
Université
de Montréal

# Projet ORAlgorithmics

# Projet ORAlgorithmics


Département d’informatique

# Projet ORAlgorithmics


Département d’informatique
et de recherche Opérationnelle

# Une collection d’algorithmes de recherche opérationnelle en code source libre

Présenté par

Anouar Ben Dhaou

Boubkre El Allani

Khalid Kandouli

Rida Bouchaib

# *Plan de la présentation*

I- Contexte

II- Introduction.

III- Spécification technique du projet .

IV- Comment utiliser ORAlgorithmics?

V- Tests JUnit .

VI- Discussion & Remerciement .

# Contexte

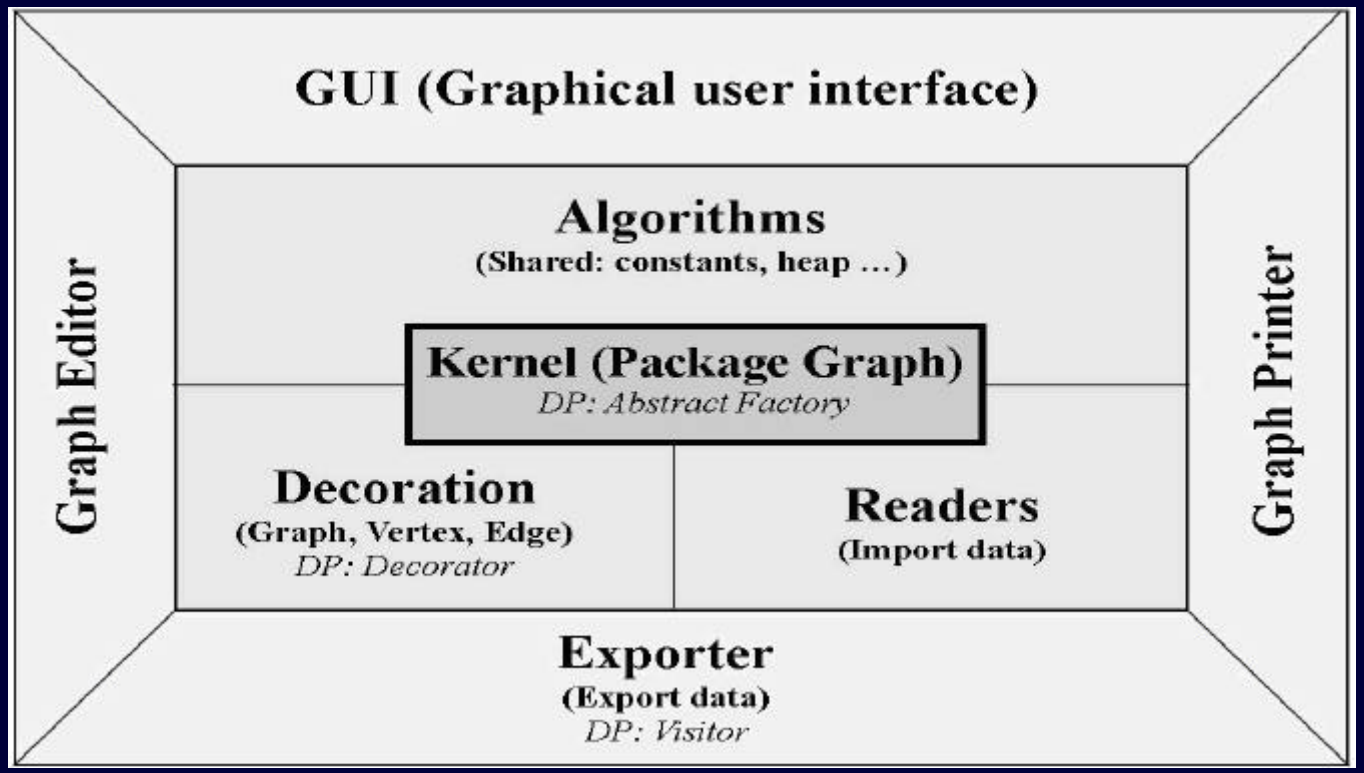
- La recherche opérationnelle est un domaine important en informatique .
- Nécessité d’un outil open source.

# Introduction

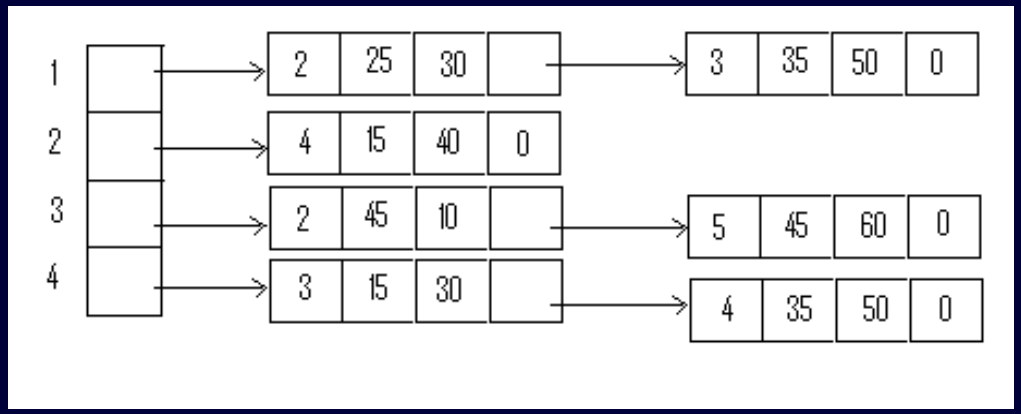
- Le but du projet : Mettre en place un outil ouvert qui permet l'implantation d'une collection d'algorithmes.
- Deux objectifs : servir l'utilisateur et le programmeur.
- Le noyau de cet outil est en fait un meta-modele pour décrire les graphes et les méthodes de base pour les manipuler.
- Importance des patrons de conception: Decorator, Visitor et Factory.
- Algorithmes implementes: BFS, DFS, Bellman, Dijkstra, Floyd, Kruskal, Postier Chinois, Prim.

# Spécification technique du projet

## Architecture du projet

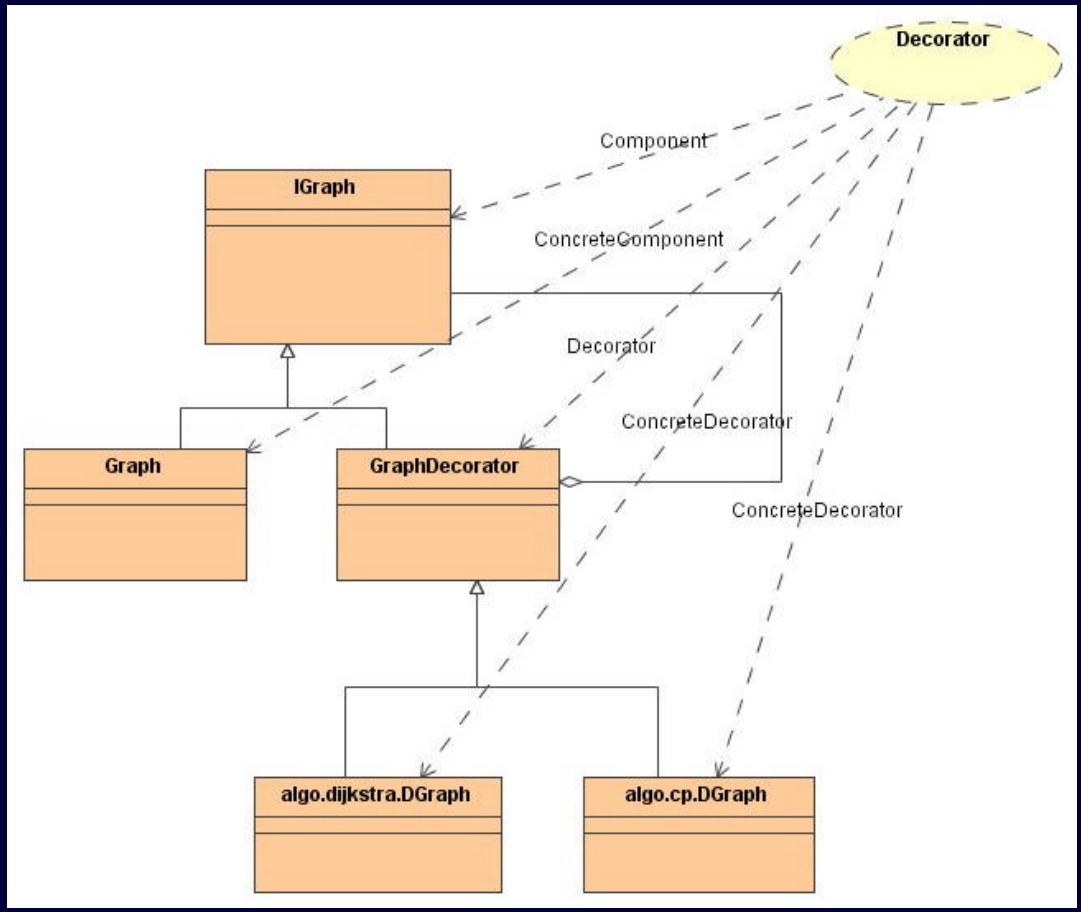


# Représentation interne des graphes



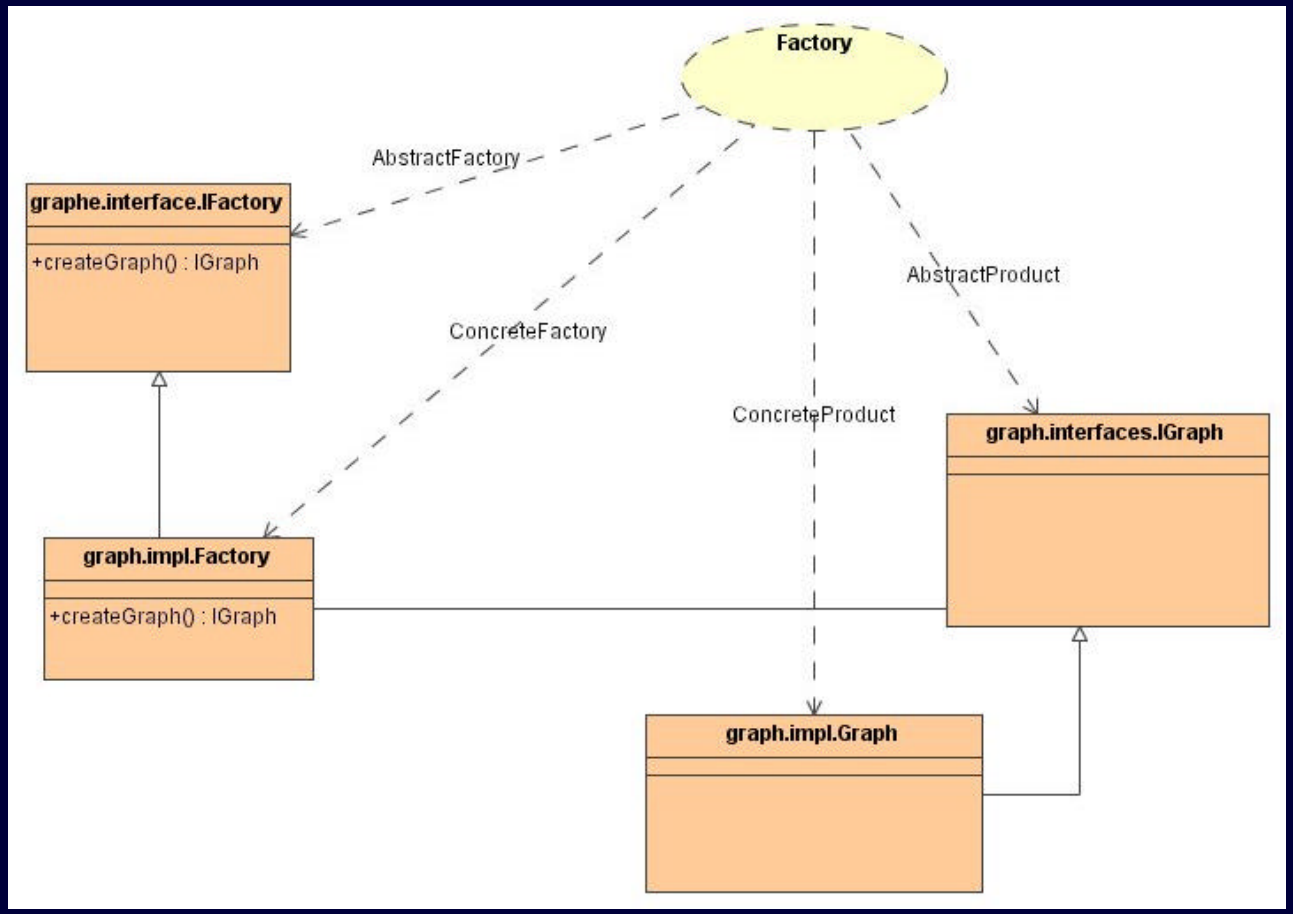

*Exemple de liste d'adjacence*

# Patron de conception (Décorateur)

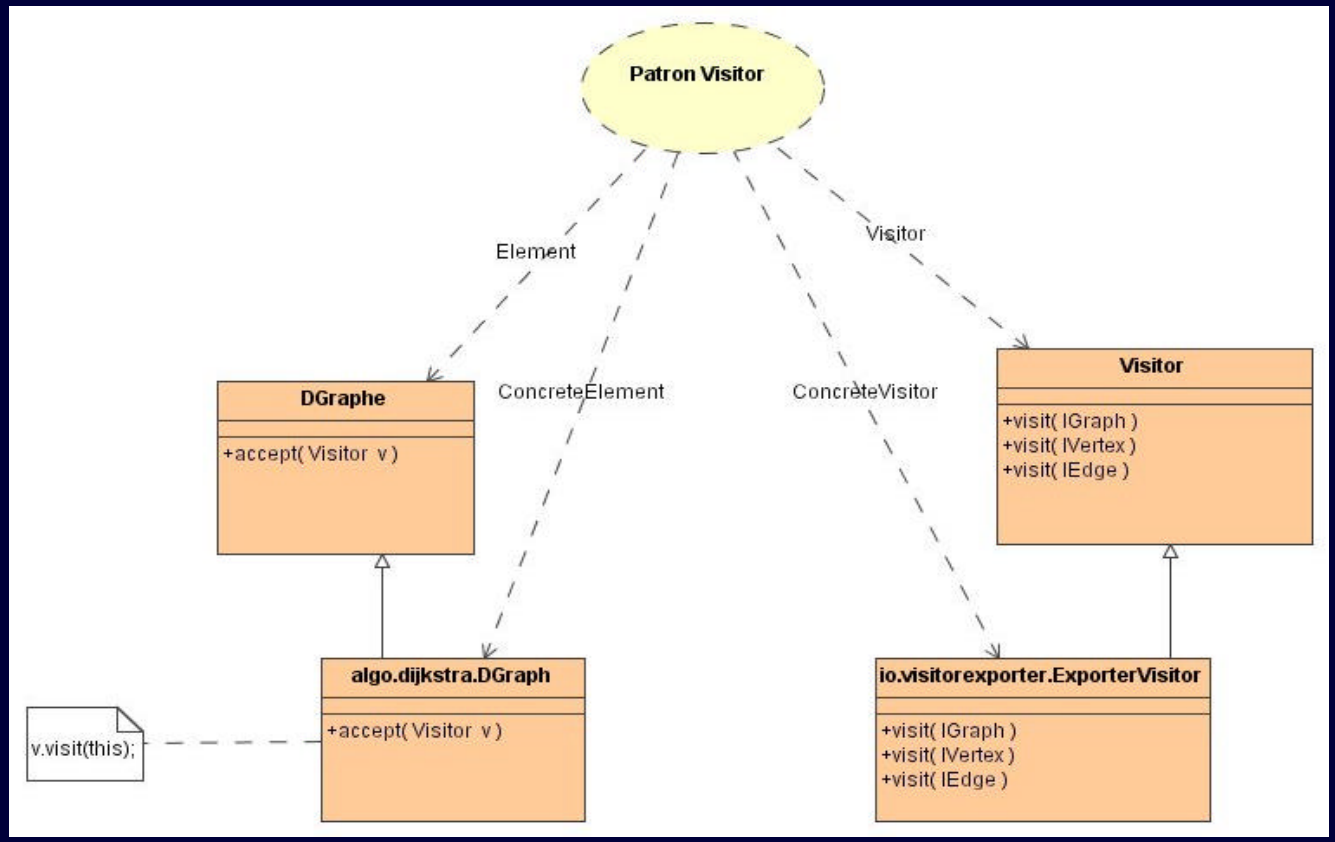


# Patron de conception

## (Abstract Factory)



# Patron de conception (Visiteur)



# Structure du fichier d'importation/exportation

| Exemple de fichier d'importation | Description |
|---|---|
| Réseau routier | Titre du graphe |
| ##4 | Nombre de sommets (doit commencer par '##') |
| Montréal | Identificateur de sommet |
| Québec | Idem |
| Ottawa | Idem |
| Toronto | Idem |
| ##5 | Nombre d'arcs (doit commencer par '##') |
| route1 Montréal Toronto 10 | Identificateur arc, source, destination, poids (décoration) |
| route2 Montréal Québec 20 | Idem |
| route3 Québec Toronto 30 | Idem |
| route4 Québec Ottawa 40 | Idem |
| route5 Ottawa Toronto 50 | Idem |

# Comment utiliser ORAlgorithms ?

Démonstration

Comment ajouter un nouvel algorithme ?

Nom de l’algorithme: ***algoNouveau.***

(Présentation code)

# Phase 1: Création du package

Dans le répertoire **src/algo**, ajouter un nouveau répertoire sous le nom *algoNouveau*.
Il représentera le package propre à *algoNouveau*.

# Phase 2: Création du reader.java

- Préparer un reader pour l'importation du fichier correspondant au nouvel algorithme. (On peut reprendre l'un des reader déjà existant)

- Dans ce reader, mettre les bonnes valeurs aux deux constantes **NBCOLVERTEX** et **NBCOLEDGE** correspondant respectivement au nombre de colonnes de sommets et au nombre d'arcs;

- Personnaliser les paramètres d'instanciation du constructeur *DVertex* dans la méthode *createVertex* et du constructeur *DEdge* dans la méthode *createEdge.*

# Phase 3: Étape de décoration

- Créer la classe *DVertex.java* qui doit étendre la classe *VertexDecorator*
- Personnaliser les arguments du constructeur *DVertex*.
- Ajouter les attributs correspondants à la décoration comme variables de classe.
- Ajouter les méthodes relatives à la décoration (comme *set* et *get*)
- Ajouter la méthode *toString*, qui est appelée lors de l'édition graphique.
- Ajouter la méthode *accept(Visitor v)* relative au patron de conception *Visitor*.
  - Ajouter dans le fichier *io/visitorexporter/ExporterVisitor.java* l'implémentation de la méthode déclarée dans l'interface.

**Remarque: même procédure pour DEdge et DGraph**

# Phase 4: Implémentation de l'algorithme

- Créer la classe *algoNouveau.java*, qui implémente concrètement le nouvel algorithme.

- Personnaliser les arguments du constructeur algoNouveau qui doit au moins recevoir un DGraph.

# Phase 5: MAJ du IHM

- oralgo.java.
- Ajouter dans le tableau *algoCollection* le nom de l'algorithme.
- Ajouter la méthode *traitalgoNouveau* qui lance l'exécution de l'algorithme.

```
void traitalgoNouveau () {
      algo.algoX.Reader  r  =  new  algo.algoX.Reader(getName());
      //…
     if (r.isReadOk()) {
                  algoX dj = new algoX(r.getGraph());
                  areaResult.setText(dj.getResult());
                  //…
```

# Phase 6: MAJ du IHM (suite)

- Dans la partie qui traite le bouton *start* de la méthode *actionPerformed*, ajouter l'appel à la méthode *traitalgoNouveau*.

```
/* indexAlgoNouveau : est l'index de algoNouveau  dans la zone
        algorithme de l'interface graphique*/
if (allAlgo[indexAlgoNouveau].isSelected()) {
          traitalgoNouveau();
}
```

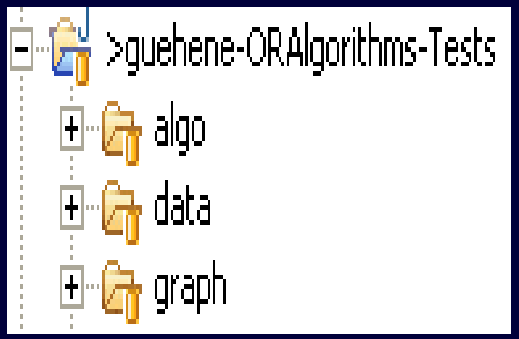
# Test JUnit

## Structure du projet test

**algo** : package des tests des algorithmes.

**data** : les fichier d'import/export.

**graph** : contient les tests du graphe de base



# Test JUnit (Suite)

## Statistiques des tests

| Package | Nombre de tests |
|---|---|
| graph | 895 |
| algo.bellman | 112 |
| algo.bfs | 85 |
| algo.dfs | 98 |
| algo.chinesepostman | 187 |
| algo.dijkstra | 171 |
| algo.floyd | 181 |
| algo.kruskal | 179 |
| algo.prim | 194 |
| algo.shared.heap | 28 |
| io (import, export) | 121 |
| **Total** | **2251** |

Démonstration

# Discussion

## Difficultés rencontrées :

- Le Postier chinois (pour un cycle non Eulérien)
- Application des patrons de conception

## Points forts du logiciel :

- L’implémentation des algorithmes est indépendante du noyau.
- L’interface graphique est indépendante de la structure interne des graphes et des algorithmes.
- Prise en compte de la complexité de certaines méthodes afin de les optimiser.  monceau (Heap.java)  en ordre de log (n).

# Discussion

## Amélioration et ajouts possibles/futures

- Implémenter d’autres algorithmes de recherche opérationnelle.
- Choisir dynamiquement la structure de données du graphe lors de l’instanciation, en fonction du nombre d’arcs et du nombre de sommets.
- Appliquer le MVC au niveau du GUI.
- Utiliser le parallélisme (exploiter le Heap : gere l’acces aux  sections critiques).
- Implémenter un graphe non orienté basé sur la structure actuelle (graphe oriente).
- Impression du graphe

# Discussion (Suite)

- **Profits personnels.**
- L’apport de ce projet dans notre formation est très important.
- Appliquer et d’approfondir plusieurs notions : génie logiciel, programmation java, recherche opérationnelle, structures de données et enfin l’algorithmique.
- Familiarisation avec l’environnement de développement Eclipse, JUnit et CVS.
- **Remerciement.**