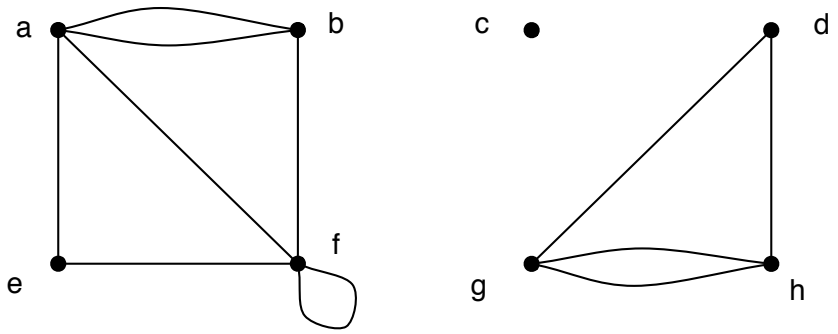
4 Conexidade

A conexidade trata de questões referentes a partes de grafos estarem conectadas ou não. O capítulo inicia com uma contextualização geral sobre o tema, seguida por uma série de definições teóricas iniciais. São então apresentados diversos algoritmos para a avaliação de conexidade de grafos não dirigidos e dirigidos.

4.1 Introdução

A conexidade trata de questões referentes a partes de grafos estarem conectadas ou não. Para ilustrar o assunto, considere o grafo não dirigido $G = (V, E)$ em que $V = \{a, b, c, d, e, f, g, h\}$ e $E = \{(a, b), (a, b), (e, a), (f, a), (b, f), (f, f), (d, g), (h, g), (h, g), (d, h), (e, f)\}$. Uma representação gráfica deste grafo é ilustrada na Figura 4.1. Intuitivamente, podemos observar que existem três partes distintas neste grafo, completamente separadas entre si. Aparentemente isto constituiria três grafos, mas na verdade trata-se de um grafo só (porque ele seria resultado da modelagem de um único problema). A conexidade refere-se a este tipo de situação.

Figura 4.1 – Exemplo de conexidade



Fonte: o autor.

4.2 Definições iniciais

A seguir, veremos algumas definições importantes para compreendermos os problemas de conexidade em grafos dirigidos e não dirigidos, e também os algoritmos que tratam esses problemas.

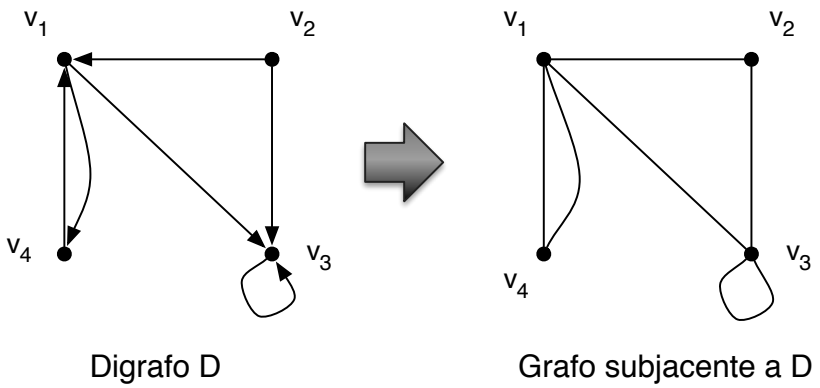
Definição 4.1 (Grafo conexo)**.** *Um grafo G é conexo se, para todos os pares de vértices u e v do grafo, existe pelo menos um caminho de u para v. Caso contrário, o grafo é chamado não conexo.*

Definição 4.2 (Componente conexa)**.** *Uma componente conexa de um grafo G é um subgrafo conexo maximal de G.*

O grafo da Figura 4.1 possui três componentes conexas, que compreendem os seguintes conjuntos de vértices: $C_1 = \{a, b, e, f\}$, $C_2 = \{c\}$ e $C_3 = \{d, g, h\}$

Definição 4.3 (Grafo subjacente)**.** *Dado um digrafo D, seu grafo subjacente é um grafo não dirigido obtido pela substituição de todas as arestas dirigidas de D por arestas não dirigidas. A Figura 4.2 mostra um grafo dirigido $D = (V, E)$ e seu grafo subjacente.*

Figura 4.2 – Exemplo de grafo subjacente a um digrafo



Fonte: o autor.

Definição 4.4 (Digrafo conexo)**.** *Um digrafo D é (fracamente) conexo se seu grafo subjacente for conexo.*

Se pensarmos do ponto de vista de programação de computadores, podemos considerar um grafo como sendo uma variável ou um objeto numa determinada linguagem de programação. Sendo assim, a configuração que resulta da adição ou remoção de um vértice ou aresta de um grafo G pode ser considerado um novo valor para G.

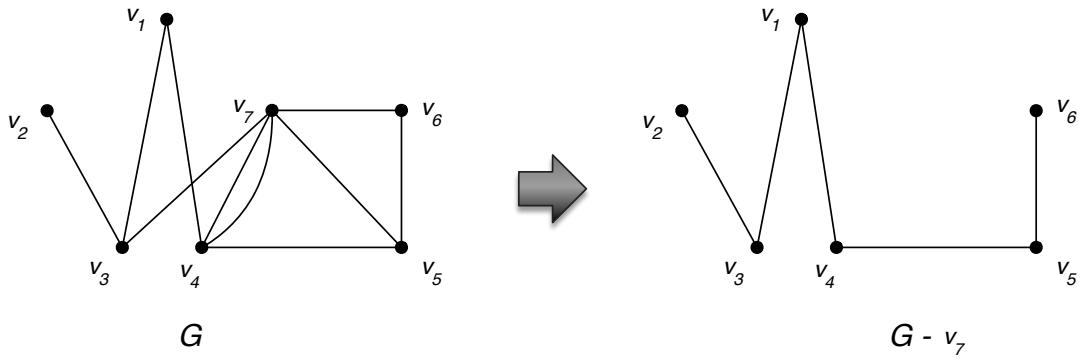
As definições abaixo estabelecem algumas operações simples em grafos e poderiam, por exemplo, serem implementadas em uma classe representativa de grafos.

Definição 4.5 (Subgrafo de remoção de vértice)**.** *Se v é um vértice do grafo G, então o subgrafo de remoção de vértice $G - v$ é o subgrafo induzido pelo conjunto de vértices $V_G - v$, isto é:*

$$V_{G-v} = V_G - v \qquad e \qquad E_{G-v} = \{e \in E_G : v \notin endpts(e)\}$$

Exemplo 4.1. *A Figura 4.3 mostra o resultado da subtração do vértice v_7 do grafo G.*

Figura 4.3 – Exemplo de remoção de vértice

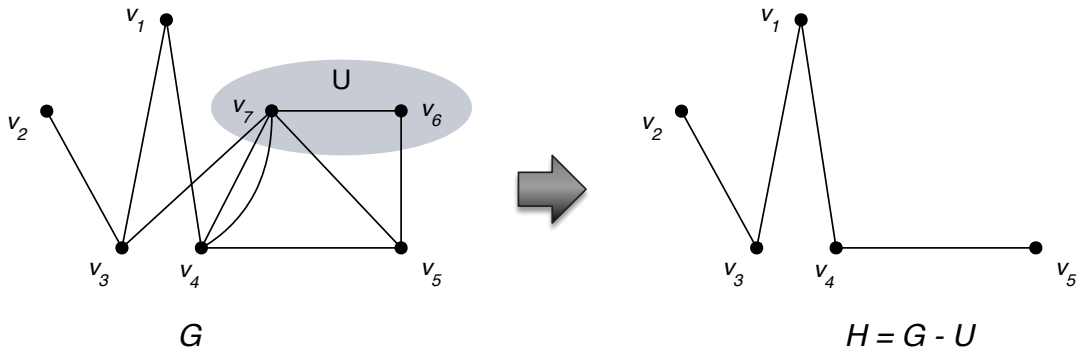


Fonte: o autor.

De forma geral, se tivermos um conjunto U tal que $U \subseteq V_G$, o resultado da aplicação sucessiva da remoção de todos os vértices de U em G é denotado por $G - U$.

Exemplo 4.2. *Na Figura 4.4, temos o grafo U definido por $V_U = \{v_6, v_7\}$ e $E_U = (v_6, v_7)$. Ao aplicarmos a operação geral de subtração de grafos $G - U$, temos como resultado o grafo H.*

Figura 4.4 – Exemplo de subtração de grafos



Fonte: o autor.

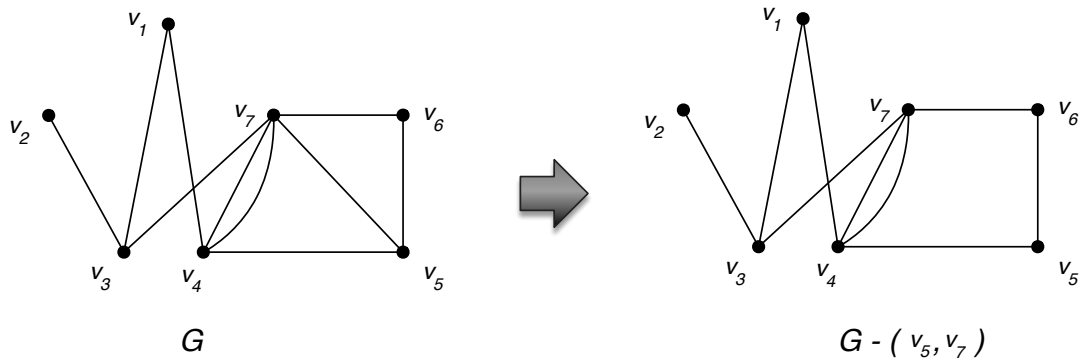
Definição 4.6 (Subgrafo de remoção de aresta)**.** *Se e é uma aresta do grafo G, então o subgrafo de remoção de aresta $G - e$ é o subgrafo induzido pelo conjunto de arestas $E_G - e$, isto é:*

$$V_{G-e} = V_G \qquad e \qquad E_{G-e} = E_G - e$$

De forma geral, se tivermos um conjunto D tal que $D \subseteq E_G$, o resultado da aplicação sucessiva da remoção de todos as arestas de D em G é denotado por $G - D$.

Exemplo 4.3. *Na Figura 4.5, temos um exemplo da operação de remoção da aresta* (v_5, v_7) *do grafo* G*.*

Figura 4.5 – Exemplo de remoção de aresta



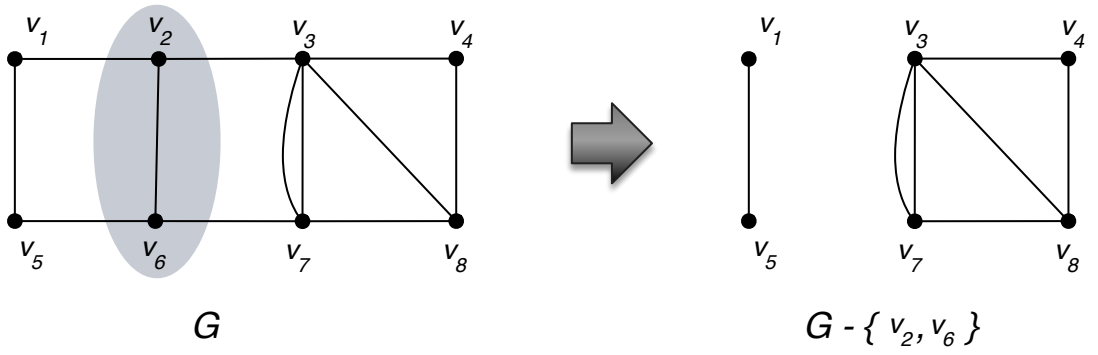
Fonte: o autor.

As definições de subgrafos de remoção de vértices e de arestas para grafos dirigidos são análogas às dos grafos não dirigidos.

Definição 4.7 (Corte de Vértices)**.** *Um corte de vértices em um grafo* G *é é um conjunto de vértices* U *tal que* $G - U$ *tem mais componentes conexas que* G*.*

Exemplo 4.4. *Na Figura 4.6, temos um exemplo de corte de vértices representado pelo subconjunto de vértices* $\{v_2, v_6\}$ *do grafo* G*. Após a operação* $G - \{v_2, v_6\}$ *o grafo inicialmente conexo passa a ter duas componentes conexas. Portanto, o conjunto* $\{v_2, v_6\}$ *é um corte de vértices do grafo* G*. Note que o grafo* G *tem outro corte de vértices, que é o conjunto* $\{v_3, v_7\}$*.*

Figura 4.6 – Exemplo de corte de vértices

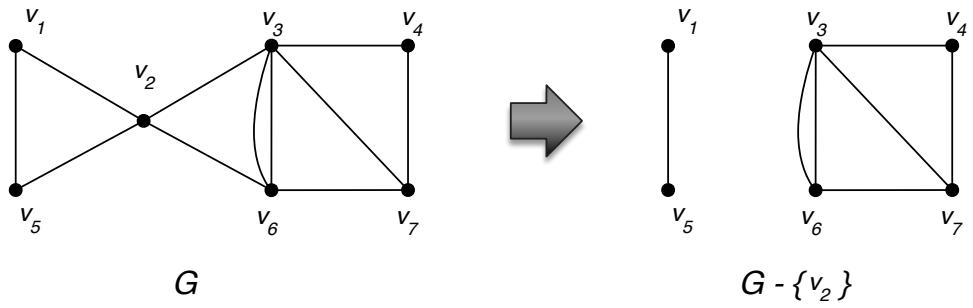


Fonte: o autor.

Definição 4.8 (Vértice de corte)**.** *Um vértice de corte é um corte de vértices constituído por apenas um vértice.*

Exemplo 4.5. *A Figura 4.7 ilustra um exemplo de vértice de corte. Neste caso, a remoção do vértice v_2 do grafo conexo G faz com que ele passe a ter duas componentes conexas.*

Figura 4.7 – Exemplo de vértice de corte



Fonte: o autor.

Definição 4.9 (Corte de arestas)**.** *Um corte de arestas em um grafo G é é um conjunto de arestas D, tal que, $G - D$ possui mais componentes conexas que G.*

Definição 4.10 (Aresta de corte ou "ponte")**.** *Uma aresta de corte ou ponte é um corte de arestas constituído por apenas uma aresta.*

As operações de remoção de vértices e de arestas nos mostram que alguns grafos são "mais conexos" que outros. Por exemplo, alguns grafos podem ser desconectados com a remoção de uma única aresta ou então um vértice. Por outro lado, há outros grafos que permaneceriam conectados a não ser que houvesse remoção de uma quantidade maior de arestas e/ou vértices. Os conceitos de conectividade de vértices e conectividade de arestas nos trazem parâmetros numéricos que indicam o grau de "conectividade" dos grafos. Estes conceitos são úteis para se avaliar o nível de vulnerabilidade de redes de informação, malhas rodoviárias ou qualquer outro tipo de rede de comunicação ou transporte.

Definição 4.11 (Conectividade de vértices)**.** *A conectividade de vértices de um grafo G, denotada por $\kappa(G)$ é o número mínimo de vértices cuja remoção pode desconectar G ou reduzi-lo a um grafo com apenas um vértice.*

Portanto, se G possuir pelo menos um par de vértices não-adjacentes, então $\kappa(G)$ é igual ao tamanho de seu menor corte de vértices.

Definição 4.12 (k-conexão)**.** *Um grafo G é k-conexo (ou k-conectado) se G é conexo e $\kappa_v(G) \geq k$. Se G possui vértices não adjacentes, então G é k-conexo se cada um de seus cortes de vértices tiver pelo menos k vértices.*

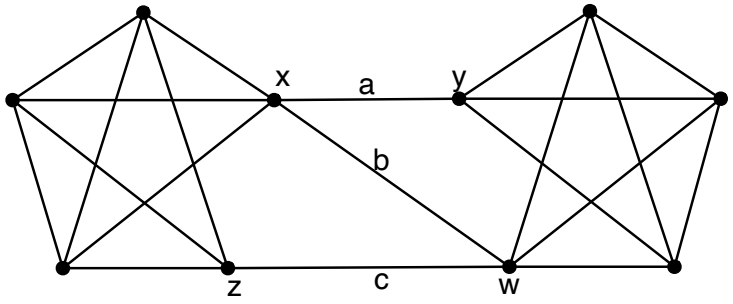
Definição 4.13 (Conectividade de arestas)**.** *A conectividade de arestas de um grafo G, denotada por $\gamma(G)$ é o número mínio de arestas cuja remoção pode desconectar G.*

Portanto, se G for um grafo conexo, então $\gamma(G)$ é igual ao tamanho de seu menor corte de arestas.

Definição 4.14 (k-conexão-de-arestas)**.** *Um grafo G é k-conexo-de-arestas (ou k-conectado-de-arestas) se G é conexo e se cada um de seus cortes de aresta tem no mínimo k arestas (isto é, $\gamma(G) \geq k$).*

Exemplo 4.6. *No grafo G da Figura 4.8, o conjunto de vértices $\{x, z\}$ é um dos seus três cortes de vértices com dois elementos. Neste grafo também é relativamente fácil perceber que não há vértice de corte. Portanto, $\kappa(G) = 2$. O conjunto de arestas $\{a, b, c\}$ é o único corte de arestas com 3 elementos no grafo G, e não há corte de arestas com menos de 3 elementos. Portanto, $\gamma(G) = 3$.*

Figura 4.8 – Grafo G com $\kappa(G) = 2$ e $\gamma(G) = 3$



Fonte: o autor.

4.3 Conexidade com algoritmo de busca

Um problema comum é avaliar se determinado grafo é conexo e, não sendo, determinar a sua quantidade de componentes conexas. Pela definição 4.1, podemos inferir uma forma de fazer isso por meio do uso de algoritmos de busca, visto que sua principal função é encontrar caminhos nos grafos. Esta solução foi descrita por Hopcroft e Tarjan em 1973 (HOPCROFT; TARJAN, 1973).

Caso se queira utilizar o algoritmo de busca em profundidade, por exemplo, basta adaptá-lo para contabilizar a quantidade de buscas recursivas que são executadas. Cada uma delas corresponde a uma componente conexa no grafo.

O algoritmo 4.1 mostra o pseudocódigo desta solução. Da mesma forma que no algoritmo anterior, a variável N_{cc} é um valor inteiro que armazena a quantidade de

componentes conexas do grafo. Consequentemente, no vetor de roteamento construído pelo algoritmo, cada árvore corresponde a uma componente conexa. A função $VisitaVertice(v)$ pode ser vista na subseção 3.3.1.

Algoritmo: ConexidadeBuscaProfundidade(G)

para cada $v \in V$ **faça**
  $estado(v) \leftarrow NAO\_VISITADO$;
  $\rho(v) \leftarrow nil$;

$tempo \leftarrow 0$;
$N_{cc} \leftarrow 0$;

para cada *vértice* $v \in V$ **faça**
  se $cor(v) = NAO\_VISITADO$ **então**
    $N_{cc} \leftarrow N_{cc} + 1$;
    $VisitaVertice(v)$;

se $N_{cc} > 1$ **então**
  imprima("Grafo não conexo, com N_{cc} componentes conexas")
senão
  imprima("Grafo conexo")

Algoritmo 4.1: Avaliação de conexidade com busca em profundidade

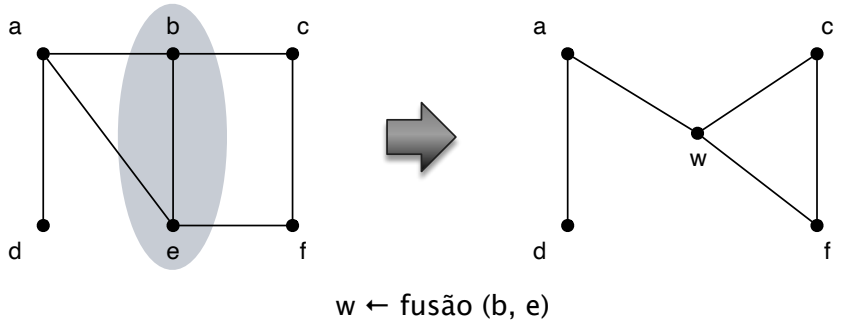
É possível fazer o mesmo tipo de adaptação no algoritmo de busca em largura. Com isso, já temos dois algoritmos para avaliar conexidade em grafos não-dirigidos. A seguir, veremos uma terceira opção utilizando estruturas de dados para conjuntos disjuntos de vértices.

4.4 Conexidade por fusão de vértices

A avaliação de conexidade pode ser feita pelo algoritmo apresentado por Goodman e Hedetniemi (GOODMAN; HEDETNIEMI, 1977). A ideia básica do algoritmo é reduzir cada componente conexa a um único vértice por meio da operação de fusão de vértices[1]. Dados dois vértices adjacentes u e v a fusão é realizada pela remoção da(s) aresta(s) (u, v) e dos vértices u e v. É criado um novo vértice w que deve ser adjacente a todos os demais vértices inicialmente adjacentes a u e v. A Figura 4.9 ilustra esta operação.

[1] Esta operação também é conhecida como *contração de aresta*.

Figura 4.9 – Operação de fusão de vértices



Fonte: o autor.

O algoritmo seleciona um vértice arbitrário e vai "colapsando" o grafo com sucessivas fusões de vértice enquanto houver vértice adjacente disponível para a operação. A partir daí, contabiliza-se uma componente conexa e o procedimento é iniciado novamente a partir de um novo vértice arbitrário, se houver. O pseudocódigo apresentado tem os seguintes elementos básicos:

- w: vértice arbitrário;
- H: cópia do grafo original G. Isto é necessário para preservar o grafo original, porque supõe-se que as operações de fusão destroem o grafo;
- N_{cc}: número de componentes conexas do grafo.

Algoritmo: ConexidadeFusao(G)

```
// Inicialização
H ← G
N_cc ← 0
// Fusão sucessiva de vértices
enquanto H ≠ ∅ faça
    Selecione um vértice w ∈ H
    enquanto w for adjacente a algum vértice v ∈ H faça
        w ← fusao(w, v)
    // Remova o vértice w do grafo
    H ← H − w
    // Contabilize uma componente conexa
    N_cc ← N_cc + 1
// Avaliacao da conexidade
se N_cc > 1 então
    imprima("Grafo não conexo, com N_cc componentes conexas")
senão
    imprima("Grafo conexo")
```

Algoritmo 4.2: Avaliação de conexidade por fusão de vértices

4.5 Conexidade por conjuntos disjuntos

Em Cormen et al. (2001), é apresentada uma solução para avaliação de conexidade baseada em estruturas de dados de conjuntos disjuntos de vértices, considerando que eles são representados por objetos em linguagem de programação. Conjuntos disjuntos são aqueles que não compartilham elementos, isto é, dados k conjuntos $A_1, A_2, \ldots, A_k$ tem se que a intersecção entre todos eles é um conjunto vazio: $A_1 \cap A_2 \cap \cdots \cap A_k = \emptyset$.

Há diferentes formas de implementar este tipo de estrutura de dados, uma delas é usando listas encadeadas para representar os conjuntos. Entretanto, vamos apresentar uma implementação que armazena conjuntos em árvores utilizando referências para nós predecessores, de forma semelhante ao que fizemos com vetores de roteamento no Capítulo 3. Cada árvore representa um conjunto e cada nó representa um elemento do respectivo conjunto. Para utilizar a estrutura na avaliação de conexidade, é necessário implementar as seguintes operações:

- $CriaConjunto(v)$: esta operação deve criar um conjunto com o vértice v e atribuir um identificador único para este conjunto, que pode ser uma referência para o próprio objeto vértice, pois sabemos que os vértices são únicos e v não estará em nenhum outro conjunto;
- $BuscaConjunto(v)$: retorna uma referência para o identificador do conjunto que contém o vértice v;
- $Uniao(u, v)$: cria um novo conjunto por meio da união dos conjuntos que contém os vértices u e v, e atribui um identificador ao conjunto resultante.

Se estas operações forem implementadas com listas encadeadas para armazenar os conjuntos, a operação $CriaConjunto(v)$ pode ser implementada em tempo $O(1)$, porém a operação $BuscaConjunto(v)$ leva tempo $O(n)$ (sendo n a quantidade de vértices do grafo), visto que precisa percorrer linearmente as listas de cada conjunto até encontrar o vértice v e retornar o identificador do conjunto correspondente. A união de duas listas de vértices $Uniao(u, v)$ pode ser feita em tempo constante, porém, como esta operação precisa chamar $BuscaConjunto(v)$ duas vezes, seu desempenho acaba sendo também linear na quantidade de vértices do grafo: O(n). Para este tipo de implementação, cada vértice v possui dois atributos:

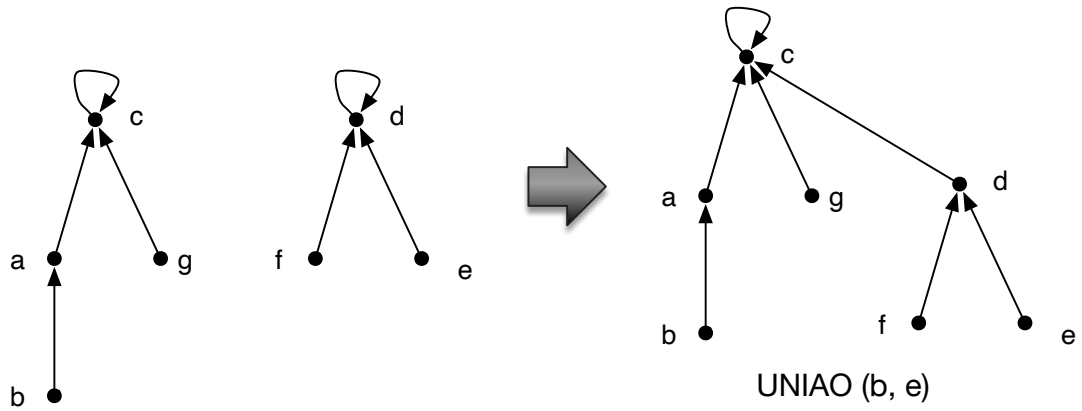
- $\rho(v)$: referência para o vértice pai de v na árvore que armazena um conjunto de vértices;
- $peso(v)$: o peso do vértice é um atributo numérico referente ao nível do vértice na árvore. Vértices folha possuem peso 0, e o vértice raiz vai ter como valor de peso a própria profundidade da árvore.

Em relação a uma implementação com listas encadeadas, a estrutura usando árvores se torna mais eficiente. A operação $CriaConjunto(v)$ leva tempo constante $O(1)$ ao criar um conjunto atribuindo v a seu próprio predecessor.

A operação $BuscaConjunto(v)$, por sua vez, precisa "subir" a árvore a partir de v em direção ao nó raiz e retornar a sua referência. Esta operação é linear na profundidade da árvore. Como veremos a seguir, o pseudocódigo contém uma heurística para compressão das árvores, reduzindo sua profundidade a cada operação de união de conjuntos. Isto é feito por meio do atributo *peso* dos vértices das duas árvores a serem unidas.

A operação $Uniao(u, v)$ pode ser feita atribuindo-se como predecessor de um vértice raiz o outro vértice raiz, conforme ilustrado na Figura 4.10. Aqui, as operações $BuscaConjunto(b)$ e $BuscaConjunto(e)$, retornam, respectivamente, referências para c e d, enquanto a operação $Uniao(b, e)$ forma um novo conjunto ao atribuir c como predecessor de d.

Figura 4.10 – Exemplo conjuntos disjuntos com árvores



Fonte: o autor.

A seguir, temos os pseudocódigos destas operações. A operação $CriaConjunto(v)$ forma um conjunto com apenas um objeto vértice simplesmente inicializando os seus atributos $\rho(v)$ e $peso(v)$. Ao aplicarmos esta operação para os n vértices de um grafo, criamos uma **floresta de árvores de conjuntos disjuntos**.

Algoritmo: $CriaConjunto(v)$

$\rho(v) \leftarrow v$
$peso(v) \leftarrow 0$

Algoritmo 4.3: Operação $CriaConjunto(v)$

A operação $BuscaConjunto(v)$ é um procedimento recursivo que inicia do vértice v e gradativamente vai subindo os nós até chegar ao nó raiz, o qual é retornado. Note

que, ao atribuir o retorno da chamada recursiva ao predecessor do vértice, o método promove uma adaptação da árvore, tornando-a menos profunda. Portanto, quanto mais este procedimento for chamado, mais otimizada fica a estrutura para executar também a união de dois conjuntos.

Algoritmo: $BuscaConjunto(v)$

se $v \neq \rho(v)$ **então**
  $\rho(v) \leftarrow BuscaConjunto(\rho(v))$
retorna $\rho(v)$

Algoritmo 4.4: Operação $BuscaConjunto(v)$

A operação de união usa um procedimento auxiliar $Link(u, v)$ para efetivamente conectar as duas árvores. Se as árvores forem de profundidades diferentes, a árvore resultante menos profunda será aquela que mantiver como raiz o nó raiz da árvore original mais profunda. Esta heurística é implementada pelo procedimento $Link(u, v)$.

Algoritmo: $Uniao(u, v)$

$Link(BuscaConjunto(u), BuscaConjunto(v))$

Algoritmo 4.5: Operação $Uniao(u, v)$

O procedimento $Link(u, v)$ recebe a raiz de duas árvores e escolhe qual das duas vai se manter como raiz da árvore resultante. Isto é feito por meio da observação do atributo *peso* de cada um dos vértices raiz. A raiz com menor peso deve se tornar filha da raiz com peso original maior.

Algoritmo: $Link(u, v)$

se $peso(u) > peso(v)$ **então**
  $\rho(v) \leftarrow u$
senão
  $\rho(u) \leftarrow v$
  se $peso(u) = peso(v)$ **então**
    $peso(v) \leftarrow peso(v) + 1$

Algoritmo 4.6: Operação $Link(u, v)$

A utilização da estrutura de conjuntos disjuntos de vértices permita que utilizemos o algoritmo a seguir para construir uma floresta, na qual cada árvore representa uma componente conexa do grafo. O procedimento começa com n conjuntos de um vértice e vai verificando aresta por aresta para promover uniões dos conjuntos caso os dois vértices incidentes a uma aresta estejam em conjuntos diferentes.

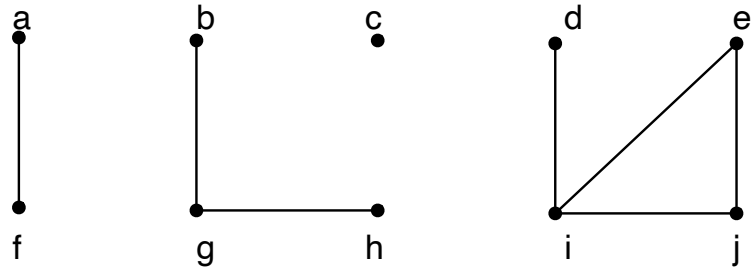
Algoritmo: $ComponentesConexasConjuntos(G)$

para *cada vértice* $v \in V$ **faça**
  CriaConjunto(v)
para *cada aresta* $(u, v) \in E$ **faça**
  se $BuscaConjunto(u) \neq BuscaConjunto(v)$ **então**
    $Uniao(u, v)$

Algoritmo 4.7: Algoritmo $ComponentesConexasConjuntos(G)$

A Figura 4.11 ilustra passo a passo a configuração dos conjuntos de vértices à medida que o algoritmo é executado no grafo mostrado.

Figura 4.11 – Exemplo de formação de conjuntos disjuntos



aresta	Coleção de conjuntos disjuntos									
inicio	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e, j)	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e, j\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	
(h, g)	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e, j\}$	$\{f\}$		$\{h, g\}$	$\{i\}$	
(f, a)	$\{a, f\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e, j\}$			$\{h, g\}$	$\{i\}$	
(d, i)	$\{a, f\}$	$\{b\}$	$\{c\}$	$\{d, i\}$	$\{e, j\}$			$\{h, g\}$		
(b, g)	$\{a, f\}$		$\{c\}$	$\{d, i\}$	$\{e, j\}$			$\{h, g, b\}$		
(i, j)	$\{a, f\}$		$\{c\}$		$\{e, j, d, i\}$			$\{h, g, b\}$		
(e, i)	$\{a, f\}$		$\{c\}$		$\{e, j, d, i\}$			$\{h, g, b\}$		

Fonte: o autor.

Após a execução do algoritmo, pode-se investigar a configuração das componentes conexas por meio da estrutura de conjuntos. O procedimento $MesmaComponente(u, v)$ ilustra uma forma de se consultar a estrutura.

Algoritmo: $MesmaComponente(u, v)$

se $BuscaConjunto(u) = BuscaConjunto(v)$ **então**
  retorna *Verdadeiro*
senão
  retorna *Falso*

Algoritmo 4.8: Algoritmo $MesmaComponente(u, v)$

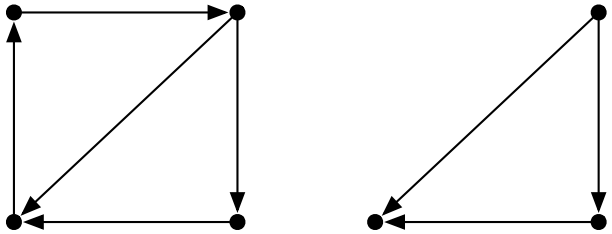
4.6 Conexidade em grafos dirigidos

Em grafos dirigidos, o conceito de conexidade também está relacionado à existência de caminhos entre vértices. Por exemplo, em termos gerais avaliaremos questões tais como se há "livre circulação" em determinadas regiões do grafo dirigido. Ou seja, se sairmos de um determinado vértice u e atingirmos um vértice v, é possível retornar a u, considerando os sentidos das arestas do grafo? Para tal avaliação, usaremos as definições a seguir, para a construção de um algoritmo baseado na Busca em Profundidade,

Definição 4.15 (Componente fortemente conexa)**.** *Uma componente fortemente conexa de um digrafo $G = (V, E)$ é um conjunto maximal de vértices $C \subseteq V$, tal que, para cada par de vértices u e v em C, existe caminho do vértice u para o vértice v e vice-versa (de v para u).*

Definição 4.16 (Grafo fortemente conexo)**.** *Um grafo dirigido é fortemente conexo se existir um caminho direto de cada vértice para cada vértice. A Figura 4.12 mostra um grafo fortemente conexo e um fracamente conexo, respectivamente.*

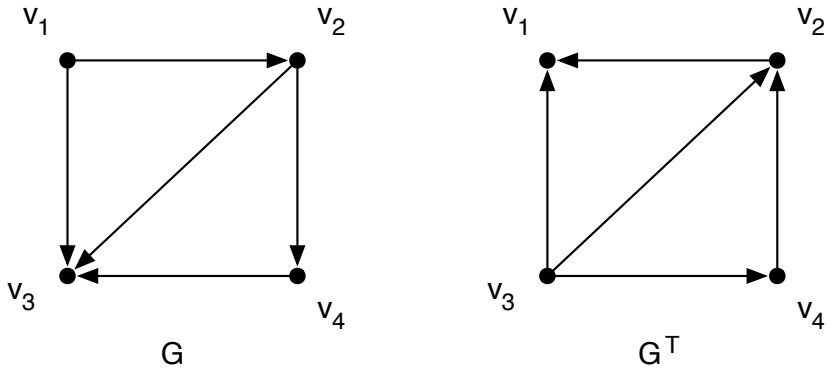
Figura 4.12 – Exemplo de grafos dirigidos forte e fracamente conexos



Fonte: o autor.

Definição 4.17 (Grafo transposto)**.** *Dado um grafo dirigido G=(V,E), seu grafo transposto é definido por $G^T = (V, E^T)$, onde $E^T = \{(u, v) : (v, u) \in E\}$. Em outras palavras, para encontrar um grafo transposto de um determinado grafo dirigido, basta que se inverta o sentido de todas as suas arestas. A Figura 4.13 mostra um grafo dirigido G e seu grafo transposto G^T.*

Figura 4.13 – Exemplo de um grafo dirigido e seu grafo transposto



Fonte: o autor.

Utilizando a busca em profundidade, o algoritmo a seguir, adaptado de Aho, Hopcroft e Ullman (1983), encontra as componentes fortemente conexas de grafos dirigidos:

Entrada: um grafo dirigido G.

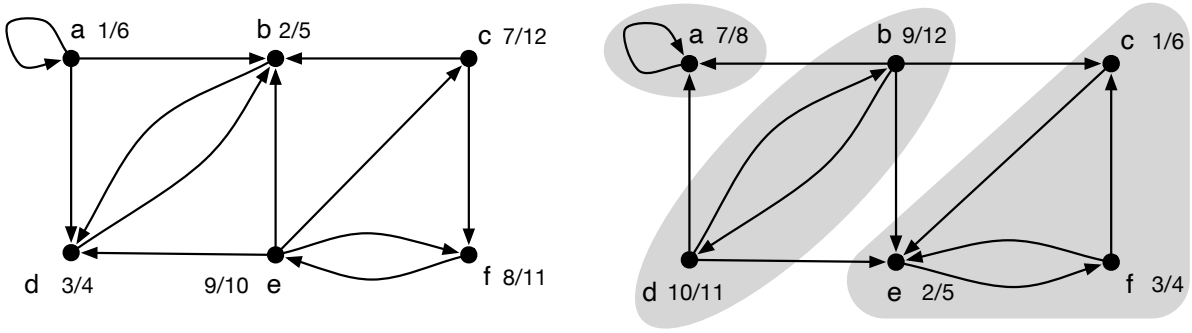
Passos:

1. Execute a busca em profundidade no grafo $BuscaProfTodos(G)$, calculando o tempo de fechamento $f(v)$ para cada um dos seus vértices;
2. Construa o grafo transposto G^T;
3. Execute a busca em profundidade no grafo transposto $BuscaProfTodos(G^T)$, porém eu seu laço principal chame $VisitaVertice(v)$ considerando a ordem decrescente de tempos de encerramento $t_e(v)$ calculados no passo 1.
4. Os vértices de cada uma das árvores de busca geradas no passo anterior formam cada uma das componentes fortemente conexas do grafo original G.

Saída: floresta de árvores de busca, em que cada árvore é uma componente fortemente conexa.

A Figura 4.14a mostra o resultado de uma busca em profundidade em um grafo dirigido G com os tempos de abertura e fechamento mostrados próximos aos vértices. A Figura 4.14b mostra o grafo transposto G^T juntamente com as três componentes fortemente conexas agrupadas pelas regiões cinza.

Figura 4.14 – Obtenção de componentes fortemente conexas

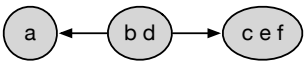


(a) Grafo G com tempos $d(v)/f(v)$ (b) Grafo G^T com as comp. fortemente conexas

Fonte: o autor.

Um resultado adicional que podemos obter é o grafo acíclico dirigido de componentes fortemente conexas mostrado na Figura 4.15. Este grafo é obtido pela contração de todas as arestas dentro de cada componente fortemente conexa de G de forma que acaba restando apenas um único vértice para cada componente fortemente conexa.

Figura 4.15 – grafo acíclico dirigido de componentes fortemente conexas



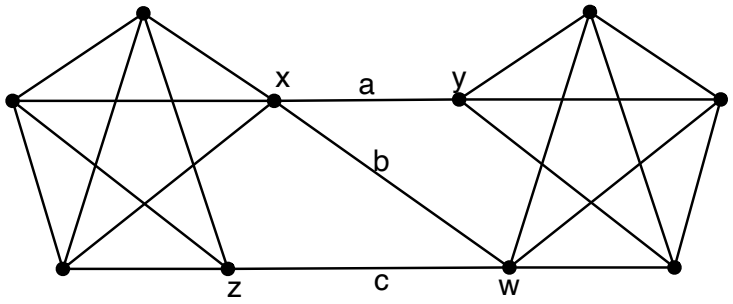
Fonte: o autor.

4.7 Exercícios

1. (PosComp – 2002) O menor número de arestas em um grafo conexo com n vértices é:

   a) 1 b) $n/2$ c) $n-1$ d) n e) n^2

2. Dado o grafo abaixo, encontre dois cortes de vértices com dois vértices cada (Obs.: rotule o grafo, caso seja necessário).



Fonte: o autor.

3. Para cada um dos itens abaixo, desenhe um grafo que atenda às especificações, ou então explique porque tal grafo não existe.

   a) Um grafo G com 6 vértices tal que $\kappa(G) = 2$ e $\gamma(G) = 2$;
   b) Um grafo conexo com 11 vértices, 10 arestas e sem vértice de corte;
   c) Um grafo 3-conexo com exatamente uma ponte;
   d) Um grafo 2-conexo com 8 vértices e exatamente duas pontes;

4. Determine $\kappa(K_{r,s})$ e $\gamma(K_{r,s})$, sendo $K_{r,s}$ um grafo bipartido completo.

5. Sendo $\delta(G)$ o grau do vértice de menor grau de um grafo G, mostre um exemplo de grafo G, quando possível, tal que:

   a) $\kappa(G) = 2$, e $\gamma(G) = 3$ e $\delta(G) = 4$;
   b) $\kappa(G) = 3$, e $\gamma(G) = 2$ e $\delta(G) = 4$;
   c) $\kappa(G) = 2$, e $\gamma(G) = 2$ e $\delta(G) = 4$.

6. No grafo de Petersen, encontre:

   a) um corte de arestas com 3 arestas;
   b) um corte de arestas com 4 arestas;
   c) um corte de arestas com 5 arestas;
   d) um corte de arestas com 6 arestas.

7. (PosComp – 2008) Denomina-se complemento de um grafo $G(V, E)$ o grafo H que tem o conjunto de vértices igual ao de G e tal que, para todo par de vértices distintos v, w em V, temos que a aresta (v, w) é aresta de G se e somente se (v, w) não é aresta de H.

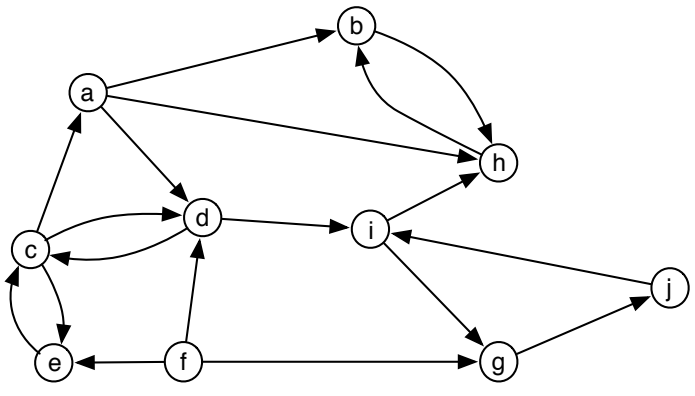
   A esse respeito, assinale a afirmativa CORRETA.

   a) G e H são grafos isomorfos.
   b) Se o grafo G é conexo, então H é conexo.
   c) Se o grafo G não é conexo, então H é conexo.
   d) Se o grafo G não é conexo, então H não é conexo.
   e) Os grafos G e H têm o mesmo número de componentes conexas.

8. Dada a seguinte matriz de adjacência, encontre as componentes fortemente conexas do respectivo dígrafo, utilizando o algoritmo baseado em busca em profundidade.

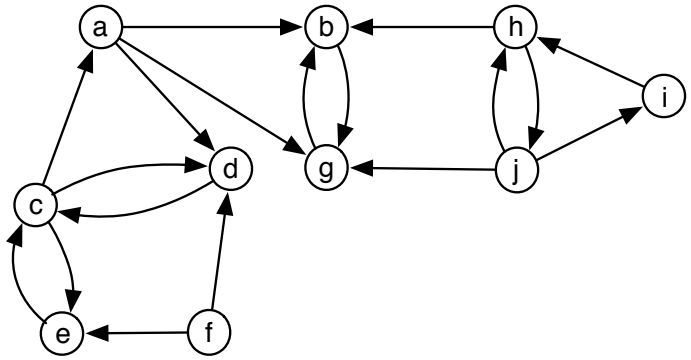
$$[A] = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

9. Dados os dígrafos (a), (b) e (c) abaixo, encontre suas componentes fortemente conexas, mostrando passo a passo o desenvolvimento da solução do problema. Sugestão: utilizar o algoritmo baseado na geração de árvores de busca em profundidade e anotar os tempos de abertura e fechamento próximos aos vértices.

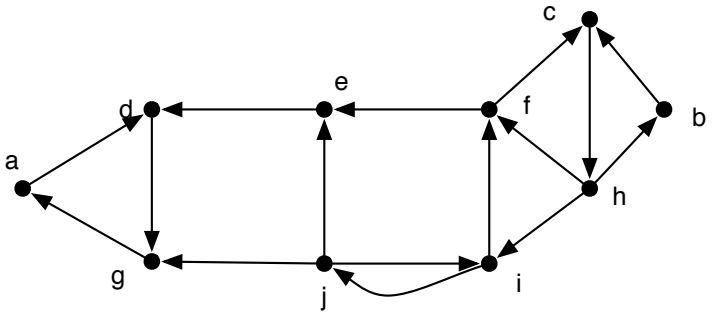


a)

b)



c)



10. Se uma nova aresta for inserida em um dígrafo, como podem mudar as sua componentes fortemente conexas?

11. Desenhe um grafo conexo com 8 vértices com a menor quantidade possível de arestas.

12. Desenhe um grafo conexo com 7 vértices de tal forma que a remoção de qualquer uma de suas arestas resulta em um grafo não conexo.

13. Desenhe um grafo conexo com 5 vértices que permaneça conexo após a remoção de duas arestas quaisquer.