2 Representação de grafos

Agora que temos um conhecimento geral dos principais conceitos e definições sobre grafos, partiremos para o desenvolvimento de estruturas de dados para implementação computacional de grafos. O tema é bastante amplo e detalhes de implementação podem variar dependendo do tipo de grafo ou do contexto de aplicação. Entretanto, longe de esgotar a discussão sobre o assunto, este capítulo pretende mostrar questões gerais básicas que podem ser úteis para a maior parte das implementações computacionais para solucionar problemas envolvendo grafos.

Começamos com uma contextualização básica e enumeramos as operações mais comuns que uma estrutura de dados de grafos deveria fornecer. São mostrados quatro tipos básicos de estruturas de dados: listas de arestas, listas de adjacência, mapas de adjacência e matrizes de adjacência. Por fim, propõe-se alguns exercícios de implementação.

2.1 Introdução

Até o presente momento, foram apresentadas duas formas de representação de grafos: a representação por definição dos conjuntos de vértices e arestas (conjuntos V e E) e a representação gráfica. Estas duas formas de representação são úteis para o estudo teórico de assuntos relacionados a grafos, mas são inadequadas para a descrição e implementação de algoritmos. Para tal, veremos neste capítulo outras quatro formas de representação em termos de estruturas de dados: as listas de arestas, as listas de adjacência, os mapas de adjacência e as matrizes de adjacência.

Para iniciarmos a discussão sobre implementação de tais estruturas, precisamos estabelecer as principais entidades envolvidas. Temos então vértices, arestas e grafos. Considerando o uso de linguagens orientadas a objetos, tais entidades poderiam ser implementadas como classes, por exemplo.

Em todas as quatro estruturas devemos manter uma coleção de vértices, os quais são objetos que contém referências para dados relevantes do problema modelado (por exemplo, os identificadores dos vértices). Por outro lado, as estruturas diferem bastante na forma com a qual tratam as arestas. Consequentemente, o desempenho das diferentes operações sobre a estrutura de dados também varia de acordo com o tipo de estrutura.

A implementação de algoritmos em grafos envolve uma série de operações sobre a estrutura de dados. As operações incluem consultas, criação e remoção de elementos dos grafos. Algumas das operações mais comuns são: obter a ordem e o tamanho do grafo, retornar uma coleção de todos os vértices ou de todas as arestas do grafo, retornar uma

coleção de todas as arestas incidentes a um vértice em grafo não dirigido, retornar uma coleção de arestas de saída ou de chegada em um vértice de grafo dirigido, retornar o grau de um vértice (ou grau de entrada e saída em grafos dirigidos), inserir e remover vértices e arestas, entre outras.

Antes de vermos em maiores detalhes cada uma dessas estruturas, vamos definir as principais operações que elas precisam implementar.

2.2 Operações em estruturas de dados de Grafos

Sendo os grafos definidos como conjuntos de vértices e arestas, é natural que implementações computacionais modelem esta abstração como três tipos de dados fundamentais: Vértices, Arestas e Grafos (e/ou Digrafos a depender de decisões de projeto). Vértices são os tipos de dados mais elementares, geralmente contendo pelo menos referência para informações adicionais vinculadas e dependentes do contexto de aplicação. O vértice, por exemplo, pode estar associado a um identificador ou um determinado código referente aos dados do problema modelado etc. Uma aresta também pode estar associada a um objeto contendo informações oriundas do problema modelado, tais como: número de voo, distância, custo, códigos identificadores diversos, entre outras.

A seguir, definimos uma série de operações comuns em algoritmos que operam em grafos. Dependendo do contexto pode-se implementar todas elas para que se tenha uma estrutura dados abstrata para grafos, ou então apenas parte delas de forma a implementar algum algoritmo específico. O Quadro 1 apresenta as operações e suas descrições. É importante frisar que não estão especificados em detalhes os tipos de dados de parâmetros de entrada e valores de retorno, podendo estes serem definidos em função de requisitos específicos de cada aplicação.

2.3 Listas de arestas

Esta estrutura de dados é, provavelmente, a mais simples de implementar, por outro lado é a menos eficiente. Sua principal característica do ponto de vista estrutural é manter uma lista não ordenada de arestas. Trata-se de uma estrutura enxuta, porém a operação de localização de uma aresta em particular não é eficiente. A localização de todas as arestas incidentes a um vértice também é ineficiente.

A Figura 2.1 mostra esquematicamente a organização dos objetos vértices e arestas na memória. Temos uma coleção iterável não-ordenada V contendo referências para os vértices e outra coleção iterável não-ordenada E contendo referências para os objetos aresta. As referências estão representadas como setas na figura. Estas coleções podem ser listas duplamente encadeadas. Cada aresta tem referências para os vértices aos quais

Quadro 1 – Operações mais comuns em estruturas de dados de Grafos

Operação	Descrição
getOrdem( )	retorna quantidade de vértices
getTamanho( )	retorna quantidade de arestas
vertices( )	retorna uma iteração de todos os vértices do grafo
arestas( )	retorna uma iteração de todas as arestas do grafo
insereV( )	instancia um novo vértice e o adiciona ao grafo
removeV(v)	remove o vértice v e todas as suas arestas incidentes
insereA(u, v)	instancia uma nova aresta incidente aos vértices u e v, e a adiciona ao grafo
removeA(e)	remove a aresta e
adj(v)	retorna uma iteração com todos os vértices adjacentes ao vértice u
getA(u, v)	retorna uma referência para a aresta de u para v ou null se os vértices não forem adjacentes. Para grafos não dirigidos, getA(u, v) e getA(v, u) produzem o mesmo resultado
grauE(v)	retorna o grau de entrada do vértice v em grafos dirigidos
grauS(v)	retorna o grau de saída do vértice v em grafos dirigidos
grau(v)	retorna o grau do vértice v em grafos não dirigidos. Alternativamente, pode-se usar os dois métodos anteriores em grafos não dirigidos de forma que grauE(v) e grauS(v) retornem o mesmo resultado
verticesA(e)	retorna o par de vértices que conectados à aresta e. Se o grafo for dirigido, o primeiro vértice do par é a origem e o segundo é o destino da aresta
oposto(v,e)	para um vértice v incidente à aresta e, retorna o outro vértice incidente à aresta
arestasE(v)	retorna uma iteração de todas as arestas de entrada do vértice v
arestasS(v)	retorna uma iteração de todas as arestas de saída do vértice v

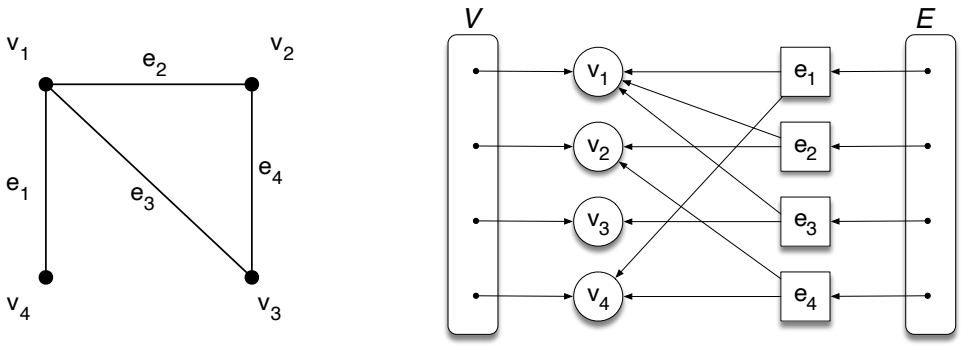
Fonte: Autor.

ela é incidente. Isto permite que as operações verticesA(e) e oposto(v,e) sejam executadas em tempo constante. Note que os vértices não dispõem de referências para suas arestas incidentes.

Considerando n e m, respectivamente, a ordem e o tamanho do grafo, a estrutura de listas de arestas consome espaço de memória $O(m + n)$, pois cada vértice ou aresta consome espaço $O(1)$ e as listas V e E usam espaço linearmente proporcional às suas quantidades de elementos. Tendo em vista que as operações vertices() e arestas() iteram sobre as listas V e E, elas têm desempenho $O(n)$ e $O(m)$, respectivamente.

A maior desvantagem desta estrutura decorre do fato de não termos referências das arestas nos objetos vértice. Com isso, as operações getA(u, v), grau(v), grauE(v), grauS(v), arestasE(v) e arestasS(v) têm desempenho $O(m)$, pois demandam uma inspeção exaustiva de

Figura 2.1 – Representação por lista de arestas de um grafo simples



Fonte: adaptado de Goodrich, Tamassia e Goldwasser (2013).

todas as arestas da lista E.

Quanto às operações de alteração do grafo, a adição de novos vértices e arestas leva tempo $O(1)$ A remoção de uma aresta pode ser realizada em $O(m)$ devido à necessidade de se percorrer a lista E para fazer a remoção do elemento correspondente. A operação removeV(v) também é $O(m)$ porque a remoção de um vértice implica na remoção de todas as suas arestas incidentes.

2.4 Listas de adjacência

A estrutura de listas de adjacência de um grafo $G = (V, E)$ consiste em uma coleção não-ordenada iterável V de vértices. Cada vértice, por sua vez, deve manter uma lista não ordenada de todas as suas arestas incidentes. Isto permite que a determinação das arestas incidentes a um vértice seja mais eficiente. Dado um vértice v, sua lista de incidência é denotada por $I(v)$.

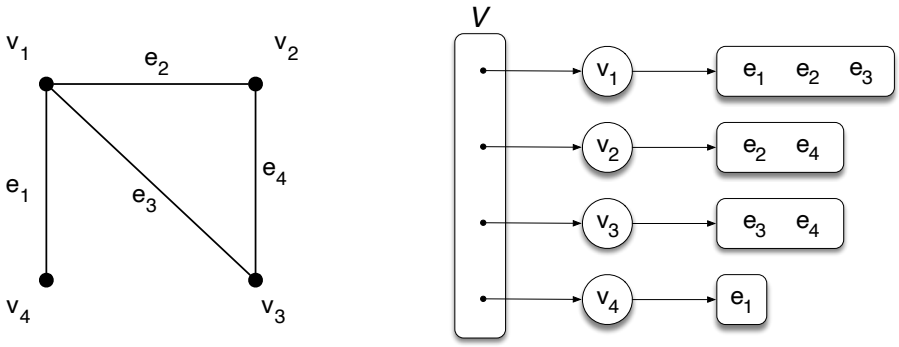
Cada lista $I(v)$ é composta por referências as arestas incidentes a v. As listas $I(v)$ podem ser compostas por coleções dinâmicas de arestas, normalmente listas duplamente encadeadas. No caso de grafos dirigidos, cada vértice mantém listas separadas para as suas arestas de entrada e de saída, respectivamente $I_e(v)$ e $I_s(v)$.

O acesso a elementos da estrutura inicialmente depende da implementação da coleção V. Em situações em que não há inserção e remoção de vértices após a criação do grafo, a coleção pode ser um array e os vértices podem possuir um identificador numérico indicando a sua posição no array. Desta forma, o acesso à lista de incidência de um determinado vértice é feito em tempo constante $O(1)$. Por outro lado, situações de natureza mais dinâmica com eventuais inserções e remoções de vértices, demandam uma implementação dinâmica para V, possivelmente por meio de listas duplamente encadeadas.

Neste caso, o acesso a determinada lista de incidência é feito em tempo $O(n)$.

A Figura 2.2 mostra a representação esquemática da estrutura de listas de adjacência para um grafo simples não dirigido. Assumimos que, cada aresta representada no diagrama é uma instância única de um objeto aresta e que todas as arestas estão organizadas em uma coleção não-ordenada iterável de arestas E.

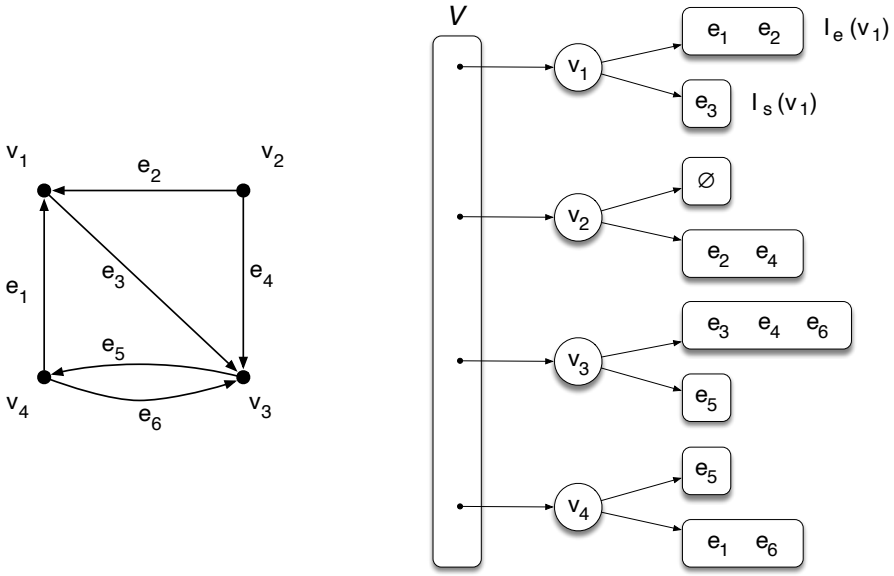
Figura 2.2 – Listas de adjacência de um grafo simples



Fonte: adaptado de Goodrich, Tamassia e Goldwasser (2013).

Conforme mencionado anteriormente, no caso de grafos dirigidos, cada vértice possui duas listas de incidência, sendo uma para as arestas de entrada e outra para as arestas de saída. A Figura 2.3 ilustra, esquematicamente, um exemplo deste tipo.

Figura 2.3 – Listas de adjacência de um grafo dirigido



Fonte: adaptado de Goodrich, Tamassia e Goldwasser (2013).

2.5 Mapas de adjacência

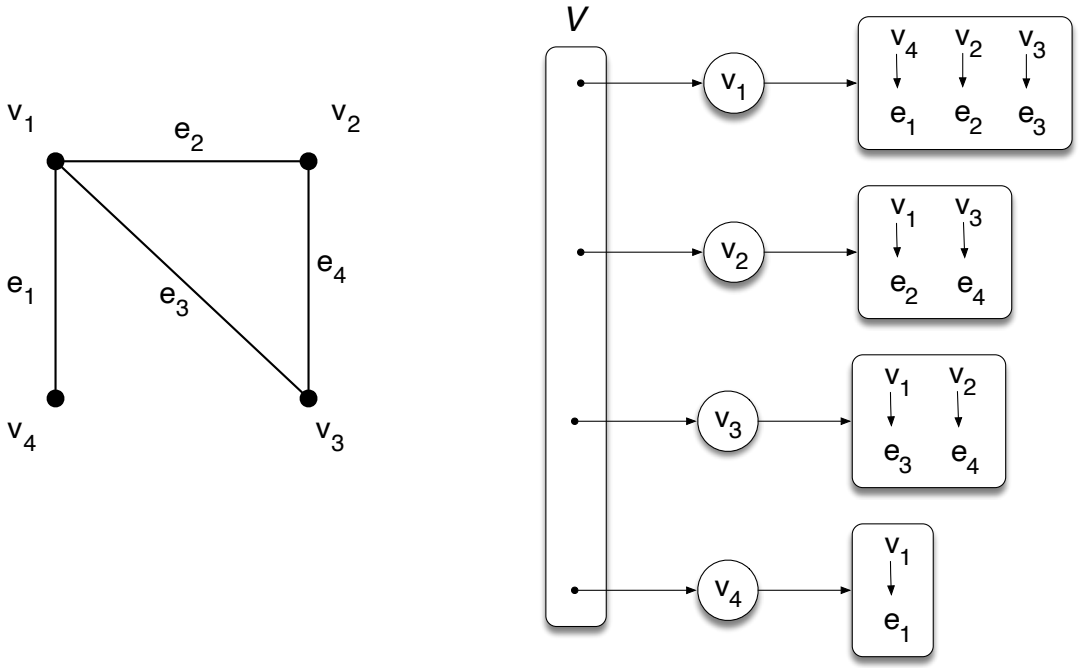
A estrutura de mapas de adjacência é semelhante à estrutura de listas de adjacência, porém ao invés de uma lista cada vértice mantém um mapa de arestas. As chaves do mapa são os vértices adjacentes ao vértice considerado. Esta estrutura permite uma maior eficiência ao acesso à uma determinada aresta feito pela operação `getA(u, v)`, desde que a implementação do mapa seja eficiente.

Ao usar mapas (ou *hash maps*) para implementar $I(v)$ de cada vértice v, fazemos o vértice incidente oposto de cada aresta ser a chave do par chave/valor do mapa. O uso de memória desta estrutura de mapas de adjacência continua sendo $O(m+n)$, visto que assim como nas listas de adjacência, cada $I(v)$ usa espaço $O(grau(v))$.

A maior vantagem dos mapas de adjacência sobre as listas de adjacência é que a operação `getA(u, v)` compreende um desempenho esperado da ordem $O(1)$. Tratando-se, portanto, de uma boa estrutura de dados para implementação de grafos de uma forma geral.

A Figura 2.4 mostra a representação esquemática da estrutura de mapas de adjacência para um grafo simples não dirigido.

Figura 2.4 – Mapas de adjacência de um grafo dirigido



Fonte: adaptado de Goodrich, Tamassia e Goldwasser (2013).

2.6 Matrizes de adjacência

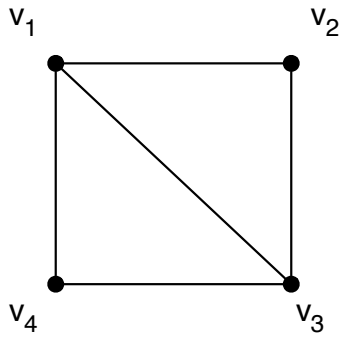
Uma matriz de adjacência, denotada por $[A]$, é uma matriz quadrada de tamanho $n \times n$. Para um grafo simples $G = (V, E)$, os elementos da matriz são definidos da seguinte forma:

$$a_{i,j} = \begin{cases} 1, & \text{se } (v_i, v_j) \in E; \\ 0, & \text{se } (v_i, v_j) \notin E. \end{cases} \tag{2.1}$$

Por esta definição, pensamos nos vértices como inteiros no conjunto $\{0, 1, 2, \ldots, n-1\}$ e nas arestas como pares destes inteiros. Dado um par de vértices, podemos acessar uma aresta em tempo constante. Armazenando estas informações em uma matriz quadrada de tamanho $n \times n$, cada vértice do grafo fica associado a uma linha e uma coluna da matriz. Se houver aresta ligando dois vértices v_i e v_j quaisquer, os elementos correspondentes às linhas i e j da matriz possuem valor igual a 1. O grafos da Figura 2.5 e da Figura 2.6 ilustram este caso.

Note que no caso de grafos não dirigidos a matriz é simétrica, guardando portanto a propriedade $[A] = [A]^T$, em consequência de $a_{i,j} = a_{j,i}$. Isto está de acordo com a definição de grafos não dirigidos vista no Capítulo 1, pois este tipo de grafo modela relações simétricas entre elementos de um problema. Para o caso de digrafos, sempre temos $[A] \neq [A]^T$.

Figura 2.5 – Matriz de adjacência de grafo simples

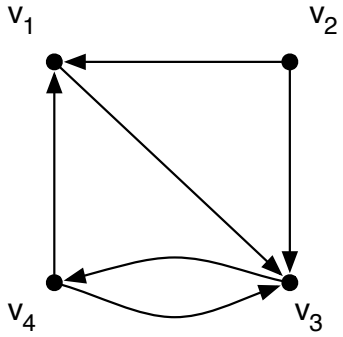


$$[A] = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Fonte: o autor.

É importante mencionar que estas matrizes apenas indicam a existência ou não de uma aresta conectando determinado par de vértices. Se optarmos por acessar os objetos aresta, podemos construir a matriz de forma que cada elemento $a_{i,j}$ guarde a referência para um objeto aresta (v_i, v_j), ao invés de conter um número inteiro. Se não houver aresta entre determinado par de vértices, o elemento correspondente da matriz deve conter uma referência nula.

Figura 2.6 – Matriz de adjacência de grafo simples dirigido

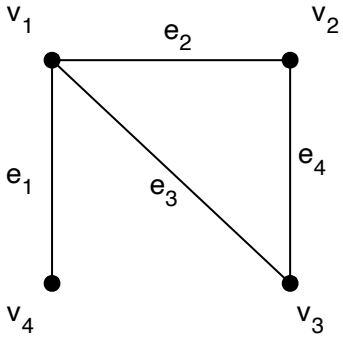


$$[A] = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Fonte: o autor.

A Figura 2.7 mostra, esquematicamente, como ficaria este tipo de matriz de adjacência para um grafo simples não dirigido.

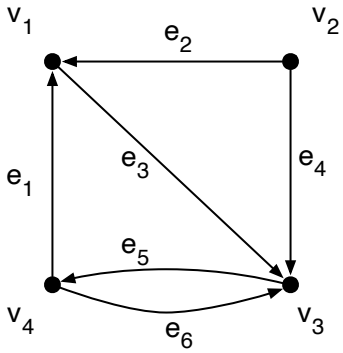
Figura 2.7 – Matriz de adjacência de grafo simples



$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} nil & e_2 & e_3 & e_1 \\ e_2 & nil & e_4 & nil \\ e_3 & e_4 & nil & nil \\ e_1 & nil & nil & nil \end{bmatrix} \end{array}$$

No caso de um grafo simples dirigido, teríamos matrizes não-simétricas com referências para as arestas, conforme ilustrado na Figura 2.8.

Figura 2.8 – Matriz de adjacência de grafo simples dirigido



$$[A] = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} nil & nil & e_3 & nil \\ e_2 & nil & e_4 & nil \\ nil & nil & nil & e_5 \\ e_1 & nil & e_6 & nil \end{bmatrix} \end{array}$$

Fonte: o autor.

Os exemplos acima tratam apenas de grafos simples. Como cada par de vértice forma apenas uma aresta, é conveniente a utilização de matrizes. Para multigrafos, especialmente no caso de arestas paralelas, é preciso adaptar a estrutura para comportar mais de uma

referência a arestas para cada par de vértice. É possível pensar em estratégias combinando listas ou mapas de adjacência com matrizes, em que cada elemento da matriz estaria associado a uma lista ou mapa de arestas. Isto resolve o problema de representação de arestas paralelas, mas por outro lado perdemos uma das grandes vantagens de se usar matrizes de adjacência, que é o acesso a arestas em tempo constante.

2.7 Exercícios

1. Quais são as principais diferenças, em termos de vantagens e desvantagens, entre o uso de listas de adjacência e matrizes de adjacência para representação de grafos?

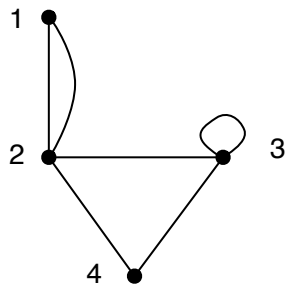
2. Escreva as listas e matrizes de adjacência dos grafos abaixo, sendo dados seus conjuntos de vértices e arestas. Faça a mesma coisa, considerando que os conjuntos abaixo definem grafos dirigidos.

   a) $V = \{1, 2, 3, 4, 5\}$ e $E = \{(1, 2), (1, 4), (1, 5), (2, 3), (3, 4), (4, 4)\}$;
   b) $V = \{1, 2, 3, 4, 5, 6\}$ e $E = \{(1, 2), (1, 4), (1, 4), (2, 3), (2, 5), (3, 5)\}$;
   c) $V = \{1, 2, 3, 4, 5, 6\}$ e
   $E = \{(1, 2), (1, 3), (1, 4), (2, 5), (2, 6), (3, 5), (3, 6), (4, 5), (4, 6)\}$;
   d) $V = \{1, 2, 3, 4, 5\}$ e $E = \{(1, 2), (1, 4), (2, 3), (2, 4), (2, 5), (3, 4), (3, 5)\}$;
   e) $V = \{1, 2, 3, 4\}$ e $E = \{(1, 2), (2, 3), (2, 4), (3, 4)\}$;
   f) $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$ e
   $E = \{(1, 2), (2, 2), (2, 3), (3, 4), (3, 5), (6, 7), (6, 8), (7, 8)\}$.

3. (PosComp – 2006) A respeito da representação de um grafo de n vértices e m arestas é correto dizer que:

   a) a representação sob a forma de matriz de adjacência exige espaço $\Omega(m^2)$.
   b) a representação sob a forma de listas de adjacência permite verificar a existência de uma aresta ligando dois vértices dados em tempo $O(1)$.
   c) a representação sob a forma de matriz de adjacência não permite verificar a existência de uma aresta ligando dois vértices dados em tempo O(1).
   d) a representação sob a forma de listas de adjacência exige espaço $\Omega(n + m)$.
   e) todas as alternativas estão corretas.

4. (PosComp – 2013) Seja o grafo G a seguir:



Com base nesse grafo, considere as afirmativas a seguir.

I. O grafo G é conexo.

II. A matriz de adjacências do grafo G é dada por $[A] = \begin{bmatrix} 0 & 2 & 0 & 0 \\ 2 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$

III. O grau do vértice 2 é igual a 2.

IV. O grafo G é denotado como Grafo Simples.

Assinale a alternativa correta.

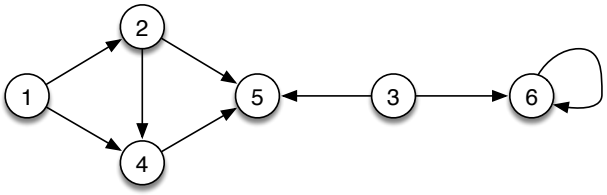
a) Somente as afirmativas I e II são corretas.
b) Somente as afirmativas I e IV são corretas.
c) Somente as afirmativas III e IV são corretas.
d) Somente as afirmativas I, II e III são corretas.
e) Somente as afirmativas II, III e IV são corretas.

5. (PosComp – 2015) Seja $G = (V, E)$ um grafo em que V é o conjunto de vértices e E é o conjunto de arestas. Considere a representação de G como uma matriz de adjacências $[A]$.

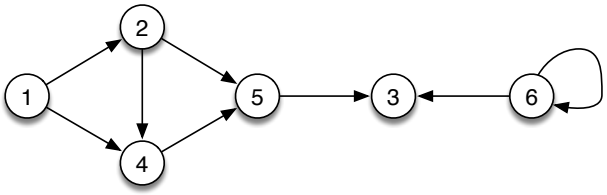
$$[A] = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

O correspondente grafo orientado G é:

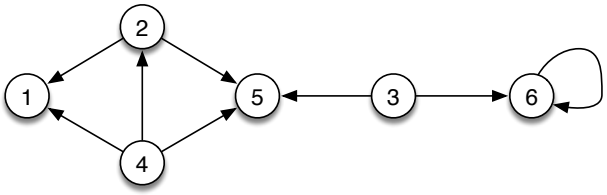
a)



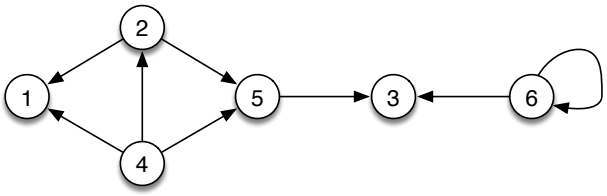
b)



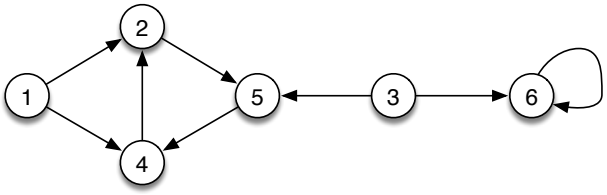
c)



d)



e)



6. (PosComp – 2016) A matriz de um grafo $G = (V, A)$ contendo n vértices é uma matriz $n \times n$ de bits, em que $A[i, j]$ é 1 (ou verdadeiro, no caso de booleanos) se e somente se existir um arco do vértice i para o vértice j. Essa definição é uma:

   a) Matriz de adjacência para grafos não ponderados.
   b) Matriz de recorrência para grafos não ponderados.
   c) Matriz de incidência para grafos não ponderados.
   d) Matriz de adjacência para grafos ponderados.
   e) Matriz de incidência para grafos ponderados.

7. Implemente em uma linguagem de programação, uma estrutura de dados, ou conjunto de classes para armazenar e manipular grafos e digrafos. Avalie as vantagens e desvantagens da sua estrutura em relação a modelagem comentada na seção 2.2.