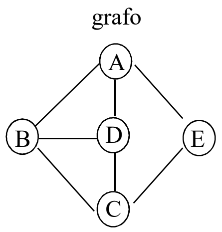
Algoritmos em Grafos: Conectividade

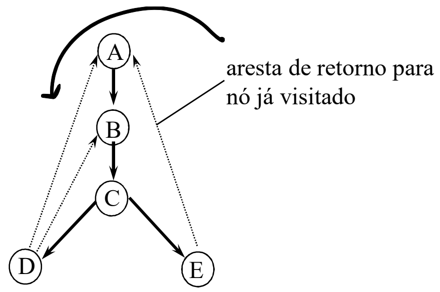
R. Rossetti, L. Ferreira, H. L. Cardoso, F. Andrade
FEUP, MIEIC, CAL

Grafos não dirigidos

Conectividade

- Um grafo não dirigido é conexo sse uma pesquisa em profundidade a começar em qualquer nó visita todos os nós

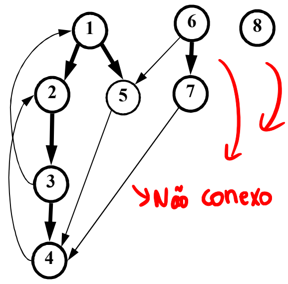




Pesquisa em profundidade

```
class Graph { …
  void dfs() {
    for (Vertex v : vertexSet)
       v.visited = false;
    for (Vertex v : vertexSet)
      if (! v.visited)
        dfs(v);
        //v passa a ser raiz duma árvore dfs
  }
  void dfs( Vertex v ) {
    v.visited = true;
      // fazer qualquer coisa c/ v aqui
    for(Edge e : v.adj)
      if( ! e.dest.visited )
        dfs( e.dest );
      // ou aqui
    }
}
```

Exemplo em grafo dirigido (vértices numerados por ordem de visita e dispostos por profundidade de recursão):



Não conexo

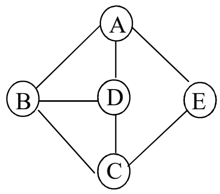
Arestas a traço forte: árvore de expansão em profundidade

Na DFS podem ser produzidas várias árvores, porque a pesquisa pode ser repetida a partir de várias fontes (ao contrário da BFS que só produz uma). O conjunto das várias árvores é conhecido como Floresta DFS.

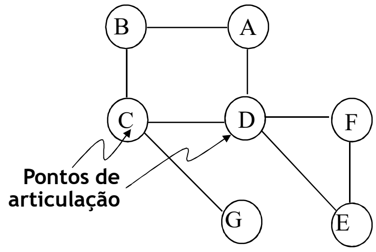
Biconectividade e Pontos de Articulação

- Grafo conexo não dirigido é **biconexo** se não existe nenhum vértice cuja remoção torne o resto do grafo desconexo
- **Pontos de articulação**: vértices que tornam o grafo desconexo
- Aplicação - rede com tolerância a falhas

Biconexo



Não biconexo



FEUP Universidade do Porto Faculdade de Engenharia

Algoritmos em Grafos: Conectividade, CAL

Algoritmo de detecção de pontos de articulação

- Início num vértice qualquer
- Pesquisa em profundidade, numerando os vértices ao visitá-los — Num(v), em pré-ordem (antes de visitar adjacentes) Numerar antes da chamada recursiva
- Para cada vértice *v*, na árvore de visita em profundidade, calcular *Low(v)*: o menor número de vértice que se atinge com zero ou mais arestas na árvore e possivelmente uma aresta de retorno → Em qualquer ponto, quão próximo é possível chegar à Raíz
- Vértice v é ponto de articulação se tiver um filho w tal que Low(w) ≥ Num(v)
- A raiz é ponto de articulação sse tiver mais que um filho na árvore

FEUP Universidade do Porto Faculdade de Engenharia

Algoritmos em Grafos: Conectividade, CAL

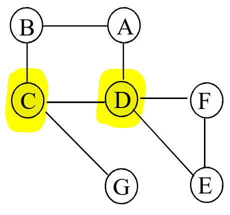
Cálculo de Low(v)

- Low(v) é mínimo de
  - Num(v)
  - o menor Num(w) de todas as arestas (v, w) de retorno
  - o menor Low(w) de todas as arestas (v, w) da árvore
- Na visita em profundidade, inicializa-se Low(v)=Num(v) antes de visitar adjacentes, e vai-se actualizando o valor de Low(v) a seguir a visita a cada adjacente
- Realizável em tempo O(|E| + |V|)

FEUP Universidade do Porto Faculdade de Engenharia Algoritmos em Grafos: Conectividade, CAL

Exemplo

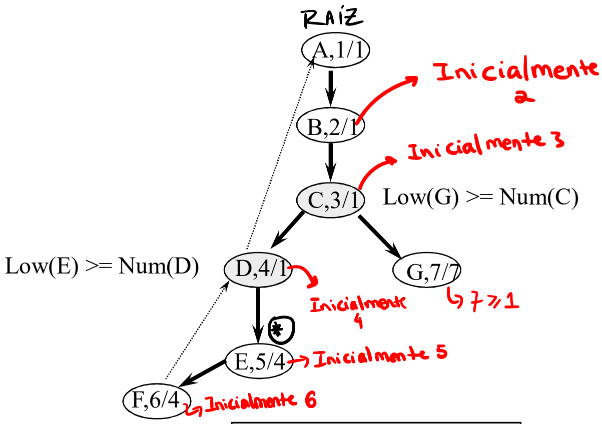
Grafo



. C e D pontos de articulação

Uma árvore de expansão em profundidade

v, Num(v)/Low(v)



Pontos de articulação: C e D

Nenhum descendente de D atinge a Raiz se este for retirado

(*) temos um descendente de E com low + baixo, também diminui

FEUP Universidade do Porto Faculdade de Engenharia Algoritmos em Grafos: Conectividade, CAL

Pseudo-código

```
// Procura Pontos de Articulação usando dfs
// Contador global e inicializado a 1
void findArt( Vertex v) {            → escolhemos arbitrariamente
     v.visited = true;
     v.low = v.num = counter++;       → para inicializar
     for each w adjacent to v         → para todos os adjacentes
          if( !w.visited ) {          → se ainda n foi visitado  // ramo da árvore
passa a ser o antecessor ← w.parent = v;
               findArt(w);            → Recursivamente
               v.low = min(v.low, w.low);  → mínimo entre v e os descendentes
               if(w.low >= v.num )    → ponto de articulação
                    System.out.println(v, "Ponto de articulação");
          }
          else                        → se já foi visitado
             if ( v.parent != w )              //aresta de retorno
               v.low = min(v.low, w.num);
}
```

$O(|E|+|V|)$

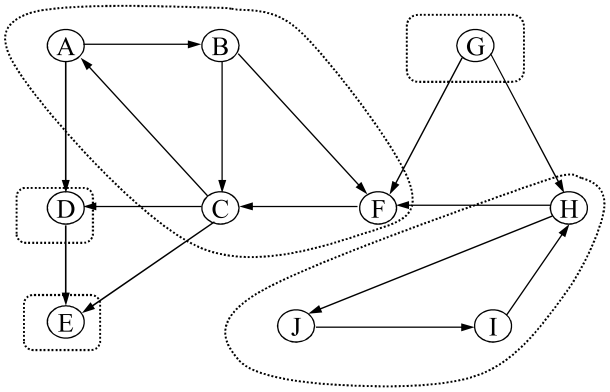
FEUP Universidade do Porto Faculdade de Engenharia — Algoritmos em Grafos: Conectividade, CAL

Grafos dirigidos

FEUP Universidade do Porto Faculdade de Engenharia — Algoritmos em Grafos: Conectividade, CAL

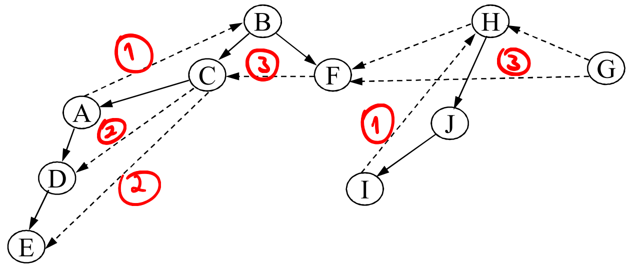
Componentes fortemente conexos

· que subgrafo permite chegar a todos os outros?



Árvore de expansão

- Pesquisa em profundidade induz uma árvore/floresta de expansão
- Para além das arestas genuínas da árvore, há arestas para nós já marcados
  1 • arestas de retorno para um antepassado – (A,B), (I,H) → Para um nível <
  2 • arestas de avanço para um descendente – (C,D), (C,E) → Para um nível >
  3 • arestas cruzadas para um nó não relacionado – (F,C), (G,F) → Para um vértice do mesmo nível

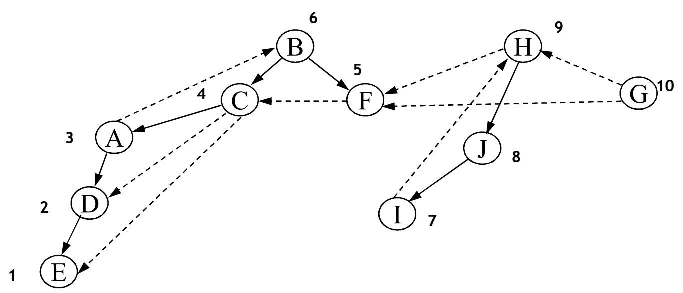


Componentes fortemente conexos

- Método:
  - Pesquisa em profundidade no grafo G determina floresta de expansão, numerando vértices em pós-ordem (ordem inversa de numeração em pré-ordem)
    ↳ primeiro visita, depois atualiza
  - Inverter todas as arestas de G (grafo resultante é Gr) → Caminho de retorno
  - Segunda pesquisa em profundidade, em Gr, começando sempre pelo vértice de numeração mais alta ainda não visitado
  - Cada árvore obtida é um componente fortemente conexo, i.e., a partir de um qualquer dos nós pode chegar-se a todos os outros

FEUP Universidade do Porto Faculdade de Engenharia
Algoritmos em Grafos: Conectividade, CAL

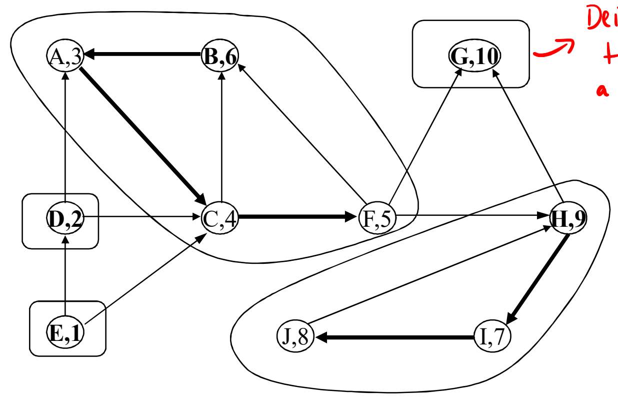
Numeração em pós-ordem



↳ Depois invertemos

FEUP Universidade do Porto Faculdade de Engenharia
Algoritmos em Grafos: Conectividade, CAL

Inversão das arestas e nova visita

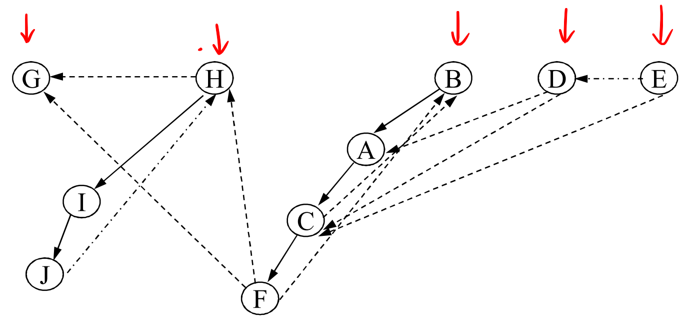


Deixa de ter acesso a outros vértices

G_r: obtido de G por inversão de todas as arestas
Numeração: da travessia de G em pós-ordem

Ao finalizar a 2ª pesquisa em profundidade, temos 5 árvores.
Cada uma destas componentes é fortemente conexa

Componentes fortemente conexos



Travessia em pós-ordem de G_r
Componentes fortemente conexos:
{G}, {H, I, J}, {B, A, C, F}, {D}, {E}