DALENCOURT Alex
GOLDACK Ludovic

TP3 Coloration d'arbres

Les implémentations d'algorithmes se trouvent dans le package algorithmes. Les algorithmes implémenté sont :

- Welsh Powel

- Brigham Dutton

- RLF

Implémentation Welsh Powel

Avant d'exécuter l'algorithme les nœuds sont trié par ordre décroissant de degré. L'algorithme parcours un a un chaque nœud et attribut la première valeur disponible parmi tous les nœuds voisins.

```java
@Override
public Graphe run(final Graphe input) {
    final int baseColor = 0;
    input.sortNodeByDegree();
    // Parcours de tous les noeud
    for (final Node node : input.getNodeList()) {
        int color = baseColor;
        for (Node tmp : node.getNeighborsOrderByColor()) {
            if (tmp.getColor() != null && tmp.getColor() > color) {
                break;
            } else if (tmp.getColor() != null && color <= tmp.getColor()) {
                color++;
            }
        }
        node.setColor(color);
    }
    return input;
}
```

Implémentation Brigham Dutton

La recherche de deux nœuds contractable se réalise grâce à la méthode contractable. Cette dernière regarde parmi toutes les association de nœuds les associations réalisable et sauvegarde à chaque fois la contraction qui présent le plus de liens communs contracté (le cacul du nombre de liens communs

contracté est délégué à la méthode finCommonTarget de graphe).

```
private Node[] contractable(final Graphe graphe) {
    int max_common_link = -1;
    Node src = null;
    Node tgt = null;
    List<Node> nodeList = graphe.getNodeList();
    // Parcours avec curseur pour ne pas traiter un lien deux fois
    for(int i = 0; i < nodeList.size() - 1; i++){
        for(int j = i + 1; j < nodeList.size(); j++){
            // Contrôle de l'existance d'un lien entre deux noeuds
            if(!graphe.existLink(nodeList.get(i),nodeList.get(j))){
                int max = graphe.findCommonTarget(nodeList.get(i), nodeList.get(j));
                // Si le nombre de voisins commun est supérieur au max rencontré sauv
                if(max > max_common_link){
                    max_common_link = max;
                    src = nodeList.get(i);
                    tgt = nodeList.get(j);
                }
            }
        }
    }
    return max_common_link > 0 ? new Node[]{src,tgt} : null;
}
```

Les nœuds ainsi trouvé sont contracté dans un métanoeud héritant de nœud afin de considéré ce dernier comme étant un nouveau nœud et qui permet de contenir une liste de nœuds contracté en plus des éléments de base. Enfin, les références des nœuds contractés sont remplacés dans le graphe par la référence du nouveau métanoeud (méthode replaceTuple de graphe)

```
public Graphe run(final Graphe input) throws NodeNotFoundException {
    Graphe copy = new Graphe(input);
    Node[] tuple;
    // Contraction des noeuds tant que c'est possible
    while((tuple = contractable(copy)) != null){
        // Création de la contraction
        MetaNode meta = new MetaNode(tuple);
        // Remplacement des noeuds contractés
        copy.replaceTuple(tuple, meta);
    }
    // Reconstruction des couleurs
    int color = 0;
    for(Node node : copy.getNodeList()){
        node.setColor(color++);
    }
    return input;
}
```

La dernière partie réalise l'attribution des couleurs. Le principe de sélection de couleur est que tout nœud n'ayant pas pu être concaténé a une couleur à lui. Enfin, l'attribution de la couleur fonctionne

sur le principe des références d'objets java ainsi attribuer une couleur à un métanoeud attribuera la couleurs à tous les nœuds qu'ils regroupe.

Implémentation RLF

Avant de commencer le programme on tri les nœuds dans l'ordre descendant des degrés. Le traitement continue tant qu'il existe des nœuds. Le programme traitera toujour le premier nœud de plus haut degré restant. Ainsi lorsqu'ils seront contracté deux nœuds dont un de plus haut degré donnera un nœud de plus haut degré. Le regroupement créé remplacera les nœuds du tuple et sera placé en première position de la liste. Si un nœud ne peux plus être contracté il sera supprimer de la liste des nœuds et une nouvelle couleur lui sera attribuée.

```
public Graphe run(final Graphe input) throws NodeNotFoundException {
    final Graphe copy = new Graphe(input);
    // On trie les noeuds par degré décroissant
    copy.sortNodeByDegree();
    int color = 0;
    // Tanqu'il existe de noeuds on réalise l'algorithme
    while(!copy.getNodeList().isEmpty()){
        // On sélectionne le noeud de plus grand degré
        Node node = copy.getNodeList().get(0);
        // On cherche un noeud contrctable selon l'algorithme d'élect
        Node contract = findContractNode(copy, node);
        if(contract != null) {
            // Si il existe un noeud contractable on réalise la contr
            Node[] tuple = new Node[]{node, contract};
            MetaNode metaNode = new MetaNode(tuple);
            // Le meta-Noeud est positionné dans la liste des noeuds
            // Le nouveau noeud reste le noeud de plus haut degré : (
            copy.replaceTuple(tuple, metaNode, idx: 0);
        } else {
            // Si plus aucun noeud ne peut être contracté il existe u
            // Suppression du noeud
            copy.removeNode(node);
            // Attribution de la première couleur disponible à tous l
            node.setColor(color++);
        }
    }
    return input;
}
```

La recherche de la meilleur contraction possible fonctionne comme pour Brigham Dutton mais ne nécessite qu'un seul parcours car nous avons un nœud invariant.

DALENCOURT Alex
GOLDACK Ludovic

```java
private Node findContractNode(final Graphe graphe, final Node node){
    int max_common_link = -1;
    Node tgt = null;
    List<Node> nodeList = graphe.getNodeList();
    // Parcours des autres noeuds du graphe
    for(int j = 1; j < nodeList.size(); j++){
        // Contrôle de l'existance d'un lien les noeuds
        if(!graphe.existLink(node,nodeList.get(j))){
            int max = graphe.findCommonTarget(node, nodeList.get(j));
            // Si le nombre de voisins commun est supérieur au max re
            if(max > max_common_link){
                max_common_link = max;
                tgt = nodeList.get(j);
            }
        }
    }
    return tgt;
}
```

Tests de performance

Chaque programme mentionné ici se trouve dans le package mains.

Afin d'assurer la validité des algorithmes le programme AlgorithmeValidator permet de calculer un graphe coloré et de l'afficher suivant la méthode choisie. Il restera à l'utilisateur de reconstruire le graphe afin de réaliser un contrôle manuel. Le programme peut être exécuté via l'exécutable AlgorithmeValidator.jar :

java -jar {Chemin vers AlgorithmeValidator.jar} {Algorithme choisi} {nombre de nœuds du graphe} {probabilité d'une arête}

Les paramètres possibles pour le choix de l'algorithme est :

- wp : Welsh Powel

- bd : Brigham Dutton

- rlf : RLF

Afin de réaliser les tests de performance deux programmes ont été réalisés PerformanceComparator et PerformanceComparatorNPArgument. Le premier programme réalise la moyenne du temps d'exécution des trois algorithmes sur 50 exécutions pour des graphes aléatoires de 100 nœuds avec une probabilité d'arête variant entre les valeurs suivantes : 0,1/0,3/0,5/0,7/0,9 (10 exécution par probabilité). Ce programme peut être exécuté via l'exécutable PerformanceComparator.jar. Le second programme réalise la moyenne du temps d'exécution des trois algorithmes sur X exécutions

DALENCOURT Alex
GOLDACK Ludovic

pour des graphes aléatoires de N nœuds avec une probabilité P d'arête (X, N et P choisi à l'exécution). Le programme peut être exécuté via l'exécutable PerformanceComparatorArgument.jar.

Java -jar {chemin vers PerformanceComparator.jar}

java -jar {chemin vers PerformanceComparatorArgument.jar} {nombre d'exécutions} {nombre de nœuds} {probabilité d'une arête}

Pour une exécution les trois algorithmes effectuent une coloration sur un même graphe généré.

Résultats :

Les paramètres utilisés sont :

- X : 50

- N : 100

- P : 0,1/0,3/0,5/0,7/0,9

Méthode WelshPowel -> Temps total : 127 ms, Nombre de couleurs total : 1107

Méthode BrighamDutton -> Temps total : 21904 ms, Nombre de couleurs total : 1064

Méthode RLF -> Temps total : 412 ms, Nombre de couleurs total : 1079

Méthode WelshPowel -> Temps moyen : 2 ms, Nombre de couleurs moyen : 22

Méthode BrighamDutton -> Temps moyen : 438 ms, Nombre de couleurs moyen : 21

Méthode RLF -> Temps moyen : 8 ms, Nombre de couleurs moyen : 21

A la vue de ces résultats on peux réaliser les observations suivantes :

- L'algorithme Welsh Powel est le programme le plus rapide en moyenne mais semble utiliser plus de couleurs que les autres algorithmes. Cela se confirme au regard des valeurs totales.

- L'algorithme Brigham Dutton est quand à lui le plus long en moyenne et cela se confirme par le temps total.

- Le nombre moyen de couleurs ne permet pas de distinguer de différences entre RLF et Brigham Dutton. Les valeurs totales ne sont pas les plus fiables car l'écart type des nombres de couleurs trouvé peut être petit pour RLF et ou grand pour Brigham Dutton. Ainsi, l'optimalité d'un ou de l'autre doit être fonction du nombre d'arête.

DALENCOURT Alex
GOLDACK Ludovic

Exécution de nouveaux tests avec les paramètres suivant :

- X : 50

- N : 100

- P : 0,4

Méthode WelshPowel -> Temps total : 125 ms, Nombre de couleurs total : 813

Méthode BrighamDutton -> Temps total : 23094 ms, Nombre de couleurs total : 772

Méthode RLF -> Temps total : 457 ms, Nombre de couleurs total : 779

Méthode WelshPowel -> Temps moyen : 2 ms, Nombre de couleurs moyen : 16

Méthode BrighamDutton -> Temps moyen : 461 ms, Nombre de couleurs moyen : 15

Méthode RLF -> Temps moyen : 9 ms, Nombre de couleurs moyen : 15

Exécution de nouveaux tests avec les paramètres suivants :

- X : 50

- N : 100

- P : 0,9

Méthode WelshPowel -> Temps total : 134 ms, Nombre de couleurs total : 2191

Méthode BrighamDutton -> Temps total : 28077 ms, Nombre de couleurs total : 2090

Méthode RLF -> Temps total : 595 ms, Nombre de couleurs total : 2156

Méthode WelshPowel -> Temps moyen : 2 ms, Nombre de couleurs moyen : 43

Méthode BrighamDutton -> Temps moyen : 561 ms, Nombre de couleurs moyen : 41

Méthode RLF -> Temps moyen : 11 ms, Nombre de couleurs moyen : 43

A la vue de ces derniers résultats on peut déduire qu'en moyenne plus le nombre d'arêtes est grand alors Brigham Dutton donne de meilleurs résultats.