Teoria dos Grafos – Implementação com Listas de Adjacências
Resolução da Atividade Hands-on em Laboratório – 02
Prof. Calvetti

1. Introdução

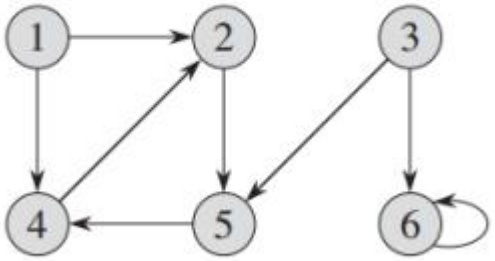
Um grafo é uma coleção de vértices e arestas. Pode-se modelar a abstração por meio de uma combinação de três tipos de dados: Vértice, Aresta e Grafo. Um vértice (VERTEX) pode ser representado por um objeto que armazena a informação fornecida pelo usuário, por exemplo, informações de um aeroporto. Uma aresta (EDGE) armazena relacionamentos entre vértices, por exemplo: número do voo, distâncias, custos, etc.

A ADT Graph deve incluir diversos métodos para se operar com grafos, podendo lidar com grafos direcionados ou não direcionados. Uma aresta (u,v) é dita direcionada de u para v se o par (u,v) é ordenado, com u precedendo v. Uma aresta (u,v) é dita não direcionada se o par (u,v) não for ordenado.

A implementação de grafos por meio de Matriz de Adjacências é feita numerando-se os vértices 1,2, ... |V| de alguma maneira arbitrária. A representação de um grafo G, de acordo com Cormen, consiste de uma matriz |V| x |V| A = (a_{ij}) tal que:

1 se (i,j) ϵ E,

Aij = 0 em caso contrário.



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

A matriz de adjacências, de acordo com Cormen, exige a memória $\Theta(V^2)$, independentemente do número de arestas do grafo.

Embora a representação de listas de adjacências seja assintoticamente pelo menos tão eficiente quanto a representação de matriz de adjacências, a simplicidade de uma matriz de adjacências pode torná-la preferível quando os grafos são razoavelmente pequenos.

Neste hands-on, iremos fazer a implementação de grafos simples, não-orientados, com o emprego de Matriz de Adjacências e com a Linguagem Java.

Para a implementação, vamos criar um Projeto Java na IDE chamado Grafo_MatAdj. Em seguida, vamos criar um package denominado br.maua.

2. Implementação da Classe Grafo

No package br.maua, iremos implementar a Classe Grafo que irá abstrair o grafo por meio de uma matriz de adjacências.

Vamos também incluir na Classe Grafo o construtor de grafos.

O tamanho da matriz é VxV, onde V é o número de vértices no grafo e o valor de uma entrada Aij é 1 ou 0, dependendo da existência de uma aresta do vértice i ao vértice j.

No caso de grafos não direcionados, a matriz é simétrica em relação à diagonal uma vez que para cada aresta (i, j), há também uma aresta (j, i).

As operações básicas, como adicionar uma aresta, remover uma aresta e verificar se existe uma aresta do vértice i ao vértice j, são operações de tempo constante extremamente eficientes.

A classe será definida por:

```
package br.maua;

public class Grafo {

	int numVertices;

	boolean [][] adjMatrix;
}
```

3. Implementação do Construtor de Grafo na Classe Grafo

Incluir na classe Grafo, o construtor do grafo que terá como parâmetro o número de vértices do grafo:

```
package br.maua;

public class Grafo {

	int numVertices;

	boolean [][] adjMatrix;

public Grafo(int numVertices) {
	this.numVertices = numVertices;
	adjMatrix = new boolean[numVertices][numVertices];
}
```

4. Implementação do método addAresta(i,j) na Classe Grafo

   Incluir na classe Grafo, o método addAresta(i,j) para adicionar uma aresta entre dois vértices referenciados por i e j:

```java
package br.maua;

public class Grafo {

	int numVertices;

	boolean [][] adjMatrix;

	public void addAresta(int i, int j) {
		adjMatrix[i][j] = true;
		adjMatrix[j][i] = true;
	}
```

5. Implementação do método removeAresta(i,j) na Classe Grafo

   Incluir na classe Grafo, o método removeAresta(i,j) para remover uma aresta entre dois vértices referenciados por i e j:

```java
package br.maua;

public class Grafo {

	int numVertices;

	boolean [][] adjMatrix;

public void removeAresta(int i, int j) {
		adjMatrix[i][j] = false;
		adjMatrix[j][i] = false;
}
```

6. Implementação do método ehAresta(i,j) na Classe Grafo

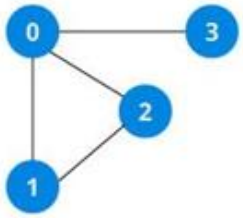
   Incluir na classe Grafo, o método ehAresta(i,j) que retorna true se existe aresta entre as referências i e j. Caso contrário, retorna false:

```java
package br.maua;

public class Grafo {

	int numVertices;

	boolean [][] adjMatrix;

public boolean EhAresta(int i, int j) {
		return adjMatrix[i][j];
}
```

7. Implementação do método retornaGrafo() na Classe Grafo

Incluir na classe Grafo, o método retornaGrafo() que retorna um String correspondente à matriz de adjacência que representa o grafo.

Por exemplo: Seja o Grafo:



	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

A função retornaGrafo() deverá gerar um String correspondente ao grafo. Ao se executar a função:

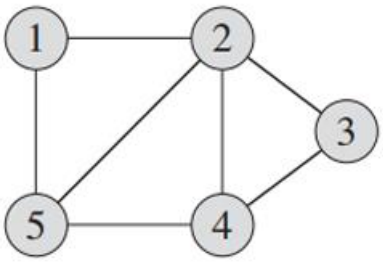
```
System.out.print(g.retornaGrafo());

/* Outputs
   0: 0 1 1 1
   1: 1 0 1 0
   2: 1 1 0 0
   3: 1 0 0 0
  */
```

```
package br.maua;

public class Grafo {

     int numVertices;

     boolean [][] adjMatrix;

public String retornaGrafo() {
        StringBuilder s = new StringBuilder();
        for (int i = 0; i < numVertices; i++) {
            s.append(i + ": ");
            for (boolean j : adjMatrix[i]) {
                s.append((j?1:0) + " ");
            }
            s.append("\n");
        }
        return s.toString();
    }
```

8. Implementação da classe Teste_Grafo

Vamos agora implementar a classe Teste_Grafo, com a função main para criarmos um grafo e exercitarmos todas as funções criadas na implementação de Grafos com Matriz de Adjacência.

Implementar o grafo e exercitar as funções construídas na classe para o seguinte grafo, obtido de Cormen:



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

```
package br.maua;

public class Teste_Grafo {

	public static void main(String[] args) {

		Grafo g = new Grafo(4);

		g.addAresta(0,1);
		g.addAresta(0,4);

		g.addAresta(1,0);
		g.addAresta(1,2);
		g.addAresta(1,3);
		g.addAresta(1,4);

		g.addAresta(2,1);
		g.addAresta(2,3);


		g.addAresta(3,1);
		g.addAresta(3,2);
		g.addAresta(3,4);

		g.addAresta(4,0);
		g.addAresta(4,1);
		g.addAresta(4,3);



		System.out.print(g.retornaGrafo());
		/*
			0: 0 1 0 0 1
			1: 1 0 1 1 1
			2: 0 1 0 1 0
			3: 0 1 1 0 1
			4: 1 1 0 1 0

		*/

	}
}
```