Rapport TP2 évolution et restructuration

Introduction

J'ai créer 2 versions du TP une première version qui fait un couplage du graphe avec des liens Unidirectionnel et une deuxième qui fait un couplage avec des liens Bidiréctionnel.

Mon TP contient 4 projets :

- TP2_restructuration qui contient les exo 1 et 2 avec un couplage Bidirectionnel.
- Tp2_rustructuration_Spoon qui contient l'exercice 3 avec un couplage Bidirectionnel.
- TP_restructuration_NonOriente qui contient les exo 1 et 2 avec un couplage unidirectionel.
- TP_restructutation_spoon_NonOriente qui contient l'exercice 3 avec un couplage unidirectionel.

Le TP contient aussi un dossier "Projet de test" qui contient des projets pour tester l'application.
Il contient aussi un dossier "Vidéo démonstration" contenant des applications montrant comment utilisé l'application.

Explications code Exo1 et 2 Bidiréctionnel

La classe principale de l'application est la classe Parser qui permet de créer l'AST de l'application et de récupérer les méthodes et leurs invocations.

J'ai fait pour le TP un graphe de couplage Unidirectionnel.
Il y a un arc de A->B si la classe A appel une méthode de la classe B.

Pour calculer le couplage entre 2 classes la moyenne entre l'arc A->B et B->A ce qui me donne le couplage entre 2 classes.

Le couplage unidirectionnel me semblait meilleur que le bidirectionnel car cela permet de bien voir si c'est surtout la classe A qui appelle les méthodes de la classe B ou l'inverse. Alors que le bidirectionnel avait comme défaut de ne pas mettre en évidence le sens des appels de méthodes. Cette méthode du graph bidirectionnelle me paraissait donc plus cohérente.

Le programme dessine le graph d'appel sous forme de PNG.

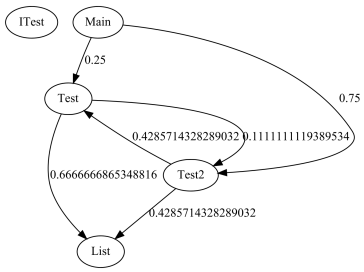


FIGURE 1 – Example représentation graph

Il dessine aussi un graph représentant le regroupement (clustering) hiérarchique à une étape i de l'algorithme passé en paramètre. Les clusters sont représentés sous forme de noeuds qui ont comme nom l'ensemble des noms des classes que le cluster contient. Par exemple si un cluster contient une classe A B et C le nom du cluster sera A :B :C.

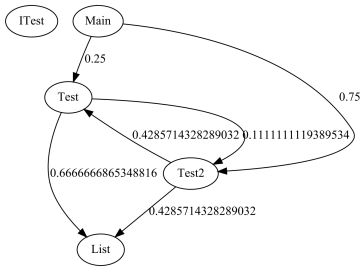


FIGURE 2 – Exemple représentation de l'algorithme Cluster a 0 étape

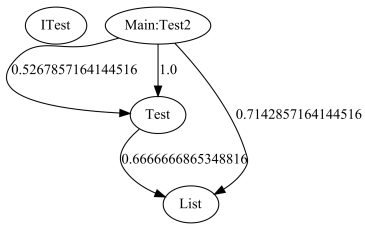


FIGURE 3 – Exemple représentation de l'algorithme Cluster a 1 étape

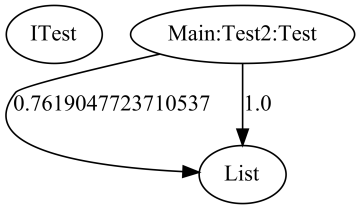


FIGURE 4 – Exemple représentation de l'algorithme Cluster a 2 étape

FIGURE 5 – Exemple représentation de l'algorithme Cluster a 3 étape

Il dessine aussi le dendogram sous forme de graph (j'ai utilisé la librairie JgraphT pour dessiner le dendogram la représentation n'est donc pas parfaite comme sur l'exemple du TP de plus l'image ne se dessine pas sur des gros projets car la taille de l'image sera trop grosse à dessiner pour JgraphT (dépassement mémoire) l'erreur se produit sur l'exemple de projet test).

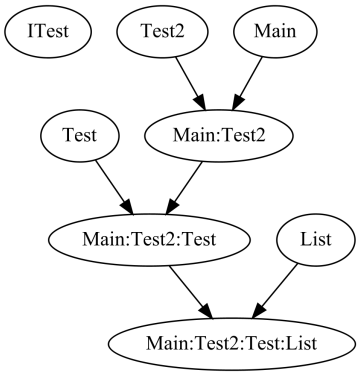


FIGURE 6 – Exemple de dendogramme

Je dessine de plus un graph représentant l'algorithme d'identification des groupes de classes couplées avec une valeur de CP passée en paramètre.

Cette fonction parcourt récursivement l'arborescence d'un dossier et retourne tous les liens des fichiers java de ce dossier.

```
1 public ArrayList<String> Scan(String Path)
```

Retourne dans un String le contenu d'un fichier passé en paramètre

```
1  public String readFileToString(String filePath)
```

Intialise CompilationUnit en fonction d'un fichier java passé en paramètre

```
1  public CompilationUnit AST(String Path)
```

Cette fonction permet en fonction d'un arraylist de lien de fichier java (qui sera créé par la fonction Scan(String Path)) et qui retourne le graph d'appel avec les différents couplages sur chaque arrêt.

```
1     public DirectedWeightedPseudograph Parse(ArrayList<String> f)
```

Cette fonction permet de créer une copie d'un graph.

```
1     public DirectedWeightedPseudograph copygraph(
      DirectedWeightedPseudograph<String, MyWeightedEdge> graph)
```

Cette fonction retourne un couple de noeuds avec le plus grand couplage d'un graph passé en paramètre.

```
1     public String[] ClusterProche(DirectedWeightedPseudograph<String,
      MyWeightedEdge> graph)
```

Cette fonction prend un graph et deux noeuds et va retourner un nouveau graphe avec un noeud contenant les 2 noeuds passés en paramètre.

```
DirectedWeightedPseudograph CreateCluster(DirectedWeightedPseudograph graph
    , String[] ex)
```

Cette fonction prend un graph et un entier i et va retourner un graph avec les diférents clusters à une etape i de l'algorithme en utilisant les fonctions écrites précédemment.

```
DirectedWeightedPseudograph HierarchieCluster(DirectedWeightedPseudograph
    hierarchie, int i)
```

Cette fonction permet de créer un nouveau noeud à partir du noeud "cible" et en lui suprimmant la classe "remove" du cluster.

```
String newNode(String remove,String replace)
```

Cette fonction utilise la methode précédente pour intégrer le nouveau noeud dans le graph et suprimer le noeud "cible"

```
DirectedWeightedPseudograph replaceVertex(String remove,String replace,
    DirectedWeightedPseudograph module)
```

Cette fonction retourne le couplage moyen d'un module en fonction du nom du module et du graph de couplage passé en paramètre.

```
float moyennecouplage(String module,DirectedWeightedPseudograph graph)
```

Cette fonction retourne le noeud d'un cluster qui est le moins couplé en fonction du nom du cluster et du graph de couplage.

```
String VertexMustBig(String module,DirectedWeightedPseudograph graph)
```

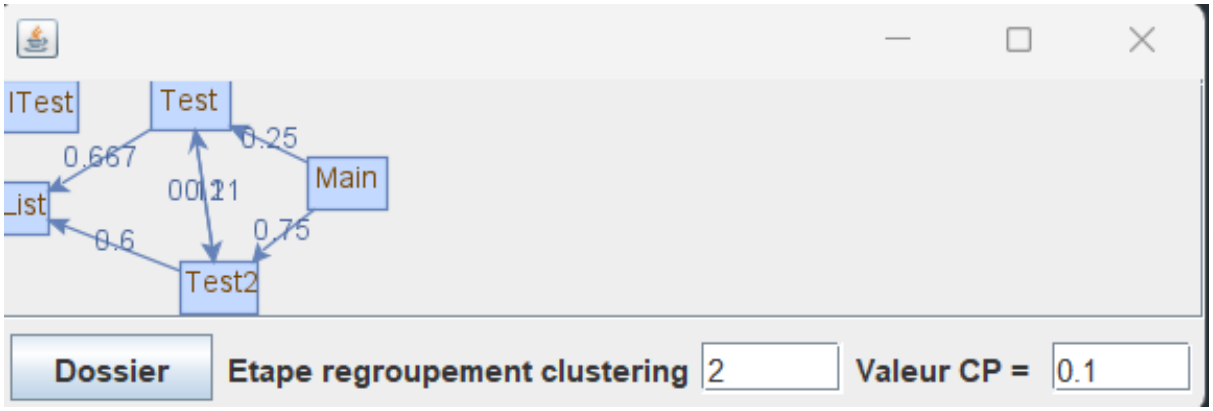
Cette fonction répond à la question 2 de l'exo2 et crée différents modules en respectant que le nombre de module ne dépasse pas M/2 où M est le nombre de classe. Que chaque classe de chaque module face partie de la même branche du dendogramme. Et que le nombre moyen de couplage par module soit supérieur a CP.

```
DirectedWeightedPseudograph GroupeClasse(DirectedWeightedPseudograph
    hierarchie, float CP)
```

Cette fonction permet de créer le dendogram à partir d'un graph d'appel donné en paramètre

```
1 SimpleDirectedGraph DrawDendrogramme(DirectedWeightedPseudograph hierarchie
    )
```

Interface



En cliquant sur le bouton "Dossier" s'ouvre une interface pour selectionner le projet à parcourir recursivement.

Le champs "Etape regroupement clustering" indique à quelle étape de l'itération on veut arreter l'algorithme de regroupement hiérarchique et le résultat sera ensuite dessiné sous forme d'un PNG dans le dossier "image". Le champ de "Valeur CP" indique quelle moyenne minimale de couplage doit avoir chaque cluster. Le resultat sera ensuite dessiné sous forme de PNG dans le dossier "image".

Exo 3 Bidiréctionnel

L'exo 3 contient les mêmes classes que l'Exo1 et 2 sauf que pour la fonction Parser on utilise les fonctions fournies par Spoon.

Explications code Exo1, 2 et 3 Unidiretionnel

Ces projets contiennent exactement les mêmes classes que la version Bidirectionnel mais en utilisant un graph Unidiretionnel.