Algorithmes et Pensée Computationnelle

Programmation orientée objet

Le code présenté dans les énoncés se trouve sur Moodle, dans le dossier **Ressources**.

1 Conception de classes pour coder des weighted graphs

1.1 Partie 1

Ici on cherche à pousser votre réflexion sur le processus de conception de classes pour implémenter un concept existant en dehors de la programmation. Pour cela il faut se placer du point de vue de l'utilisateur pour savoir ce dont il aurait besoin. Il faut aussi savoir de quoi est constitué le concept que l'on cherche à implémenter. C'est à dire, pour utiliser une analogie, les différentes briques qui sont utilisées pour construire le mur qui est votre objet final.
Cette partie de l'exercice a pour but de poser des questions théoriques qui sont une aide pour comprendre la logique de conception.
Question 1: (*Durée 15 minutes*) Exercice 1

1. Quels sont les 3 composants d'un weighted graph ?
2. Combien de classes sont nécéssaires pour représenter tous les composants du graph et le graph lui même ?
3. Sachant que vous avez une classe qui représente les arêtes et que les sommets sont représentés par des chaînes de caractères. Déterminer quels sont les attributs de la class graph.
4. Maintenant que vous avec le schéma général de votre classe graph, il faut déterminer quelles méthodes sont nécéssaires au fonctionnement de cet objet.

   Il faut donc réfléchir sur les questions suivantes :

   (a) Est-ce que l'utilisateur aura besoin de modifier l'objet une fois celui-ci initialisé ?

   (b) 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 ?

   (c) Est-il nécéssaire de définir des méthodes qui ne seront pas utiles pour l'utilisateur mais utiles pour éviter la redondance du code ?

Une fois ces questions répondues, le schéma des méthodes nécessaires devient plus claire.
Il reste enfin à traiter l'implémentation de tout cela. C'est à dire, comment coder la classe pour qu'elle soit fonctionnelle.

Conseil

1. Il faut décomposer ce qui constitue un graph. Revenir sur le cours de la semaine 7 sur moodle.
2. Réfléchir aux différents objets du type les arêtes qui consituent un graph. Réflechir si ils ont besoin d'être représenté avec une classe ou est-ce que les types de "base" sont suffisants.
3. Les attributs sont les briques qui constituent votre classe. Il faut donc poser les variables nécéssaires et leur type.
4. Penser de la même manière que lorsque vous utilisez une application et critiquez (en bien ou en mal) les fonctionnalitées de celle-ci. Ici, il faut réfléchir à ce qu'une personne qui va utiliser votre "application" (la classe) aurait besoin.

Solution

Certaines questions n'ont pas qu'une réponse car il y a toujours plus d'une manière d'implémenter une classe. Voici une façon de répondre :

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 Edges 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.
4. Les réponses à cette partie sont courtes mais assez essentielles pour savoir quelles méthodes coder :
   (a) Oui, l'utilisateur voudra sûrement rajouter des arêtes, des sommets ou potentiellement changer le poids d'une arête. Il faut donc lui laisser cette possibilité
   (b) Les graphs sont souvent accompagnés par des algorithmes de recherche, par exemple l'algorithme de Kruskal pour les weighted graphs. Il est donc utile de pouvoir identifier si un sommet ou une arête sont des composants de l'instance.
   (c) Certaines méthodes pourraient avoir un code très long si on ne ségmentarise pas leur contenu. Cela peut entraver la relecture de votre code par vous ou par un tiers.
   De plus dans un cadre d'optimisation, il faut éviter un maximum la redondance du code. Il est donc de bonne pratique de définir des méthodes qui vont éviter cela dans une classe.

1.2 Partie 2

Le code pour la classe Edges est fourni dans le dossier ressources sur moodle. Utiliser le fichier Main.java dans le dossier ressources sur moodle pour effectuer des tests. Il devrait retourner :

```
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 2: (*Durée 20 minutes*) Exercice 2
Voici une partie de la classe graph codée. Implémentez les méthodes "update$_w eight$", "$new_e dge$" et "" $edge_e xist$".
Java :

```java
import java.util.List;
import java.util.HashMap;
import java.util.Vector;

public class graph_empty {

    // Les attributs de la class graph
    public List<Edges> edges = new Vector();   // Utilisation de vector car il faut que l'on puisse rajouter ou supprimer des é
        léments de la liste
    public List<String> vertices = new Vector();



    // Methode qui permet l'ajout d'un sommet au graph.
    public void add_vertex(String name){
        this.vertices.add(name); // Méthode qui permet d'ajouter un sommet au graph
    }

    // Cette méthode va tester si le sommet demandé existe dans le graph. Si oui retourne le poids, sinon retourne 0.
    public double edge_exist(String from_vertex, String to_vertex){
        // crire votre code ci-dessous



        // Fin de la zone d'écriture
    }
    // L'implémentation de la méthode ci dessous n'est pas importante pour vous à comprendre. Elle vous est utile pour
    // générer une arête lorsque vous cherchez à en ajouter une à votre graph. Elle fait aussi le test si jamais
    // les sommets utilisés font partis du graph ou non. Si non, elle va les ajouter au graph. Cette méthode peut
    // être utile dans la méthode new_edge
    private void generate_edge(String from_vertex, String to_vertex, double weight){
        if (this.vertices.contains(from_vertex) & this.vertices.contains(to_vertex)){
            Edges new_edge = new Edges(from_vertex,to_vertex,weight);
            this.edges.add(new_edge);
        }
        else {
            if (!this.vertices.contains(from_vertex)){
                this.vertices.add(from_vertex);
            }
            if (!this.vertices.contains(to_vertex)){
                this.vertices.add(to_vertex);
            }
            Edges new_edge = new Edges(from_vertex,to_vertex,weight);
```

```
            this.edges.add(new_edge);
        }

    }

    public void update_weight(String from_vertex, String to_vertex, double weight){
        // crire votre code ci-dessous









        // Fin de la zone d'écriture
    }
    // Méthode qui va ajouter l'arête dans le graph.
    public void new_edge(String from_vertex, String to_vertex, double weight){
        // crire votre code ci-dessous









        // Fin de la zone d'écriture
    }

    // Méthode nous permettant de supprimer une arête du graph.
    public void del_edge(String from_vertex, String to_vertex){
        for( Edges edge : this.edges ){
            if (edge.from_vertex == from_vertex & edge.to_vertex == to_vertex){
                this.edges.remove(edge);
                System.out.println("Edge between " + from_vertex + " and " + to_vertex + " has been deleted.");
                break;
            }

        }

    }

    // Fonction qui permet d'imprimer les composants d'un graph
    public void print(){
        for(int i=0; i< this.vertices.size(); ++i){
            System.out.println("The vertex number " + (i+1) + " has a value of: " + this.vertices.get(i));
        }
        for (Edges edge : this.edges){
            edge.print();
        }
    }
}
```

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 imprimera une phrase indiquant que cette arête n'existe pas.
2. La méthode "edge exist" qui va prendre en paramètre le sommet d'origine et le sommet d'arrivée. Si cette arête est dans le graph alors la méthode ressort son poids, 0 sinon.
3. La méthode "new edge" doit créer une instance de Edges 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 avec 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 comme 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 Edges pour changer le poids.
2. Il faut tester pour chaque arête (itération) si elle est égale à celle rentrée 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 celui indiqué en paramètre de la méthode.
   (c) Si l’arête existante n’a pas le même poids que celui indiqué en paramètre de la méthode.
   (d) Utiliser 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
5  public class Edges {
6     public String from_vertex; // node de départ
7     public String to_vertex;  // node d'arrivée ( chaîne de caractère)
8     public double weight;  // poids de l'arête
9
10    public Edges(String from_vertex, String to_vertex, double weight) {
11       this.from_vertex = from_vertex;
12       this.to_vertex = to_vertex;
13       this.weight = weight;
14    }
15
16    public void print(){
17       Map<String, String> edge_rep = new HashMap <String,String>(); // Création d'un dictionnaire pour
           pouvoir afficher une arête
18       edge_rep.put(”from_vertex”,this.from_vertex);
19       edge_rep.put(”to_vertex”,this.to_vertex);
20       edge_rep.put(”weight”, String.valueOf(this.weight));
21       System.out.println(edge_rep);
22    }
23
24 }
```

Solution

Java :

```java
public class graph {
    // Les attributs de la class graph
    public List<Edges> edges = new Vector();   // Utilisation de vector car il faut que l'on puisse rajouter ou supprimer des éléments de la liste
    public List<String> vertices = new Vector();



    // Methode qui permet l'ajout d'un sommet au graph.
    public void add_vertex(String name){
        this.vertices.add(name); // Méthode qui permet d'ajouter un sommet au graph
    }

    // Cette méthode va tester si le sommet demandé existe dans le graph. Si oui retourne le poids, sinon retourne 0.
    public double edge_exist(String from_vertex, String to_vertex){
        for (Edges edge : this.edges) {
            if (edge.from_vertex == from_vertex & edge.to_vertex == to_vertex) {
                return edge.weight;
            }
        }
        return 0;
    }
    // L'implémentation de la méthode ci dessous n'est pas importante pour vous à comprendre. Elle vous est utile pour
    // générer une arête lorsque vous cherchez à en ajouter une à votre graph. Elle fait aussi le test si jamais
    // les sommets utilisés font partis du graph ou non. Si non, elle va les ajouter au graph. Cette méthode peut
    // être utile dans la méthode new_edge
    private void generate_edge(String from_vertex, String to_vertex, double weight){
        if (this.vertices.contains(from_vertex) & this.vertices.contains(to_vertex)){
            Edges new_edge = new Edges(from_vertex,to_vertex,weight);
            this.edges.add(new_edge);
        }
        else {
            if (!this.vertices.contains(from_vertex)){
                this.vertices.add(from_vertex);
            }
            if (!this.vertices.contains(to_vertex)){
                this.vertices.add(to_vertex);
            }
            Edges new_edge = new Edges(from_vertex,to_vertex,weight);
            this.edges.add(new_edge);
        }

    }

}
```

Solution

Java :

```
1  public void update_weight(String from_vertex, String to_vertex, double weight){
2      for (Edges edge : this.edges){
3      if(edge.from_vertex == from_vertex & edge.to_vertex == to_vertex){
4      edge.weight = weight;
5      System.out.println("Weight of" + edge + "has been updated");
6          }
7      else {
8      System.out.println("The vertex between the two nodes given does not exist, it will be created.");
9          }
10     }
11
12 }
13 // Méthode qui va ajouter l'arête dans le graph.
14 public void new_edge(String from_vertex, String to_vertex, double weight){
15     double test_existence = this.edge_exist(from_vertex,to_vertex); // Peut valoir soit le point de l'arête soit 0 si
         elle n'existe pas.
16     if ( test_existence == weight){
17     System.out.println("Edge between" + from_vertex + " and " + to_vertex + " with the same weight already
         exists");
18     }
19     else{
20     if ( test_existence != 0) {
21     System.out.println("Edge between" + from_vertex + " and " + to_vertex + "exists but with a different weight
         and will be overwritten");
22     this.update_weight(from_vertex, to_vertex, weight);
23     }
24     else{
25     this.generate_edge(from_vertex, to_vertex, weight);
26         }
27     }
28
29 }
30
31 // Méthode nous permettant de supprimer une arête du graph.
32 public void del_edge(String from_vertex, String to_vertex){
33     for( Edges edge : this.edges ){
34     if (edge.from_vertex == from_vertex & edge.to_vertex == to_vertex){
35     this.edges.remove(edge);
36     System.out.println("Edge between " + from_vertex + " and " + to_vertex + " has been deleted.");
37     break;
38         }
39     }
40 }
41
42 public void print(){
43     for(int i=0; i< this.vertices.size(); ++i){
44     System.out.println("The vertex number " + (i+1) + " has a value of: " + this.vertices.get(i));
45         }
46     for (Edges edge : this.edges){
47     edge.print();
48         }
49     }
50 }
```