TADS - Teoria dos Grafos
Lab 1 - Implementação de Grafo v 1.1

Prof. Dr. Paulo César Rodacki Gomes - IFC

4 de agosto de 2017

1 Objetivo

O objetivo desta atividade prática em laboratório é implementar classes básicas para manutenção de grafos **não dirigidos**, de acordo com diagrama de classes da figura 1.

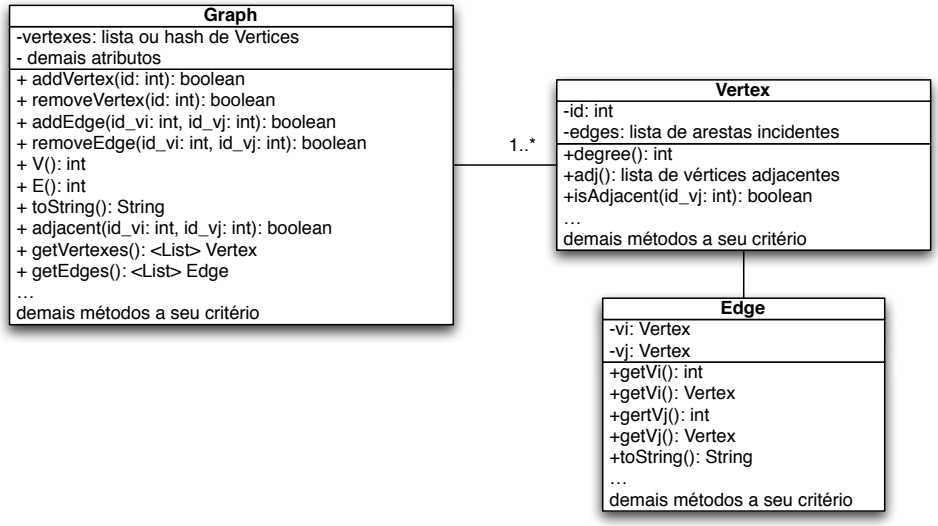

Figura 1: Diagrama de classes para **Grafo não dirigido**

Descrição geral:

A implementação de grafos pode ser feita de várias maneiras diferentes, dependendo dos critérios de projeto mais importantes. Neste exercício estamos propondo a implementação de uma estrutura **dinâmica** para grafos

não dirigidos. Obviamente a implementação pode ter diferentes graus de complexidade tanto em termos de métodos oferecidos quanto na questão de design de classes. Aqui, estamos propondo uma estrutura dinâmica, porém simples e enxuta.

A implementação se divide em três classes: *Graph* ,*Vertex* e *Edge*. A classe *Graph* mantém apenas uma estrutura dinâmica contendo todos os vértices. Cada vértice possui um rótulo (id) numérico que pode ser definido pelo usuário das classes, e uma lista das arestas incidentes no vértice.

Note que não existe lista de arestas na classe grafo, e cada objeto aresta é referenciado duas vezes, uma no vértice v_i e outra no vértice v_j.

Descrição dos principais métodos:

Classe Graph:

1. `addVertex(int id)`: cria um novo vértice no grafo. O método recebe um id inteiro que será o rótulo do vértice. Deve instanciar um objeto da classe vértice e inclui-lo no grafo (na lista, conjunto ou hash map de vértices).

2. `removeVertex(int id)`: retira o vértice do grafo. IMPORTANTE: este método deve remover todas as arestas incidentes ao vertice removido.

3. `addEdge(int id_vi, int id_vj)`: cria uma nova Aresta no grafo. O método recebe os 2 ids dos vértices incidentes à aresta. Note que os 2 vértices precisam existir para que a operação seja efetuada com sucesso.

4. `removeEdge(int id_vi, int id vj)`: retira a aresta do grafo, caso exista.

5. `boolean adjacent(int id_vi, int id_vj)`: retorna verdadeiro se os dois vértices são adjacentes.

6. `getVertexes()`: retorna uma lista de todos os Vertices (objetos) do grafo.

7. `getEdges()`: retorna uma lista de todas as Arestas (objetos) do grafo.

8. `int V()`: retorna a ordem do grafo.

9. `int E()`: retorna o tamanho do grafo.

10. `String toString()`: imprime o grafo de acordo com a organização em listas de adjacência. Portanto, um grafo igual ao da figura 2 seria impresso da seguinte forma:

```
1: 2 3 4 4
2: 1 3
3: 1 2 3
4: 1 1
```

Observação: a ordem dos vértices é arbitrária, portanto **não** é necessário que os *ids* estejam em ordem crescente.

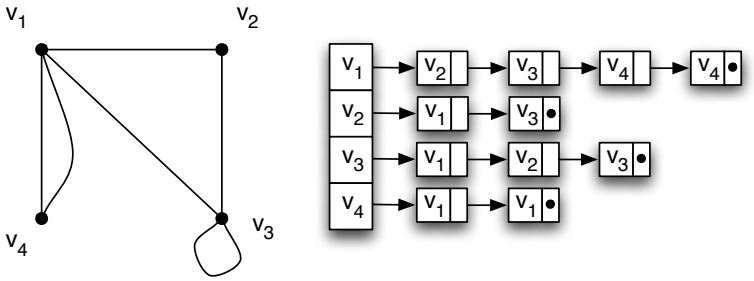

Figura 2: Exemplo de listas de adjacência de grafo não dirigido