

Algoritmos em Grafos

Wladimir Araújo Tavares¹

¹Universidade Federal do Ceará - Campus de Quixadá

26 de abril de 2017

Grafos

- Uma estrutura matemática formada por um conjunto de n vértices e um conjunto de m arestas.
- As arestas conectam pares de vértices. As arestas podem ser direcionadas ou não.
-

- 1 Busca em largura
 - Diâmetro de uma árvore
 - Centro de uma árvore
 - Diâmetro de um grafo
- 2 Busca em profundidade
 - Ordenação Topológica
 - Componentes Conexas de grafos não direcionados
 - Grafo direcionado fortemente conexo
 - Componentes Fortemente Conexas em grafos direcionados
 - Ponto de articulação
 - Pontes
- 3 Coloração Gulosa
- 4 Exercício: k-coloração
- 5 Exercício: Número de caminhos
- 6 Exercício: Percurso de Euler em grafos não direcionados
- 7 Exercício: Percurso de Euler em grafos direcionados

Busca em largura

Algorithm 1 Algoritmo BFS

```
1: function BFS( $G, s$ )
2:   for cada  $v \in V(G)$  do
3:      $cor[v] \leftarrow \text{BRANCO}$ 
4:      $\pi[v] \leftarrow \text{NULL}$ 
5:      $d[v] \leftarrow \infty$ 
6:    $cor[s] \leftarrow \text{CINZA}$ 
7:    $d[s] \leftarrow 0$ 
8:    $Q \leftarrow \emptyset$ 
9:   ENQUEUE( $s$ )
10:  while  $Q \neq \emptyset$  do
11:     $u \leftarrow \text{DEQUEUE}(Q)$ 
12:    for cada  $v \in \text{Adj}[u]$  do
13:      if  $cor[v] = \text{BRANCO}$  then
14:         $cor[v] \leftarrow \text{CINZA}$ 
15:         $d[v] \leftarrow d[u] + 1$ 
16:         $\pi[v] \leftarrow u$ 
17:        ENQUEUE( $Q, v$ )
18:   $cor[u] \leftarrow \text{PRETO}$ 
```

Bons e maus sujeitos

- 1 Existem dois tipos de lutadores profissionais: "bons sujeitos" e "maus sujeitos". Entre qualquer par de lutadores profissionais pode ou não haver rivalidade. Suponha que temos n lutadores e uma lista de m pares de lutadores para os quais existem rivalidades. Forneça um algoritmo de tempo $O(n + m)$ que determina se é possível designar alguns dos lutadores como bons sujeitos e os restantes como maus sujeitos, de tal forma que a rivalidade ocorra em cada caso entre um bom sujeito e um mau sujeito.

Bons e maus sujeitos

Algorithm 2 Algoritmo BOM_E_MAL

- 1: **function** BOM_E_MAL(G)
 - 2: Execute a quantidade de BFS necessárias até que todos os vértices sejam visitados.
 - 3: Atribua a todos os lutadores cuja distância é par para ser "bom sujeito" e os lutadores cuja distância é ímpar para ser "mal sujeito".
 - 4: Verifique se a rivalidade em cada aresta acontece entre um "bom sujeito" e um "mal sujeito."
-

Diâmetro de uma árvore

- 2 O diâmetro de uma árvore $T = (V, E)$ é dado por $\max_{u,v \in V} d(u, v)$, onde $d(u, v)$ é a distância mínima entre os vértices u e v , ou seja, o diâmetro é a maior das menores distâncias na árvore. Forneça um algoritmo eficiente para calcular o diâmetro de uma árvore e analise o tempo de execução de seu algoritmo.

Diâmetro de uma árvore

Algorithm 3 Algoritmo diâmetro de uma árvore

- 1: **function** DIAMETER(T)
 - 2: Execute uma BFS a partir de um vértice qualquer. Seja u o vértice mais distante encontrado pela BFS.
 - 3: Execute uma BFS a partir de u .
 - 4: A maior distância encontrada pela segunda BFS é o diâmetro da árvore T .
-

Centro de uma árvore

- 3 A excentricidade de um vértice v , denotado por $E(v)$, é a distância mínima de v até o vértice mais longe de v . O centro de um grafo são os vértices que tem o menor valor de excentricidade. Encontre o centro de uma árvore T .

Diâmetro de um grafo

- 2 O diâmetro de um grafo $G = (V, E)$ é dado por $\max_{v \in V} E(v)$, onde $E(v)$ é a distância mínima entre o vértice v até o vértice mais longe de v , ou seja, o diâmetro é a maior das menores distâncias em um grafo. Forneça um algoritmo eficiente para calcular o diâmetro de um grafo e analise o tempo de execução de seu algoritmo.

- 1 Busca em largura
 - Diâmetro de uma árvore
 - Centro de uma árvore
 - Diâmetro de um grafo
- 2 Busca em profundidade
 - Ordenação Topológica
 - Componentes Conexas de grafos não direcionados
 - Grafo direcionado fortemente conexo
 - Componentes Fortemente Conexas em grafos direcionados
 - Ponto de articulação
 - Pontes
- 3 Coloração Gulosa
- 4 Exercício: k-coloração
- 5 Exercício: Número de caminhos
- 6 Exercício: Percurso de Euler em grafos não direcionados
- 7 Exercício: Percurso de Euler em grafos direcionados

Busca em Profundidade

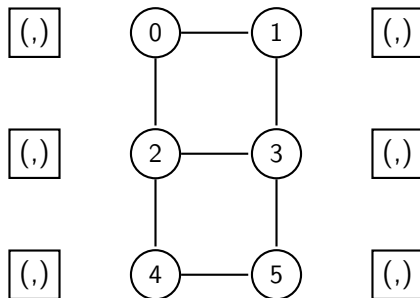
Algorithm 4 Algoritmo DFS

```
1: function DFS( $G$ )
2:   for cada  $v \in V(G)$  do
3:      $cor[v] \leftarrow BRANCO$ 
4:      $\pi[v] \leftarrow NULL$ 
5:    $tempo \leftarrow 0$ 
6:   for cada  $v \in V(G)$  do
7:     if  $cor[v] = BRANCO$  then
8:        $DFS\_VISIT(v)$ 
```

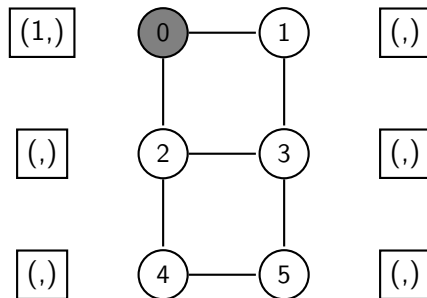
Algorithm 5 Algoritmo DFS_VIST

```
1: function DFS_VISIT( $v$ )
2:    $cor[v] \leftarrow CINZA$ 
3:    $tempo \leftarrow tempo + 1$ 
4:    $chegada[v] \leftarrow tempo$ 
5:   for cada  $u \in Adj[v]$  do
6:     if  $cor[u] = BRANCO$  then
7:        $\pi[u] \leftarrow v$ 
8:        $DFS\_VISIT(u)$ 
9:    $cor[u] \leftarrow PRETO$ 
10:   $tempo \leftarrow tempo + 1$ 
11:   $partida[u] \leftarrow tempo$ 
```

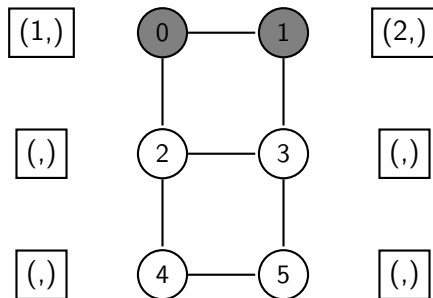
Busca em profundidade



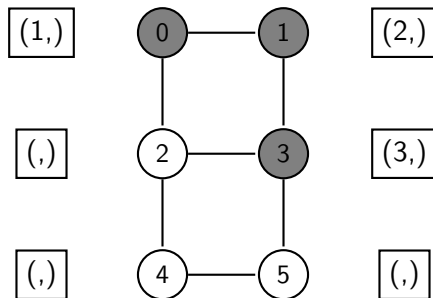
Busca em profundidade



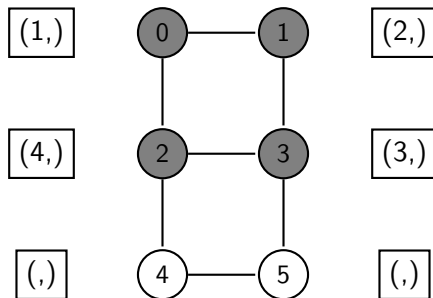
Busca em profundidade



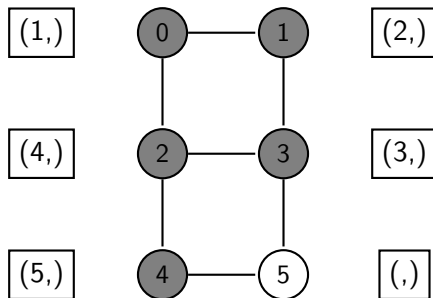
Busca em profundidade



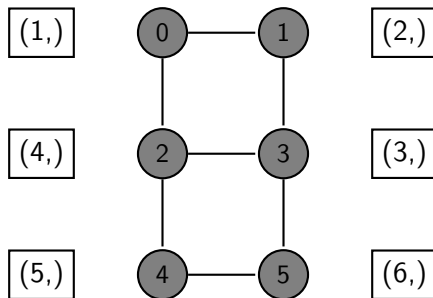
Busca em profundidade



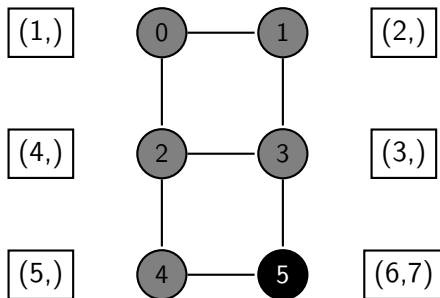
Busca em profundidade



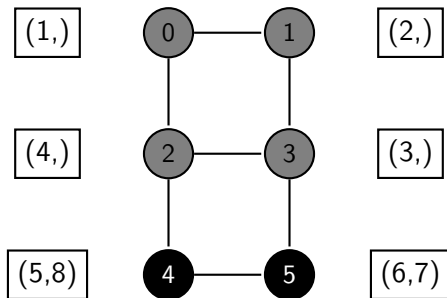
Busca em profundidade



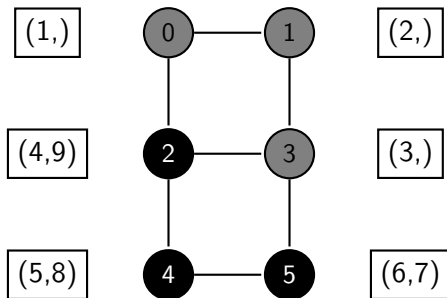
Busca em profundidade



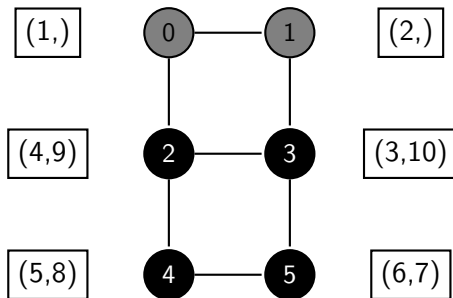
Busca em profundidade



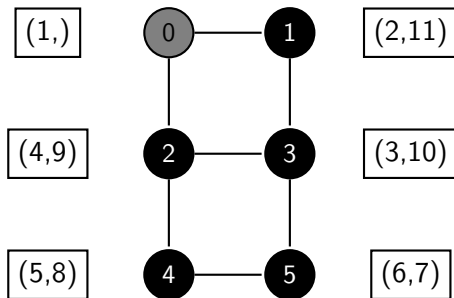
Busca em profundidade



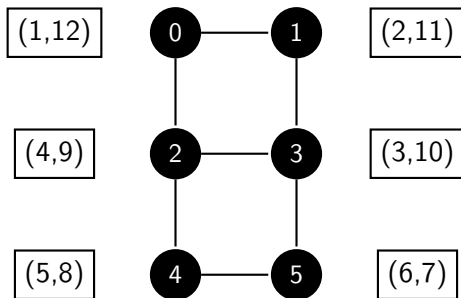
Busca em profundidade



Busca em profundidade



Busca em profundidade



Classificação de arestas em grafos não-direcionado

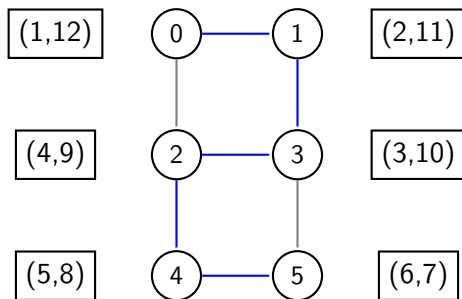
- A aresta (u, v) é uma **aresta da árvore** se v foi descoberto primeiro pela exploração da aresta (u, v) .

$$chegada[u] < chegada[v] < partida[v] < partida[u] \quad (1)$$

- A aresta (u, v) é uma **aresta de retorno** se ela conecta um vértice u a um ancestral v em uma árvore da busca em profundidade.. Então,

$$chegada[v] < chegada[u] < partida[u] < partida[v] \quad (2)$$

Classificação de arestas em grafos não-direcionado



Classificação de arestas em grafos não-direcionado

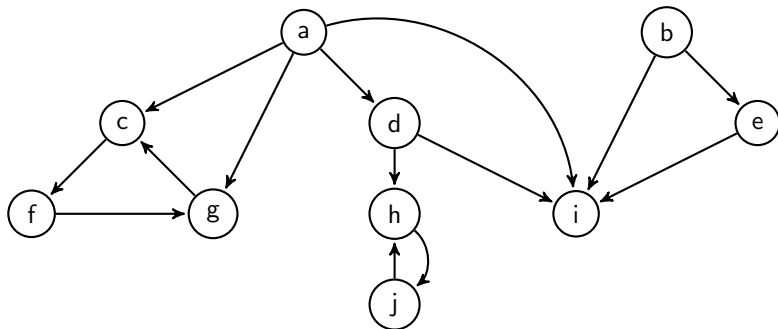
- A aresta (u, v) é uma **aresta da árvore** se v foi descoberto primeiro pela exploração da aresta (u, v) .
- A aresta (u, v) é uma **aresta de retorno** se ela conecta um vértice u a um ancestral v em uma árvore da busca em profundidade.
- A aresta (u, v) é uma **aresta direta** se ela conecta um vértice u a um descendente v em uma árvore de busca em profundidade.
- A aresta (u, v) é uma **aresta cruzada** se ela conecta um vértice que não seja ancestral do outro.

Classificação de arestas em grafos direcionado

A classificação pode ser definida quando a aresta (u, v) é explorada primeiro, se a cor de v for

- BRANCO indica uma aresta de árvore.
- CINZA indica uma aresta de retorno.
- PRETO e $chegada[u] < chegada[v]$ indica uma aresta direta.
- PRETO e $chegada[u] > chegada[v]$ indica uma aresta cruzada.

Classificação de arestas em grafos direcionado



Aplicações da DFS com tempo de chegada e partida:

- Ordenação topológica em um grafo direcionado acíclico.
- Detecção de componentes conexas em grafos não-direcionados.
- Determina se um grafo direcionado é fortemente conexo
- Detecção de componentes fortemente conexa em grafos direcionados.
- Detecção de ciclos em grafos direcionados e não-direcionados.
- Encontrar pontes e articulações.

Ordenação Topológica

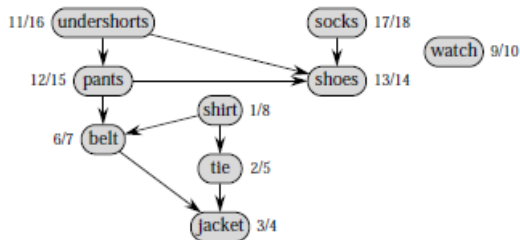


Figura O professor Bumstead ordena topologicamente sua roupa ao se vestir. Cada aresta orientada (u, v) representa que a peça u deve ser vestida antes da peça v

Ordenação Topológica

Algorithm 6 Algoritmo Ordenação Topológica(G)

- 1: **function** TOPOLOGICAL_SORT(G)
 - 2: CHAMAR DFS(S) para calcular *partida*[v] para cada vértice v
 - 3: à medida que cada vértice é terminado, inserir o vértice à frente de uma lista ligada
 - 4: **return** lista ligada de vértices
-

Ordenação Topológica

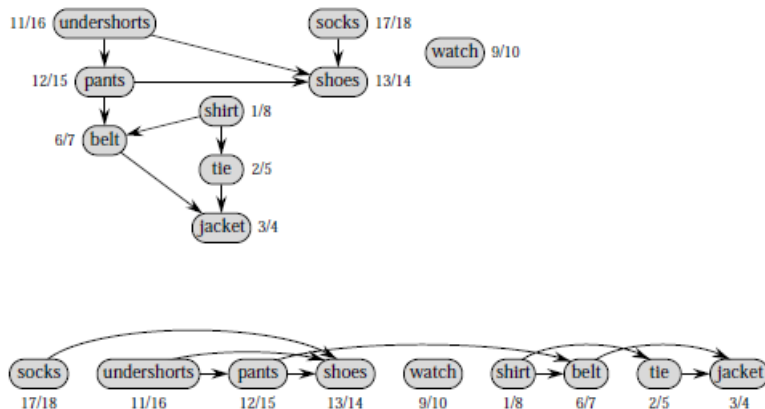


Figura O professor Bumstead ordena topologicamente sua roupa ao se vestir. Cada aresta orientada (u, v) representa que a peça u deve ser vestida antes da peça v

Componentes Conexas em grafos não direcionados

Algorithm 7 Algoritmo DFS_COMPONENTE

```
1: function DFS_COMPONENTE( $G$ )
2:   for cada  $v \in V(G)$  do
3:      $cor[v] \leftarrow \text{BRANCO}$ 
4:      $\pi[v] \leftarrow \text{NULL}$ 
5:    $tempo \leftarrow 0$ 
6:    $cont \leftarrow 0$ 
7:   for cada  $v \in V(G)$  do
8:     if  $cor[v] = \text{BRANCO}$  then
9:        $cont \leftarrow cont + 1$ 
10:       $DFS\_VISIT(v, cont)$ 
```

Componentes Conexas em grafos não direcionados

Algorithm 8 Algoritmo DFS_VIST

```
1: function DFS_VISIT( $v, cont$ )
2:    $cor[v] \leftarrow CINZA$ 
3:    $cc[v] \leftarrow cont$ 
4:    $tempo \leftarrow tempo + 1$ 
5:    $chegada[v] \leftarrow tempo$ 
6:   for cada  $u \in Adj[v]$  do
7:     if  $cor[u] = BRANCO$  then
8:        $\pi[u] \leftarrow v$ 
9:        $DFS\_VISIT(u, cont)$ 
10:   $cor[u] \leftarrow PRETO$ 
11:   $tempo \leftarrow tempo + 1$ 
12:   $partida[u] \leftarrow tempo$ 
```

Componentes Conexas em grafos não direcionados

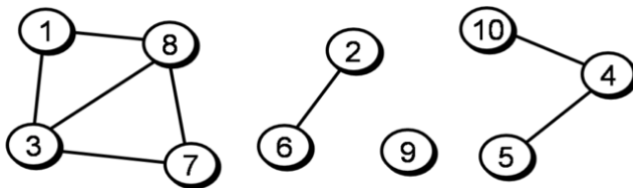


Figura O algoritmo encontra 4 componentes conexas: $\{1,3,7,8\}$, $\{2,6\}$, $\{9\}$ e $\{4,5,10\}$

Grafo direcionado fortemente conexo

Um grafo direcionado é fortemente conexo se:

- $DFS_VISIT(G, v)$ visita todos os vértices de G , então existe um caminho de v para todos outros vértices de G .
- $DFS_VISIT(G^T, v)$ visita todos os vértices de G , então existe um caminho de todos os outros vértices de G para v .

Grafo direcionado fortemente conexo

Algorithm 9 STRONGLY_CONNECTED(G)

```
1: function STRONGLY_CONNECTED( $G$ )
2:   for cada  $v \in V(G)$  do
3:      $cor[v] \leftarrow \text{BRANCO}$ 
4:      $\pi[v] \leftarrow \text{NULL}$ 
5:   CHAMAR  $\text{DFS\_VISIT}(G, v)$ 
6:   if  $\exists v \text{ } cor[v] = \text{BRANCO}$  then
7:     return false
8:   Calcule  $G^T$ 
9:   for cada  $v \in V(G)$  do
10:     $cor[v] \leftarrow \text{BRANCO}$ 
11:     $\pi[v] \leftarrow \text{NULL}$ 
12:   CHAMAR  $\text{DFS\_VISIT}(G^T, v)$ 
13:   if  $\exists v \text{ } cor[v] = \text{BRANCO}$  then
14:     return false
15:   else
16:     return true
```

Componentes Fortemente Conexas em grafos direcionados

- Um componente fortemente conexa em um grafo direcionado em $G = (V, E)$ é um conjunto maximal de vértice $C \subseteq V$ tal que para todo par de vértice u e v em C , temos ao mesmo tempo $u \rightarrow v$ e $v \rightarrow u$.
- Se u e v são acessíveis um a partir do outro em G se e somente se eles são acessíveis um a partir do outro em G^T .

Componentes Fortemente Conexas em grafos direcionados

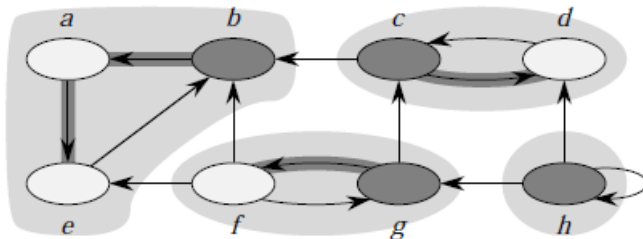


Figura As componentes fortemente conexas de G são mostradas como regiões sombreadas.

Componentes Fortemente Conexas em grafos direcionados

Algorithm 10 STRONGLY_CONNECTED_COMPONENTS(G)

- 1: **function** STRONGLY_CONNECTED_COMPONENTS(G)
 - 2: CHAMAR DFS(G) para calcular $partida[v]$ para cada vértice v em $O(V + E)$
 - 3: Calcular G^T em $O(V + E)$
 - 4: CHAMAR DFS(G^T) mas , no loop principal do DFS, considerar os vértices em ordem decrescente de $partida[v]$
 - 5: Dar saída aos vértices de cada árvore da floresta da busca primeiro na profundidade como uma componente fortemente conexa separadamente.
-

Componentes Fortemente Conexos em grafos direcionados

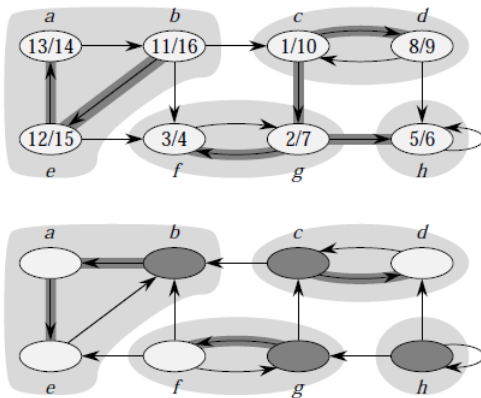


Figura As componentes fortemente conexas de G são mostradas como regiões sombreadas.

Algoritmo de Tarjan - SCC

Algorithm 11 Algoritmo TARJAN

```
1: function TARJAN( $G$ )
2:    $index \leftarrow 0$ 
3:   for cada  $v \in V$  do
4:      $v.index \leftarrow NULL$ 
5:      $v.onStack \leftarrow false$ 
6:   for cada  $v \in V(G)$  do
7:     if  $v.index = NULL$  then
8:        $strongconnect(v)$ 
```

Algoritmo de Tarjan - SCC

Algorithm 12 Algoritmo strongconnect(v)

```
1: function strongconnect( $v$ )
2:    $v.index \leftarrow index$ 
3:    $v.lowlink \leftarrow index$ 
4:    $index \leftarrow index + 1$ 
5:    $push(S, v)$ 
6:    $v.onStack \leftarrow true$ 
7:   for cada  $u \in Adj[v]$  do
8:     if  $u.index = NULL$  then
9:        $strongconnect(u)$ 
10:     $v.lowlink \leftarrow \min(v.lowlink, u.lowlink)$ 
11:   else if  $u.onStack$  then
12:      $v.lowlink \leftarrow \min(v.lowlink, u.index)$ 
13:   if  $v.lowlink = v.index$  then
14:     inicialize uma componente fortemente conexa
15:     repeat
16:        $w \leftarrow pop(S)$ 
17:        $w.onStack \leftarrow false$ 
18:       Adiciona  $w$  a componente fortemente conexa atual
19:     until  $w = v$ 
20:     Imprima a componente fortemente conexa
```

Ponto de articulação

- Um vértice v em um grafo conexo não orientado $G = (V, E)$ é uma **articulação** se sua remoção torna o grafo desconexo.
- Uma aresta e em um grafo conexo não orientado $G = (V, E)$ é uma **ponte** se sua remoção torna o grafo desconexo

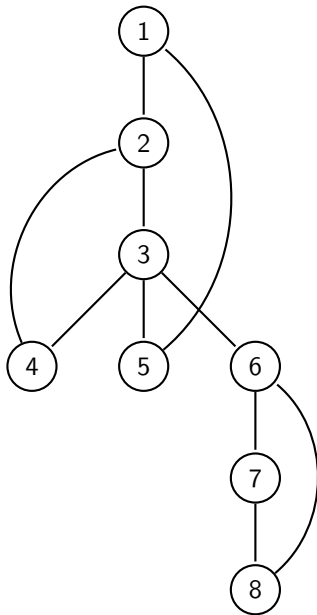


Figura Os vértices 3 e 6 são pontos de articulação e aresta $\{3,6\}$ é uma ponte.

Ponto de articulação

Lemma

Seja v a raiz de G_π , então v é ponto de articulação se e somente se ele tem pelo menos dois filhos.

Demonstração.

Se v tem somente um filho, isso significa que qualquer vértice pode ser alcançado a partir desse filho. Logo, tirando v de G não tornaria desconexo. Se v tem mais de um filho, isso significa que não podemos alcançar todos os vértices a partir de qualquer um dos filhos. Nesse caso, devemos passar por v para ir para vértices em subárvores diferentes. Logo, retirar v torna o G desconexo. □

Ponto de articulação

Lemma

Se v é um ponto de articulação se e somente se v tem um filho s tal que não existe uma aresta de retorno de s ou de qualquer descendente de s para um ancestral próprio de v .

Demonstração.

se v tem um filho s tal que não existe uma aresta de retorno de s ou de qualquer descendente de s para um ancestral próprio de v . Então a retirada de v torna o grafo desconexo, os vértices da sub-árvore de s não alcançam os vértices ancestrais próprio de v . Caso contrário, se retiramos v , um caminho de s ou qualquer descendente de s para um ancestral próprio de v existe usando a aresta de retorno. □

Algorithm 13 Algoritmo DFS_ARTICULACAO

```
1: function DFS_ARTICULACAO( $G$ )
2:   for cada  $v \in V(G)$  do
3:      $cor[v] \leftarrow \text{BRANCO}$ 
4:      $\pi[v] \leftarrow \text{NULL}$ 
5:      $articulacao[v] \leftarrow \text{false}$ 
6:    $altura \leftarrow 0$ 
7:   for cada  $v \in V(G)$  do
8:     if  $cor[v] = \text{BRANCO}$  then
9:        $\text{DFS\_VISIT}(v, altura)$ 
```

Algorithm 14 Algoritmo DFS_VISIT(v , altura)

```
1: function DFS_VISIT( $v$ , altura)
2:    $cor[v] \leftarrow CINZA$ 
3:    $altura \leftarrow altura + 1$ 
4:    $nivel[v] \leftarrow menor[v] \leftarrow altura$ 
5:    $filhos \leftarrow 0$ 
6:   for cada  $u \in Adj[v]$  do
7:     if  $cor[u] = BRANCO$  then
8:        $filhos \leftarrow filhos + 1$ 
9:        $\pi[u] \leftarrow v$ 
10:      DFS_VISIT( $u$ , altura)
11:       $menor[v] \leftarrow \min(menor[v], menor[u])$ 
12:      if  $nivel[v] \neq 1 \wedge menor[u] \geq nivel[v]$  then
13:         $articulacao[v] \leftarrow true$ 
14:      if  $cor[u] = CINZA \wedge \pi[v] \neq u$  then
15:         $menor[v] \leftarrow \min(menor[v], nivel[u])$ 
16:  if  $nivel[v] = 1 \wedge filhos > 1$  then
17:     $articulacao[v] \leftarrow true$ 
18:   $cor[v] \leftarrow PRETO$ 
```

Ponte

- Uma aresta de G é uma ponte se e somente se ela não reside em qualquer ciclo simples de G .
- Uma aresta $e = \{u, v\}$ é uma ponte em G_Π se e somente se $menor[v] = nivel[v]$.

Algorithm 15 Algoritmo DFS_VISIT(v , altura)

```
1: function DFS_VISIT( $v$ , altura)
2:    $cor[v] \leftarrow CINZA$ 
3:    $altura \leftarrow altura + 1$ 
4:    $nivel[v] \leftarrow menor[v] \leftarrow altura$ 
5:    $filhos \leftarrow 0$ 
6:   for cada  $u \in Adj[v]$  do
7:     if  $cor[u] = BRANCO$  then
8:        $filhos \leftarrow filhos + 1$ 
9:        $\pi[u] \leftarrow v$ 
10:      DFS_VISIT( $u$ , altura)
11:       $menor[v] \leftarrow \min(menor[v], menor[u])$ 
12:    if  $cor[u] = CINZA \wedge \pi[v] \neq u$  then
13:       $menor[v] \leftarrow \min(menor[v], nivel[u])$ 
14:  if  $menor[v] == nivel[v] \wedge \pi[v] \neq NULL$  then
15:     $(\pi[v], v)$  é uma ponte
16:   $cor[u] \leftarrow PRETO$ 
```

- 1 Busca em largura
 - Diâmetro de uma árvore
 - Centro de uma árvore
 - Diâmetro de um grafo
- 2 Busca em profundidade
 - Ordenação Topológica
 - Componentes Conexas de grafos não direcionados
 - Grafo direcionado fortemente conexo
 - Componentes Fortemente Conexas em grafos direcionados
 - Ponto de articulação
 - Pontes
- 3 Coloração Gulosa
- 4 Exercício: k-coloração
- 5 Exercício: Número de caminhos
- 6 Exercício: Percurso de Euler em grafos não direcionados
- 7 Exercício: Percurso de Euler em grafos direcionados

Coloração Gulosa

- Uma coloração de um grafo $G = (V, E)$ é um função $cor : V \rightarrow C$, onde C é um conjunto de cores, tal que para cada aresta $\{u, v\} \in E$, tem-se $cor(u) \neq cor(v)$

Coloração Gulosa

- Uma coloração de um grafo $G = (V, E)$ é um função $cor : V \rightarrow C$, onde C é um conjunto de cores, tal que para cada aresta $\{u, v\} \in E$, tem-se $cor(u) \neq cor(v)$
- Uma k -coloração de G é uma coloração que utiliza k cores.

Coloração Gulosa

- Uma coloração de um grafo $G = (V, E)$ é um função $cor : V \rightarrow C$, onde C é um conjunto de cores, tal que para cada aresta $\{u, v\} \in E$, tem-se $cor(u) \neq cor(v)$
- Uma k -coloração de G é uma coloração que utiliza k cores.
- O número cromático de um grafo G , denotado por $\chi(G)$, é o menor número de cores k tal que existe uma k -coloração de G .

Coloração Gulosa

- Uma coloração de um grafo $G = (V, E)$ é um função $cor : V \rightarrow C$, onde C é um conjunto de cores, tal que para cada aresta $\{u, v\} \in E$, tem-se $cor(u) \neq cor(v)$
- Uma k -coloração de G é uma coloração que utiliza k cores.
- O número cromático de um grafo G , denotado por $\chi(G)$, é o menor número de cores k tal que existe uma k -coloração de G .
- Uma coloração gulosa considera os vértices de um grafo em uma ordem específica v_1, v_2, \dots, v_n e atribui a cada vértice a menor cor disponível que não está sendo usada por nenhum vizinho de v_i .

Algorithm 16 Algoritmo GREEDY_COLORING(G , $order$)

```
1: function GREEDY_COLORING( $G$ ,  $order$ )
2:   for  $v \in V$  do
3:      $cor[v] \leftarrow 0$ 
4:    $max\_color \leftarrow 0$ 
5:   for  $i \leftarrow 1$  até  $|V|$  do
6:      $v \leftarrow order[i]$ 
7:     for  $j \leftarrow 1$  até  $max\_color$  do
8:        $disponivel[j] \leftarrow true$ 
9:     for  $u \in Adj[v]$  do
10:      if  $cor[u] \neq 0$  then
11:         $disponivel[cor[u]] \leftarrow false$ 
12:      $k \leftarrow 0$ 
13:     for  $j \leftarrow 1$  até  $max\_color$  do
14:       if  $disponivel[j]$  then
15:          $k \leftarrow j$ 
16:     if  $k = 0$  then
17:        $max\_color \leftarrow max\_color + 1$ 
18:        $cor[v] \leftarrow max\_color$ 
19:     else
20:        $cor[v] \leftarrow k$ 
```

- 1 Busca em largura
 - Diâmetro de uma árvore
 - Centro de uma árvore
 - Diâmetro de um grafo
- 2 Busca em profundidade
 - Ordenação Topológica
 - Componentes Conexas de grafos não direcionados
 - Grafo direcionado fortemente conexo
 - Componentes Fortemente Conexas em grafos direcionados
 - Ponto de articulação
 - Pontes
- 3 Coloração Gulosa
- 4 Exercício: k-coloração
- 5 Exercício: Número de caminhos
- 6 Exercício: Percurso de Euler em grafos não direcionados
- 7 Exercício: Percurso de Euler em grafos direcionados

k-coloração

- 1 Forneça um algoritmo que dado um grafo G , determine se G admite uma k -coloração ou não e mostre todas as k -coloração possíveis.

- 1 Busca em largura
 - Diâmetro de uma árvore
 - Centro de uma árvore
 - Diâmetro de um grafo
- 2 Busca em profundidade
 - Ordenação Topológica
 - Componentes Conexas de grafos não direcionados
 - Grafo direcionado fortemente conexo
 - Componentes Fortemente Conexas em grafos direcionados
 - Ponto de articulação
 - Pontes
- 3 Coloração Gulosa
- 4 Exercício: k-coloração
- 5 Exercício: Número de caminhos
- 6 Exercício: Percurso de Euler em grafos não direcionados
- 7 Exercício: Percurso de Euler em grafos direcionados

Exercícios

- 2 Forneça um algoritmo de tempo linear que tome como entrada um grafo acíclico orientado $G = (V, E)$ e dois vértices s e t , e devolva o número de caminhos de s para t em G . Por exemplo, existem exatamente quatro caminhos do vértice p para o vértice v : pov , $poryu$, $posryv$ e $psryu$. (Seu algoritmo só precisa contar os caminhos, não listá-los.)

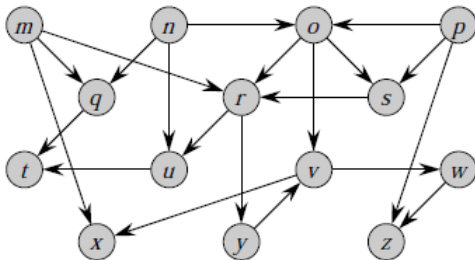


Figura Grafo Direcionado Acíclico

- 1 Busca em largura
 - Diâmetro de uma árvore
 - Centro de uma árvore
 - Diâmetro de um grafo
- 2 Busca em profundidade
 - Ordenação Topológica
 - Componentes Conexas de grafos não direcionados
 - Grafo direcionado fortemente conexo
 - Componentes Fortemente Conexas em grafos direcionados
 - Ponto de articulação
 - Pontes
- 3 Coloração Gulosa
- 4 Exercício: k-coloração
- 5 Exercício: Número de caminhos
- 6 Exercício: Percurso de Euler em grafos não direcionados
- 7 Exercício: Percurso de Euler em grafos direcionados

- 3 Um percurso de Euler de um grafo não direcionado conectado de Euler $G = (V, E)$ é um ciclo que percorre cada aresta de G exatamente uma vez, embora possa alcançar um vértice mais de uma vez.
- 1 Mostre que G tem um percurso de Euler se e somente se o grau(v) é par, para cada vértice $v \in V(G)$.
 - 2 Descreva um algoritmo de tempo $O(E)$ para encontrar um percurso de Euler de G se existir um.

- 1 Busca em largura
 - Diâmetro de uma árvore
 - Centro de uma árvore
 - Diâmetro de um grafo
- 2 Busca em profundidade
 - Ordenação Topológica
 - Componentes Conexas de grafos não direcionados
 - Grafo direcionado fortemente conexo
 - Componentes Fortemente Conexas em grafos direcionados
 - Ponto de articulação
 - Pontes
- 3 Coloração Gulosa
- 4 Exercício: k-coloração
- 5 Exercício: Número de caminhos
- 6 Exercício: Percurso de Euler em grafos não direcionados
- 7 Exercício: Percurso de Euler em grafos direcionados

- 3 Um percurso de Euler de um grafo direcionado conectado de Euler $G = (V, E)$ é um ciclo que percorre cada aresta de G exatamente uma vez, embora possa alcançar um vértice mais de uma vez.
- 1 Mostre que G tem um percurso de Euler se e somente se o $\text{grau_entrada}(v)$ for igual ao $\text{grau_saída}[v]$, para cada vértice $v \in V(G)$.
 - 2 Descreva um algoritmo de tempo $O(E)$ para encontrar um percurso de Euler de G se existir um.