IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Programação Avançada

Implementação do TAD –Graph
Programação Avançada – 2020-21

Bruno Silva, Patrícia Macedo

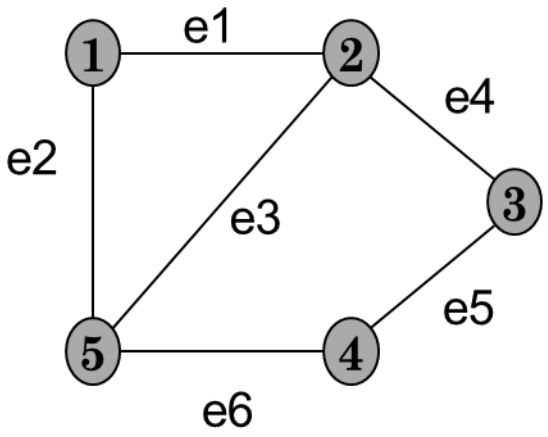
Sumário 

- Especificação do TAD Graph (revisão)
- Noção de Tipo de Dados Vertex e de Edge
- Interface Graph
- Implementação usando uma Lista de Arestas

TAD Graph - Especificação

Um grafo G(V, E) é definido pelos conjuntos V e E, onde:

- V é um conjunto não vazio: Vértices.
  `{v1,v2,v3,v4,v5}`
- E é um conjunto de pares ordenados e=(v,w) com v e w pertencente a V: arestas (edges).
  `{ e1=(v1,v2) ,e2=v2,v5), e3=(v2,v5), e4=(v2,v3), e5=(v4,v3)}`



TAD Graph – Especificação das operações

- **insertVertex(v):** insere v como sendo vértice do grafo.
- **insertEdge(u, v, e):** insere uma aresta ***x*** entre os vértices ***u*** e v; devolve erro se v e x não conrespondem a vértices do grafo
- **removeVertex(v)**: remove o vértice **v** e todas as suas arestas adjacentas, devolve erro se v não existir no grafo.
- **removeEdge(e)**: remove a aresta ***e*** , devolve erro se e não existir no grafo.
- **numvertices():** devolve o número de vértices
- **numEdges():** devolve o número de arestas
- **edges() :** devolve uma coleção iterável das arestas do grafo.
- **vertices() :** devolve uma coleção iterável dos vértices do grafo.
- **opposite(v, e):** devolve o vértice da aresta e oposto ao vértice v, devolve erro se v ou e não existem no grafo, ou se v mão é vértice da resta e.
- **degree(v):** devolve o grau do vértice v, devolve erro se v não existe no garfo.
- **incidentEdges(v):** devolve uma coleção iterável das arestas incidentes ao vértice v, devolve erro de v não existe no grafo.
- **areAdjacent(v,w):** devolve um valor lógico (true/false) que indica se os vértices v o w são adjacentes , devolve erro se v ou w não existire como vértices do grafo.

Implementação em JAVA

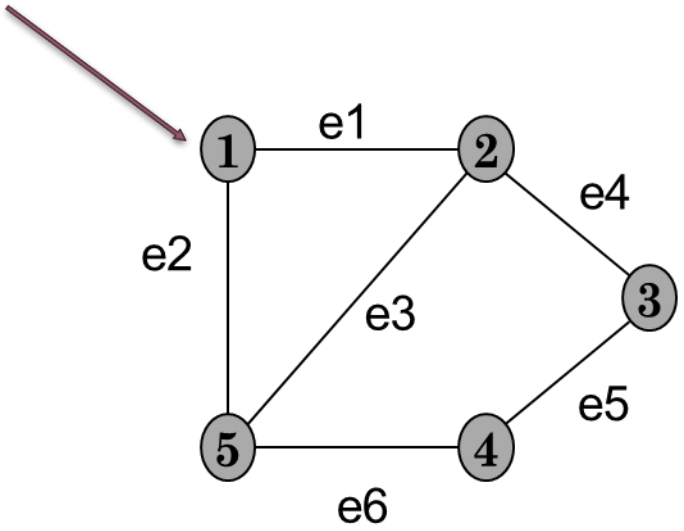
Para implementarmos o TAD Graph em JAVA, temos que decidir como implementar a noção de Vértice e de Aresta (Edge).

Considerações:

- Os grafos tem as arestas e os vértices rotulados
- Uma aresta e um vértice, podem ter elementos de tipos diferentes.
  Exemplo: Um mapa com os Caminho de Ferro de Portugal, os vértices teriam elementos do tipo Estação de Caminho de Ferro e as Arestas informação sobre o percurso que liga as duas estações.
- Usam-se as interfaces em Java para definir o tipo Vertex (vértice) e o tipo Edge (aresta).
- O tipo Vertex caracteriza-se pelo seu rótulo (elemento)
- O tipo Edge caracteriza-se pelo seu rótulo (elemento) e pelos par de vértices que conecta.

Tipo Vertex - Implementação

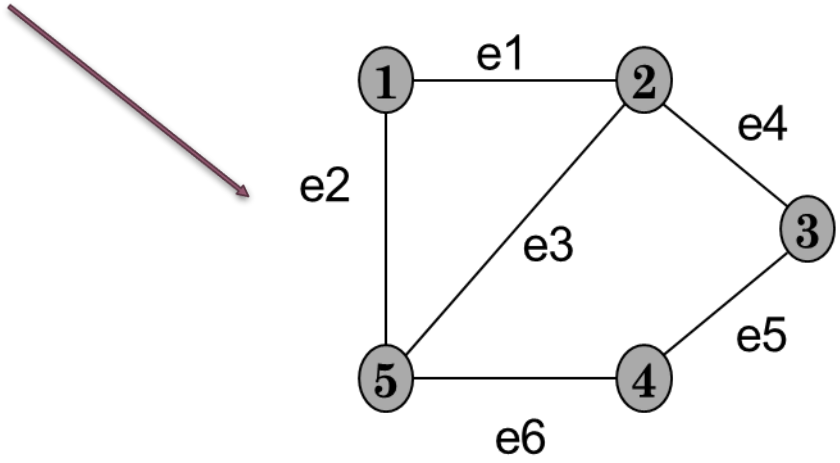
- Um vértice caracteriza-se por guardar um elemento do tipo genérico V.




```
public interface Vertex<V> {

    public V element();
}
```

Tipo Edge - Implementação

- Uma aresta (edge) caracteriza-se por guardar a referência para o par de vértices que conecta e um elemento do tipo E .



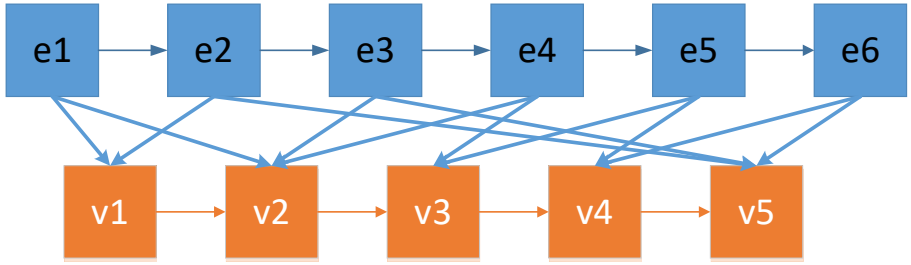

```
public interface Edge<E, V> {

    public E element();

    public Vertex<V>[] vertices();
}
```

Interface Graph

```
public interface Graph<V, E> {

    public int numVertices();

    public int numEdges();

    public Collection<Vertex<V>> vertices();

    public Collection<Edge<E, V>> edges();

    public Collection<Edge<E, V>> incidentEdges(Vertex<V> v)throws InvalidVertexException;

    public Vertex<V> opposite(Vertex<V> v, Edge<E, V> e) throws InvalidVertexException, InvalidEdgeException;

    public boolean areAdjacent(Vertex<V> u, Vertex<V> v) throws InvalidVertexException;

    public Vertex<V> insertVertex(V vElement) throws InvalidVertexException;

    public Edge<E, V> insertEdge(Vertex<V> u, Vertex<V> v, E edgeElement)
                                            throws InvalidVertexException, InvalidEdgeException;

    public  V removeVertex(Vertex<V> v) throws InvalidVertexException;

    public E removeEdge(Edge<E, V> e) throws InvalidEdgeException;
}
```

Implementação usando a estrutura de dados : Listas de arestas

- Map de Arestas.
- Map de Vertices
- Usam-se dicionarios apra mais facilmente obter uma aresta ou um vertice em função do seu elemnto.




```
public class GraphEdgeList<V, E> implements Graph<V, E>

private Map<V,Vertex<V>> vertices;
private Map<E,Edge<E, V>> edges;

public GraphEdgeList() {
    this.vertices = new HashMap<>();
    this.edges = new HashMap<>();
}
```

Implementação da interface Vertex<V>

- A interface Vertex<V> é implementada como uma inner class de GraphEdgeList: **MyVertex**

```
class MyVertex implements Vertex<V> {

    V element;

    public MyVertex(V element) {
        this.element = element;
    }

    @Override
    public V element() {
        return this.element;
    }

    @Override
    public String toString() {
        return "Vertex{" + element + '}';
    }
}
```

Implementação da interface Edge<E,V>

- A interface Edge<E,V> é implementada como uma inner class de GraphEdgeList : **MyEdge**

```
class MyEdge implements Edge<E, V> {

    E element;
    Vertex<V> vertexOutbound;
    Vertex<V> vertexInbound;

    public MyEdge(E element, Vertex<V> vertexOutbound, Vertex<V> vertexInbound) {
        this.element = element;
        this.vertexOutbound = vertexOutbound;
        this.vertexInbound = vertexInbound;
    }

    @Override
    public E element() {
        return this.element;
    }

    public boolean contains(Vertex<V> v) {
        return (vertexOutbound == v || vertexInbound == v);
    }

    @Override
    public Vertex<V>[] vertices() {
        Vertex[] vertices = new Vertex[2];
        vertices[0] = vertexOutbound;
        vertices[1] = vertexInbound;

        return vertices;
    }
```

CheckVertex : método auxiliar

Método que tem como objetivo converter o tipo Vertex no objecto concreto, verificando se o mesmo pertence ao grafo.

```
private MyVertex checkVertex(Vertex<V> v) throws InvalidVertexException {
    if(v == null) throw new InvalidVertexException("Null vertex.");

    MyVertex vertex;
    try {
        vertex = (MyVertex) v;
    } catch (ClassCastException e) {
        throw new InvalidVertexException("Not a vertex.");
    }

    if (!vertices.containsKey(vertex.element)) {
        throw new InvalidVertexException("Vertex does not belong to this graph.");
    }

    return vertex;
}
```

CheckEdge : método auxiliar

Método que tem como objetivo converter o tipo Edge no objecto concreto, verificando se o mesmo pertence ao grafo.

```
private MyEdge checkEdge(Edge<E, V> e) throws InvalidEdgeException {
    if(e == null) throw new InvalidEdgeException("Null edge.");

    MyEdge edge;
    try {
        edge = (MyEdge) e;
    } catch (ClassCastException ex) {
        throw new InvalidVertexException("Not an adge.");
    }

    if (!edges.containsKey(edge.element)) {
        throw new InvalidEdgeException("Edge does not belong to this graph.");
    }

    return edge;
}
```

Implementação: Inserir um Vértice

- Não é permitido haver dois vértices idênticos.
- Insere-se o vértice no map de vértices.

```
public Vertex<V> insertVertex(V vElement) throws InvalidVertexException {
    if (existsVertexWith(vElement)) {
        throw new InvalidVertexException("There's already a vertex with this element.");
    }

    MyVertex newVertex = new MyVertex(vElement);

    vertices.put(vElement, newVertex);

    return newVertex;
}
```

Implementação: Remover um Vértice

1. Remove todas as arestas incidentes
   - Usa o método incidentEdges para determinar a lista de arestas a remover
   - Remove cada aresta da lista de arestas
2. Remove o vértice da lista

```
public synchronized V removeVertex(Vertex<V> v) throws InvalidVertexException {

    checkVertex(v);

    V element = v.element();

    //remove incident edges
    Iterable<Edge<E, V>> incidentEdges = incidentEdges(v);
    for (Edge<E, V> edge : incidentEdges) {
        edges.remove(edge.element());
    }

    vertices.remove(v.element());

    return element;
}
```

ADT Graph | Exercícios de implementação

1. Faça *clone* do projeto base **ADTGraph_Template** (projeto **IntelliJ**) do *GitHub*:

   https://github.com/pa-estsetubal-ips-pt/ADTGraph_FXSmartGraph_Template

2. Forneça o código dos métodos por implementar, i.e., os que estão a lançar `NotImplementedException` ;

3. Compile e teste o programa fornecido.

2. Altere o método main, de forma a construir o grafo da figura abaixo.