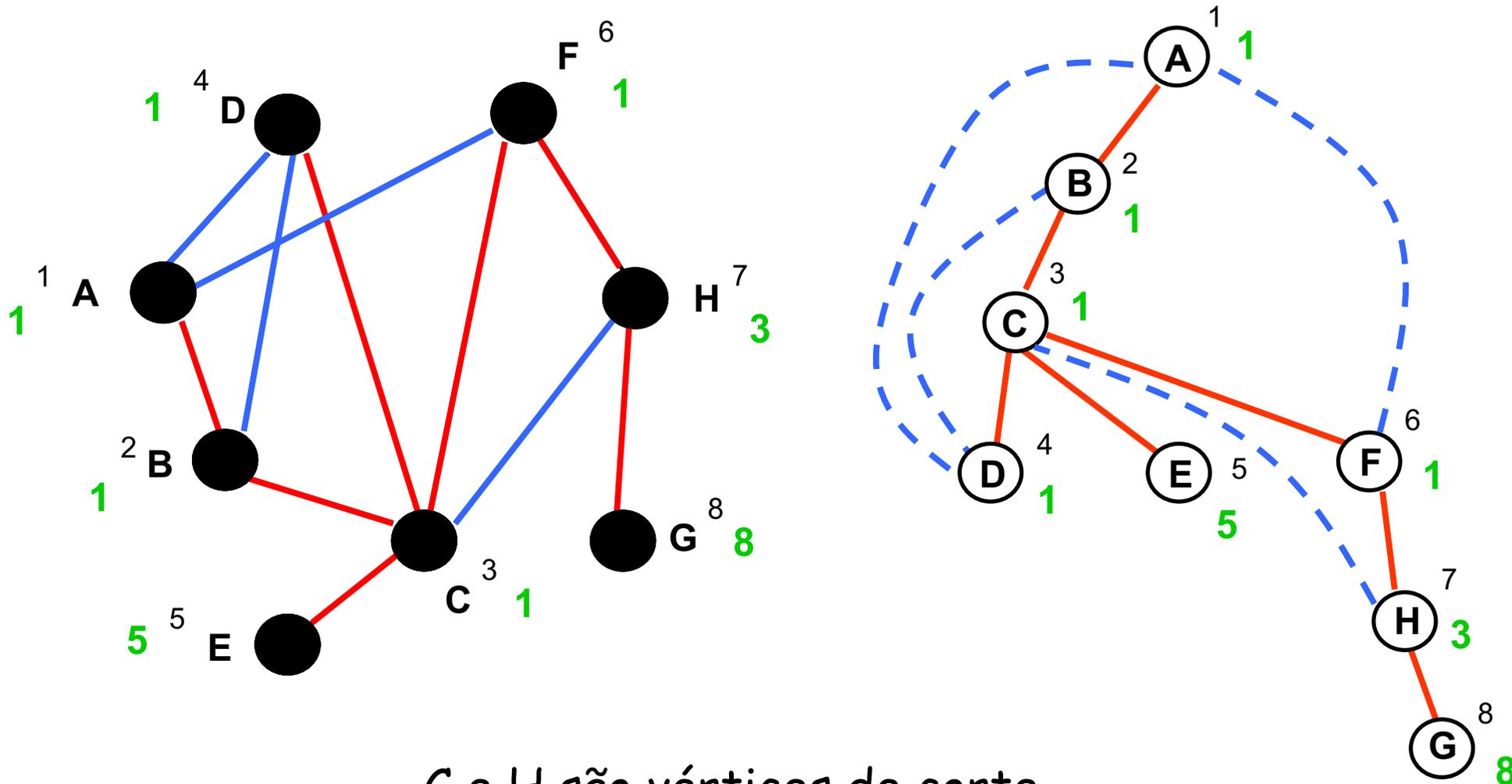
# Vértices de corte

- Uma variante do algoritmo de Tarjan pode encontrar os vértices de corte (ou pontos de articulação) de um grafo  $G=(V,E)$  *conexo não-orientado, sem laços ou arestas repetidas*.

Desse modo, se fizéssemos uma exploração em profundidade a partir de cada vértice do grafo, poderíamos identificar todos os vértices de corte (no entanto, há outra solução mais eficiente)

- **Ideia:**
  - Considere a árvore  $T$  de exploração em profundidade e a numeração  $expl[v]$  para cada  $v \in V$ .
  - **Raiz:** será vértice de corte se tiver pelo menos dois filhos em  $T$ .
  - **Demais vértices:**
    - $v$  será vértice de corte se tiver algum filho sem retorno para nenhum dos ancestrais de  $v$ .
    - É calculado  $m[v] = \min\{expl[v], expl[x]\}$ , onde  $x$  é um vértice que  $v$  (ou um de seus descendentes) atinge em  $T$  através de uma *única aresta de retorno*.
    - Portanto,  $v$  será vértice de corte se tiver algum filho  $u$  tal que  $m[u] \geq expl[v]$ .

# Exemplo



C e H são vértices de corte

# Algoritmo

```
TarjanVC(r) {  
  // válido se for conexo e  
  // não tiver laços ou  
  // arestas repetidas  
  int ce = 0;  
  for v ∈ V {  
    expl[v] = 0;  
    pai[v] = null;  
    nfilhos[v] = 0;  
    VC[v] = false;  
  }  
  DFSVC(r);  
  for v ∈ V-{r} {  
    p = pai[v];  
    VC[p] = VC[p] || (m[v] >= expl[p]);  
  }  
  VC[r] = (nfilhos[r] > 1);  
  for v ∈ V  
    if (VC[v]) v é vértice de corte;  
}
```

Trata as arestas de retorno, tanto na ida como na volta

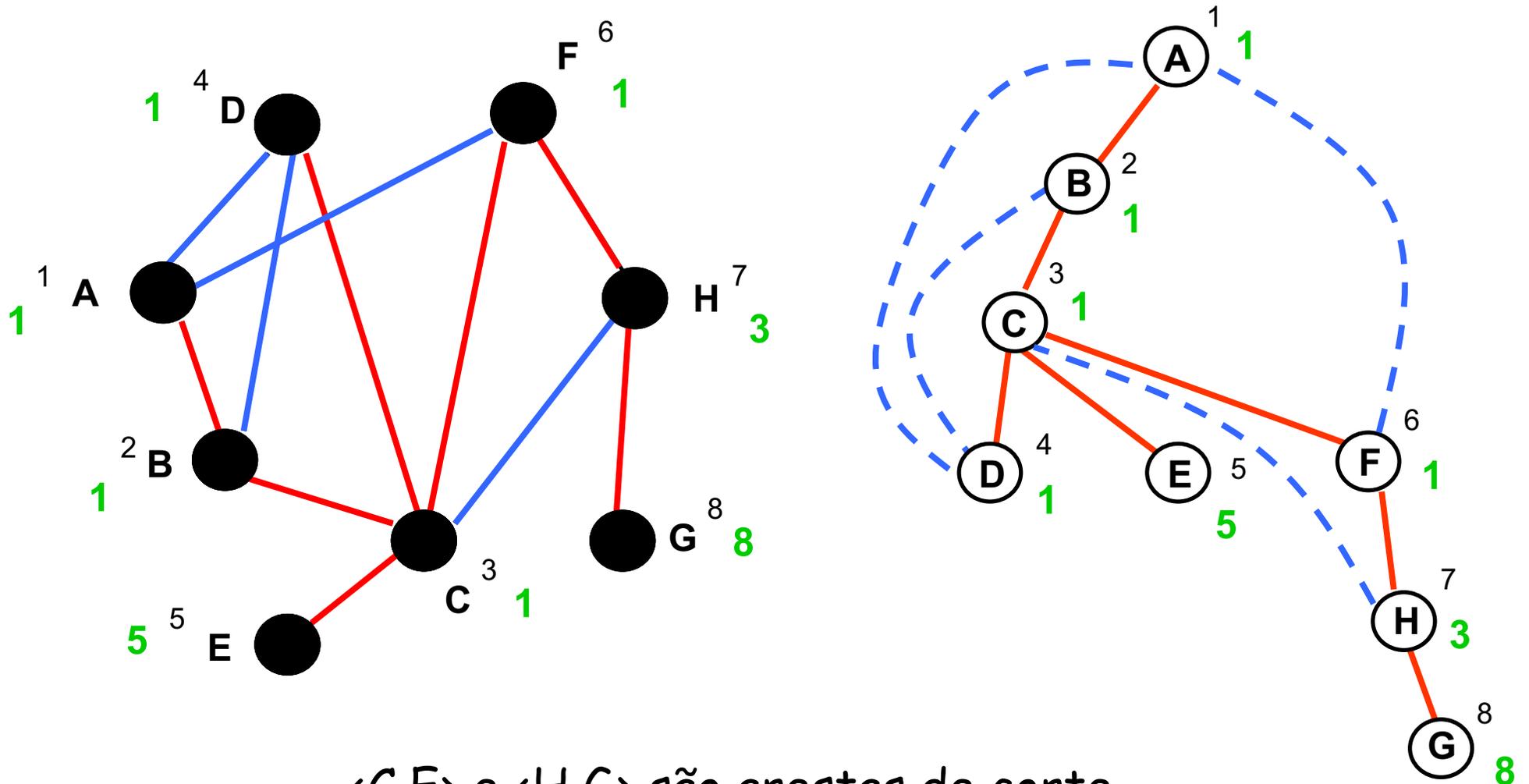
```
DFSVC(v) {  
  expl[v] = ++ce;  
  m[v] = expl[v];  
  for <v,u> ∈ E  
    if (expl[u] == 0) {  
      pai[u] = v;  
      nfilhos[v]++;  
      DFSVC(u);  
      m[v] = min{m[v], m[u]};  
    }  
  else // arestas de retorno  
    if (u != pai[v])  
      m[v] = min{m[v], expl[u]};  
}
```

Complexidade de tempo:  $\Theta(n+m)$

# Arestas de corte

- A identificação das arestas de corte (ou pontes) é realizada de maneira semelhante:
  - Encontrar uma árvore de exploração  $T$ , calculando as mesmas numerações  $expl$  e  $m$  para os vértices.
  - É fácil constatar que nenhuma aresta de retorno dessa exploração pode ser de corte.
  - Uma aresta  $\langle v, u \rangle \in T$  será de corte se  $m[u] = expl[u]$ .

# No exemplo anterior



$\langle C, E \rangle$  e  $\langle H, G \rangle$  são arestas de corte

# Algoritmo

```
TarjanAC(r) {  
    // válido se for conexo e  
    // não tiver laços ou  
    // arestas repetidas  
    int ce = 0;  
    for v ∈ V {  
        expl[v] = 0;  
        pai[v] = null;  
    }  
    DFSAC(r);  
}
```

```
DFSAC(v) {  
    expl[v] = ++ce;  
    m[v] = expl[v];  
    for <v,u> ∈ E  
        if (expl[u] == 0) {  
            pai[u] = v;  
            DFSAC(u);  
            m[v] = min{m[v],m[u]};  
            if (m[u] == expl[u])  
                <v,u> é aresta de corte;  
        }  
    else // arestas de retorno  
        if (u != pai[v])  
            m[v] = min{m[v],expl[u]};  
}
```

Complexidade de tempo:  $\Theta(n+m)$