

# Algorithmes et Pensée Computationnelle

## Programmation orientée objet - Exercices avancés

Le but de cette séance est de se familiariser avec un paradigme de programmation couramment utilisé : la Programmation Orientée Objet (POO). Ce paradigme consiste en la définition et en l'interaction avec des briques logicielles appelées **Objets**. Dans les exercices suivants, nous manipulerons des objets, aborderons les notions de classe, méthodes, attributs et encapsulation. Au terme de cette séance, vous serez en mesure d'écrire des programmes mieux structurés. Afin d'atteindre ces objectifs, nous utiliserons principalement le langage **Java** qui offre une panoplie d'outils pour mieux comprendre ce paradigme de programmation. Le code présenté dans les énoncés se trouvent sur Moodle, dans le dossier **Ressources**.

## 1 Manipulation de graphes en POO (Java)

Cette partie constitue un rappel sur la notion de graphe orienté. Avant de traiter les exercices de cette partie, veuillez répondre aux questions suivantes :

1. Quels sont les composantes d'un **graphe orienté** ?
2. Combien de classes sont nécessaires pour représenter un graphe orienté ainsi que toutes ses composantes ?
3. Sachant que vous avez une classe qui représente les **arêtes** et que les **sommets** sont représentés par des chaînes de caractères. Quels pourraient être les attributs de la classe **graph**.

Maintenant que vous avez une idée des attributs de la classe **graph** à définir, déterminer quelles méthodes sont nécessaires au fonctionnement de cet objet. Pour ce faire, vous pouvez répondre aux questions suivantes :

1. Est-ce que l'utilisateur aura besoin de modifier l'objet une fois celui-ci aura été initialisé ?
2. Est-ce que l'utilisateur aura besoin de vérifier l'état de l'objet (existence d'attributs, vérification de valeurs...) ou de certaines parties de l'objet ?

### Conseil

N'hésitez pas à revenir sur le cours de la semaine 7 (algorithmes de graphes) disponible sur Moodle.

Voici les réponses aux questions posées ci-dessus.

**Certaines questions peuvent avoir plus d'une réponse étant donné qu'il existe plusieurs manières de représenter un graphe en programmation.**

1. (a) Les **sommets**.  
(b) Les **arêtes**.  
(c) Le **poids** de chaque arête.
2. Il faut 2 classes ( ou 3 si on cherche à avoir plus d'informations dans les sommets) :  
(a) Une classe **Edge** qui va représenter les arêtes.  
(b) Une classe **Graph**.
3. La classe **Graph** va avoir 2 attributs : un ensemble contenant les **sommets** et un autre ensemble contenant les **arêtes** de celui-ci.

Le code de la classe **Edge** est fourni dans le dossier ressources sur Moodle. Utiliser le fichier **Main.java** dans le dossier **Code** sur Moodle pour effectuer des tests. Une fois exécuté, il devrait afficher :

```
The vertex number 1 has a value of: Lausanne
The vertex number 2 has a value of: Geneve
The vertex number 3 has a value of: Berne
{from_vertex=Geneve, weight=35.0, to_vertex=Lausanne}
{from_vertex=Lausanne, weight=100.0, to_vertex=Berne}
{from_vertex=Geneve, weight=120.0, to_vertex=Berne}
Edge between Geneve and Berne has been deleted.
The vertex number 1 has a value of: Lausanne
The vertex number 2 has a value of: Geneve
The vertex number 3 has a value of: Berne
{from_vertex=Geneve, weight=35.0, to_vertex=Lausanne}
{from_vertex=Lausanne, weight=100.0, to_vertex=Berne}
```

```
Process finished with exit code 0
```

### Question 1: (🕒 20 minutes)

Voici une partie de la classe `graph` que vous devez compléter. Implémentez les méthodes `update_weight()`, `new_edge()` et `edge_exist()`.

```
1  import java.util.List;
2  import java.util.HashMap;
3  import java.util.Vector;
4
5  public class graph_empty {
6
7      // Les attributs de la classe graphe
8      public List<Edge> edges = new Vector(); // Utilisation de Cector car il faut que l on puisse rajouter ou supprimer des é
          éléments de la liste
9      public List<String> vertices = new Vector();
10
11      // Méthode qui permet l ajout d un sommet au graphe.
12      public void addVertex(String name){
13          this.vertices.add(name); // Méthode qui permet d ajouter un sommet au graphe
14      }
15
16      // Cette méthode va tester si le sommet demandé existe dans le graphe. Si oui retourne le poids, sinon retourne 0.
17
18      public double edgeExist(String from_vertex, String toVertex){
19
20          // Ecrire votre code ici
21
22      }
23
24      // La méthode ci-dessous vous permet de générer une arête lorsque vous cherchez à en ajouter une à votre graphe. Elle
          fait aussi le test si jamais les sommets utilisés font partis du graphe ou non. Si non, elle va les ajouter au graphe. Cette
          méthode peut être utile dans la méthode new_edge
25
26      private void generate_edge(String from_vertex, String toVertex, double weight){
27          if (this.vertices.contains(from_vertex) & this.vertices.contains(toVertex)){
28              Edge new_edge = new Edge(from_vertex,toVertex,weight);
29              this.edges.add(new_edge);
30          }
31          else {
32              if (!this.vertices.contains(from_vertex)){
33                  this.vertices.add(from_vertex);
34              }
35              if (!this.vertices.contains(toVertex)){
36                  this.vertices.add(toVertex);
37              }
38              Edge new_edge = new Edge(from_vertex,toVertex,weight);
39              this.edges.add(new_edge);
40          }
41      }
42
43
44      public void updateWeight(String from_vertex, String toVertex, double weight){
45          // Ecrire votre code ici
46      }
47      // Méthode qui va ajouter l arête dans le graphe.
48      public void new_edge(String from_vertex, String toVertex, double weight){
49          // Ecrire votre code ici
50      }
51
52      // Méthode nous permettant de supprimer une arête du graphe.
53      public void delEdge(String from_vertex, String toVertex){
54          for(Edge edge : this.edges){
55              if (edge.from_vertex == from_vertex & edge.toVertex == toVertex){
56                  this.edges.remove(edge);
57                  System.out.println("Edge between " + from_vertex + " and " + toVertex + " has been deleted.");
58                  break;
59              }
60          }
61      }
62
63      // Fonction qui permet d imprimer les composants d un graphe
64      public void print(){
65          for(int i=0; i< this.vertices.size(); ++i){
```

```

66     System.out.println("The vertex number " + (i+1) + " has a value of: " + this.vertices.get(i));
67 }
68 for (Edge edge : this.edges){
69     edge.print();
70 }
71 }
72 }

```

1. La méthode `update_weight()` doit prendre en paramètres : le **sommet** d'origine, le **sommet** d'arrivée ainsi que le poids d'une **arête**. Si cette **arête** existe alors elle change son poids. Sinon, elle affichera une phrase indiquant que l'**arête** n'existe pas.
2. La méthode `edge_exist()` prendra en paramètre le **sommet** d'origine et le **sommet** d'arrivée. Si cette **arête** est dans le **graph** alors la méthode renvoie son poids, sinon, elle renvoie 0.
3. La méthode `new_edge` doit créer une instance de `Edge` et l'ajouter à l'ensemble `edges` si la connexion n'existe pas déjà. Si elle existe avec un autre poids mettre à jour le poids. Si elle existe de façon identique alors retournez la dans la console en utilisant un `System.out.print`. Enfin, si on est dans aucun des deux cas précédents utiliser la méthode `generate_edge` qui vous est donnée pour créer et ajouter cette arête au **graph**. La méthode `new_edge` aura pour paramètres : le **sommet** d'origine, le **sommet** d'arrivée, le poids.

#### Conseil

1. Utiliser une boucle `for` pour parcourir toutes les **arêtes** dans le **graph**. Faire un test sur les attributs de `Edge` pour changer le poids.
2. Il faut tester pour chaque **arête** ( itération) si elle est égale à celle donnée en paramètres.
3. Il faut utiliser les méthodes `edge_exist()`, `update_edge()` et `generate_edge()` pour écrire cette méthode. Il y a 4 tests à effectuer :
  - (a) Si l'**arête** existe.
  - (b) Si l'**arête** existante a le même poids que celle indiquée en paramètre de la méthode.
  - (c) Si l'**arête** existante n'a pas le même poids que celle indiquée en paramètre de la méthode.
  - (d) Utilisez le résultat de `edge_exist` pour simplifier ces tests.

#### >\_ Solution

##### Java :

```

1  import java.util.List;
2  import java.util.HashMap;
3  import java.util.Map;
4  import java.util.Vector;
5
6  class Edge {
7      public String fromVertex; // node de départ
8      public String toVertex; // node d arrivée (chaîne de caractère)
9      public double weight; // poids de l arête
10
11     public Edge(String fromVertex, String toVertex, double weight) {
12         this.fromVertex = fromVertex;
13         this.toVertex = toVertex;
14         this.weight = weight;
15     }
16
17     public void print(){
18         Map<String, String> edgeRep = new HashMap <String,String>(); // Création d un dictionnaire pour
19         pouvoir afficher une arête
20         edgeRep.put("From Vertex",this.fromVertex);
21         edgeRep.put("To Vertex",this.toVertex);
22         edgeRep.put("Weight", String.valueOf(this.weight));
23         System.out.println(edgeRep);
24     }
25 }

```

## >\_ Solution

Java :

```
1 class Graph {
2     // Attributs de la class graph
3     public List<Edge> edges = new Vector(); // Utilisation de vector car il faut que l on puisse rajouter ou
        supprimer des éléments de la liste
4     public List<String> vertices = new Vector();
5
6     // Méthode qui permet l ajout d un sommet au graph.
7     public void addVertex(String name){
8         this.vertices.add(name); // Méthode qui permet d ajouter un sommet au graph
9     }
10
11     // Cette méthode va tester si le sommet demandé existe dans le graph. Si oui retourne le poids, sinon retourne 0.
12     public double edgeExist(String fromVertex, String toVertex){
13         for (Edge edge : this.edges) {
14             if (edge.fromVertex == fromVertex & edge.toVertex == toVertex) {
15                 return edge.weight;
16             }
17         }
18         return 0;
19     }
20     // Cette méthode permet de générer une arête lorsque vous cherchez à en ajouter une à votre graphe. Elle fait
        aussi le test si jamais les sommets utilisés font partis du graphe ou non. Si non, elle va les ajouter au
        graphe. Cette méthode peut être utile dans la méthode newEdge
21     private void generateEdge(String fromVertex, String toVertex, double weight){
22         if (this.vertices.contains(fromVertex) & this.vertices.contains(toVertex)){
23             Edge newEdge = new Edge(fromVertex,toVertex,weight);
24             this.edges.add(newEdge);
25         }
26         else {
27             if (!this.vertices.contains(fromVertex)){
28                 this.vertices.add(fromVertex);
29             }
30             if (!this.vertices.contains(toVertex)){
31                 this.vertices.add(toVertex);
32             }
33             Edge newEdge = new Edge(fromVertex,toVertex,weight);
34             this.edges.add(newEdge);
35         }
36     }
37
38     public void updateWeight(String fromVertex, String toVertex, double weight){
39         for (Edge edge : this.edges){
40             if(edge.fromVertex == fromVertex & edge.toVertex == toVertex){
41                 edge.weight = weight;
42                 System.out.println("Weight of " + edge + "has been updated");
43             }
44             else {
45                 System.out.println("The vertex between the two nodes given does not exist, it will be created.");
46             }
47         }
48     }
49
50     // Méthode nous permettant de supprimer une arête du graph.
51     public void delEdge(String fromVertex, String toVertex){
52         for( Edge edge : this.edges ){
53             if (edge.fromVertex == fromVertex & edge.toVertex == toVertex){
54                 this.edges.remove(edge);
55                 System.out.println("Edge between " + fromVertex + " and " + toVertex + " has been deleted.");
56                 break;
57             }
58         }
59     }
```

## >\_ Solution

Java :

```
1  public void print(){
2      for(int i=0; i< this.vertices.size(); ++i){
3          System.out.println("The vertex number " + (i+1) + " has a value of: " + this.vertices.get(i));
4      }
5      for (Edge edge : this.edges){
6          edge.print();
7      }
8  }
9  // "Méthode qui va ajouter l'arête dans le graph."
10 public void newEdge(String fromVertex, String toVertex, double weight){
11     double testExistence = this.edgeExist(fromVertex,toVertex); // Peut valoir soit le point de l'arête soit 0 si elle
12     // n'existe pas.
13     if (testExistence == weight){
14         System.out.println("Edge between" + fromVertex + " and " + toVertex + " with the same weight already
15         exists");
16     }
17     else{
18         if (testExistence != 0) {
19             System.out.println("Edge between" + fromVertex + " and " + toVertex + "exists but with a different
20             weight and will be overwritten");
21             this.updateWeight(fromVertex, toVertex, weight);
22         }
23         else{
24             this.generateEdge(fromVertex, toVertex, weight);
25         }
26     }
27 }
```