IPS Instituto Politécnico de Setúbal

Escola Superior de Tecnologia de Setúbal

Programação Avançada 2021-22

[3b] Grafos | TAD Graph

Bruno Silva, Patrícia Macedo

Sumário 📝

- Especificação do TAD Graph;
- Noção de Tipo de Dados Vertex e de Edge;
- Interface Graph, Vertex , Edge;
- Utilização do Tipo Graph;
- Exercícios.

TAD Graph | Especificação

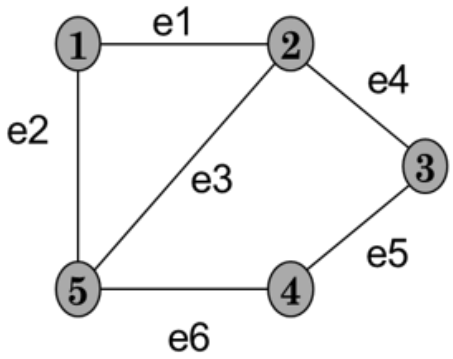
Um grafo G(V, E) é definido pelos conjuntos V e E, onde:

- **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=(v1,v5), e3=(v2,v5), e4=(v2,v3), e5=(v4,v3), e6=(v5,v4)}`



TAD Graph | Especificação

As operações **modificadoras** do TAD Graph são:

- `insertVertex(v)` : Insere **v** como vértice do grafo;
- `insertEdge(u, v, e)` : Insere a aresta **e** entre os vértices **u** e **v**. Devolve erro se **u** e **v** não corresponderem a vértices do grafo;
- `removeVertex(v)` : Remove o vértice **v** e todas as suas arestas adjacentes. Devolve erro se **v** não existir no grafo;
- `removeEdge(e)` : Remove a aresta **e**. Devolve erro se **e** não existir no grafo.

As operações inspetoras (devolvem informação sobre o estado do 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 existirem no grafo, ou se `v` não é vértice da aresta `e`;
- `degree(v)` : Devolve o grau do vértice `v`. Devolve erro se `v` não existe no garfo;
- `incidentEdges(v)` : Devolve a coleção iterável das arestas incidentes ao vértice `v`. Devolve erro se `v` não existir no grafo;
- `areAdjacent(v,w)` : Devolve um valor lógico que indica se os vértices `v` e `w` são adjacentes. Devolve erro se `v` ou `w` não existirem como vértices do grafo.

O TAD Graph : Implementação em JAVA

- Para implementarmos o TAD Graph, precisamos de especificar adicionalmente os seguintes tipos de dados:
  - **Vertex**: Define o tipo **vértice** e caracteriza-se por guardar um elemento do tipo genérico **V**.
  - **Edge**: Define o tipo **aresta** e caracteriza-se por guardar um elemento do tipo genérico **E** e os vértices que conecta.

Interface Vertex

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

- Podemos definir uma interface para esse tipo.

```
public interface Vertex<V> {
    public V element();
}
```

NOTA : Todos as instâncias de **Vertex** disponibilizam a operação `element()` .

Interface Edge

Por simplicidade, iremos sempre assumir que as arestas se caracterizam por ter um rótulo (ou valor) associado. Assim uma aresta (*edge*) caracteriza-se por:

- Guardar a referência para o par de vértices que liga;
- Conter m elemento do tipo E.

Podemos apresentar uma interface que defina esse tipo.

```
public interface Edge<E, V> {
    public E element();
    public Vertex<V>[] vertices();
}
```

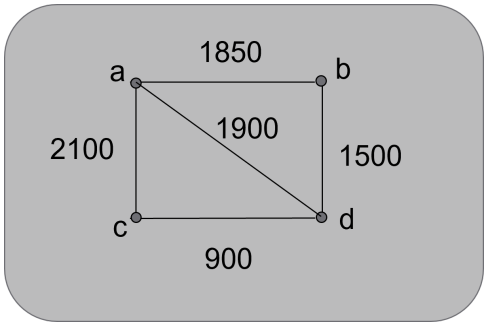
NOTA : Todos as instâncias de *Edge*, disponibilizam a operação `element()` e `vertices`

Interface Graph | Tipos Genéricos

- O tipo **Graph**, sendo um tipo Genérico, vai ter dois parâmetros (à semelhança da interface Map<K,V>).
  - **V** - Tipo de dados correspondente aos elementos do vértice;
  - **E** - Tipo de dados correspondente aos elementos da aresta.

Exemplo:

- Considere o grafo da figura que representa a distância entre locais num mapa:
  - **V** - Tipo *Char* (letra que define o local);
  - **E** - Tipo *Integer* (valor da distância em km).



Interface Graph | Exceções

- Na especificação estão categorizados dois tipos de erros:
  - Devolve erro se o vértice `v` não existir no grafo;
  - Devolve erro se a aresta `e` não existir no grafo.
- Que correspondem a dois tipos de exceções:
  - `InvalidVertexException`
  - `InvalidEdgeException`

Interface Graph | Operações

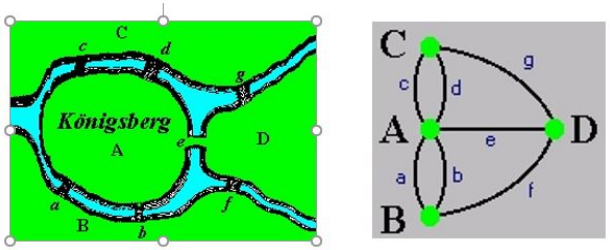
```
public interface Graph<V, E> {

    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;
}
```

Exercícios (1)

1. Faça clone do projeto : https://github.com/estsetubal-pa-geral/ADTGraphTemplate.git
Este projeto contem integrada a API ADT_Graph.jar que disponibiliza uma implementação do ADT Graph apresentado acima.

Pretende-se manipular o ADT Graph. Para tal define-se uma instância de um Grafo usando para vértice objetos do tipo `Local` e para arestas objetos do tipo `Bridge`

2. Complete o método `main` de forma a
   a) Construir o grafo apresentado na figura;



Exercícios (1) Continuação

b) Mostrar todas as pontes que saem do Local `A`;

c) Mostrar todos os locais vizinhos do Local `D`;

d) Determinar o número de pontes que partem de `C`;

e) Dados dois locais, verificar se existe uma ponte que os liga diretamente.

Exercícios (2)

Relembrando o pseudocódigo do DFS

```
Algorithm: DFS(Graph,vertice_root)
BEGIN
    setAsVisited(vertice_root)
    push(s,vertice_root)
    WHILE s is not EMPTY
        BEGIN
            v <- pop(s)
            process(v)
            FOR EACH w adjacents(v)
                IF w is not visited THEN
                    BEGIN
                        setAsVisited(w)
                        push(s,w)
                    END
        END
END
```

Exercícios (2) cont

Escreva um método na classe `Main` que implemente o método DFS.

Nota : Relembra-se que poderá usar a classe `java.util.Stack`